עיבוד מקדים של נתוני טקסט

ביחידה זו נלמד על ארכיטקטורות רשת מיוחדות עבור נתונים סדרתיים, כגון מדידות של אוסף משתנים בפרקי זמן קבועים (למשל מדידה יומית של טמפרטורה ולחות), או משפטים בשפה טבעית – בהם כמובן יש לסדר הופעת המילים השפעה על משמעות המשפט. כדוגמה חשבו על ההבדל בין שני המשפטים הבאים, המשתמשים באותן המילים, רק בסדר שונה:

"טוב מאוד, לא רע"
"רע מאוד, לא טוב"

אם ברצוננו לאמן אלגוריתם לזהות את הרגש הבא לידי ביטוי בשני המשפטים הנ"ל (חיובי/שלילי), הוא יהיה חייב ללמוד שסדר הופעת המילים במשפט משפיע על הרגש המובע. רשת בעלת קישוריות מלאה המקבלת כקלט את המשפט כולו תוכל ללמוד את החשיבות של הסדר, אך לכך ידרשו דוגמאות רבות ותהליך אימון סבוך. על כן, כמו ביחידת הלימוד הקודמת, שם תיכננו מראש רשתות המיועדות למשימות עיבוד תמונה, ביחידה זו נתכנן רשתות הלוקחות בחשבון באופן מובנה את מימד הסדר של הנתונים, אלו רשתות נשנות (Recurrent Neural Networks – RNN). הדוגמה אשר תלווה אותנו היא אוסף משפטים בשפה האנגלית, המביעים רגש שלילי או חיובי, ובפרק הנוכחי נלמד על עבודת ההכנה הדרושה בכדי להזינם כקלט לרשת נוירונים.

אוסף הנתונים

מאגר המשאבים GLUE מורכב מתשעה אוספי נתונים שונים בתחום עיבוד השפה הטבעית המשמשים לבדיקת ביצועיהם של אלגוריתמי עיבוד שפה טבעית (Natural Language Processing – NLP) במגוון משימות שונות. אחת המשימות היא זיהוי הרגש (שלילי או חיובי) המובע במשפט נתון, ואוסף הנתונים הנבחר למשימה זו הוא SST2: אוסף של כ-70 אלף משפטים המתוייגים לשתי מחלקות: 0 – רגש שלילי, 1 – רגש חיובי.

אוסף נתונים זה (ורבים אחרים), ניתן לטעון בעזרת הספרייה Datasets, אך ראשית יש להתקין אותה בסביבת העבודה, שכן היא לא מותקנת כברירת מחדל בהפצות הסטנדרטיות של פייתון. לשם כך, הריצו את הפקודה הבאה.

```
pip install datasets
```

לאחר ההתקנה ניתן לייבא את הספרייה, ובשורת קוד אחת לטעון את הנתונים הרצויים.

```
import datasets as ds
dataset = ds.load_dataset("glue", "sst2")
```

מתוך אוסף הנתונים ניקח את 67 אלף המשפטים המשמים לאימון, ונטען לזכרון את המשפטים והסיווג שלהם כרשימות סטנדרטיות של פייתון.

```
sentence_list=ldataset["train"]["sentence"]
labels_list=dataset["train"]["label"]
```

כעת נדפיס משפט אחד וסיווגו, על מנת לוודא שטעינת הנתונים בוצעה כהלכה.

```
print(sentence_list[1],labels_list[1],sep="\n")
```

פלט:

```
contains no wit , only labored gags
0
```

עיבוד מקדים

לפני הזנת הנתונים לאלגוריתם הלמידה, עלינו לבצע עליהם עיבוד מקדים. בעוד שנתונים ויזואליים כגון תמונות יכולנו להזין ישירות לרשתות קונבולוציה, זה אינו המצב עבור רשתות נשנות: עלינו להמיר את המשפטים לטנזורים מספריים בצורה שיטתית, כאשר השלב הראשון בתהליך הוא טוקניזציה (tokenization) של המשפט: חלוקתו לרכיבים קטנים. הגישה הבסיסית ביותר למשימה זו היא חלוקת המשפט למילים לפי סימן הרווח המופיע בהן, כלהלן

```
sentence = sentence_list[1]
print(sentence, type(sentence))
print(sentence.split())
```

פלט:

```
contains no wit , only labored gags  <class 'str'>
['contains', 'no', 'wit', ',', 'only', 'labored', 'gags']
```

שימוש במתודה `split` על משתנה מטיפוס `str` יניב רשימה של משתנים מאותו טיפוס – אלו הטוקנים (tokens) של המשפט המקורי.

