Assignments
M3224.000300 Natural Language Processing with Neural Networks
Fall 2024

Graduate School of Data Science
Seoul National University
Instructor: Jay-yoon Lee (lee.jayyoon@snu.ac.kr)
TA: Hye Won Jeon (pingpong0926@snu.ac.kr)

Due on Thursday Oct. 24, 2024 by 23:59 pm

(Assignment 3: Recurrent Neural Networks)

Honor Pledge for Graded Assignments
"I, YOUR NAME HERE , affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own."

0 Instructions

- Total score cannot exceed 100 points. For example, if you score 98 points from non-bonus questions and 3 points are added from bonus questions, your score will be 100 points, not 101 points.
- Skeleton codes for problem 3 are at the directory `/q3`.
- Run the `bash collect_submission.sh` script to produce your 2000_00000_coding.zip file. Please make sure to modify collect_submission.sh file before running this command. (**2000_00000** stands for your student id.)
- Modify this tex file into `2000_00000_written.pdf` with your written solutions.
- Upload both `2000_00000_coding.zip` and `2000_00000_written.pdf` to etl website.
- **If the submission instructions are not followed, `4 points` will be deducted.**

1 NLP tasks with RNN (20 pts)

RNNs are versatile! In class, we learned that this family of neural networks have many important advantages and can be used in a variety of tasks. They are commonly used in many state-of-the-art architectures for NLP.

For each of the following tasks, state how you would run RNN to do that task. In particular, specify how the RNN would be used at test time (not training time), and specify

- How many outputs i.e. number of times the softmax $\hat{y}^{(t)}$ is called from your RNN. If the number of outputs is not fixed, state it as arbitrary.
- What each $\hat{y}^{(t)}$ is a probability distribution over (e.g. distributed over all species of categories).
- Which inputs are fed to produce each output $\hat{y}^{(t)}$.

The inputs for each of the tasks are specified below.

(a) **Movie Rating (5 pts)**: Classify sentiment of a movie review ranging from negative to positive (integer values from 1 to 5).
Inputs: A sentence containing n words. **Answer:**
1. How many outputs: The softmax function $\hat{y}$ is called **once**.

2. What each $\hat{y}$ is a probability distribution over: $\hat{y}$ is a probability distribution over the **5 possible sentiment ratings** (integer values from 1 to 5), ranging from negative to positive sentiment.

3. Which inputs are fed to produce each output $\hat{y}$: The entire sentence containing n words is fed into the RNN. The RNN processes all n words sequentially, and uses the **final hidden state** to compute the output $\hat{y}$.

(b) **Part-of-speech Tagging (5 pts)**: For each word in a sentence, categorize that word in correspondence with a particular part-of-speech such as either nouns, verbs, adjectives, adverbs, etc.
Inputs: A sentence containing n words. **Answer:**
1. How many outputs: The softmax function $\hat{y}(t)$ is called n **times**, once for each word in the sentence.

2. What each $\hat{y}(t)$ is a probability distribution over: Each $\hat{y}(t)$ is a probability distribution over the **set of all possible part-of-speech tags**, such as nouns, verbs, adjectives, adverbs, etc.

3. Which inputs are fed to produce each output $\hat{y}(t)$: The entire sentence containing n words is fed into the RNN. At each time step t, the RNN processes up to the t-th word and generates a hidden state h_t, which is used to compute the output $\hat{y}(t)$ for the t-th word.

(c) **Text Generation (5 pts)**: Generate text from a chatbot that was trained to speak like a news anchor by predicting the next word in the sequence.
Input: A single start word or token that is fed into the first time step of the RNN. **Answer:**
1. How many outputs: The softmax function $\hat{y}(t)$ is called an **arbitrary** number of times, depending on the desired length of the generated text.

2. What each $\hat{y}(t)$ is a probability distribution over: Each $\hat{y}(t)$ is a probability distribution over the **entire vocabulary of possible words**, representing the probability of each word being the next word in the sequence.

