אופרטורים טנזוריים

אחד היתרונות של שימוש בטנזורים הוא האפשרות לבצע פעולות איבר-איבר על כל הטנזור באופן חסכוני ומהיר, לדוגמה:

```
x = torch.tensor([1, 2, 3, 4])
y = torch.tensor([5, 6, 7, 8])
print(x + y)
```

פלט:

```
tensor([ 6,  8, 10, 12])
```

ניתן לראות כי הפעולה `x + y` הניבה טנזור אשר כל איבר שלו הוא סכום האיברים בעלי אותו אינדקס בטנזורים המקוריים. על כך נאמר כי האופרטור "+" עבר **הרמה** לפעול איבר-איבר על טנזורים. רוב הפעולות המתמטיות הבסיסיות הורמו בצורה דומה, ראו למשל (וזכרו כי `**` היא פעולת החזקה):

```
print(x - y, x * y, x / y, x**y, sep='\n')
```

פלט:

```
tensor([-4, -4, -4, -4])
tensor([ 5, 12, 21, 32])
tensor([0.2000, 0.3333, 0.4286, 0.5000])
tensor([    1,    64,  2187, 65536])
```

מלבד היתרון של תמציתיות הכתיבה והקריאות של השימוש באופרטורים מורמים, לרוב השימוש בהם מהיר בסדרי גודל על פני מימוש נאיבי של הפעולה בעזרות לולאות. ראו למשל את ההשוואה הבאה, המשתמשת בפקודת ה-magic `%%timeit` למדידת זמן הריצה של תא במחברת ג'ופיטר. לאחר יצירת שני טנזורים חד ממדיים בעלי 10,000 איברים,

```
x = torch.ones(size=(10**5,))
y = torch.randint(low=0,high=10,size=(10**5,))
```

נתזמן את פעולת הכפל איבר-איבר בשתי דרכים שונות:

1. פעולת הכפל איבר-איבר הטנזורית:

```
%%timeit
x*y
```

תוצאה טיפוסית של הרצת תא זה היא כ200 מיקרו-שניות.

2. ביצוע פעולת הכפל בעזרת לולאה:

```
%%timeit
for i in range(x.numel()):
  x[i]*y[i]
```

תוצאה טיפוסית של הרצת תא זה היא כ600 מילי-שניות, פי 3000 מהתוצאה הקודמת!

ההבדל בזמן הריצה אינו ספציפי עבור פעולת הכפל, ואף יש פעולות מורמות אחרות אשר הפער עבורן גדול יותר. הסיבות לכך חורגות מתחום הדיון הנוכחי, אך עלינו לזכור שלמען מהירות זמן הריצה של הקוד, נעדיף להשתמש בפעולות טנזוריות ככל האפשר. למזלנו, לא רק הפעולות

האריתמטיות הבסיסיות הורמו לטנזורים, גם אופרטורים בוליאניים וכן אופרטורים המוכרים לנו מאלגברה לינארית כגון כפל מטריצות וחישוב מכפלות פנימיות וחיצוניות. למשל:

```
x = x.reshape(1,4)
y = y.reshape(4,1)
print(x,y,x@y,y@x,sep='\n')
```

פלט:

```
tensor([[1, 2, 3, 4]])
tensor([[5],
        [6],
        [7],
        [8]])
tensor([[70]])
tensor([[ 5, 10, 15, 20],
        [ 6, 12, 18, 24],
        [ 7, 14, 21, 28],
        [ 8, 16, 24, 32]])
```

שימו לב שכאשר שני הטנזורים הם בעלי ממד אחד, האופרטור "@" מחשב את המכפלה הפנימית שלהם, אך אם נשנה אותם לווקטור עמודה ווקטור שורה דו ממדיים, נוכל לקבל את תוצאת כפל המטריצות על ידי שימוש באותו אופרטור:

```
x = torch.tensor([1, 2, 3, 4])
y = torch.tensor([5, 6, 7, 8])
print(x@y,x==1,sep='\n')
```