השלב הבא בתהליך העיבוד הוא יצירת אוצר המילים: איסוף כל הטוקנים באוסף הנתונים, ומיפוים למספרים סידוריים באופן חד-חד ערכי. לצורך זו נשתמש בספרייה torchtext.

```
from torchtext.vocab import build_vocab_from_iterator
tokenizer = lambda x: x.split()
tokenized = list(map(tokenizer, sentence_list))

vocab = build_vocab_from_iterator(tokenized)
print(vocab(sentence.split()))
print(vocab("one two three".split()))
```

פלט:

```
[2923, 60, 329, 1, 88, 1992, 548]
[28, 128, 356]
```

ראו כיצד בעזרת הפונקציה `map` ביצענו טוקניזציה לכל המשפטים בשורת קוד אחת. המשפטים הועברו לבנאי אוצר המילים המובנה, והפלט שהתקבל ממנו הוא מקודד טוקנים למספריהם. כך ניתן לראות למשל שהמספר 28 מייצג את המילה "one".

מכיוון שכל מספר שייך למילה אחת בלבד, ניתן לפרש קידוד נתון בחזרה לטוקנים המתאימים. בדוגמה הבאה אנחנו מפרשים את סדרת עשרת המספרים הראשונים בחזרה לטוקנים, אלו הם הטוקנים הנפוצים ביותר באוסף הנתונים.

```
print(vocab.get_itos()[0:10]) # integer to string
```

פלט:

```
['the', ',', 'a', 'and', 'of', '.', 'to', "'s", 'is', 'that']
```

אוצר המילים נבנה על בסיס סט נתוני האימון, אך לאחר אימון האלגוריתם נרצה להפעיל אותו על קלט כלשהו, לאו דווקא כזה המורכב מאוצר המילים הקיים. במצב הנוכחי, אם ננסה לבצע טוקניזציה והמרה למספרים סידוריים למשפט שמכיל מילים חדשות נקבל שגיאה.

```
vocab("hello world".split())
```

פלט:

```
RuntimeError: Token hello not found and default index is not
set
```

על כן, נוסיף טוקן מיוחד עבור מילים מחוץ לאוצר המילים, ובאותה הזדמנות נמחק ממנו מילים נדירות מאוד, שכן אוצר מילים גדול מאוד יכביד על תהליך הלמידה.

```
vocab=build_vocab_from_iterator(tokenized,
                                specials=["<UNK>"],min_freq=5)
vocab.set_default_index(0)
vocab("hello world".split())
```

פלט:

```
[0, 152]
```

כעת אנו רואים כי למרות שהמילה "hello" לא נמצאת באוצר המילים, לא התקבלה שגיאה – מילה זו מופתה לטוקן של המילים הלא ידועות.

לבסוף, נשאר לנו רק לחלק את כל המשפטים באוסף הנתונים לטוקנים ולהמירם למספרים הסידוריים שלהם באוצר המילים.

```
stoi = lambda x: torch.tensor(vocab(x)) # string to integer
integer_tokens = list(map(stoi, tokenized))
print(integer_tokens[1])
```

פלט:

```
tensor([2924,   61,  330,    2,   89, 1993,  549])
```

במשתנה `integer_tokens` התקבלה רשימה פייתונית של טנזורים, המכילים מספרים טבעיים: המספרים הסידוריים באוצר המילים של טוקני המשפטים המקוריים.

רשתות נוירונים מטבען פועלות על מספרים **ממשיים** ולא טבעיים. על כן, השלב האחרון בעיבוד הנתונים יהיה להמיר את הטוקנים השלמים לייצוג ממשי: לכל מילה נתאים וקטור ממשי במרחב רב מימדי, זהו שיכון המילה (word embedding).

```
embedding_layer = nn.Embedding(len(vocab),3)
integer=stoi(["word"])
print(integer)
print(embedding_layer(integer))
```

פלט:

```
tensor([1234])
tensor([[ 0.4685,  1.1884, -0.3000]],
grad_fn=<EmbeddingBackward0>)
```

בקוד זה הגדרנו שכבה המבצעת שיכון מילים למרחב תלת מימדי, המרנו את המילה "word" למספר הסידורי שלה במילון: 1234 וחישבנו את השיכון שלו. כאשר במשפט נתון תופיע המילה word, טנזור תלת מימדי זה הוא שיוזן לרשת הנוירונים במקומה.

שימו לב כי מערכת הגזירה האוטומטית עוקבת אחרי השיכונים – אלו הם פרמטרים אשר יילמדו יחד עם אימון הרשת, תוך ציפייה שהאלגוריתם ילמד לייצג את המילים בצורה מועילה לניתוח הרגש של משפט נתון. בקטע הקוד הבא נדגים כיצד השיכונים שמורים בשכבה: כמטריצה אשר השורה ה-k שלה מכילה את השיכון של המילה ה-k באוצר המילים.

```
W=embedding_layer.weight
print(W.size())
print(W[integer,:])
```

פלט:

```
torch.Size([11222, 3])
tensor([[ 0.4685,  1.1884, -0.3000]], grad_fn=<IndexBackward0>)
```

למידת שיכון מוצלח היא משימה קשה לא פחות מאימון רשת הנוירונים עצמה: בדרך כלל השיכון יהיה במרחב ממימד גבוה, ובאוצר המילים קיימות עשרות אלפי מילים. על כן, רק בשכבת השיכון ייתכן ויהיו יותר פרמטרים מבכל שאר הרשת. על מנת להתמודד עם אתגר זה, ניתן להשתמש בשיכונים מאומנים מראש (pretrained embeddings) על אוסף נתונים גדול. בקטע הקוד הבא נדגים כיצד לטעון את הגרסה הקטנה ביותר של GloVe: שיכוני מילים אשר אומנו על כל וויקיפדיה בשנת 2014.

```
from torchtext.vocab import GloVe
glove_embedder=GloVe(name='6B',dim=50)
embedded=glove_embedder.get_vecs_by_tokens(["It's", "cool"])
print(embedded.size(),embedded.dtype,embedded.requires_grad)
```

פלט:

```
torch.Size([2, 50]) torch.float32 False
```

העברת משפט לאחר טוקניזציה ל-GloVe תניב טנזור המכיל את השיכונים של המילים במרחב 50 מימדי, אותם נזין לרשת הנוירונים במקום פלט שכבת השיכון הגדרנו בעבר. שימו לב שמערכת ה-Autograd אינה עוקבת אחרי שיכוני המילים, שכן ערכי השיכון כבר מאומנים ואין לנו רצון שהם ישתנו בתהליך אימון הרשת אשר עתיד לבוא.

במבט עמוק יותר לתוצאה נגלה כי השיכון של המילה "It's" הוא טנזור האפס: זהו השיכון של מילים מחוץ לאוצר המילים של GloVe. בעת שימוש בשיכון מאומן מראש עלינו לתת את הדעת גם לאוצר המילים, ותהליך הטוקניזציה שבו השתמשו בעת למידת השיכון, אליהם נרצה להתאים את תהליך עיבוד הנתונים הגולמיים שלנו ככל האפשר.

בנוסף, יש לקחת בחשבון גם את העובדה ששיכונים אלו לא אומנו על אוסף הנתונים איתו אנו עובדים, וכן לא למטרת סיווג הרגש המובע במשפט כך שייתכן שאין הם אידיאלים למטרה זו. אף מעבר לכך, אוסף הנתונים הגדול עליו שיכונים אלו אומנו לא מתאים ללמידה מפוקחת, שכן אין זה סביר לתייג את הרגש המובע בכל משפט בוויקיפדיה: עלות התיוג באופן אנושי תהיה יקרה להחריד. עקב כך, שיכוני GloVe נלמדו בפיקוח עצמי (Self Supervised Learning – SSL): כשלב הכנה לאלגוריתם האימון חושבו מספר המופעים המשותפים של כל זוג מילים באוסף המשפטים. מטריצת המופעים המשותפים בתורה שימשה ליצירת פונקציית המטרה של אלגוריתם הלמידה, כאשר התוצאה היא שכל הלמידה התרחשה על אוסף נתונים לא מתוייג, מבלי להידרש להתערבות אנושית ביצירת אוסף הנתונים.

שאלות לתרגול

1. המילים במשפט "It's cool" נפוצות בשימוש ועל כן לא סביר שהן לא נלמדו בתהליך האימון של GloVe. נסו לבצע טוקניזציה שונה למשפט, באופן ידני, ולראות האם בדרך זו אפשר למצוא שיכון בעל ערך לכל טוקן.
   הערה: טעינת השיכונים של GloVe כרוכה בהורדה של כ-1GB נתונים.