גזירה אוטומטית

בפרק זה נכיר את אחד הכלים השימושיים ביותר בספרייה PyTorch – הגזירה האוטומטית. כלי זה, הממומש בחבילה Autograd של הספרייה, מאפשר לנו להצהיר מראש על טנזורים כ"משתנים" אשר בהמשך נרצה לגזור פונקציות שלהם וזאת על ידי שינוי המאפיין `requires_grad` לערך אמת (לרוב בעת יצירת הטנזור) כלהלן:

```
x = torch.tensor([2.], requires_grad=True)
```

לאחר הצהרה זו, הספרייה תתחיל לעקוב אחרי כל פעולת חשבון שמבוצעת על הטנזור, ולבסוף כאשר ברצוננו לחשב את הנגזרת של פונקציה כלשהי ביחס ל-`x`, כל שעלינו הוא להריץ את המתודה `backward()` והנגזרת תחושב באופן אוטומטי ותישמר במאפיין `grad` של הטנזור. ראו למשל:

```
y=x**2
z=y+x
z.backward()
print(x.grad)
```

פלט:

```
tensor([5.])
```

לפני המשך הקריאה, וודאו שאתם מבינים כי הקשר הפונקציונלי בין `z` ל-`x` הוא $z = x^2 + x$, לכן הנגזרת היא $\frac{dz}{dx} = 2x+1$ ובהצבת הסקלר 2 ב-`x` אכן מתקבל הערך 5.

פונקציונליות זו עובדת כמובן גם ביחס לנגזרות חלקיות כפי שניתן לראות בשתי הדוגמאות הבאות:

דוגמה 1

```
x = torch.tensor([2.,3.],requires_grad=True)
y = x[0]*x[1]
y.backward()
print(x.grad)
```

פלט:

```
tensor([3., 2.])
```

בדוגמה זו אנו רואים כי הקשר הפונקציונלי בין `y` לבין **הוקטור** `x` הוא $y = x_0 \cdot x_1$ ובמאפיין `grad` נשמר **הגרדיאנט** $\nabla y = \left(\frac{\partial y}{\partial x_0}, \frac{\partial y}{\partial x_1}\right)$.

דוגמה 2

```
x = torch.tensor([2.,3.],requires_grad=True)
y = torch.tensor([4.,5.],requires_grad=True)
w=x.sum()**2
z=w+y.sum()**3+x[0]+y[1]
z.backward()
print(x.grad,y.grad,sep='\n')
```

פלט:

```
tensor([11., 10.])
tensor([243., 244.])
```

בדוגמה זו אנו עוקבים אחרי פעולות החשבון שמבוצעות על שני טנזורים שונים **במקביל**, ובעת הרצת המתודה `backward()`, מחושב במקביל הגרדיאנט של `z` ביחס לכל אחד מהם. שימו לב לקשר הפונקציונלי בין הטנזורים המופיעים בדוגמה,

$$z=\left(x_0+x_1\right)^2+\left(y_0+y_1\right)^3+x_0+y_1$$

ובהתאם, ודאו כי אתם מבינים את חישוב הנגזרות:

$$\frac{\partial z}{\partial x}=\left(\frac{\partial z}{\partial x_0},\frac{\partial z}{\partial x_1}\right)=\left(2\left(x_0+x_1\right)+1,2\left(x_0+x_1\right)\right)$$

$$.\frac{\partial z}{\partial y}=\left(\frac{\partial z}{\partial y_0},\frac{\partial z}{\partial y_1}\right)=\left(3\left(y_0+y_1\right)^2,3\left(y_0+y_1\right)^2+1\right)$$