פלט:

```
tensor(70)
tensor([ True, False, False, False])
```

סוג שימושי נוסף של אופרטורים הפועלים על טנזורים הם מתודות הפחתה – מתודות של הטנזור המחזירות סקלר, לרוב מדדי סיכום כגון ערכי מינימום, מקסימום, ממוצע או נורמה. דוגמאות ספורות לכך ניתן לראות כאן:

```
x =  torch.ones(size=(3,))
y =  torch.ones(size=(3,3))
z =  torch.ones(size=(3,3,3))
print(x.norm(),y.min(),z.sum(),sep='\n')
```

פלט:

```
tensor(1.7321)
tensor(1.)
tensor(27.)
```

יש לשים לב שמתודות ההפחתה תמיד מחזירות סקלר ללא תלות במימדי הטנזור המקורי. יחד עם זאת, ניתן להעביר את אחד המימדים כפרמטר, ההפחתה תתבצע רק לאורך מימד זה, ויוחזר טנזור קטן במימד אחד מטנזור הקלט. ראו למשל:

```
x =  torch.arange(24).reshape(2,3,4)
print(x,x.size())

y = x.sum()
print(y,y.size())

z = x.sum(axis=0)
print(z,z.size())

w = x.sum(axis=1)
print(w,w.size())
```

פלט:

```
tensor([[[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11]],

        [[12, 13, 14, 15],
         [16, 17, 18, 19],
         [20, 21, 22, 23]]]) torch.Size([2, 3, 4])
tensor(276) torch.Size([])
tensor([[12, 14, 16, 18],
        [20, 22, 24, 26],
        [28, 30, 32, 34]]) torch.Size([3, 4])
tensor([[12, 15, 18, 21],
        [48, 51, 54, 57]]) torch.Size([2, 4])
```

שימו לב בדוגמה זו שהסכום ב-`z` התבצע לאורך המטריצות (מימד 0) והתקבלה מטריצה אחת מאותו מימד כמו השתיים בטנזור המקורי, בעוד שב-`w` הסכום התבצע על עמודות המטריצות (מימד 1 של `x`) המקוריות, וכל שורה במטריצת הפלט מייצגת את אחת מהן.

בדומה לקונסטרקטורים, אין צורך לזכור בעל פה את כל האופרטורים שניתן להפעיל על טנזורים, אך לפני מימוש נאיבי פעולה מתמטית או לוגית סטנדרטית בעזרת לולאות, חיוני לבדוק בתיעוד האם כבר קיים אופרטור מובנה יעיל. לרוב התשובה תהיה חיובית ותחסוך זמן תכנות וזמן ריצה.

שאלות לתרגול

1. צרו טנזור תלת מימדי מגודל `Size=[100, 2, 2]` אשר מכיל מספרים אקראיים מהתפלגות אחידה סטנדרטית..
   א. חשבו את הערכים העצמיים של כל המטריצות מסדר 2X2 המתקבלות עבור כל ערך של המימד הראשון. מה תוכלו להגיד על הערך המדומה של הערכים העצמיים של המטריצות? האם תוכלו להסביר תופעה זו?:
   ב. חשבו את נורמת פרובניוס של כל אחת מהמטריצות.
   ג. חשבו את הנורמה האוקלידית של הטנזור המקורי, כאשר כל ערכיו נלקחים בחשבון.
   ד. חשבו את נורמת אינסוף של שתי המטריצות מסדר 100X2 המתקבלות עבור כל אחד מערכי המימד האחרון.

   הערה: היעזרו בתיעוד הספרייה `torch.linalg` למציאת הפונקציות המתאימות, ונסו לבצע כל חישוב בעזרת שורת קוד אחת.

2. השוו את זמן הריצה של פעולת החילוק והאקספוננט אשר הורמו לפעול על טנזורים, לחישוב המקביל בעזרת לולאה. בצעו השוואה זו בעזרת הפקודה `%%timeit`. האם תוכלו להסביר את ההבדלים בין שתי הגישות?