אימון הנוירון - מימוש עם PyTorch

בפרק הקודם מימשנו את תהליך אימון הנוירון מהיסוד, וכעת נראה כיצד בעזרת PyTorch נוכל לעשות זאת באופן תמציתי יותר ובו זמנית באופן יעיל יותר מבחינה חישובית.

ראשית נגדיר את מודל הנוירון היחיד שלנו בעזרת הספרייה `torch.nn` המכילה את כל אבני הבניין של רשתות נוירונים אשר להם נזדקק.

```
from torch import nn
z=nn.Linear(2,1)
y=nn.Sigmoid()
print(z)
```

פלט:

```
Linear(in_features=2, out_features=1, bias=True)
```

בעת יצירת המשתנה z, הפרמטרים w_0, w_1, b אותחלו באקראי ולכן אנו יכולים לחשב את תחזית המודל על אחת מהנקודות באוסף הנתונים שלנו, למשל:

```
y(z(X[0,:]))
```

פלט:

```
tensor([0.1221], grad_fn=<SigmoidBackward0>)
```

כמו כן, ניתן לגשת לפרמטרים של המודל ישירות, שכן הם מאפיינים של z, לבדוק שאכן החישוב המבוצע בקריאה ל-z כפונקציה הוא $z(x_0, x_1) = w_0 x_0 + w_1 x_1 + b$:

```
print(z.weight,z.bias,sep='\n')
assert(z.weight[0,0]*X[0,0]+z.weight[0,1]*X[0,1]+z.bias[0] ==
       z(X[0,:]))
```

פלט:

```
Parameter containing:
tensor([[-0.2000,  0.2818]], requires_grad=True)
Parameter containing:
tensor([-0.0825], requires_grad=True)
```

בנוסף, אם ברצוננו לשנות את ערכי הפרמטרים, ניתן לעשות זאת בנקל, אך יש לזכור שמערכת ה-Autograd פעילה, ועוקבת אחרי הפעולות המתבצעות על פרמטרים אלו בכדי שבעתיד נוכל לקבל את הנגזרות לפיהם באופן אוטומטי. מכיוון ששינוי הפרמטרים אינו חלק מתהליך האימון (ולמעשה הוא אף אינו פעולה גזירה!), יש להצהיר במפורש שעל פעולה זו להתבצע ללא חישוב נגזרות, אחת הדרכים לעשות זאת היא שימוש ב-Context Manager:

```
with torch.no_grad():
  z.weight[0,0], z.weight[0,1], z.bias[0] = (1, -1, 5.5)
print(z.weight,z.bias,sep='\n')
```

פלט:

```
Parameter containing:
tensor([[ 1., -1.]], requires_grad=True)
Parameter containing:
tensor([5.5000], requires_grad=True)
```

כל הפעולות אשר מבוצעות בתוך ההקשר של `no_grad()` מבוצעות ללא מערכת הגזירה האוטומטית.

כעת נריץ את כל הנתונים קדימה ברשת, ונקבל את תחזית המודל עבורם בשורת קוד אחת, וכך גם את פונקציית המחיר שלנו (שימו לב שאנו משתמשים כאן באופרטורים שהורמו לפעול איבר איבר על טנזורים):

```
y_model = torch.squeeze(y(z(X)))
CE_loss = -1/len(Y)*torch.sum(Y*y_model+(1-Y)*(1-y_model))
print(CE_loss)
```

פלט:

```
tensor(-0.5340, grad_fn=<MulBackward0>)
```

אחרי שחישבנו את פונקציית המחיר, נחשב את הגרדיאנט שלה לפי הפרמטרים. כאן בא לידי ביטוי היתרון הגדול ביותר של PyTorch: כל שיש הוא להריץ את הפקודה `backward()` ולקבל את הנגזרות באופן אוטומטי.

```
CE_loss.backward()
print(z.weight.grad,z.bias.grad,sep='\n')
```

פלט:

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

נשאר רק לעדכן את ערכי הפרמטרים על ידי חיסור הערך $\alpha\nabla C$ (שוב ללא מערכת הגזירה האוטומטית), ולחזור על התהליך עד להתכנסות. אם כן, ליבת לולאת האימון נראית כך:

```
z.zero_grad()
y_model = torch.squeeze(y(z(X)))
CE_loss = -1/len(Y)*torch.sum(Y*torch.log(y_model)+(1-
Y)*torch.log(1-y_model))
CE_loss.backward()
with torch.no_grad():
  z.weight -= alpha*z.weight.grad
  z.bias   -= alpha*z.bias.grad
```

זכרו שלפני כל איטרציה יש לאפס את הגרדיאנט, שכן ברירת המחדל של `backward()` היא הוספת הערכים המחושבים באיטרציה זו לאלו הקיימים.

שאלות לתרגול

1. כתבו את לולאת האימון, הריצו אותה והשוו את התוצאות למימוש הקודם.
   א. האם מתקבלות תוצאות זהות?
   ב. איזה קוד מהיר יותר? בדקו בעזרת פקודת ה-Magic `%%timeit`.