Assignments

Applications of Natural Language Processing (Intro to NLP and its recent progress)

Fall 2022

Graduate School of Data Science
Seoul National University

Instructor: Dr.Jay-yoon Lee (lee.jayyoon@snu.ac.kr)
Head TA: Jonghyun Song (hyeongoon11@snu.ac.kr)
TA: Yikyung Kim (k2y1513@snu.ac.kr)

Due on Tuesday Oct. 18, 2022 by 3:30pm (before class)

(Assignment 2: Recurrent Neural Networks & Transformers)

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 (3)

- Total score cannot exceed 100 points. For example, if you score 95 points from non-bonus questions and 10 points are added from bonus questions, your score will be 100 points, not 105 points.
- You can download the skeleton code file and TeX file HERE
- Skeleton codes for problem 2 and 3 are at the directory `/q2` and `/q3` each. Problem 1 does not have coding problems.
- Run the `bash collect_submission.sh` script to produce your 2000_00000_coding.zip file. Please make sure to modify collect_sumbssion.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. **(4pts)**

1 NLP tasks with RNN (9+3)

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 at each time step to produce each output
- **(Bonus)** What is one advantage that an RNN would have over a neural window-based model for this task?

The inputs for each of the tasks are specified below.

(a) **Movie Rating (4pts)**: Classify sentiment of a movie review ranging from negative to positive (integer values from 1 to 5).
Inputs: A sentence containing n words. **Answer:**

(b) **Part-of-speech Tagging (4pts)**: 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:**

(c) **Text Generation (4pts)**: 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:**

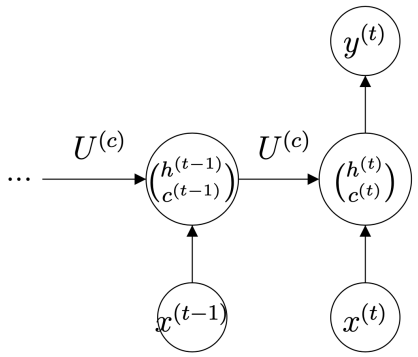
2 GRU & LSTM (52+6)

2.1 Backprop on LSTM

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. **(3 pts) Answer:**

(b) Which part of the gradient $\frac{\delta J}{\delta U^{(c)}}$ allows LSTM to mitigate the effect of the vanishing gradient problem? Explain in two sentences or less how this would help classify the correct sentiment for the sentence in 1.2 part (b). **(3 pts) Answer:**

(c) Rather than using the last hidden state to output the sentiment of a sentence, what could be a better solution to improve the performance of the sentiment analysis task? **(3 pts) Answer:**

2.2 Neural Machine Translation with LSTM

In Nerual 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. In this section, we describe the **training procedure** for the proposed NMT system, which uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder. After training, you can find out how the NMT system

is better than you in translating Jeju dialect. The model is trained and evaluated on JIT (Jejueo interview transcripts) dataset [1]

