אימון רשת בעזרת מאיץ גרפי

חלק נכבד מהפעולות החישוביות אשר אנו מבצעים בעת אימון רשת נוירונים ניתנות לביצוע במקביל. הראשונה שבהן, אך לא היחידה, היא כפל מטריצות, הבאה לידי ביטוי, בין השאר, בשכבות ליניאריות. נניח שוקטור הקלט לשכבה ליניארית נתונה הוא $X=(x_1,...,x_d)$, אז בעת פעפוע קדימה ברשת מתבצע החישוב הבא (עבור דגימה יחידה),

$$\begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{md} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = \begin{pmatrix} \sum_k w_{1k} x_k \\ \vdots \\ \sum_k w_{mk} x_k \end{pmatrix},$$

ועבור minibatch (בגודל N),

$$\begin{pmatrix} z_1^{[1]} & \cdots & z_1^{[N]} \\ \vdots & \ddots & \vdots \\ z_m^{[1]} & \cdots & z_m^{[N]} \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{md} \end{pmatrix} \begin{pmatrix} x_1^{[1]} & \cdots & x_1^{[N]} \\ \vdots & \ddots & \vdots \\ x_d^{[1]} & \cdots & x_d^{[N]} \end{pmatrix}$$

ראו כי עבור כל איבר במטריצת הפלט יש לחשב את הביטוי

$$z_a^{[b]} = \sum_k w_{ak} x_a^{[b]},$$

ולכן למעשה, כל ביטוי מהצורה $w_{ak} x_a^{[b]}$ ניתן לחישוב במקביל בליבה נפרדת ולבסוף אפשר לסכום את התוצאות לקבלת הפלט. יש לציין שקיימים אלגוריתמים מתקדמים לכפל המטריצות, וחבילות התוכנה בהן אנו משתמשים ממשות אותם, אך גם הם ניתנים למימוש במקביל בדומה לחישוב הנאיבי.

אם כן, יש לנו צורך ברכיב חומרה בעל מספר רב של ליבות המסוגלות לבצע פעולות פשוטות (כגון כפל שני סקלרים). מאיץ גרפי הוא בדיוק רכיב שכזה ולכן היווה מועמד טבעי להאצת החישובים של מודלי למידה עמוקה. כך אכן היה באופן היסטורי: רשתות הנוירונים הראשונות אשר ניצחו את המתחרים בתחרויות למידת מכונה כגון ImageNet, ובישרו את עלייתה של הלמידה העמוקה לבכורה בעשור השני לשנות האלפיים אומנו על גבי מאיצים גרפיים. מאז פותחו רכיבים ייחודיים להאצת החישובים הנפוצים במימוש רשתות נוירונים, כן ממשקי הפשטה לעבודה עם רכיבים אלו, כך שכיום באופן כמעט מוחלט תהליך האימון (וכן לעתים גם החיזוי) מבוצע במקביל.

השימוש במאיץ גרפי בעזרת PyTorch פשוט. ראשית יש לבדוק שבסביבה בה עובדים מותקן רכיב תואם CUDA, זהו הממשק בו ממומשת יכולת זו ב-PyTorch. ניתן לעשות זאת בעזרת פקודת ה-shell `nvidia-smi`, אשר תחזיר פירוט של המאיצים הזמינים, או דרך PyTorch ישירות:

```
torch.cuda.is_available()
```

פלט:

```
True
```

מספר שירותי ענן מספקים סביבות עבודה עם מאיצים גרפיים, ביניהם Colab של גוגל: היכנסו לתפריט Edit בחרו באפשרות Notebook settings וסמנו ב-GPU את מאיץ החומרה המבוקש.

כעת, כשיש יותר מרכיב חומרה יחיד זמין, עלינו להיות מודעים למאפיין `device` הקיים בכל טנזור. מאפיין זה מציין את הרכיב עליו ממוקם הטנזור. ברירת המחדל היא המעבד הראשי (CPU):

```
A=torch.empty(1)
print(A.device)
```

פלט:

```
cpu
```

ביכולתנו להעביר טנזורים מרכיב אחד לאחר בעזרת המתודה `to()`, ראו להלן.

```
device = torch.device('cuda:0')
A=A.to(device)
print(A.device)
```

פלט:

```
cuda:0
```

שימו לב ששם המאיץ הגרפי (הראשון/ היחיד בסביבת העבודה) הוא `cuda:0`. כמו כן, שימו לב שהעברת המשתנה מהמעבד הראשי אל המאיץ היא פעולה הכרוכה בהעתקת נתוני המשתנה מזכרון המעבד אל זכרון המאיץ, ולכן התוצאה של המתודה `to()` הוא תמיד **עותק** של המשתנה המקורי.

