תרגום מכונה באמצעות רשת מקודד-מפרש נשנית

בפרק זה נלמד כיצד לשלב בארכיטקטורת מקודד-מפרש רכיבים של רשתות נשנות, ולאמנן לבצע תרגום אוטומטי של קטע טקסט נתון. ראשית נדון באוסף הנתונים המתאים למשימה זו: Tatoeba. זהו אוסף של זוגות משפטים בשפות שונות בעלי משמעות זהה. המשפטים מתורגמים על ידי משתמשי הפרוייקט באופן התנדבותי כך שמצד אחד אוסף הנתונים עוד ממשיך להתעדכן אך מצד שני משתמשי האתר לרוב אינם מתרגמים במקצועם ולכן קיימות בו לא מעט שגיאות, קטנות וגדולות. לצרכינו בהמשך היחידה אוסף זה מספק, אך יש לזכור את מגבלותיו בבואנו לאמן רשת למטרות החורגות מהדגמת יכולת.

אחד היתרונות של אוסף זה הוא מגוון השפות הגדול הזמין בו. נשתמש שוב בספרייה Datasets לצורך טעינת הנתונים, כאשר לצורך הדוגמה שפת המקור שנבחר היא אנגלית, והיעד: צרפתית.

```
import datasets as ds
src="en"
tgt="fr"
dataset = ds.load_dataset("tatoeba", lang1=src, lang2=tgt)
```

הפרמטרים `lang1` ו-`lang2` שולטים בשפות של אוסף הנתונים המתקבל, ועל ידי החלפתם תוכלו להשתמש באותו קוד המופיע בהמשך פרק זה לצורך אימון רשת לתרגום מעברית לאנגלית, למשל.

נתבונן בדגימה אחת מאוסף הנתונים המתקבל.

```
dataset["train"]["translation"][10]
```

פלט:

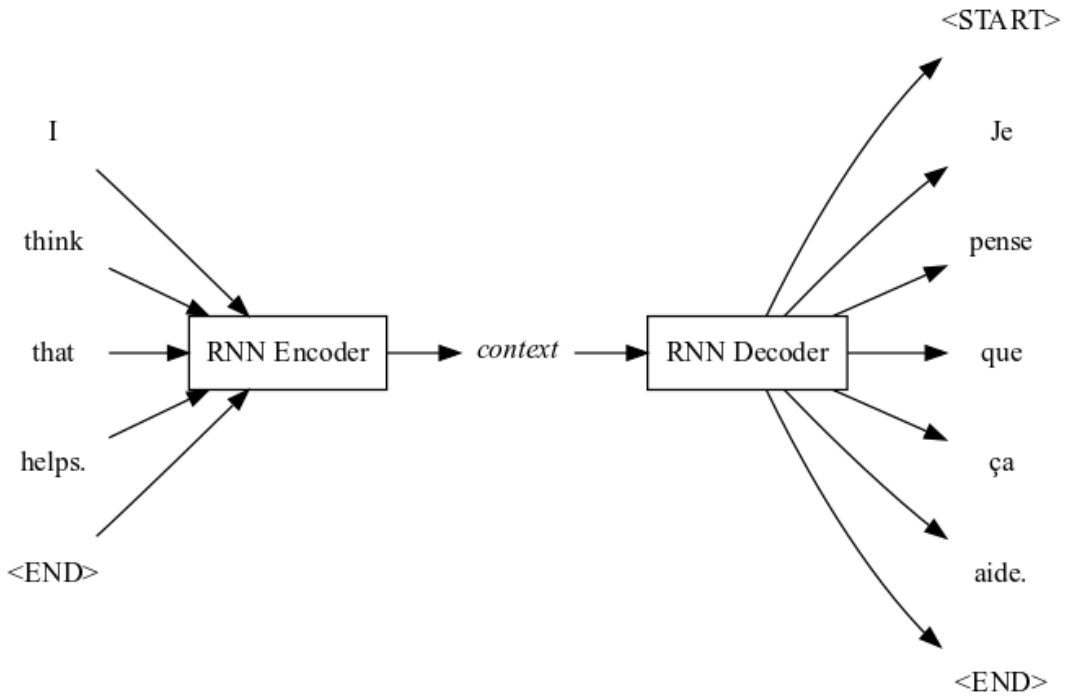
```
{'en': "Today is June 18th and it is Muiriel's birthday!",
 'fr': "Aujourd'hui nous sommes le 18 juin et c'est
l'anniversaire de Muiriel !"}
```

