SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

50.040 Natural Language Processing, Summer 2022
Homework 3

Due 29 July 2022, 5pm

Homework 3 will be graded by Ming Shan

Introduction In this homework, you will learn how to build a **statistical machine translation** model and a **neural machine translation** model.

Part 1 (25 points) In Part 1, we'll implement "IBM Model 1" using the "expectation–maximization (EM)" algorithm. We need to estimate the translation probabilities $t(f|e)$ on a parallel corpus, where e is a word from the English sentences and f is a word from the corresponding foreign sentences.

Note that there's a constraint for such probabilities:

$$\sum_f t(f|e) = 1, \quad t(f|e) \geq 0 \tag{1}$$

We'll use this constraint when initializing the translation probabilities in subsequent tasks.

Data We'll use the English-French parallel corpus under the folder “data/en-fr“, which contains a set of translation instances. As can be seen below each instance consists of an English-French sentence pair (note that we are translating from French into English, but as we discussed in class, when working on the translation model using IBM model 1, we are interested in generating French from English).

```
Hop in.    Montez.
Hug me.    Serre-moi dans tes bras !
I left.    Je suis parti.
```

The dataset is obtained from MXNET(`http://data.mxnet.io/data/fra-eng.zip`).

Question 1 [code] (3 points) Implement "word_pairs_in_corpus" function which returns all the possible word pairs (e,f) that have appeared in the corpus. List down the 10 most frequent pairs and count the number of unique word pairs.

Question 2 [code] (2 points) Implement "corpus_log_prob" function according to

$$\sum_k \sum_{i=1}^{m_k} \log \left(\sum_{j=0}^{n_k} t(f_i^{(k)} | e_j^{(k)}) \right) \tag{2}$$

where k is the index of each sentence pair, m_k, n_k are the lengths of each French, English sentences, $f_i^{(k)}, e_j^{(k)}$ are the French, English words at i-th, j-th positions, respectively. This is the objective function we try to maximize.

Question 3 [code] (10 points)

1. Implement "Init" function which initializes the translation probability dictionary t according to equation (1). You may need to `numpy.random.rand()` function here. (2 points)

2. Implement "hard_EM" function which returns an updated translation probability dictionary t. It is possible that in the hard EM algorithm a word $\tilde{e}$ from an English sentence may not be aligned with any word from the corresponding French sentence. In this case, let us set the corresponding probabilities $t(f|\tilde{e}) = \frac{1}{|V_f|}$ where $|V_f|$ is the size of the French vocabulary (in this case, the number of unique French words that ever appear in the training parallel corpus). (8 points)

3. Run the training code we provide for you.

Question 4 [code] (2 points)

1. Implement "visualize_alignment" function to visualize the word alignment (namely $t(f|e)$) of an original English sentence and the corresponding French sentence. You need to use "sns.heatmap()" in this part. For example, for the sentence pair

   ```
   NULL tom loves chocolate .
   tom adore le chocolat .
   ```

   the output of function "visualize_alignment" should be similar to Fig. 1

2. Run the code we provide for you to visualize the word alignments of the example below:

   ```
   NULL it was a very exciting game .
   c'était un jeu vraiment très excitant .
   ```

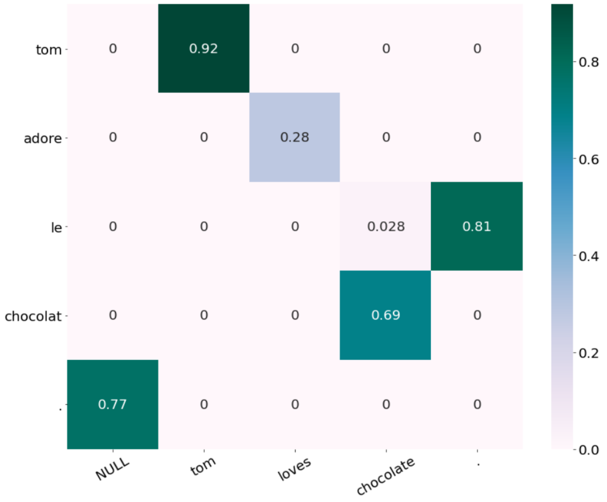

Figure 1

Question 5 [code] (8 points)

1. Implement the "soft_EM" function which returns an updated translation probability dictionary t by using soft EM algorithm.
2. Run the training code and visualization code we provide for you.

Part 2 (25 points) In this part, you will learn to build a sequence to sequence neural machine translation (NMT) model with attention mechanism. This model consists of a Bidirectional-LSTM *Encoder* and a Unidirectional-LSTM *Decoder*. The encoder can compress the source language sentence into a vector while the decoder can decode the vector and outputs a target language sentence. We'll describe how the Seq2Seq model works and then you will implement some key components in this model.

Given a sentence $(x_1, x_2, ..., x_m) \in \mathbb{R}^{e \times m}$ in source language, where e is the embedding size of each word x_i, m is the sentence length, we send these embeddings to the Bi-LSTM encoder, yielding *hidden states* and *cell states* for every position $i \in \{1, 2, ..., m\}$ in both the forward LSTM and backward LSTM. We denote the hidden/cell states in the forward LSTM as $\overrightarrow{h_i^{enc}} \in \mathbb{R}^{h \times 1}$, $\overrightarrow{c_i^{enc}} \in \mathbb{R}^{h \times 1}, i \in \{0, 1, 2, ..., m\}$, where $\overrightarrow{h_0^{enc}}, \overrightarrow{c_0^{enc}}$ are the initial hidden/cell states, and in the backward LSTM as $\overleftarrow{h_i^{enc}} \in \mathbb{R}^{h \times 1}$, $\overleftarrow{c_i^{enc}} \in \mathbb{R}^{h \times 1}, i \in \{0, 1, 2, ..., m\}$, where $\overleftarrow{h_m^{enc}}, \overleftarrow{c_m^{enc}}$ are the initial hidden/cell states. In practice, we concatenate LSTM left and right hidden/cell states, $h_i^{enc} = [\overleftarrow{h_i^{enc}}; \overrightarrow{h_i^{enc}}] \in \mathbb{R}^{2h \times 1}$, $c_i^{enc} = [\overleftarrow{c_i^{enc}}; \overrightarrow{c_i^{enc}}] \in \mathbb{R}^{2h \times 1}$. We treat the LSTM outputs as the latent representation for each encoded word. Next, we initialize the decoder hidden/cell states with a linear transformation of the last hidden/cell states from forward LSTM/ backward LSTM.

$$h_0^{dec} = W_h[\overleftarrow{h_0^{enc}}; \overrightarrow{h_m^{enc}}], \quad W_h \in \mathbb{R}^{h\times 2h}, h_0^{dec} \in \mathbb{R}^{h\times 1} \tag{3}$$

$$c_0^{dec} = W_c[\overleftarrow{c_0^{enc}}; \overrightarrow{c_m^{enc}}], \quad W_c \in \mathbb{R}^{h\times 2h}, c_0^{dec} \in \mathbb{R}^{h\times 1} \tag{4}$$

After initialization, we feed the decoder a matching sentence in the target language. At t^{th} time step, we look up the embedding matrix for the t^{th} word, $y_t \in \mathbb{R}^{e\times 1}$. When $t = 1$, y_1 is a special token "START". We then concatenate y_t and the *combined output* $o_{t-1}^{dec} \in \mathbb{R}^{h\times 1}$ from previous time step to produce (we will see how to derive this term later) $\bar{y}_t \in \mathbb{R}^{(e+h)\times 1}$. For $t = 1$, o_0^{dec} is a zero vector. We then feed $\bar{y}_t$ as input to the decoder LSTM.

$$h_t^{dec}, c_t^{dec} = Decoder(\bar{y}_t, h_{t-1}^{dec}, c_{t-1}^{dec}), \quad h_t^{dec}, c_t^{dec} \in \mathbb{R}^{h\times 1} \tag{5}$$

A potential issue with this model it that the encoder Bi-LSTM needs to be able to compress all necessary information of a source sentence into a fixed length vector, that is, h_0^{dec}. This may make it difficult for the encoder LSTM to cope with long sentences. Thus, we will use *attention mechanism* to address this problem. Specifically, we will map the encoder outputs $o_1^{enc}, ..., o_m^{enc}$ to the same dimensional space (using linear transformation) and use h_t^{dec} to compute attention weights over $o_1^{enc}, ..., o_m^{enc}$. The attention weights are computed by concatenating h_t^{dec} with $o_1^{enc}, ..., o_m^{enc}$, performing a linear transformation on the concatenated representations, applying a $Tanh(\cdot)$ function and multiplying it with an alignment matrix.

$$o_i^{enc} = W_{outputs}(o_i^{enc}), \quad i \in \{1, 2, ..., m\}, W_{outputs} \in \mathbb{R}^{h\times 2h} \tag{6}$$

$$\begin{gathered} e_{t,i} = W_{alignment} \cdot tanh(W_{combined}([h_t^{dec}; o_i^{enc}])) \\ i \in \{1, 2, ..., m\}, W_{combined} \in \mathbb{R}^{h\times h}, W_{combined} \in \mathbb{R}^{h\times 1} \end{gathered} \tag{7}$$

$e_{t,i}$ represents the alignment scores based on both the decoder hidden state and encoder outputs. We then normalize the alignment scores $e_t = (e_{t,1}, ..., e_{t,m})$, obtaining a probability distribution α_t. Last, we compute a weighted sum of all the encoder outputs, yielding a_t. This a_t contains necessary information from the source sentence and thus releases the burden of h_0^{dec}.

$$\alpha_t = softmax(e_t), \quad \alpha_t \in \mathbb{R}^{m\times 1} \tag{8}$$

$$a_t = \sum_{i=1}^{m} \alpha_{t,i} o_i^{enc}, \quad a_t \in \mathbb{R}^{2h\times 1} \tag{9}$$

We then concatenate a_t with the decoder hidden state h_t^{dec} and feed it through a linear layer, a $Tanh(\cdot)$ function, to get the *combined output* o_t^{dec}.

$$u_t = [a_t; h_t^{dec}], \quad u_t \in \mathbb{R}^{3h\times 1} \tag{10}$$

$$o_t^{dec} = \text{Tanh}(W_u u_t), \quad W_u \in \mathbb{R}^{h\times 3h}, o_t^{dec} \in \mathbb{R}^{h\times 1} \tag{11}$$

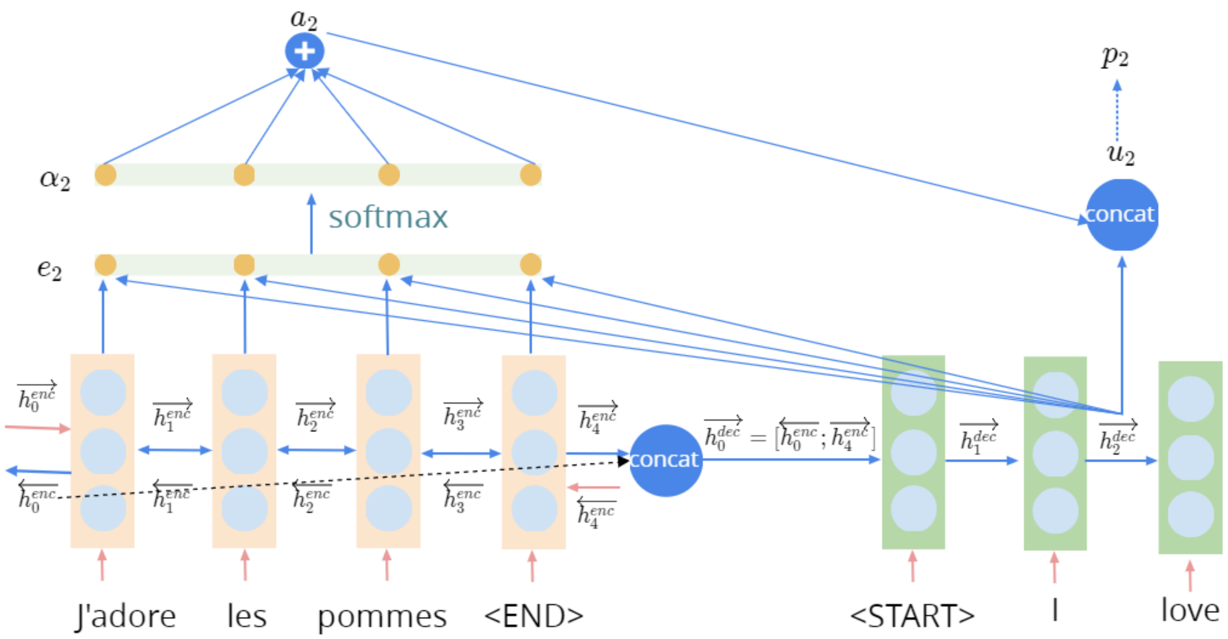
a_2
p_2
u_2
concat
α_2
softmax
e_2
$\overrightarrow{h_0^{enc}}$
$\overrightarrow{h_1^{enc}}$
$\overrightarrow{h_2^{enc}}$
$\overrightarrow{h_3^{enc}}$
$\overrightarrow{h_4^{enc}}$
$\overleftarrow{h_0^{enc}}$
$\overleftarrow{h_1^{enc}}$
$\overleftarrow{h_2^{enc}}$
$\overleftarrow{h_3^{enc}}$
$\overleftarrow{h_4^{enc}}$
concat
$\overrightarrow{h_0^{dec}} = [\overleftarrow{h_0^{enc}}; \overrightarrow{h_4^{enc}}]$
$\overrightarrow{h_1^{dec}}$
$\overrightarrow{h_2^{dec}}$
J'adore
les
pommes
<END>
<START>
I
love

Figure 2: Seq2Seq NMT (cell state is ignored in the figure)

W can produce a probability distribution over the target words vocabulary V at t^{th} time step:

$$p_t = \text{softmax}(W_{vocab} o_t^{dec}), \quad p_t \in \mathbb{R}^{|V| \times 1}, W_{vocab} \in \mathbb{R}^{|V| \times h} \tag{12}$$

where $|V|$ is the vocabulary size. Lastly, we compute the cross entropy loss between p_t and our target label l_t, where y_t is a one-hot vector of the target word at timestep t.

$$CE(p_t, y_t) = \sum_{i}^{|V|} y_{t,i} \ln p_{t,i} \tag{13}$$

Question 6 [code] (2 points) Before we build our model, we need to preprocess our data. Implement the *read_corpus* function in “utils” section.

Question 7 [code] (2 points) Implement part of the *__init__* function in “Encoder” class and “Decoder” class.

Question 8 [code] (5 points) Implement the *__init__* and *forward* function “Attention“ class. This function will generate attention distribution α_t.

Question 9 [code] (3 points) Implement the *forward* function in “Encoder” class . This function converts source sentences into word embedding tensors X, generates $h_1^{enc}, h_2^{enc}, ..., h_m^{enc}$ and computes initial hidden state h_0^{dec}, and initial cell state c_0^{dec}.

Question 10 [code] (7 points) Implement the *forward* function in "Decoder" class. This function constructs $\bar{y}_t$ and runs the *decode_one_step* function over every time step of the input sentence.

Question 11 [code] (6 points) Implement *decode_one_step* function in "Decoder" class. This function applies the decoder's LSTM Cell for a single time step, computing the encoding of the target word h_t^{dec}, the attention distribution α_t, attention output a_t and the combined output o_t. *Hint: You should be using the implemented "Attention" class for the computation of the attention distribution.*

Now you have finished building your NMT model. Run the remaining in you notebook. The BLEU score should be around 0.35. Run the model several times and report the **best** BLEU score.

How to submit

1. Fill up your student ID and name in the Jupyter Notebook.
2. Click the Save button at the top of the Jupyter Notebook.
3. Select Cell - All Output - Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
4. Select Cell Run All.
5. Once you've rerun everything, select File - Download as - PDF via LaTeX.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see! Submit your PDF on eDimension.