לעתים נרצה לבצע חישוב על טנזורים אחריהם אנו עוקבים ולהשתמש בתוצאה **כקבוע**, ולא כפונקציה של הטנזורים המקוריים, אותה יהיה לגזור בעת הרצת `backward()`. למטרה זו ניתן להשתמש במתודה היוצרת עותק של הטנזור המקורי, אך ללא מעקב אחרי פעולות החשבון: מבחינת מערכת ה-Autograd, טנזור התוצאה יהיה קבוע. השוו את הדוגמה הבאה לקודמת, וודאו כי אתם מבינים שלמעשה כעת מתקיים

$$z=s+\left(y_0+y_1\right)^3+x_0+y_1$$

ללא התחשבות בעובדה ש-`s` חושב כפונקציה של הטנזור `x`.

```
x = torch.tensor([2.,3.],requires_grad=True)
y = torch.tensor([4.,5.],requires_grad=True)
w=x.sum()**2
s=w.detach()
z=s+y.sum()**3+x[0]+y[1]
z.backward()
print(x.grad,y.grad,sep='\n')
```

פלט:

```
tensor([1., 0.])
tensor([243., 244.])
```

לסיום, נשים לב שמערכת הגזירה האוטומטית עוקבת **בעת הריצה** אחרי פעולות החשבון המבוצעות על הטנזורים. בהתאם, אם בכל ריצה מחושבת פונקציה שונה, למשל על ידי החלטות שונות של בקרת הזרימה, רק הפעולות שבוצעו בפועל הן אלו שיילקחו בחשבון בעת הגזירה. ראו בדוגמאות הבאות כיצד אנו גוזרים "דרך" תנאים ולולאות באין מפריע:

דוגמה 3

```
y = torch.zeros(10,dtype=int)
x = torch.tensor([2.],requires_grad=True)
for i in range(10):
  z=2*x
  while torch.rand(1)<0.7:
    z=2*z
  z.backward()
  y[i]=x.grad
  x.grad=None
print(y)
```

פלט:

```
tensor([ 8, 32,  2,  4, 64,  2, 64, 16,  2, 32])
```

שימו לב שבקטע קוד זה הקשר הפונקציונלי בין z ל-x תלוי בהגרלת מספרים אקראיים: $z = 2^{1+k} x$ כאשר k הוא מספר הפעמים שהיה עלינו להגריל מ"מ אחידים סטנדרטיים עד שהתקבל אחד שערכו גדול מספיק. בהתאם בכל איטרציה של הלולאה החיצונית מתקבל קשר אחר ולכן גם ערכי הנגזרת שונים לבסוף.

כמו כן, שימו לב לפקודה `x.grad=None`. כל קריאה למתודה `backward()` מוסיפה את הערך שחושב לזה הקיים במאפיין `x.grad`, **מבלי לאפס אותו קודם לכן**. בהתאם, פלט טיפוסי של הקוד הנ"ל, ללא פקודה זו ייראה כך:

```
tensor([  2,  18,  20,  52,  84, 148, 150, 406, 414, 422])
```

דוגמה 4

כעת נראה שאותו עקרון כנ"ל תקף גם ביחס לתנאי if-else: בכל איטרציה הגזירה מתבצעת ביחס לענף הנבחר בבקרת הזרימה.

```
y = torch.zeros(5)
for i in range(5):
  x = torch.randn(1,requires_grad=True)
  if x>0:
     z=torch.exp(x)
  else:
      z=-x
  z.backward()
  y[i]=x.grad
print(y)
```

פלט:

```
tensor([ 1.1121, -1.0000, -1.0000,  2.4863, -1.0000])
```

שאלות לתרגול

1. הריצו את דוגמה 1 בפרק זה, ומיד לאחר הריצה - קראו שוב למתודת `backward()`. האם תוכלו להסביר את מקור השגיאה המתקבלת?
2. הריצו שוב את דוגמה 1 אך כעת בעת הקריאה הראשונה למתודה `backward()` הוסיפו את הפרמטר `create_graph=True`. כעת הריצו את הקוד הבא,

```
t=x.grad
x.grad=None
t.backward()
```

ונסו להסביר מהי התוצאה המתקבלת אחרי כן במשתנה `x.grad`.