כרגיל במשימות עיבוד שפה טבעית, עלינו לבצע עיבוד מקדים על אוסף הנתונים לפני הזנתם לרשת נוירונים. מכיוון שכעת אנו עוסקים בשתי שפות, יש ליצור **שני** אוצרות מילים שונים: הראשון ייבנה מכל המשפטים בשפת המקור, אנגלית, והשני מכל המשפטים בשפת היעד, צרפתית. כמו כן, כפי שנראה מיד, ארכיטקטורת הרשת הנבחרת דורשת לאותת למקודד על סוף משפט קלט, והמפרש מצפה לקבל איתות נוסף גם על תחילתו של המשפט. על כן נוסיף לאוצרות המילים טוקנים מתאימים. דוגמה למשפטים לאחר טוקניזציה מופיעה להלן.

```
Source:
['I', 'think', 'that', 'helps.', '<END>']
tensor([   2,  140,   43, 4795,    1])
Target:
['<START>', 'Je', 'pense', 'que', 'ça', 'aide.', '<END>']
tensor([  2,   4, 184,  39,  75, 644,   1])
```

הקלט לרשת יהיה הטנזור המספרי ראשון, והטנזור השני הוא הפלט הצפוי מהרשת, בעזרתו נחשב את פונקציית המחיר.

באיור הבא תראו את ארכיטקטורת הרשת ברמת ההפשטה הגבוהה ביותר: מקודד-מפרש אשר שני חלקיו מבוססים על רשתות נשנות.



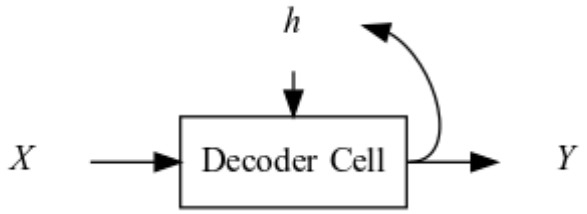
המקודד בארכיטקטורה זו יקבל משפט מקור יחיד וימיר אותו לייצוג חבוי, הקרוי בהקשר הנוכחי וקטור ה"הקשר" (context). בידינו כבר כל הרכיבים לכתיבת המקודד, זהו רכיב חילוץ המאפיינים ב-RNN אשר שימשה אותנו לסיווג הרגש המובע במשפט. השוו את קטע הקוד הבא לזה של הרשת `FasterDeepRNNClassifier` אשר כתבנו ביחידה 6, וודאו כי אתם רואים את הדימיון.

```
class Encoder(nn.Module):
    def __init__(self, embed_dim, hidden_dim, RNNlayers):
        super().__init__()
        self.src_embedding  = nn.Embedding(len(src_vocab),
                                           embed_dim)
        self.rnn_stack      = nn.LSTM(embed_dim,
                                      hidden_dim,
                                      RNNlayers)

    def forward(self, src_tokens):
      all_embeddings          = self.src_embedding(src_tokens)
      all_embeddings          = all_embeddings.unsqueeze(1)
      hidden_state_history, _= self.rnn_stack(all_embeddings)
      context                 = hidden_state_history[-1,0,:]
      return context
```

זכרו שפלט המודול `nn.LSTM` הוא סדרת המצבים החבויים של התא הנשנה בראש ערמת ה-RNN, ולכן עבור וקטור ההקשר אנו בוחרים את האחרון שבה.