כברירת מחדל, תוצאת פעולת חשבון המבוצעת על שני טנזורים הנמצאים באותו רכיב – נשמרת גם היא ברכיב זה. למשל,

```
A=torch.zeros(1,device=device)
B=torch.ones(1,device=device)
C=A+B
print(C.device)
```

פלט:

```
cuda:0
```

ראו בנוסף שניתן להגדיר את ה-`device` של הטנזור כבר בשלב האתחול.

מכיוון שהעתקת נתונים מרכיב חומרה אחד לאחר היא פעולה איטית מאוד, היא אינה מתבצעת באופן אוטומטי אלא רק לאחר הוראה מפורשת של המשתמש. לכן, אם ננסה לבצע פעולת חשבון בין שני טנזורים הנמצאים על רכיבים שונים תתקבל שגיאה. ראו למשל:

```
A=torch.zeros(1,device=device)
B=torch.ones(1,device='cpu')
C=A+B
```

פלט:

```
RuntimeError: Expected all tensors to be on the same device,
but found at least two devices, cuda:0 and cpu!
```

לא רק טנזורים אלא גם מודלים שלמים (כגון כאלו אשר יצרנו בעזרת הפקודה `nn.Sequential`) ניתן להעביר אל המאיץ הגרפי, שכן גם להם קיימת המתודה `to()`.

הפרט האחרון שצריך לקחת בחשבון הוא שספריות פייתון אחרות כגון NumPy או matplotlib אינן עובדות עם המאיץ באופן מובנה. לכן קודם לשימוש בהן, למשל לצורך ציור גרף, עלינו להעתיק את הקלט שלהן בחזרה ל-CPU. זאת אפשר לבצע בעזרת המתודה `tensor.cpu()`. אם הטנזור כבר נמצא על המעבד הראשי, לא מתבצעת העתקה נוספת ללא צורך ולכן ניתן להשתמש במתודה זו באופן חופשי.

לסיכום, בכדי לאמן מודל על גבי המאיץ הגרפי כל שיש לעשות הוא להעביר אליו את המודל לאחר הגדרתו, וכן להעביר אליו כל minibatch של נתונים, לאחר טעינתו. שאר לולאת האימון אינה משתנה. שימו לב שכוחו של המאיץ הגרפי בא לידי ביטוי היכן שפעולות החשבון הן צוואר הבקבוק של תהליך האימון: כאשר רשת הנוירונים גדולה מספיק. עבור רשתות קטנות, ייתכן שהשימוש במאיץ **אף יאט** את תהליך האימון, עקב המחיר של העתקת הנתונים מרכיב אחד לרכיב אחר.

שאלות לתרגול

1. שנו את הקוד לאימון רשת הנוירונים לסיווג התמונות של אוסף הנתונים Fashion-MNIST שכתבתם בעבר כך שראשית לאימון תבדקו האם קיים בסביבה מאיץ גרפי, ואם הוא קיים – הרשת תאומן על גבי המאיץ. **הנחיות:** אין צורך לכתוב את לולאת האימון מחדש. מספיק להגדיר בסביבה משתנה `device` בעל ערך המתאים לרכיבים הזמינים, ולהשתמש במתודה `to(device)`.
2. השוו את זמני האימון של הרשת על גבי ה-CPU ועל גבי ה-GPU עבור ארכיטקטורות רשת שונות – רשת קטנה (מעט פרמטרים) ורשת גדולה (הרבה פרמטרים).
3. בנוסף לחישוב כפל המטריצה במעבר קדימה דרך שכבה ליניארית, פעולה זו מבוצעת גם בפעפוע לאחור. כתבו בפירוט את המטריצות המשתתפות בפעולה זו. הניחו שגרדיאנט פונקציית המחיר לפי פלט השכבה, עבור כל דגימה ב-minibatch הנתון, חושב זה מכבר.