2.2.1 Training Procedure

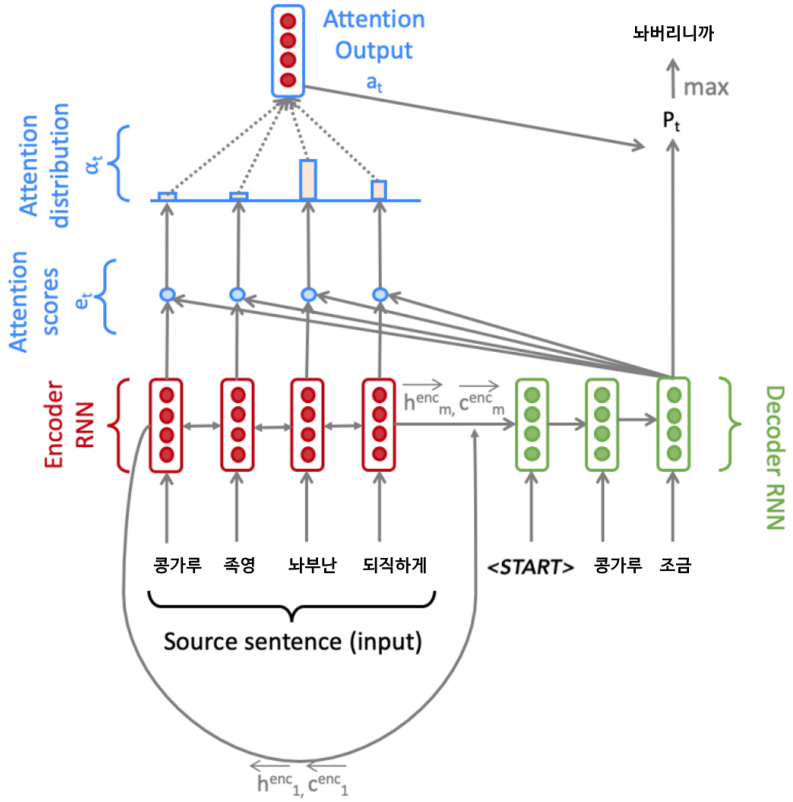


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. (NOTE: Embedding of NMT in the assignment differs from described above.)

Given a sentence in the source language (Jeju dialect), we look up the subword embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \ldots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e \times 1}$), 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 give hidden states $\mathbf{h}_i^{enc}$ and cell states $\mathbf{c}_i^{enc}$:

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

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

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]

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

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

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

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

[2] If it's not obvious, think about why we regard $[\overleftarrow{\mathbf{h}_1^{enc}}, \overrightarrow{\mathbf{h}_m^{enc}}]$ as the 'final hidden state' of the Encoder.

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 $\mathbf{h}_1^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$:

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

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

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

$\mathbf{e}_{t,i}$ is a scalar, the ith element of $\mathbf{e}_t \in \mathbb{R}^{m\times 1}$, computed using the hidden state of the decoder at the tth 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 ith 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 then 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 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 step t of the decoder. Now that we have described the model, let's try implementing it for Jeju dialect to Korean translation!

2.2.2 Setting up Virtual Environment

In this part, we will set up a virtual environment for implementing the NMT machine. Before you begin, make sure to read instructions about using GSDS cluster on the ETL board. Run the following commands within the assignment directory (`a2`) to create the appropriate conda environment. This guarantees that you have all the necessary packages to complete the assignment. You will be asked to implement LSTM cells and the seq2seq model using the PyTorch package.

```
bash env.sh
conda activate a2
```

2.2.3 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. **(3 pts)**

(b) Implement the code of class `LSTMCell` in the file `assignment_code.py`. LSTMCell contains two functions: initialization `__init__()` and forward `forward()`. You can refer to the PyTorch documentation[3] or GRUcell class which is implemented on the skeleton code. You can run sanity check by executing `python sanity_check.py 1c` **(3 pts)**

(c) Implement the `__init__` function in `model_embeddings.py` and `nmt_model.py` to initialize the necessary model embeddings and layers for the NMT system. **(3 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` **(3 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` **(3 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` **(3 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. **(0 pts)**

(h) After training your model, execute the following command to test the model:

```
sh run.sh test
```

To get a full credit, BLEU score should be larger than 50. **(3 pts)**

2.2.4 Written Questions

(a) The alphabet ㆍ(*araea* (아래아)), which are not used in contemporary Korean, is frequently used in jeju dialect. Because of this, conventional preprocessing tools for Korean may cause problems. Analyze `vocab.py` and state how the skeleton code fixed up the issue. **(3 pts) Answer:**

(b) **(Bonus)** As you can see in `vocab.py`, we modeled our NMT problem at a subword-level. That is, given a sentence in the source language, we looked up subword components from an embeddings matrix. Alternatively, we could have modeled the NMT problem at the word-level, by looking up whole words from the embeddings matrix. Why might it be important to model our Jejueo-to-Korean NMT problem at the subword-level vs. the whole word-level? **(3 pts) Answer:**

(c) Here we present a series of error we found in the outputs of NMT model. For each example of a reference Korean translation and NMT Korean translation, (1) please identify the error in NMT translation, (2) provide possible reasons why the model may have made the errors, and (3) Describe **ONE** possible way to fix the observed error. ($< unk >$ refers to *unknown* token) **6 pts)**

(i) **Source Sentence:** 가운데 대가 잇주 , 가운데 . 가운데 대 놔 가지고서 이제 만들면은 그 실 고망 , 고망 안터렐이 그 불로 무시거 불깍살로 영 헤근에 불부트민 불깍살 꺼 버리면 그디 불 잇잖느냐이 ?

Reference Translation: 가운데 대가 있지 , 가운데 . 가운데 대 놔 가지고서 이제 만들면 그 실 구멍 , 구멍 안으로 그 불로 무엇 성냥개비로 이렇게 해서 불붙으면 성냥개비 꺼 버리면 거기 불 있잖니 ?