3. Which inputs are fed to produce each output $\hat{y}(t)$:

- At the first time step, the input is the given start word or token.
- For each subsequent time step t, the input is the word generated at the previous time step $\hat{y}(t-1)$ (typically the word with the highest probability from $\hat{y}(t-1)$).
- The RNN uses the input word and the previous hidden state to produce the new hidden state and output $\hat{y}(t)$.

(d) **Machine Translation (5 pts)**: Translate the given sentence into another language.
Input: A sentence containing n words. **Answer:**
1. How many outputs: The softmax function $\hat{y}(t)$ is called an **arbitrary** number of times, corresponding to the number of words in the translated sentence, which may differ from n.

2. What each $\hat{y}(t)$ is a probability distribution over: Each $\hat{y}(t)$ is a probability distribution over the **vocabulary of the target language**, representing possible words that can be generated at each time step.

3. Which inputs are fed to produce each output $\hat{y}(t)$:

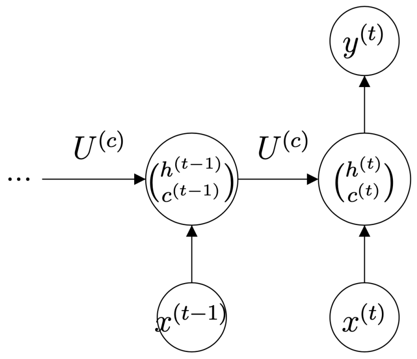
- The input sentence containing n words is first fed into an **encoder RNN**, which processes the sequence and produces a context vector.
- During decoding, at the first time step, the decoder RNN receives a special start-of-sequence token as input, and its initial hidden state is initialized with the context vector from the encoder.
- For each subsequent time step t, the decoder RNN takes the previously generated word (from time step $t-1$) as input.
- The decoder RNN uses the input word and its hidden state to produce the output $\hat{y}(t)$.

2 Backprop on LSTM (12 pts)

In class, we learned about Long Short-Term Memory (LSTM) model. Recall the units of an LSTM cell are defined as

$$\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) \\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \\
\tilde{c}_t &= tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ tanh(c_t)
\end{aligned}$$

where the final output of the last lstm cell is defined by $\hat{y}_t = softmax(h_t W + b)$. The final cost function J uses the cross-entropy loss. Consider an LSTM for two time steps, t and $t-1$.



(a) Derive the gradient $\frac{\delta J}{\delta U^{(c)}}$ in terms of the following gradients: $\frac{\delta h_t}{\delta h_{t-1}}, \frac{\delta h_{t-1}}{\delta U^{(c)}}, \frac{\delta J}{\delta h_t}, \frac{\delta c_t}{\delta U^{(c)}}, \frac{\delta c_{t-1}}{\delta U^{(c)}}, \frac{\delta c_t}{\delta c_{t-1}}, \frac{\delta h_t}{\delta c_t}$, and $\frac{\delta h_t}{\delta o_t}$. *Not all of the gradients may be used.* You can leave the answer in the form of chain rule and do not have to calculate any individual gradients in your final result. **(8 pts) Answer:**

$$\frac{\delta J}{\delta U^{(c)}} = \frac{\delta J}{\delta h_t}\left(\frac{\delta h_t}{\delta c_t}\frac{\delta c_t}{\delta U^{(c)}} + \frac{\delta h_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta U^{(c)}}\right)$$

(b) Which part of the gradient $\frac{\delta J}{\delta U^{(c)}}$ allows LSTM to mitigate the effect of the vanishing gradient problem? **(4 pts) Answer:**
The part of the gradient $\frac{\delta J}{\delta \mathbf{h}^{(t)}}$ that allows LSTMs to mitigate the effect of the vanishing gradient problem is the term:

$$\frac{\delta c_t}{\delta U^{(c)}} = \frac{\delta c_t}{\delta c_{t-1}}\frac{\delta c_{t-1}}{\delta U^{(c)}} + \frac{\delta c_t}{\delta \tilde{c}_t}\frac{\delta \tilde{c}_t}{\delta U^{(c)}}$$

specifically $\frac{\delta c_t}{\delta c_{t-1}}$.

