

עיבוד שפה טבעית - תרגיל 2 שאלה 3

על ידי: פן אייל - 208722058, קורן אביטבול - 318796448

התקנות

כמו במקרה הקודם, נוכל להשתמש בחבילות עיבוד השפה של [האניגמייס](#) על-מנת לטעון את הדאטא. כיוון שאנחנו טוענים דאטא בפורמט מטוקנז של UD, נצטרך להתקין גם חבילה התומכת בו.

```
!pip install datasets -q
!pip install tokenizers -q
!pip install conllu -q
```

ייבוא

נצטרך כמעט את כל החבילות שהתקנו עבור הרשת הנשנית למסמכים.

```
import matplotlib.pyplot as plt # for plotting
import pandas as pd # only to show some data in a nice table
import torch
import time

from tokenizers import Tokenizer
from tokenizers.models import WordLevel
from tokenizers.pre_tokenizers import WhitespaceSplit
from tokenizers.trainers import WordLevelTrainer

from datasets import load_dataset
from torch import nn, optim
from torch.utils.data import DataLoader
from tqdm.notebook import tqdm # progress bar
```

נתעלם משיגאות warning ונמחק את progress barn לאחר שסיים בשביל הדפסת המחברת ל PDF בצורה יפה יותר.

```
import warnings
warnings.filterwarnings('ignore')

TQDM_LEAVE = False
```

זרע הפורענות

משהו שכדאי לעשות בזמן שמפתחים מודל הוא להמעיט באקראיות ככל הניתן. בחבילות הנומריות של פייתון כמו numpy ו-pytorch יש הרבה מאוד אלמנטים אקראיים, כמו איך פרמטרים מאותחלים, או באיזה סדר נפל דאטאסט אם אנחנו בוחרים לערבב אותו בכל איטרציה (מומלץ באופן כללי, לא נממש הפעם). הדרך שלנו לשלוט באלמנטים האלה כדי שיהיו זהים בכל הרצה (ובין היתר, להקל על בדיקת התרגיל) היא לקבוע זרע אקראי בתחילת ההרצה, שממנו תנבע האקראיות באופן דטרמיניסטי. הבה:

```
SEED = 5784



import random
from numpy import random as nprnd

random.seed(SEED)
nprnd.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
```

מה בדאטא?

כמו בניתוח סנטימנט, אנחנו רוצים לדעת עם מה יש לנו עסק. לטעון מתוך UD זה קל עם חבילת datasets, ובשביל לגשת לשדות השונים, ניתן להיעזר ב[בדף התיעוד](#). אותנו מעניין רק הטקסט וחלקי הדיבר UPOS (בשלב הזה) ולכן נסתכל על כמה דוגמאות:

```
pd.set_option('display.max_colwidth', None)
dataset = load_dataset(
    'universal_dependencies', 'cu_proiel')
dataset.set_format(type="pandas", columns=["text", "tokens", "upos"])
dataset['validation'][:10]
```

	text	tokens	upos	
0	остави тоу даръ твои прѣдъ	[остави, тоу, даръ, твои, прѣдъ,	[16, 14, 0, 6, 2,	
	олътаремъ ѿ шедъ прѣжде	олътаремъ, ѿ, шедъ, прѣжде,	0, 9, 16, 14,	
	съмири сѧ съ братромъ своимъ	съмири, сѧ, съ, братромъ, своимъ,	16, 11, 2, 0, 6,	
	и тогда пришедъ принеси даръ твои	и, тогда, пришедъ, принеси, даръ, твои]	9, 14, 16, 16, 0, 6]	
1	Бѣди оувѣщаѧ сѧ съ сѣпъремъ	[Бѣди, оувѣщаѧ, сѧ, съ, сѣпъремъ,	[17, 16, 11, 2,	
	своимъ скоро донъдеже еси на пѣти съ нимъ да не прѣдасть	своимъ, скоро, донъдеже, еси, на, пѣти, съ, нимъ, да, не, прѣдасть,	0, 6, 14, 5, 17, 2, 0, 2, 11, 5,	
	тебе сѣдии ѿ сѣдии тѧ прѣдасть	тебе, сѣдии, ѿ, сѣдии, тѧ, прѣдасть,	14, 16, 11, 0, 9,	
	слоуѣ ѿ въ темъницѧ въврѣжетъ тѧ	слоуѣ, ѿ, въ, темъницѧ, въврѣжетъ, тѧ]	0, 11, 16, 0, 9, 2, 0, 16, 11]	
2	аминь ѿлюжъ тебѣ	[аминь, ѿлюжъ, тебѣ]	[15, 16, 11]	
3	не изидеши отъ тѣдѣ донъдеже въздаси послѣдънии кодрантъ	[не, изидеши, отъ, тѣдѣ, донъдеже, въздаси, послѣдънии, кодрантъ]	[14, 16, 14, 14, 5, 16, 6, 0]	
4	Слышасте ѣко речено бѣ~ древънимъ	[Слышасте, ѣко, речено, бѣ~, древънимъ]	[16, 5, 16, 17, 6]	
5	не прѣлюбы сътвориши	[не, прѣлюбы, сътвориши]	[14, 0, 16]	
6	азъ же ѿлюжъ вамъ ѣко въсѣкъ	[азъ, же, ѿлюжъ, вамъ, ѣко, въсѣкъ,	[11, 14, 16, 11,	
	иже възърить на женѧ съ	иже, възърить, на, женѧ, съ,	5, 6, 11, 16, 2,	
	похотиѧ юже любы сътвори съ	похотиѧ, юже, любы, сътвори, съ,	0, 2, 0, 14, 0,	
			16 2 14 2 0	