NMT Translation: 가운데 대가 있지 , 가운데 . 가운데 대 놔 가지고서 이제 만들면 그 실 구멍 , 구멍 안터$< unk >$이 그 불로 무엇 불깍살로 이렇게 해서 불붙으면 불깍살 거 버리면 거기 불 있잖아 ?
Answer:

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

(ii) **Source Sentence:** ㅎ、ㄴ 치가 얼만곤 허민 성냥쌀 ㅎ、나가 ㅎ、ㄴ 치라 .

Reference Translation: 한 치가 얼마인가 하면 성냥개비 하나가 한 치야 .

NMT Translation: 한 치가 얼만곤 하면 성냥개 하나가 한 치라 .
Answer:

(iii) **Source Sentence:** 여름은 나민 저 거시기 요센 말로 난닝구 닮게시리 .

Reference Translation: 여름은 되면 저 거시기 요샌 말로 러닝 같게끔 .

NMT Translation: 여름은 나면 저 거시기 요샌 말로 $< unk >$게끔 .
Answer:

(d) You can use dot product attention, multiplicative attention, and additive attention to calculate attention score. **(4 pts)**

(i) explain ONE advantage and disadvantage of dot product attention ($\mathbf{s}^T\mathbf{h}_i$) compared to multiplicative attention ($\mathbf{s}^T\mathbf{W}\mathbf{h}_i$). **Answer:**

(ii) explain ONE advantage and disadvantage of dot product attention ($\mathbf{s}^T\mathbf{h}_i$) compared to additive attention ($\mathbf{v}^T \tanh(\mathbf{W_1 h_i} + \mathbf{W_2 s_t})$). **Answer:**

(e) 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.[4] Suppose we have a source sentence $\mathbf{s}$, a set of k reference translations $\mathbf{r}_1, \ldots, \mathbf{r}_k$, 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:

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

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

Next, we compute the *brevity penalty* BP. Let $len(c)$ be the length of $\mathbf{c}$ 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 $\mathbf{c}$ 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.

(i) Consider this example.

- Source Sentence $\mathbf{s}$: **그때는 뭐 사먹을 것도 엇일 때고 주로 뭐 학습장이나 작기장이나 사던지 그렇게 헤엇던 거 가따**
- Reference Translation $\mathbf{r}_1$: 그때는 뭐 사먹을 것도 없을 때고 주로 뭐 학습장이나 종합장이나 사던지 그렇게 했었던 거 같아

[4] 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

- Reference Translation $\mathbf{r}_2$: 그때는 뭐 사먹을 것도 없을 시절이고 주로 뭐 학습장이나 잡기장이나 사든지 그렇게 했었던 거 같다
- NMT Translation $\mathbf{c}_1$: 그때는 뭐 사먹을 것도 없을 때고 주로 뭐 학습장이나 작기장이나 사든지 그렇게 했었던 거 가

Please compute the BLEU scores for $\mathbf{c}_1$ and $\mathbf{c}_2$. 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). Note that the BLEU scores can be expressed between 0 and 1 or between 0 and 100. The code is using the 0 to 100 scale while in this question we are using the **0 to 1** scale. **(6 pts) Answer:**

(ii) 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 transitions versus a single reference translation. **(3 pts) Answer:**

(iii) **(BONUS)** List ONE advantage and disadvantage of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation. **(3 pts) Answer:**

3 Transformer (36)

3.1 Attention Exploration

Multi-headed self-attention is the core modeling component of Transformers. In this question, we'll get some practice working with the self-attention equations, and motivate why multi-headed self-attention can be preferable to single-headed self-attention.

Recall that attention can be viewed as an operation on a *query* $q \in \mathbb{R}^d$, a set of *value* vectors $\{v_1, \ldots, v_n\}, v_i \in \mathbb{R}^d$, and a set of *key* vectors $\{k_1, \ldots, k_n\}, k_i \in \mathbb{R}^d$, specified as follows:

$$c = \sum_{i=1}^{n} v_i \alpha_i \tag{18}$$

$$\alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)} \tag{19}$$

with α_i termed the "attention weights". Observe that the output $c \in \mathbb{R}^d$ is an average over the value vectors weighted with respect to α_i.
NOTE: Do not take this problem so seriously. Every answer can be stated within 5 lines!