This derivative is given by:

$$\frac{\delta c_t}{\delta c_{t-1}} = f_t$$

Due to the additive nature of the cell state update in LSTMs:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

the gradient can flow through many time steps without vanishing. The forget gate f_t controls the preservation of information, and when f_t is close to 1, the gradient $\frac{\delta c_t}{\delta c_{t-1}}$ remains close to 1, allowing gradients to be effectively propagated backward through time.

3 Neural Machine Translation with LSTM (68+3 pts)

In Neural Machine Translation (NMT), our goal is to convert a sentence from the *source* language to the *target* language. In this assignment, we will implement a sequence-to-sequence (Seq2Seq) network with attention, to build a Neural Machine Translation (NMT) system between Jeju dialect and Korean.

3.1 Training Procedure

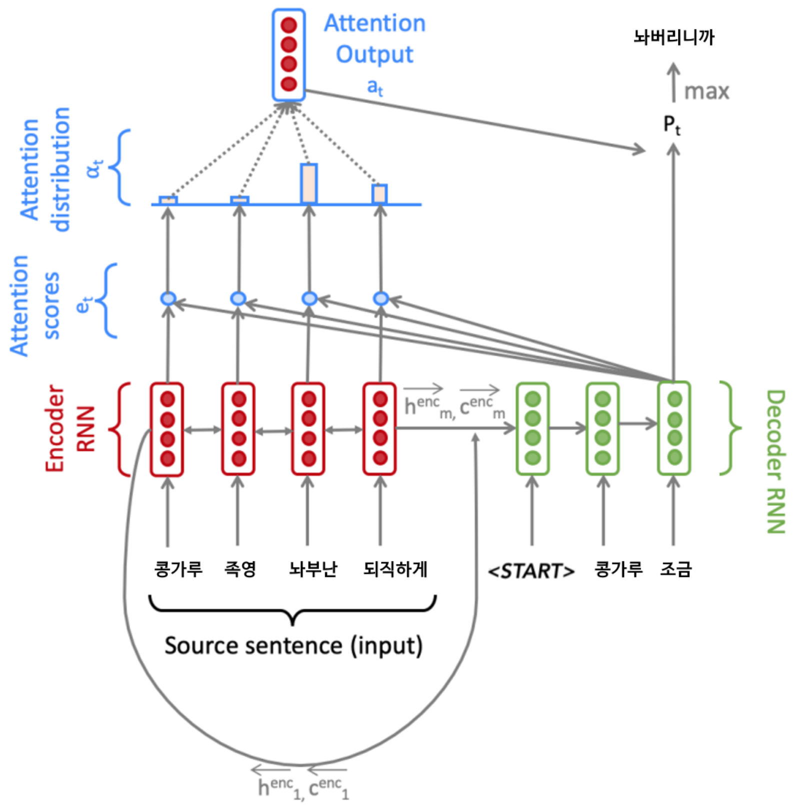


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. (NOTE: Embedding of NMT in the assignment differs from described above.)

In this section, we describe the **training procedure** for the proposed NMT system, which uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder. The model is trained and evaluated on JIT (Jejueo interview transcripts) dataset[1]. Given a sentence in the source language (Jeju dialect), we look up the subword embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \dots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e\times 1}$), where m is the length of the source sentence and e is the embedding size. We feed these embeddings to the bidirectional encoder, yielding hidden states and cell states for both the forwards ($\rightarrow$) and backwards ($\leftarrow$) LSTMs. The forwards and backwards versions are concatenated to make hidden states $\mathbf{h}_i^{enc}$ and cell states $\mathbf{c}_i^{enc}$:

$$\mathbf{h}_i^{\text{enc}} = [\overleftarrow{\mathbf{h}_i^{enc}}; \overrightarrow{\mathbf{h}_i^{enc}}] \text{ where } \mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\mathbf{h}_i^{enc}}, \overrightarrow{\mathbf{h}_i^{enc}} \in \mathbb{R}^{h\times 1} \qquad 1 \le i \le m \qquad (1)$$

$$\mathbf{c}_i^{\text{enc}} = [\overleftarrow{\mathbf{c}_i^{enc}}; \overrightarrow{\mathbf{c}_i^{enc}}] \text{ where } \mathbf{c}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\mathbf{c}_i^{enc}}, \overrightarrow{\mathbf{c}_i^{enc}} \in \mathbb{R}^{h\times 1} \qquad 1 \le i \le m \qquad (2)$$

We then initialize the decoder's first hidden state $\mathbf{h}_0^{\text{dec}}$ and cell state $\mathbf{c}_0^{\text{dec}}$ with a linear projection of the encoder's final hidden state and final cell state.[2]

[1] https://www.kaggle.com/datasets/bryanpark/jit-dataset

[2] Note that we regard $[\overleftarrow{\mathbf{h}_1^{enc}}, \overrightarrow{\mathbf{h}_m^{enc}}]$ as the 'final hidden state' of the Encoder.

$$\mathbf{h}_0^{\text{dec}} = \mathbf{W}_h[\overleftarrow{\mathbf{h}_1^{enc}}; \overrightarrow{\mathbf{h}_m^{enc}}] \text{ where } \mathbf{h}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_h \in \mathbb{R}^{h\times 2h} \tag{3}$$

$$\mathbf{c}_0^{\text{dec}} = \mathbf{W}_c[\overleftarrow{\mathbf{c}_1^{enc}}; \overrightarrow{\mathbf{c}_m^{enc}}] \text{ where } \mathbf{c}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_c \in \mathbb{R}^{h\times 2h} \tag{4}$$

With the decoder initialized, we must now feed it a target sentence. On the t^{th} step, we look up the embedding for the t^{th} subword, $\mathbf{y}_t \in \mathbb{R}^{e\times 1}$. We then concatenate $\mathbf{y}_t$ with the *combined-output vector* $\mathbf{o}_{t-1} \in \mathbb{R}^{h\times 1}$ from the previous timestep (we will explain what this is later down this page!) to produce $\overline{\mathbf{y}_t} \in \mathbb{R}^{(e+h)\times 1}$. Note that for the first target subword (i.e. the start token) $\mathbf{o}_0$ is a zero-vector. We then feed $\overline{\mathbf{y}_t}$ as input to the decoder.

$$\mathbf{h}_t^{\text{dec}}, \mathbf{c}_t^{\text{dec}} = \text{Decoder}(\overline{\mathbf{y}_t}, \mathbf{h}_{t-1}^{\text{dec}}, \mathbf{c}_{t-1}^{\text{dec}}) \text{ where } \mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{c}_t^{\text{dec}} \in \mathbb{R}^{h\times 1} \tag{5}$$

$$\tag{6}$$

We then use $\mathbf{h}_t^{\text{dec}}$ to compute multiplicative attention over $\mathbf{h}_1^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$:

$$\mathbf{e}_{t,i} = (\mathbf{h}_t^{\text{dec}})^T \mathbf{W}_{\text{attProj}} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{e}_t \in \mathbb{R}^{m\times 1}, \mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h} \qquad 1 \le i \le m \tag{7}$$

$$\alpha_t = \text{softmax}(\mathbf{e}_t) \text{ where } \alpha_t \in \mathbb{R}^{m\times 1} \tag{8}$$

$$\mathbf{a}_t = \sum_{i=1}^{m} \alpha_{t,i} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{a}_t \in \mathbb{R}^{2h\times 1} \tag{9}$$

Here, $\mathbf{e}_{t,i}$ is a scalar, the i^{th} element of $\mathbf{e}_t \in \mathbb{R}^{m\times 1}$, computed using the hidden state of the decoder at the t^{th} step $\mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h\times 1}$, the attention projection $\mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h}$, and the hidden state of the encoder at the i^{th} step $\mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}$.