המממ. קצת פחות ברור מה שקורה פה מאשר בדוגמת המסמכים השלמים. קודם כל, אנחנו לא דוברים סלאבית כנסייתית עתיקה. [הנה](#) ערך הויקיפדיה שלה ו**הנה** הסבר נוסף, והם יספרו לנו על תכונות של השפה שאולי יעזרו לנו להבין בהמשך אם יש תופעות שכדאי להתייחס אליהן מפורשות.

הדבר השני שאנחנו מתקשים איתו הוא פורמט התגים, שמופיעים כאן אחרי שכבר מופו לאינדקסים ע"י מי שהזין אותם לשרתי האיגנפייס. למזלנו, דאגו לנו גם לשמירת המיפוי בתוך הדאטאסט, וניתן לגשת אליו מתוך כל אחד מחיתוכי הדאטא (splits).

```
val_tags = dataset['validation'].features['upos'].feature.names
[f'{i:2}: {p}' for (i,p) in enumerate(val_tags)]
```

```
[' 0: NOUN',
 ' 1: PUNCT',
 ' 2: ADP',
 ' 3: NUM',
 ' 4: SYM',
 ' 5: SCONJ',
 ' 6: ADJ',
 ' 7: PART',
 ' 8: DET',
 ' 9: CCONJ',
 '10: PROPN',
 '11: PRON',
 '12: X',
 '13: _',
 '14: ADV',
 '15: INTJ',
 '16: VERB',
 '17: AUX']
```

סקירת-שפיות קצרה של הדאטא מהתא הקודם תראה לנו שבאמת ברוב המשפטים יש פועל (16), כפי שאנו מצפים מכל שפה, ושיש כמות סבירה של שמות עצם (0). נשים לב שאו שזה מקור שנכתב (בעת העתיקה) ללא סימני פיסוק (1), או שהם הוסרו בעת עריכת המשאב.

1. מצאו משפט מהקישור השני על השפה (לעיל) שמסביר מדוע הפעלים נמצאים במקומות מגוונים על-פני המשפטים שדגמנו.

"Old Church Slavonic had apparently a free word order, all possible orders of subject, object and verb are found"

- כיוון שאנחנו חשדנים.ות מטבענו, ולא נרצה שנאמן את המודל על רשימת תגים מסוימת ואח"כ נבחן אותו על אינדקסים לא-תואמים (מה שיכול לגרום לכך שכל החיזויים שלנו לשמות עצם יפורשו כחיזויים לפעלים), נכתוב קוד קצר שמוודא שסדר התגים זהה עבור אימון, ולידציה ומבחן.
2. כתבו קוד שמוודא את סדר התגים לשלושת חלקי הדאטאסט. ניתן להשתמש בפונקציית assert.

```
### for exercise 2 ###
```

```
array = []
for group in ['train', 'validation', 'test']:
    tags = dataset[group].features['upos'].feature.names
    array.append(tags)
assert array[0] == array[1] == array[2]
```

טוקנייזר

בעית הטיקנוז שלנו קלה אפילו יותר מזו שהיתה בניתוח סנטימנט. המסמכים שלנו באים עם חלוקה לטוקנים מראש, ולכן אפשר פשוט להעביר לטוקנייזר שנבנה בתור ברירת מחדל את הפרמטר לפיו המידע שהוא מקבל כבר עבר "טרום-טוקניזציה". על-כן:

1. נטען את הדאטאסט
2. ניצור טוקנייזר פשוט
3. נגדיר לו את התבנית למילים לא ידועות UNK ואת תבנית הריפוד PAD
4. נאמן את הטוקנייזר
5. נספר לו איך לרפד.

אחר-כך נצטרך לטפל גם בטוקנייזר עבור התגים, בעיקר כדי לשמור על מדיניות ריפוד אחידה (המודולים ב-torch יצפו לאורך רצפים אחידים בין הקלט לתגים). בשני המקרים נעשה משהו קצת מלוכלך בשביל ליצור לטוקנייזר מחרוזת קלט. עבור הטוקנים, נגדיר פונקציה שמכניסה רווחים שאחר-כל האימון של הטוקנייזר יוציא, ועבור התגים, כיוון שהם נטענ בפורמט int, נריך עליהם פונקציית str().

```
PAD_ID = 0
```

```
def map_instance_to_whitespace_tokenizable_text(inst) -> str:
    return " ".join(inst['tokens'])
```

```
def make_tokenizers():
    dataset = load_dataset("universal_dependencies", "cu_proiel", split="train")
    tokenizer = Tokenizer(WordLevel(unk_token="<UNK>"))
    tokenizer.pre_tokenizer = WhitespaceSplit()
    trainer = WordLevelTrainer(special_tokens=["<PAD>", "<UNK>"])
    tokenizer.train_from_iterator([map_instance_to_whitespace_tokenizable_text(i) for i in dataset],
                                trainer=trainer,
                                length=len(dataset))
    tokenizer.enable_padding(pad_id=PAD_ID, pad_token="<PAD>")

    tag_tokenizer = Tokenizer(WordLevel(vocab={str(i): i for i in range(len(val_tags)+1)}))
    tag_tokenizer.enable_padding(pad_id=len(val_tags), pad_token=str(len(val_tags)))
    return tokenizer, tag_tokenizer
```

```
tokenizer, tag_tokenizer = make_tokenizers()
```

```
tokenizer.save("ud-cu-tokenizer.json", pretty=True)
tag_tokenizer.save("ud-cu-tag-tokenizer.json", pretty=True)
```

```
print(tokenizer.get_vocab_size())
print(tag_tokenizer.get_vocab_size())
```

```
7745
19
```

שימו לב שקיבלנו אוצר מילים די קטן לטוקנים, כיאה לדאטאסט קטן. הולכים להיות לנו הרבה מאוד UNK בולידציה ובטסט. 19 תגים זה המספר הרצוי, שכן יש 18 תגים בסכימת UPOS, והוספנו תג ריפוד.

בניית מתייג חלקי דיבר

נתחיל בהמרת המעבד שלנו ל-GPU:

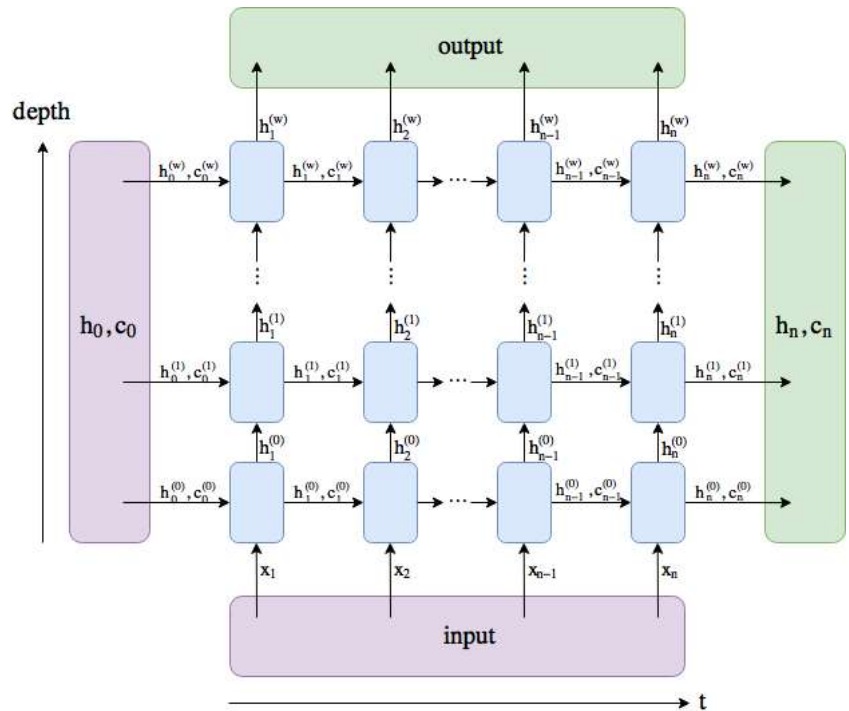
```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

הגדרת המודל

המודל הבסיסי שלנו זהה לזה שהשתמשנו בו עבור הרשת לסיווג טקסט שלם, עם הבדל אחד מאוד חשוב: הפלט שלה צריך להיות ברמת הטוקן, לא לוקחים רק את האחרון אלא את כל הסט שיוצא מה-LSTM, ואותו מעבירים לשכבה לינארית.

שימו לב: בהמשך נראה שפונקציית ההפסד שלנו מקבלת ציונים לא-מנורמלים עבור הקלאסים (חלקי דיבר, במקרה שלנו) ומבצעת את ה-softmax כחלק מהחישוב. אל תעבירו את הפלט בפונקציה שמבצעת חישוב דומה.

3. ממשו את `__init__()` ואת `forward()`. שימו לב לפרמטרים שהשתנו ביחס להדרכת תיג המסמכים.



```
class PosTagger(nn.Module):
    def __init__(self,
                  embedding_dim: int,
                  hidden_size: int,
                  num_tags: int,
                  num_layers: int) -> None:
        super().__init__()

        ### for exercise 3.1 ###

        # The embedding layer converts token indices to vectors
        self.embedding = nn.Embedding(tokenizer.get_vocab_size(),
                                       embedding_dim,
                                       padding_idx=tokenizer.padding["pad_id"])
        # The LSTM takes embedded sequences as input
        self.lstm = nn.LSTM(input_size=embedding_dim,
                             hidden_size=hidden_size,
                             num_layers=num_layers,
                             batch_first=True)
        # The linear layer maps from hidden state space to tag space
        self.fc = nn.Linear(hidden_size, num_tags)

    def forward(self, x) -> torch.Tensor:

        ### for exercise 3.2 ###

        # Pass input through the embedding layer
        embedded = self.embedding(x)
        # Pass the embedded sequence through the LSTM layer
        lstm_out, _ = self.lstm(embedded)
        # Convert LSTM output to tag space using the linear layer
        tag_space = self.fc(lstm_out)

        # Apply softmax to convert tag scores to probabilities
        tag_scores = nn.functional.softmax(tag_space, dim=2)

        return tag_scores
```

אימון ו-ולידציה

ההבדל העיקרי בפרקטיקת האימון אל מול סיווג מסמכים היא שיש גרדיאנט שמחושב עבור כל אחד מהטוקנים בנפרד: לכל אצווה (batch) יהיו לנו מספר תחזיות לא גודלה אלא כסכום אורכי המשפטים שבה. עיין בקפידה ב**תיעוד** מודול ההפסד שנשתמש בו וחישוב היטב מה הקלט שלו צריך להיות.

4.1 ממשו את חישוב ההפסד ופעפועו. שימו לב לגדלי הטנסורים המשתתפים. שני רמזים בהקשר זה:

- אין לדאוג עדיין לטוקני הריפוד. אנחנו לא נרצה להשתמש בתחזיות שלהם כמובן, אבל הטיפול בהם קורה בשלב אתחול ההפסד.

• גשו לתיעוד של torch.tensor ועמדו על ההבדל בין הפעולות view () ו-permute ().

4.2. ממשו את חישוב המטריקה. אנו נמדוד דיוק פשוט (accuracy), ולצורך כך אותחלו עבורכם משתני-עזר לפני הלולאה. כאן אנחנו כן דואגים לטוקני הריפוד.

```
def train(model: PostTagger,
          optimizer: optim.Optimizer,
          loss_fn: nn.CrossEntropyLoss,
          dataloader: DataLoader,
          curr_epoch: int) -> dict:
    model.train()
    total = 0
    correct = 0
    for batch in tqdm(dataloader, desc=f"Epoch {curr_epoch} - Training", leave=TQDM_LEAVE):
        optimizer.zero_grad()
        sentences = batch["input_ids"].to(device)
        labels = batch["labels"].to(device)
        probs = model(sentences)

        ### for exercise 4.1 ###

        one_hot_labels = nn.functional.one_hot(labels, num_classes=19).float()
        loss = loss_fn(probs.view(-1), one_hot_labels.view(-1))
        loss.backward()
        optimizer.step()

        ###

        preds = probs.argmax(dim=2)

        ### for exercise 4.2 ###

        # clean preds and labels from pad tokens
        pad_id = tag_tokenizer.padding['pad_id']
        clean_preds = preds[labels != pad_id]
        clean_labels = labels[labels != pad_id]

        correct += (clean_preds == clean_labels).sum().item()
        total += clean_labels.numel()

        ###

    return correct / total

def evaluate(model: PostTagger,
             dataloader: torch.utils.data.DataLoader) -> dict:
    model.eval()
    total = 0
    correct = 0
    with torch.no_grad(): # operations done in this block will not contribute to gradients
        for batch in tqdm(dataloader, desc=f"Evaluation", leave=TQDM_LEAVE):
            sentences = batch["input_ids"].to(device)
            labels = batch["labels"].to(device)
            probs = model(sentences)
            preds = probs.argmax(dim=2)

            ### for exercise 4.2 ###

            # clean preds and labels from pad tokens
            pad_id = tag_tokenizer.padding['pad_id']
            clean_preds = preds[labels != pad_id]
            clean_labels = labels[labels != pad_id]

            correct += (clean_preds == clean_labels).sum().item()
            total += clean_labels.numel()

    return correct / total
```

דגע האמת

נחבר הכל ביחד. נגדיר את הקבועים שלנו, שבהמשך יהיו היפר-פרמטרים.

```
BATCH_SIZE = 24
EMB_DIM = 100
HIDDEN_DIM = 64
NUM_LAYERS = 2
EPOCHS = 5
```

```
def deep_stringify(x):
    if type(x) == int:
        return str(x)
    return [deep_stringify(a) for a in x]

dataset = load_dataset("universal_dependencies", "cu_proiel")
dataset = dataset.map(lambda ins: {
    "input_ids": [e.ids for e in tokenizer.encode_batch(ins['tokens'],
                                                         is_pretokenized=True)],
    "labels": [e.ids for e in tag_tokenizer.encode_batch(deep_stringify(ins['upos']),
                                                         is_pretokenized=True)],
}, batched=True, batch_size=BATCH_SIZE)
dataset.set_format(type="torch", columns=["input_ids", "labels"])

train_dataloader = DataLoader(dataset["train"], batch_size=BATCH_SIZE)
val_dataloader = DataLoader(dataset["validation"], batch_size=BATCH_SIZE)
test_dataloader = DataLoader(dataset["test"], batch_size=BATCH_SIZE)
```

נאתחל מודל, מאפטם ופונקציית הפסד.

5. אתחלו את פונקציית ההפסד.

```
model = PostTagger(embedding_dim=EMB_DIM,
                  hidden_size=HIDDEN_DIM,
                  num_layers=NUM_LAYERS,
                  num_tags=tag_tokenizer.get_vocab_size()).to(device)
optimizer = optim.Adam(model.parameters())

### for exercise 5

loss_fn = nn.CrossEntropyLoss()

###

start = time.time()

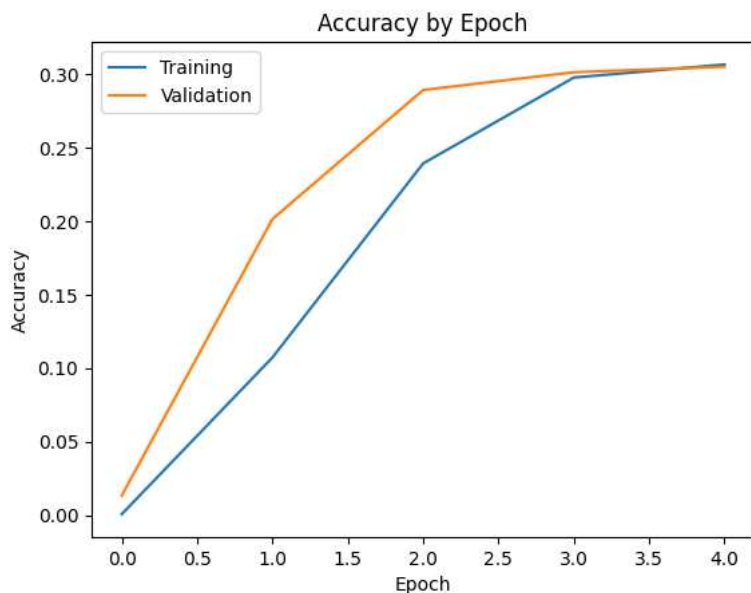
train_accuracies = []
validation_accuracies = []
for epoch in range(EPOCHS):
    train_acc = train(model, optimizer, loss_fn, train_dataloader, epoch)
    val_acc = evaluate(model, val_dataloader)
    print(f"Epoch {epoch + 1}:")
    print(f"Training Accuracy: {100 * train_acc:.2f}%")
    print(f"Validation Accuracy: {100 * val_acc:.2f}%")
    train_accuracies.append(train_acc)
    validation_accuracies.append(val_acc)

end = time.time()
print(f"\nTime to finish: {end - start} seconds")

plt.title("Accuracy by Epoch")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.plot(train_accuracies, label="Training")
plt.plot(validation_accuracies, label="Validation")
plt.legend();
```

Epoch 1:
Training Accuracy: 0.07%
Validation Accuracy: 1.34%
Epoch 2:
Training Accuracy: 10.72%
Validation Accuracy: 20.13%
Epoch 3:
Training Accuracy: 23.91%
Validation Accuracy: 28.90%
Epoch 4:
Training Accuracy: 29.75%
Validation Accuracy: 30.11%
Epoch 5:
Training Accuracy: 30.64%
Validation Accuracy: 30.49%

Time to finish: 8.102805137634277 seconds



ניתוח

6. בתיבת הטקסט להלן, תארו את הגרף שיצא לכם לעיל. הציעו לפחות שני שינויים בהיפר-פרמטרים שלדעתכם עשויים לשפר את התוצאה ו**ממשו אותם**. לכל ניסוי שכפלו את תיבת הקוד לעיל, שנו את מה שצריך, והריצו מחדש. **השתמשו בשמות משתנים חדשים עבור המודלים והתוצאות**. ניתן לוותר על הדפסות המספרים באפוקי הביניים ולהסתפק בתוצאות הסוף ובגרף.

7. שנו מאפיין של הניסוי **שאינו** אחד מההיפר-פרמטרים המוגדרים. למשל, השתמשו בקלאס אחר מ-pytorch עבור הרשת הנשנית, המאפטם, או משהו אחר לבחירתכם. או הוסיפו שכבה לינארית למודל.

האם השינויים אכן הועילו?

ניתוח - תשובות

6. אנו רואים מהגרף שככל שמספר האפוקים גדל המודל לומד יותר ויותר טוב והדיוק שלו עולה גם עבור מדגם האימון וגם עבור מדגם הולידציה. שני שינויים אפשריים להיפר-פרמטרים שנבדוק הם:

- הגדלת מספר השכבות ברשת
- הגדלת ייצוג האמבדינגס

בניסוי הראשון הגדלנו את מספר האפוקים (מ 5 ל 100) ובניסוי השני את מספר האמבדינגס (מ 100 ל 1000). בשני הניסויים ראינו שיפור ניכר בביצועים, ולכן נרציץ פעם שלישית עם שני השינויים בייחד.

7. שינינו את האופטימיזר להיות גרדיאנט דיסנט סטוכסטי. יש הרעה בתוצאות.

```

### for exercise 6 ###

def run_and_plot(model, optimizer, loss_fn, epochs):

    start = time.time()

    train_accuracies = []
    validation_accuracies = []
    for epoch in range(EPOCHS):
        train_acc = train(model, optimizer, loss_fn, train_dataloader, epoch)
        val_acc = evaluate(model, val_dataloader)
        train_accuracies.append(train_acc)
        validation_accuracies.append(val_acc)

    end = time.time()
    print(f"\nTime to finish: {end - start} seconds")

    plt.title("Accuracy by Epoch")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.plot(train_accuracies, label="Training")
    plt.plot(validation_accuracies, label="Validation")
    plt.legend();

# changing the EPOCHS from 5 to 100
EPOCHS = 100

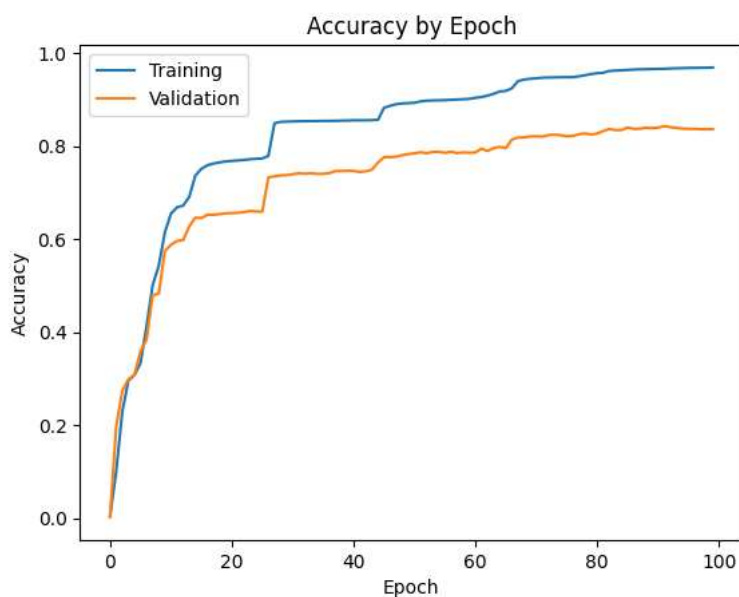
model_1 = PosTagger(embedding_dim=EMB_DIM,
                    hidden_size=HIDDEN_DIM,
                    num_layers=NUM_LAYERS,
                    num_tags=tag_tokenizer.get_vocab_size()).to(device)
optimizer_1 = optim.Adam(model_1.parameters())
loss_fn_1 = nn.CrossEntropyLoss()

run_and_plot(model_1, optimizer_1, loss_fn_1, EPOCHS)

EPOCHS = 5

```

Time to finish: 101.3928496837616 seconds



```

### for exercise 6 ###

# changing the EMB_DIM from 100 to 1000
EMB_DIM = 1000

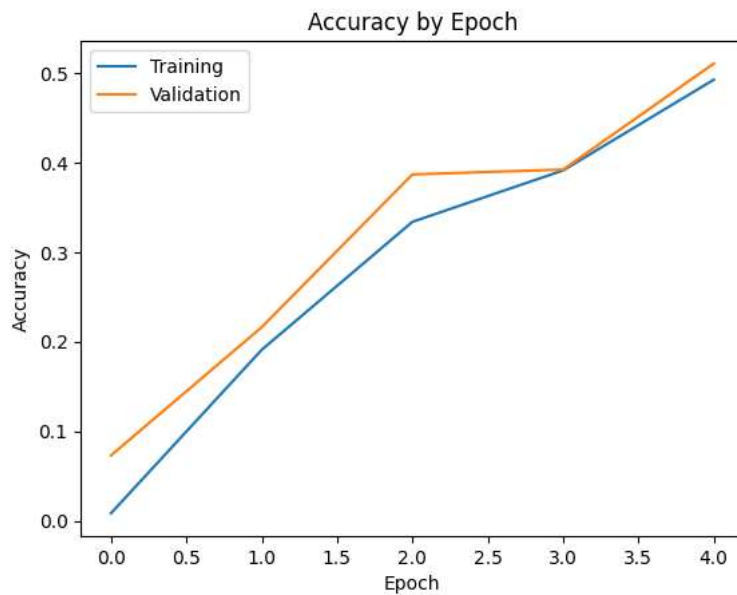
model_2 = PosTagger(embedding_dim=EMB_DIM,
                    hidden_size=HIDDEN_DIM,
                    num_layers=NUM_LAYERS,
                    num_tags=tag_tokenizer.get_vocab_size()).to(device)
optimizer_2 = optim.Adam(model_2.parameters())
loss_fn_2 = nn.CrossEntropyLoss()

run_and_plot(model_2, optimizer_2, loss_fn_2, EPOCHS)

EMB_DIM = 100

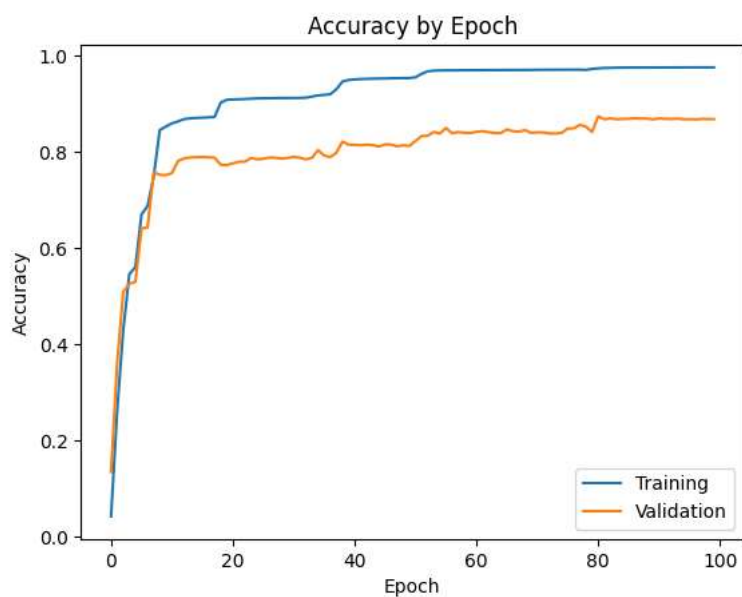
```


Time to finish: 6.557162761688232 seconds



```
### for exercise 6 ###  
  
# changing both the EPOCHS from 5 to 100 and the EMB_DIM from 100 to 1000  
  
EMB_DIM = 1000  
EPOCHS = 100  
  
model_3 = PostTagger(embedding_dim=EMB_DIM,  
                      hidden_size=HIDDEN_DIM,  
                      num_layers=NUM_LAYERS,  
                      num_tags=tag_tokenizer.get_vocab_size()).to(device)  
optimizer_3 = optim.Adam(model_3.parameters())  
loss_fn_3 = nn.CrossEntropyLoss()  
  
run_and_plot(model_3, optimizer_3, loss_fn_3, EPOCHS)  
  
EMB_DIM = 100  
EPOCHS = 5
```

Time to finish: 141.02222776412964 seconds

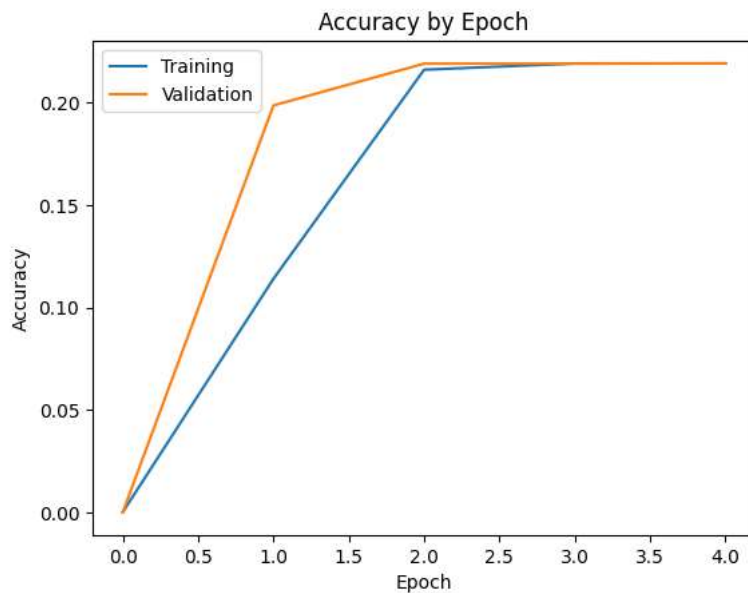


```
### for exercise 7 ###
```

```
model_4 = PosTagger(embedding_dim=EMB_DIM,
                    hidden_size=HIDDEN_DIM,
                    num_layers=NUM_LAYERS,
                    num_tags=tag_tokenizer.get_vocab_size()).to(device)
# changing the optimizer from adam to SGD
optimizer_4 = optim.SGD(model_4.parameters(), lr=0.01)
loss_fn_4 = nn.CrossEntropyLoss()

run_and_plot(model_4, optimizer_4, loss_fn_4, EPOCHS)
```

Time to finish: 4.749161958694458 seconds



טוטט

8. מצאו את המודל הטוב ביותר מאלה שניסיתם עד כה והריצו (פעם אחת בלבד) על הטסט. הקבוצה עם התוצאה הטובה ביותר תקבל בונוס נקודה לציון הסופי בקורס.

```
### for exercise 8 ###
```

```
best_model = model_3

test_acc = evaluate(best_model, test_dataloader)
print(f"Test Accuracy: {100 * test_acc:.2f}%")

Test Accuracy: 85.63%
```