(a) **An average of two.** A Transformer model might want to incorporate information from *multiple* source vectors. Consider the case where we want to incorporate information from **two** vectors v_a and v_b, with corresponding key vectors k_a and k_b.

(i) How should we combine two d-dimensional vectors v_a, v_b into one output vector c in a way that preserves information from both vectors? In machine learning, one common way to do so is to take the average: $c = \frac{1}{2}(v_a + v_b)$. It might seem hard to extract information about the original vectors v_a and v_b from the resulting c, but under certain conditions one can do so. In this problem, we'll see why this is the case.

Suppose that although we don't know v_a or v_b, we do know that v_a lies in a subspace A formed by the m basis vectors $\{a_1, a_2, \ldots, a_m\}$, while v_b lies in a subspace B formed by the p basis vectors $\{b_1, b_2, \ldots, b_p\}$. (This means that any v_a can be expressed as a linear combination of its basis vectors, as can v_b. All basis vectors have norm 1 and orthogonal to each other.) Additionally, suppose that the two subspaces are orthogonal; i.e. $a_j^\top b_k = 0$ for all j, k.
Using the basis vectors $\{a_1, a_2, \ldots, a_m\}$, construct a matrix M such that for arbitrary vectors $v_a \in A$ and $v_b \in B$, we can use M to extract v_a from the sum vector $s = v_a + v_b$. In other words, we want to construct M such that for any v_a, v_b, $Ms = v_a$).
Note: both M and v_a, v_b should be expressed as a vector in $\mathbb{R}^d$, not in terms of vectors from A and B.

Hint: Given that the vectors $\{a_1, a_2, \ldots, a_m\}$ are both *orthogonal* and *form a basis* for v_a, we know that there exist some $c_1, c_2, \ldots, c_m$ such that $v_a = c_1 a_1 + c_2 a_2 + \cdots + c_m a_m$. Can you create a vector of these weights c? **(3 pts)**

Answer:

(ii) As before, let v_a and v_b be two value vectors corresponding to key vectors k_a and k_b, respectively. Assume that (1) all key vectors are orthogonal, so $k_i^\top k_j = 0$ for all $i \neq j$; and (2) all key vectors have norm 1.[5] **Find an expression** for a query vector q such that $c \approx \frac{1}{2}(v_a + v_b)$.
(HINT: while the softmax function will never *exactly* average the two vectors, you can get close by using a large scalar multiple in the expression.) **(3 pts) Answer:**

(b) **Drawbacks of single-headed attention:** In the previous part, we saw how it was *possible* for a single-headed attention to focus equally on two values. The same concept could easily be extended to any subset of values. In this question we'll see why it's not a *practical* solution. Consider a set of key vectors $\{k_1, \ldots, k_n\}$ that are now randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means $\mu_i \in \mathbb{R}^d$ are known to you, but the covariances Σ_i are unknown. Further, assume that the means μ_i are all perpendicular; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

(i) Assume that the covariance matrices are $\Sigma_i = \alpha I \forall i \in \{1, 2, \ldots, n\}$, for vanishingly small α. Design a query q in terms of the μ_i such that as before, $c \approx \frac{1}{2}(v_a + v_b)$, and provide a brief argument as to why it works. **(3 pts) Answer:**

(ii) Though single-headed attention is resistant to small perturbations in the keys, some types of larger perturbations may pose a bigger issue. Specifically, in some cases, one key vector k_a may be larger or smaller in norm than the others, while still pointing in the same direction as μ_a. As an example, let us consider a covariance for item a as $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α (as shown in figure 2). This causes k_a to point in roughly the same direction as μ_a, but with large variances in magnitude. Further, let $\Sigma_i = \alpha I$ for all $i \neq a$.

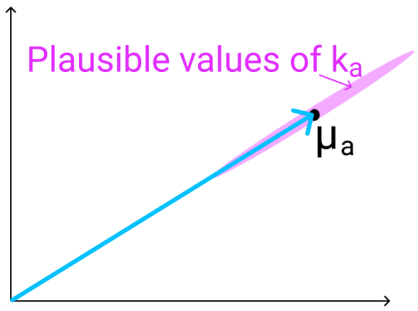


Figure 2: The vector μ_a (shown here in 2D as an example), with the range of possible values of k_a shown in red. As mentioned previously, k_a points in roughly the same direction as μ_a, but may have larger or smaller magnitude.