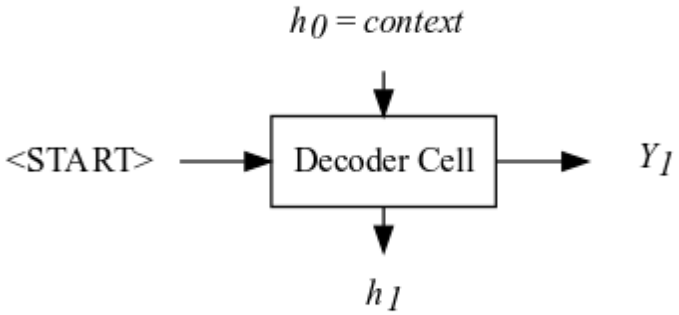
בבואנו לתכנן את המפרש ניתקל באתגר: עליו **לייצר משפט** בשפת היעד כפלט. נממש זאת בעזרת תא נשנה אשר יקבל כקלט את וקטור ההקשר מהמקודד וייצר את משפט הפלט טוקן לאחר טוקן מתחילת המשפט ועד לסופו. על כן, עלינו להוסיף לארכיטקטורת התא הנשנה הפשוט אפשרות לייצר פלט, כפי שמאוייר להלן.



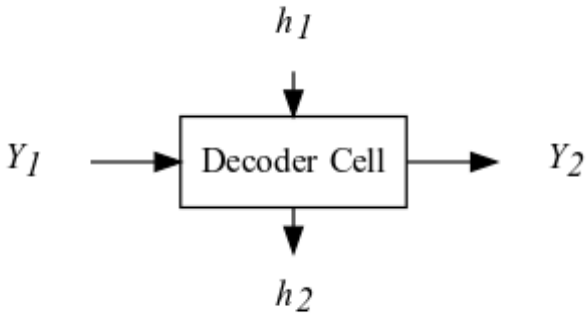
קלט התא X הוא טוקן (משוכן) בשפת היעד, צרפתית, וכך גם פלט התא, Y.

לפני כתיבת תא זה כמודול של PyTorch, נתאר את הדרך בה המפרש ישתמש בו. ראשית נדון במצב אימון, בו המפרש יקבל כקלט את וקטור ההקשר מהמקודד, וכן את סדרת הטוקנים של משפט המטרה, אותם נסמן ב- $(Z_0,...,Z_T)$, זהו בן הזוג של משפט המקור שהוזן למקודד. נשים לב לכך שלפי העיבוד המקדים אשר את תוצאתו ראינו לעיל, Z_0 הוא תמיד הטוקן המייצג את תחילת המשפט, ו- Z_T מייצג את סוף המשפט.

פלט המפרש יהיה גם הוא סדרת טוקנים, אותה נסמן ב- $(Y_0,...,Y_T)$. בתחילת החישוב הרקורסיבי בתא נאתחל את המצב החבוי בערכו של וקטור ההקשר ונגדיר באופן ידני את Y_0 להיות טוקן תחילת המשפט. כעת נזין טוקן זה בתור הקלט המסומן ב- X לעיל (לאחר שיכון כמובן), וכך נסמן למפרש שעליו להתחיל לתרגם את משפט המקור, על סמך וקטור ההקשר. ראו את הצעד הראשון במפרש באיור הבא.



כעת, לאחר קבלת הפלט Y_1, נזין אותו בחזרה לתא, הפעם בתפקיד הקלט, ונקבל את Y_2.



נחזור על פעולה זו עד לקבלת הטוקן Y_T, אשר על מנת לייצרו נידרש לעדכן את המצב החבוי $T-1$ פעמים, ולייצר את כל הטוקנים הקודמים לו. ראו מימוש חישוב זה בקוד להלן.

```
class TrainingDecoder(nn.Module):
    def __init__(self, embed_dim, hidden_dim):
        super().__init__()
        self.tgt_embedding  = nn.Embedding(len(tgt_vocab),
                                           embed_dim)
        self.RNNcell        = DecoderRNNCell(embed_dim,
                                             hidden_dim)
    def forward(self, context, tgt_tokens):
      self.RNNcell.hidden_state = context
      translated_tokens = [START_Token]
      for idx in range(len(tgt_tokens)-1):
        previous_token  = translated_tokens[idx]
        embedded_token  = self.tgt_embedding(previous_token)
        predicted_token = self.RNNcell(embedded_token)
        translated_tokens.append(predicted_token.detach())
      return translated_tokens
```

שימו לב שאנו מאתחלים אובייקט `DecoderRNNCell` בתוך בנאי המפרש, אותו נגדיר רק בהמשך הפרק. לעת עתה ניתן להבין מהו הפלט הדרוש ממנו לפי השימוש בו במתודת ה-`forward` והאיורים הנ"ל.

ייתכן שבאחד משלבי הביניים הטוקן החזוי יהיה טוקן סיום המשפט, שהרי גם הוא חלק מאוצר המילים של שפת היעד. במקרה זה נפרש זאת כאיתות של הרשת על כך שהיא סיימה את עבודת התרגום, ועל כן נוסיף ללולאה במתודת ה-`forward` את האפשרות לסיים מוקדם מהצפוי בעזרת שורות הקוד הבאות.

```
        if predicted_token == END_Token:
          break
```

לבסוף, נזכור שעל הרשת לעבור אימון, ולמטרה זו יש לבחור פונקציית מחיר מתאימה. מובן שנרצה לתגמל במחיר נמוך מודל אשר חזה נכונה טוקן Y_t הזהה לטוקן המתאים ממשפט המטרה, Z_t, אך מעבר לכך, נרצה לתגמל במחיר נמוך עוד יותר מודל העושה זאת בבטחון רב. באופן דומה, נרצה להעניש מודל אשר כלל לא היה קרוב לחזות את Z_t בעונש גדול יותר מאשר מודל אשר כמעט עשה זאת, אך בסוף "התבלבל".

שיקולים אלו מובילים אותנו לחשוב על בעיית יצירת הטוקן Y_t כבעיית סיווג קלאסית: התא הנשנה מקבל כקלט את המצב החבוי והטוקן הקודם, ועליו לייצר לכל טוקן באוצר המילים של שפת היעד את ההסתברות $P(Y_t = token)$, זהו פלט פונקציית ה-Softmax המוכרת לנו. לבסוף, הטוקן החזוי יהיה זה בעל הסתברות הסיווג הגבוהה ביותר, ה-argmax. היתרון הגדול בנקודת מבט זו הוא שפונקציית המחיר המבטאת את כל רצונותינו ידועה כבר, הלא היא האנטרופיה הצולבת.

כעת אנו מוכנים לפרט את התהליך החישובי המבוצע בתא הנשנה, המורכב משני שלבים:

1. חישוב המצב החבוי החדש. שלב זה יבוצע כבעבר, לפי הנוסחה

$$h_{t+1} = \tanh\left(W_{input}X + b_{input} + W_{hidden}h_t + b_{hidden}\right)$$

כאשר X הוא הטוקן המתקבל כקלט, h_t הוא המצב החבוי הקודם

2. חישוב הסתברויות הסיווג $P(Y = token)$, זאת נעשה על ידי הזנת המצב החבוי החדש לשכבה ליניארית ומיד אחריה – פונקציית ה-Softmax. אם כן, פלט התא יהיה וקטור בעל ערך לכל טוקן באוצר המילים, המחושב כך: $\text{Softmax}(W_{out}h_{t+1} + b_{out})$.

פרמטרי התא הנשנה הם משקלי השכבות הליניאריות המופיעות לעיל וערכי ה-bias שלהן.

מימוש התא מופיע בקטע הקוד הבא.

```
class DecoderRNNCell(nn.Module):
    def __init__(self, embed_dim, hidden_dim):
        super().__init__()
        self.hidden_state  = torch.zeros(hidden_dim)
        self.RNNcell       = nn.RNNCell(embed_dim,hidden_dim)
        self.output_linear = nn.Linear(in_features=hidden_dim,
                                       out_features=len(tgt_vocab))
        self.logsoftmax    = nn.LogSoftmax(dim=0)

    def forward(self, one_embedded_token):
        new_state          = self.RNNcell(one_embedded_token,
                                          self.hidden_state)
        tgt_token_scores   = self.output_linear(new_state)
        tgt_token_logprobs = self.logsoftmax(tgt_token_scores)

        self.hidden_state = new_state
        return tgt_token_logprobs
```

ראו כי השתמשנו בתא הנשנה המובנה ב-PyTorch, `nn.RNNCell`, אשר מבצע את החישוב הרשום בסעיף 1 לעיל.

עבודתנו עוד לא הסתיימה, שכן עלינו לחשב את המחיר עבור כל דגימה לאחר הזנתה ברשת. זאת יהיה נוח לעשות **תוך כדי** המעבר קדימה ברשת, שכן כך לא נידרש לשמור את היסטוריית וקטורי הסתברויות הסיווג של כל הטוקנים $(Y_0,...,Y_T)$ עד לסוף תרגום המשפט. בעודנו זוכרים שהאנטרופיה הצולבת היא מינוס לוגריתם ההסתברות שהמפרש מקנה לטוקן הנכון וכן שהטוקנים הנכונים נתונים במשתנה `tgt_tokens`, נוסיף חישוב זה גם הוא למתודת ה-`forward` של `TrainingDecoder`, המופיעה בשלמותה להלן. תוספת חישוב המחיר מסומנת ב-# בקוד.

```
    def forward(self, context, tgt_tokens):
      self.RNNcell.hidden_state = context
      translated_tokens = [START_Token]
      sentence_loss = 0
      for idx in range(len(tgt_tokens)-1):
        previous_token  = translated_tokens[idx]
        embedded_token  = self.tgt_embedding(previous_token)
        logprobs        = self.RNNcell(embedded_token)
        predicted_token = logprobs.argmax()
        translated_tokens.append(predicted_token.detach())

        correct_token   = tgt_tokens[idx+1]                    #
        token_loss      = -logprobs[correct_token]             #
        sentence_loss  += token_loss                           #

        if predicted_token == END_Token:
          break
      return translated_tokens, sentence_loss
```

ראו כי כעת המפרש מחזיר למשתמש גם את המחיר הכולל של המשפט, זהו סכום המחירים של כל אחד מהטוקנים במשפט המתורגם. מחיר זה יעניש מודל אשר חוזה טוקנים שגויים, ובנוסף גם מודל אשר מסיים עבודתו מוקדם מהצפוי, שכן אז הוא יחזה את טוקן סיום המשפט במקום הלא הנכון, וישלם על כך מחיר באנטרופיה הצולבת של טוקן זה.

כעת נשאר לנו רק לחבר את פלט המקודד אל המפרש, ולאמנם יחדיו. לאחר האימון, נרצה להשתמש ברשת לתרגום משפט בשפת המקור עבורו אין לנו תרגום מוכן בסט האימון. על כן, יש להוסיף למפרש מצב חיזוי, בו הוא מקבל כקלט רק את וקטור ההקשר, ומייצר טוקנים עד ליצירת טוקן סוף המשפט (או מספר מקסימלי של טוקנים קבוע מראש). מובן שבמצב חיזוי המפרש לא יחזיר את מחיר המשפט, שכן לא ניתן לחשבו כלל.

שאלות לתרגול

1. השלימו את שלב עיבוד הנתונים המקדים עבור שתי השפות, כך שלבסוף יתקבל אוצר המילים של כל שפה, וטנזורים המכילים את המספר הסידורי של הטוקנים באוצר המילים, לכל משפט בכל שפה.
2. מהם מימדי הפרמטרים W_{out}, b_{out}, המשמשים לחישוב טוקן הפלט בתא הנשנה החדש שהגדרנו לעיל?
3. כתבו בקוד את מצב האימון של המפרש, כמתואר בפסקה האחרונה בפרק זה.
4.
   א. חברו את המקודד והמפרש למודול PyTorch יחיד.
   ב. בחרו זוג שפות המוכר לכם וטענו את האוסף המתאים להן לזכרון המחשב.
   ג. אמנו רשת תרגום מכונה על batch קטן של זוגות משפטים.
   ד. הדפיסו מספר משפטים מתורגמים לצד התרגום הנכון שלהם, כפי שהוא מופיע באוסף הנתונים.
5. ניתן לאמן את המפרש על ידי הזנת הטוקן הנכון לתא הנשנה בכל איטרציה במקום הטוקן האחרון שהמפרש חזה, וזאת על מנת לייצר את הטוקן הבא במשפט המתורגם. כמובן שאפשרות זו, הנקראת באנגלית Teacher Forcing, קיימת רק במצב אימון, שכן רק אז בידינו הטוקנים הנכונים.
   א. הוסיפו את השימוש ב-Teacher Forcing למפרש, ואמנו שוב את הרשת על batch קטן.

ב. ציירו על גרף אחד את המחיר הממוצע כפונקציה של epoch האימון עבור שני המודלים שאימנתם: בשאלה זו ובשאלה הקודמת.