We now concatenate the attention output $\mathbf{a}_t$ with the decoder hidden state $\mathbf{h}_t^{\text{dec}}$ and pass this through a linear layer, tanh, and dropout to attain the *combined-output* vector $\mathbf{o}_t$.

$$\mathbf{u}_t = [\mathbf{a}_t; \mathbf{h}_t^{\text{dec}}] \text{ where } \mathbf{u}_t \in \mathbb{R}^{3h\times 1} \tag{10}$$

$$\mathbf{v}_t = \mathbf{W}_u \mathbf{u}_t \text{ where } \mathbf{v}_t \in \mathbb{R}^{h\times 1}, \mathbf{W}_u \in \mathbb{R}^{h\times 3h} \tag{11}$$

$$\mathbf{o}_t = \text{dropout}(\tanh(\mathbf{v}_t)) \text{ where } \mathbf{o}_t \in \mathbb{R}^{h\times 1} \tag{12}$$

Then, we produce a probability distribution $\mathbf{P}_t$ over target subwords at the t^{th} timestep:

$$\mathbf{P}_t = \text{softmax}(\mathbf{W}_{\text{vocab}} \mathbf{o}_t) \text{ where } \mathbf{P}_t \in \mathbb{R}^{V_t\times 1}, \mathbf{W}_{\text{vocab}} \in \mathbb{R}^{V_t\times h} \tag{13}$$

Here, V_t is the size of the target vocabulary. Finally, to train the network we compute the cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the one-hot vector of the target subword at the timestep t:

$$J_t(\theta) = \text{CrossEntropy}(\mathbf{P}_t, \mathbf{g}_t) \tag{14}$$

Here, θ represents all the parameters of the model and $J_t(\theta)$ is the loss on the step t of the decoder. Now that we have described the model, let's try implementing it for Jeju dialect to Korean translation!

3.2 Setting up Virtual Environment

In this part, we will set up a virtual environment for implementing the NMT machine. Please note that the following instructions are based on the gsds server as announced on the ETL board [3]. Run the following commands within the assignment directory (`/q3`) to create the appropriate conda environment. This guarantees that you have all the necessary packages to complete the assignment.

[3]https://myetl.snu.ac.kr/courses/264449/discussion_topics/202254

```
conda create -n a3q3 python=3.12
conda activate a3q3
srun --gres=gpu:1 bash env.sh
```

3.3 Implementation Questions

(a) To ensure the sentences in a given batch are of the same length, we must pad shorter sentences to be the same length after identifying the longest sentence in a batch. Implement the `pad_sents` function in `utils.py`, which returns padded sentences. **(5 pts)**

(b) Implement the code of class `LSTMCell_assignment` in `assignment_code.py`. LSTMCell contains two functions: initialization `__init__()` and forward `forward()`. You can refer to the PyTorch documentations [4] or `GRUCell_assignment` class which is implemented on the skeleton code. **(10 pts)**

(c) Implement the `__init__` function of `NMT` class in `nmt_model.py` to initialize layers for the NMT system. You can run sanity check by executing `python sanity_check.py 1c` **(5 pts)**

(d) Implement the `encode` function in `nmt_model.py`. This function converts the padded source sentences into the tensor $\mathbf{X}$, generates $\mathbf{h}_1^{enc}...\mathbf{h}_m^{enc}$, and computes the initial state $\mathbf{h}_0^{dec}$ and initial cell $\mathbf{c}_0^{dec}$ for the Decoder. You can run sanity check by executing `python sanity_check.py 1d` **(8 pts)**

(e) Implement the `decode` function in `nmt_model.py`. This function constructs $\overline{\mathbf{y}}$ and runs the step function over every timestep for the input. You can run sanity check by executing `python sanity_check.py 1e` **(8 pts)**

(f) Implement the `step` function in `nmt_model.py`. This function applies the Decoder's LSTM cell for a single timestep, computing the encoding of the target subword $\mathbf{h}_t^{\text{dec}}$, the attention scores $\mathbf{e}_t$, attention distribution α_t, the attention output $\mathbf{a}_t$, and finally the combined output $\mathbf{o}_t$. You can run a non-comprehensive sanity check by executing `python sanity_check.py 1f` **(5 pts)**

(g) Let's train the model! execute the following command:

```
sh run.sh vocab
sh run.sh train
```

Check out the model is running on GPU when training. Training takes within one GPU hour. After training your model, execute the following command to test the model:

```
sh run.sh test
```

Write down the execution time and BLEU score. To get a full credit, BLEU score should be larger than 50. **(5 pts)** **Answer:**

Decoding time: 122.89064693450928 sec

Corpus BLEU: 62.62832645210676

(h) There are a few different methods to generate text from a decoder model such as greedy decoding, beam search, top-k sampling, and top-p sampling. In this code, beam search with a default beam size of 10 is utilized. You can modify the beam size by passing it as an argument in the following way:

```
sh run.sh test <beam-size>
```

Now, perform the decoding with beam size of 1, 3, 5, 10 and 25. Note that beam search with a beam size of 1 is equivalent to greedy decoding. Compare the execution time and performance with different beam sizes. Explain the observed trends as well as the potential reasons for the trends. Discuss distinctions between beam search (beam size greater than 1) and greedy decoding, considering expected and observed differences. **(8 pts)**

[4] LSTMCell: https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html
GRUCell: https://pytorch.org/docs/stable/generated/torch.nn.GRUCell.html

Answer:

Beam Size	Decoding Time (sec)	BLEU Score
1 (Greedy)	80.24	62.94
3	99.90	63.27
5	102.85	63.07
10	122.89	62.63
25	178.94	60.95

Table 1: Comparison of Different Beam Sizes

Initially, I hypothesized that larger beam sizes would yield better translation quality. Based on experimental results with beam sizes 1, 3, 5, 10, and 25, I observed that larger beam sizes did not necessarily lead to better performance. While greedy decoding (beam size=1) makes locally optimal choices at each step, beam search ($beamsize > 1$) maintains multiple hypotheses, theoretically allowing for better global optimization. However, only beam size 3 showed meaningful improvement over greedy decoding (BLEU 63.27 vs 62.94), while further increases in beam size resulted in degraded performance, with beam size 25 showing the lowest score (60.95). The decoding time increased linearly with beam size, from 80.24 seconds for greedy decoding to 178.94 seconds for beam size 25, suggesting that the optimal balance between performance and computational cost is achieved with a relatively small beam size of 3.

(i) **(BONUS)** Conduct additional experiments using various beam sizes to determine the best one. Record your chosen beam size and justify your decision. Research established guidelines for beam size selection (or any rule-of-thumb value for beam size) and contrast your choice or reasoning with these conventions. **(3 pts)** **Answer:**

Beam Size	Decoding Time (sec)	BLEU Score
2	83.40	63.19
3	99.90	63.27
4	101.95	63.27

Table 2: Fine-grained Beam Size Comparison

Based on my comprehensive experiments with beam sizes ranging from 1 to 25, and additional fine-grained testing around beam size 3, I recommend using a beam size of 3 for our NMT system. While conventional wisdom and research guidelines often suggest larger beam sizes (typically 5-10), our experiments revealed that beam size 3 achieves optimal performance (BLEU score 63.27) with relatively lower computational cost. Additional experiments with beam sizes 2 and 4 showed nearly identical BLEU scores but slightly longer decoding times, further confirming that beam size 3 represents the sweet spot between translation quality and computational efficiency in my specific implementation.

3.4 Written Questions

BLEU score is the most commonly used automatic evaluation metric for NMT systems. It is usually calculated across the entire test set, but here we will consider BLEU defined for a single example.[5] Suppose we have a source sentence $\mathbf{s}$, a set of k reference translations $\mathbf{r}_1, \dots, \mathbf{r}_k$, and a candidate translation $\mathbf{c}$. To compute the BLEU score of $\mathbf{c}$, we first compute the *modified n-gram precision* p_n of $\mathbf{c}$, for each of $n = 1, 2, 3, 4$, where n is the n in n-gram:

$$p_n = \frac{\displaystyle\sum_{\text{ngram}\in\mathbf{c}} \min\left(\max_{i=1,\dots,k} \text{Count}_{\mathbf{r}_i}(\text{ngram}),\ \ \text{Count}_{\mathbf{c}}(\text{ngram}) \right)}{\displaystyle\sum_{\text{ngram}\in\mathbf{c}} \text{Count}_{\mathbf{c}}(\text{ngram})} \tag{15}$$

[5]This definition of sentence-level BLEU score matches the `sentence_bleu()` function in the `nltk` Python package. Note that the NLTK function is sensitive to capitalization. In this question, all text is lowercased, so capitalization is irrelevant.
http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleu

Here, for each of the n-grams that appear in the candidate translation **c**, we count the maximum number of times it appears in any one reference translation, capped by the number of times it appears in **c** (this is the numerator). We divide this by the number of n-grams in **c** (denominator).

Next, we compute the *brevity penalty* BP. Let $len(c)$ be the length of **c** and let $len(r)$ be the length of the reference translation that is closest to $len(c)$ (in the case of two equally-close reference translation lengths, choose $len(r)$ as the shorter one).

$$BP = \begin{cases} 1 & \text{if } len(c) \geq len(r) \\ \exp\left(1 - \frac{len(r)}{len(c)}\right) & \text{otherwise} \end{cases} \tag{16}$$

Lastly, the BLEU score for candidate **c** with respect to $\mathbf{r}_1, \ldots, \mathbf{r}_k$ is:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{4} \lambda_n \log p_n\right) \tag{17}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights that sum to 1. The log here is natural log.

(a) Consider this example.

- Source Sentence **s**: **그때는 뭐 사먹을 것도 엇일 때고 학습장이나 사던지 헤엇던 거 가따**
- Reference Translation $\mathbf{r}_1$: 그때는 뭐 사먹을 것도 없을 때고 학습장이나 사던지 했었던 거 같아
- Reference Translation $\mathbf{r}_2$: 그때는 뭐 사먹을 것도 없을 시절이고 학습장이나 사든지 했었던 거 같다
- NMT Translation **c**: 그때는 뭐 사먹을 것도 없을 때고 학습장이나 사든지 했었던 거 가

Please compute the BLEU scores for **c**. Let $\lambda_i = 0.5$ for $i \in \{1, 2\}$ and $\lambda_i = 0$ for $i \in \{3, 4\}$ (**this means we ignore 3-grams and 4-grams**, i.e., don't compute p_3 or p_4). When computing BLEU scores, show your working (i.e., show your computed values for p_1, p_2, $len(c)$, $len(r)$ and BP). You can solve this problem by implementing your own Python code, using Excel, or employing other similar methods. Note that the BLEU scores can be expressed between 0 and 1 or between 0 and 100. In this question we are using the **0 to 1** scale. **(10 pts)** **Answer:**

1-gram precision (p_1): 0.9091

2-gram precision (p_2): 0.9000

Length of candidate (len_c): 11

Closest reference length (len_r): 11

Brevity Penalty (BP): 1.0000

$\therefore$ BLEU score: 0.9045

(b) Due to data availability, NMT systems are often evaluated with respect to only a single reference translation. Please explain (in a few sentences) why this may be problematic. In your explanation, discuss how the BLEU score metric assesses the quality of NMT translations when there are multiple reference translations versus a single reference translation. **(4 pts)** **Answer:** Evaluating NMT systems with a single reference translation presents several critical limitations. First, it fails to account for multiple valid translations that convey the same meaning, as natural languages often allow various ways to express identical concepts. Second, it cannot capture the contextual flexibility of translations, where different expressions might be more appropriate depending on the context. Finally, using a single reference can lead to biased evaluation towards specific domains or styles, potentially underestimating or misrepresenting the actual performance of translation systems that produce accurate but differently worded translations.