When you sample $\{k_1, \ldots, k_n\}$ multiple times, and use the q vector that you defined in part i., what qualitatively do you expect the vector c will look like for different samples? **(3 pts)**

Answer:

(c) **Benefits of multi-headed attention:** Now we'll see some of the power of multi-headed attention. We'll consider a simple version of multi-headed attention which is identical to single-headed self-attention as we've presented it in this homework, except two query vectors (q_1 and q_2) are defined, which leads to a pair of vectors (c_1 and c_2), each the output of single-headed attention given its respective query vector. The final output of the multi-headed attention is their average, $\frac{1}{2}(c_1 + c_2)$. As in question 1((b)), consider a set of key vectors $\{k_1, \ldots, k_n\}$ that are randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Also as before, assume that the means μ_i are mutually orthogonal; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

(i) Assume that the covariance matrices are $\Sigma_i = \alpha I$, for vanishingly small α. Design q_1 and q_2 such that c is approximately equal to $\frac{1}{2}(v_a + v_b)$. **(3 pts)**

Answer:

[5] Recall that a vector x has norm 1 iff $x^\top x = 1$.

(ii) Assume that the covariance matrices are $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α, and $\Sigma_i = \alpha I$ for all $i \neq a$. Take the query vectors q_1 and q_2 that you designed in part i. What, qualitatively, do you expect the output c to look like across different samples of the key vectors? Please briefly explain why. You can ignore cases in which $k_a^\top q_i < 0$. **(3 pts)**

Answer:

3.2 Implement Minimalist Version of BERT

In this assignment, you will implement some important components of the BERT model to better understand its architecture. You will then perform sentence classification with the model you implemented. There are no written questions in this section. For more details about transformer architecture, please refer to the paper *Attention is all you need*: https://arxiv.org/pdf/1706.03762.pdf
Run `setup.sh` to properly set up the environment. You are NOT allowed to use external libraries such as `transformers` in torch.

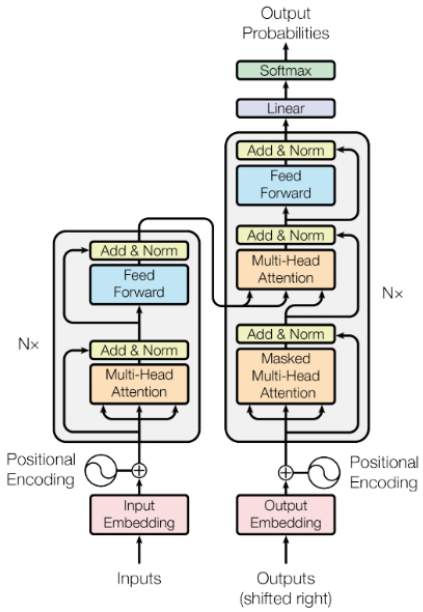


Figure 1: The Transformer - model architecture.

Figure 3: Model architecture of the transformer

(a) Implement `attention` function in `BertSelfAttention` class of `bert.py`. Scaled dot-product attention are computed as below:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{20}$$

(3 pts)

(b) Implement `add_norm` function in `BertLayer` class of `bert.py`. It corresponds to Add & Norm Layer in the figure 3, which is defined as $\text{LayerNorm}(x + \text{Sublayer}(x))$ **(3 pts)**

(c) Implement `forward` function in `BertLayer` class of `bert.py`. It corresponds to encoder stacks which are described on the left of the figure 3. After completing the codes, run `python sanity_check.py` to check your implementation. **(3 pts)**

(d) Implement `step` function in `AdamW` class of `optimizer.py`. Implement parameter update of Adam optimizer. For sanity check, run `python optimizer_test.py` **(3 pts)**

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{21}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t \tag{22}$$

$$\theta_t - = \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}\hat{m}_t} \text{ where } \hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \text{ and } \hat{v}_t = \frac{v_t}{1 - (\beta_2)^t} \tag{23}$$

(e) Implement `train` function in `classifier.py`. This function carries out backpropagation through the model parameters. **(3 pts)**

(f) Show time! Run `python3 classifier.py --option finetune --epochs NUM_EPOCHS --lr LR --train data/ssh-train.txt --dev data/ssh-dev.txt --test data/ssh-test.txt --use_gpu` . Report hyperparameters NUM_EPOCHS and LR which shows test accuracy higher than 0.5 and submit **ONE** `.pt` file accordingly.
(HINT: Think about how learning rate is likely to be when fine-tune the transformer model) **(3 pts)**
Answer: