חסכון בזיכרון

בהמשך הקורס נשתמש בטנזורים גדולים מאוד, ועל כן פעולות של הקצאת זיכרון וכתיבה אליו יהוו צוואר בקבוק שננסה להתחמק ממנו ככל האפשר. לרוב זה יתבטא בשתי גישות שונות:

1. כאשר אפשר, למשל כשאיננו רוצים לשנות את ערכי המשתנים, ניצור משתנים חדשים בעלי **מבט (view)** אל הנתונים של משתנה אחר, **במקום להעתיקם מחדש**. משתנים אלו יקבלו גישה אל המקום בזיכרון בו שמורים הנתונים של המשתנה הקודם.
2. ככל האפשר ננסה **לבצע פעולות במקום (in-place)** – תוצאות חישוב על טנזור נתון יישמרו באותו מקום בזיכרון בו שמורים הנתונים הקודמים של הטנזור.

יש בספרייה PyTorch כלים שיעזרו לנו לממש גישות אלו, ולמעשה במקרים מסוימים הן מתממשות באופן אוטומטי. למשל, כאשר אנו לוקחים חתך של טנזור ומציבים אותו בטנזור חדש, מתקבל מבט, שכן אין צורך להעתיק את הנתונים:

```
x = torch.arange(2*4*5).reshape(shape=(2,4,5))
y = x[:,0::2,2:4]
print(x,y,sep='\n')
```

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, 24],
         [25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34],
         [35, 36, 37, 38, 39]]])
tensor([[[ 2,  3],
         [12, 13]],

        [[22, 23],
         [32, 33]]])
```

בדוגמה זו חתכנו את `x` כדי לקבל את האיברים אשר הממד השני שלהם (שורות המטריצה) הוא 0 או 2 והממד השלישי (עמודות המטריצה) הוא 2,3 או 4. ודאו שאתם מבינים זאת ואם לא – חזרו על נושא האינדקסים קודם תמשיכו. את החתך הצבנו במשתנה `y` אך עלינו להיות מודעים לכך ש-`x` ו-`y` חולקים את הנתונים, וכן ששינוי באחד יוביל לשינוי בשני:

```
y[-1]=100
print(y,x,sep='\n')
```

פלט:

```
tensor([[[  2,   3,   4],
         [ 12,  13,  14]],

        [[100, 100, 100],
         [100, 100, 100]]])
tensor([[[  0,   1,   2,   3,   4],
         [  5,   6,   7,   8,   9],
         [ 10,  11,  12,  13,  14],
         [ 15,  16,  17,  18,  19]],

        [[ 20,  21, 100, 100, 100],
         [ 25,  26,  27,  28,  29],
         [ 30,  31, 100, 100, 100],
         [ 35,  36,  37,  38,  39]]])
```

מצב זה לרוב רצוי, שכן כברירת המחדל אנו מעוניינים בחסכון בזיכרון. עם זאת, הוא יכול להוביל לשגיאות לא צפויות. כאשר נרצה בכל זאת להעתיק את הנתונים למקום חדש, ולנתק את הקשר בין המשתנים, נעשה זאת בעזרת המתודה `clone()` אשר יכולה לפעול על כל טנזור. בהרצת שורת הקוד הבאה נקבל בטנזור `z` עותק חדש של הנתונים, ושינוי ערכיו לא ישפיע על ערכי `x` או `y`.

```
z = x[:,0::2,2:5].clone()
```

עלינו לשים לכך שכאשר אנו משתמשים באינדקס בוליאני, התוצאה המתקבלת היא תמיד עותק. ראו בדוגמה הבאה כיצד שינוי ב`y` אינו משפיע על הטנזור `x` וזאת למרות שהראשון נחתך מהשני.

```
x = torch.arange(5)
index = (x>=2) & (x<4)
y = x[index]
y[-1]=0
print(x,index,y,sep='\n')
```

פלט:

```
tensor([0, 1, 2, 3, 4])
tensor([False, False,  True,  True, False])
tensor([2, 0])
```

כעת נעבור לדון בגישה השנייה לחסכון בזיכרון, ביצוע פעולות החשבון במקום (in-place), ודרך זאת מחזור מקום בזיכרון שכבר הוקצה בעבר. בדוגמה הבאה הממחישה את הבעיה, נשתמש בפונקציה `id()` המחזירה את כתובת הזיכרון של נתוני הטנזור.

```
x = torch.ones(5)
address = id(x)
x = torch.exp(x)
print(address == id(x))
```

פלט:

```
False
```

ניכר שהפעלת האקספוננט על x דרשה הקצאה של מקום חדש בזיכרון, ופעולת ההצבה של תוצאת החישוב בx עצמו, היא למעשה שינוי המצביע לנתונים של הטנזור אל המקום החדש בזיכרון, ובכך נוצר "בזבוז" של מקום.

הגישה המקובלת להתמודדות עם בעיה זו היא לסמוך על האופטימיזציות הקיימות הספרייה PyTorch ולהימנע מפתרונה באופן ישיר, שכן כך עלולה להיווצר התנגשות עם אחת המערכת היסודיות של הספרייה, הגזירה האוטומטית, עליה נלמד בהמשך. למרות זאת, באפשרותנו להכתיב את דרך הפעולה ישירות בשתי דרכים שונות. ראשית אפשר להציב ישירות את תוצאת החישוב לתוך המקום הישן בזיכרון, על ידי שימוש בחיתוך:

```
address = id(x)
x[:] = torch.exp(x)
print(address == id(x))
```

פלט:

```
True
```

שנית, אפשר להשתמש בפונקציות של PyTorch המובנות לבצע את החישוב במקום, ברוב המקרים פונקציות אלו יהיו בעלות שמות זהים לפונקציות המקוריות עם קו תחתון יחיד בסוף השם, למשל הפקודה `torch.exp_(x)` תבצע את פעולת האקספוננט על ערכי הטנזור ובנוסף תציב את הערכים החדשים במקום המקורי בזיכרון, כפי שניתן לראות בדוגמה הבאה.

```
x = torch.ones(5)
print(x)
torch.exp_(x)
print(x)
```

פלט:

```
tensor([1., 1., 1., 1., 1.])
tensor([2.7183, 2.7183, 2.7183, 2.7183, 2.7183])
```

שאלות לתרגול

1. הפכו את הטנזור x הנ"ל לוקטור עמודה וחברו אותו לעצמו תוך שמירת התוצאה באותו המקום בזיכרון.