Exam – CS 685

due Thursday Nov 11 at 9:30am EST on Gradescope

- This exam should take a couple hours to complete. However, we've given you 48 hours, so you can complete and submit it anytime during that window. Submission to Gradescope should be in PDF format.
- Feel free to use any resources (e.g., lecture slides, videos, conference papers, random websites) to help you answer the problems.
- We strongly prefer that you complete this exam using LaTeX (you can just copy this Overleaf project). If you'd rather type your answers using some different software, however, please feel free to do so. If you want to handwrite your answers and upload a scan of your exam, that's also fine. Just make sure we can easily read everything you wrote.
- Do **NOT** collaborate with other students on this exam. Do **NOT** even discuss exam problems with other students. Each answer should be your own work. We will check a random subset of answers for potential plagiarism using automatic NLP-based tools. Violations will be referred to the UMass Academic Honesty Board.
- If you have clarification questions for the instructors, please use **private Piazza posts** to ask them. We will delete any public posts about the exam. If the entire class would benefit from the answer to a privately-posted question (e.g., fixing a typo or unclear wording in a problem), the instructors will add them to a public Piazza post about exam-related issues.
- We will try to regularly reply to Piazza questions during the day (i.e., 9am-5pm EST Tuesday and Wednesday). If you post a question on Thursday at 3:30AM, do not expect to hear back from us :)

section	points
1. Prompt-based learning	/ 25
2. Long context language models	/ 30
3. Tokenization in language models	/ 20
4. Machine translation	/ 25
Total	/ 100

1 Prompt-based learning (25 pts)

Gilbert works for Zeta, a company that produces fashionable VR goggles. His boss tasks him with figuring out how much praise Zeta's latest luxury goggles are getting on the Internet. Gilbert collects a small dataset of 300 Tweets that contain mentions of these goggles. He tags each Tweet with whether it contains praise or not (i.e., a binary label). Since his dataset is so small, he decides to leverage one of Zeta's enormous pretrained left-to-right language models to solve his classification task via prompt-based learning.

Question 1.1. *[10 points]* Gilbert decides to start out with discrete prompts. First, he manually writes some prompts himself (e.g., *Does the following Tweet praise Zeta's amazing goggles?*), and then he applies a paraphrase generation model to obtain even more prompts. However, he eventually discovers that none of his prompts is able to match the validation performance of a continuous prompt (a sequence of 100 learned embeddings) obtained via prompt tuning.[1] If Gilbert enumerated every possible k-token-long discrete prompt where $1 < k < 100$, would he be guaranteed to find one that matches the validation performance of the learned continuous prompt? Why or why not?

Answer:

- **No**, Gilbert will **still not be guaranteed to match the validation performance** of the learned continuous prompts.
- Using a prompt tuning approach, we search in a continuous search space which is very exhaustive and hence serves as the upper bound in our case. Even when Gilbert enumerates all the option, he would might still miss out on most-optimal setting. The thing to note is that it is not required that most optimal setting will be achieved using words, it can be something intermediate captured by embeddings.
- To understand this, can take a loosely analogous example of hyperparameter tuning while finding most optimal setting in a neural network. Suppose we are trying to find best combination of learning rate and L1 dropout for the network. If we decide to search all the integer combinations of the two parameters, the possibility of finding the most optimal parameter with this discrete setting is rare and not guaranteed but with search in a uniform exhaustive search space and with a more optimized strategy like Bayes Opt (loosely analogous to back-propagation in network for prompt tuning), we have much greater chances of finding an optimal set of parameters.
- Also, in prompt tuning approach, we perform fine-tuning of the continuous initialized prompts which is a gradient based approach and is expected to more optimal.

[1] While the method in the prompt tuning paper (which we discussed in class) is implemented within the T5 encoder/decoder model, it can also be trivially applied to a decoder-only model like the one Gilbert is using.

Also, according to Liu et. al. and Shin et. al., we find using experiments that continuous soft prompts have greater representation ability and also that discrete search space is harder to optimize. Given we have 300 tweets to learn using fine-tuning, prompt tuning will always be a better choice.

- All these points lead us to the conclusion that Gilbert's approach is not guaranteed to match the validation performance.

Question 1.2. *[10 points]* Gilbert determines that enumerating every possible discrete prompt is computationally infeasible. He decides to instead develop a method that automatically maps discrete prompts to continuous prompts. To facilitate this process, he collects a large dataset of *(discrete prompt, learned continuous prompt)* pairs across a variety of tasks, where each continuous prompt is a sequence of 100 embeddings. Concretely, assume each discrete prompt contains tokens $d_1, d_2, \ldots, d_m$ where m varies across the prompts, while each continuous prompt contains tokens $c_1, c_2, \ldots, c_{100}$. Fully detail a model architecture, objective function, and training process that Gilbert can use to solve this problem, and make sure to also specify how he can use this model to produce new continuous prompts from discrete prompts at test-time.

Answer:

Model Architecture: We will have to use an encoder-decoder model to generate an embedded representation at the final layer. For this we can use T5 base from the Hugging face library which has 768 dimension for each token and has 12 layered architectures. At the decoder side of the T5 transformer, we can add a feed-forward layer which we would fine-tune during the training stage to learn the mappings from input in encoder part to the output in the decoder part. (Selection of single or multiple feed-forward layer in architecture depends upon the number of instances that we have to learn; here I assume that a single layer would be sufficient to learn the mappings and would not overfit much). Input embeddings would be generated from tokenizer of the T5 and concatenated with the positional embeddings.

Objective Function: We can use Mean Squared Error loss. (Alternatively since we are trying to create a similar distribution, we can also use KL divergence loss in place of Mean Squared Loss for training as the objective function to minimize in our case). A gradient based update would help the network learn most optimal weights.

Training Process: In order to generate the desired mapping, we just finetune the head layer (final feed-forward layer) using our examples. We keep the entire T5's layers and embedding (taken from T5's tokenizer) frozen in the process (doesn't make much sense to update the weights of these discrete prompt tokens as T5's embedding already have context ingrained in them from pre-training stage of T5). Also in the output reference we will have to add an [EOS] token after every 100 words. The idea is that with large amount of training data, model will eventually learn that it has to stop at the c100 token which is required in our question. The updates on weights during the training stage are carried out using back-propagation.

Test Time Process: At test time, let's say we encounter a new prompt: "summarize new text:". We will generate embeddings of these tokens using T5's tokenizer and do a forward pass in the model described above in model architecture section. At decoder, we may use nucleus sampling to generate token at each next time step. Since we have learnt the optimal mappings during the fine-tuning process, the continuous embedded

representation would be optimal.

Question 1.3. *[5 points]* Assume Gilbert successfully trains a model for the task in the previous subproblem and that for any held-out task, the generated continuous prompts exhibit no downstream performance degradation compared to those obtained via prompt tuning. What are two advantages of Gilbert's approach over prompt tuning?

Answer: The two advantages that I can think of when using this model are as follows:

- We would be able to **learn a direct mapping using this model between a discrete prompt and a optimal continuous representation**. This generates an optimal continuous prompt and would save us the trouble of back-propagating in the model every time a new task or prompt is encountered.
- We see above that it saves from the process of back-propagation for prompt-tuning process. One added advantage is that the learning is one time process and since we already have a mapping **we don't need to gather additional data for every new task** and thus saves us from data gathering exercise as well.
- In order to interpret continuous prompt, people have used different techniques like K-Means clustering to figure out what kind of topics these similar clusters represent just to understand these continuous prompts. With present mapping, we start with discrete prompts which are much **more interpretable prompt** in nature than continuous prompts.

  Since we have a direct mapping between continuous prompt and discrete prompt, we would be able to analyze better and develop intuition to try and interpret what kind of prompts perform better and in what cases.

2 Long context language models (30 pts)

Julia comes up with a clever approach to pack more context into Transformer language models without overly increasing the sequence length. Given an input sequence with tokens $x_1, x_2, \ldots, x_n$, she decides that instead of directly feeding the corresponding token embeddings $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ into the first Transformer block of her language model, she will perform a segment-level aggregation step to reduce the sequence length. Specifically, she decides to encode non-overlapping contiguous segments of 5 tokens (e.g., $s_1 = x_1x_2x_3x_4x_5$, $s_2 = x_6x_7x_8x_9x_{10}$, and so on) by individually feeding each segment s_i into a pretrained BERT model and extracting the final-layer [CLS] representation $\boldsymbol{s}_i$.

Question 2.1. *[10 points]* In her first attempt to build this model, she simply passes the resulting sequence of segment representations $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{\lceil n/5 \rceil}$ through a standard 16-layer Transformer. She intends to train her model for next word prediction and compute the cross entropy loss across all timesteps in parallel, just like in a normal token-level Transformer. However, as she sets about implementing the loss function, she discovers a fundamental flaw in her design. What's wrong with her setup?

Answer:

We are using the [CLS] token to get representation of s_i using BERT. Now we observe that at each time step, we pass sequence of segment representations $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{\lceil n/5 \rceil}$ through the 16 layer transformer. Based on clarification in a Piazza post by Mohit, since $\lceil n/5 \rceil$ is a ceil operation:

- This would mean that if we want to predict x1, we would be feeding ceil(1/5) = 1 i.e. s1 representation. This would be applicable till we predict x5. Now, the thing to note here is that since s1 is the embedded representation of words x1x2x3x4x5 from BERT, we are essentially leaking information from our input into the output, thus we won't be able to train weights properly.

- Though I have explained above considering the prediction of x1, the same issue persists through out the input sequence. Consider one more example that we want to predict x7, then based on information in question we would feed s1,s2 into the model as input. Now s2 already has information about x7 present in it too.

- So, our model has a direct source of information and would not be learning anything and would perform extremely poorly at the test time. To be explicit, the fact that she is passing information from future into the present model is the fundamental flaw in her design.

Question 2.2. *[10 points]* How would you modify Julia's approach to properly train a language model for next word prediction using her segment-level input representation idea? Do not add any new components to the model, and assume computational efficiency is not a concern.

Answer:

- We saw in the previous question that the issue is the leakage of information from the segment embeddings. For example, if we want to predict x8, we were passing on information from x6x7x8x9x10.
- Here x8x9x10 come from future which our model should not have seen. So we need to modify the segment representation such that it has only information till the previous word.
- This modification can be directly done while we feed our input information to the BERT model. So, when we want to generate prediction for word x8, we would mask out the input representations after x7 in s2 input while feeding it into the BERT model i.e. the input representation for getting s2 for x8 prediction (for feeding into 16 layer transformer model) would look like [x6,x7,MASK,MASK,MASK]. Now for prediction in decoder we feed s1 and modified s2 i.e. [x6,x7,MASK,MASK,MASK]
- Similarly, for x9 prediction we would unmask x8 while feeding into BERT model and input would look like s1,[x6,x7,x8,MASK,MASK] and so on.
- We can see here that for every word prediction, we would need to perform separate computation of segment representation and hence it might be computationally inefficient but since we are not concerned with efficiency, this should work.

Question 2.3. *[10 points]* After successfully implementing her model, Julia decides to experiment with a different task: given a sequence of segment representations $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_i$, retrieve the next *segment* s_{i+1} from the set of all possible five-word segments in the training set. She wants to formulate this as a retrieval problem, inspired by recent work in retrieval such as REALM and DPR. Describe how she could set up a new model and training objective for this task using both words and math. Feel free to introduce additional notation if you need to!

Answer:

- Based on lecture on REALM in class, there are essentially two ways in which we can possibly do this: (1) Replicating Mechanism followed by REALM (2) Nearest Neighbours Search Approach complimented with our model developed in question 2.2 above. I detail both of them one by one below:

REALM STYLE IMPLEMENTATION:

- Input would look like [s1 s2 s3 s4 s5 ... s_i [MASK]] whose representation from BERT would serve as x (query) similar to REALM paper
- Since, as a part of training we have to predict next segment s_{i+1}, given we have contiguous segments till s_i. We will use contiguous segments to create our knowledge corpus.
- Now, we compute the dot product between all the combinations in Z with selected input representation generated by BERT for x above. The retrieved document by the neural language retriever would be representation of z. In the next step, we concatenate (x,z) separated by [SEP] token. Then we use a knowledge-augmented encoder (BERT) to predict the values of the MASK segment representation using argmax in the final step that predicts the value of the [MASK] segment token.
- This entire architecture is trained end-to-end using back-propagation. Training objectives and mathematical equations can be found in the diagram attached below
- This approach of predicting the next segment would work with REALM give we have previous contexts and we need to predict [MASK] at only one time step. But if for example we had to generate next 5 segment tokens, we might not have enough information about previous tokens to predict [MASK] tokens in the end.

NEAREST NEIGHBOUR SEARCH APPROACH:

- In this there would be two components: (1) Nearest Neighbour Search for Retrieval using FAISS (2) Using the decoder and BERT model that we developed in previous sub-part of this question.

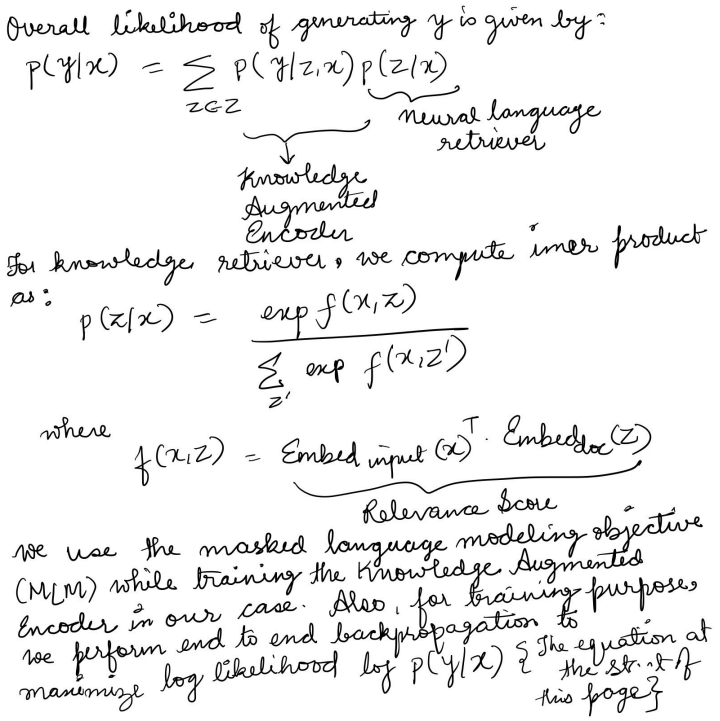


Figure 1: Mathematical Equations for REALM

- We take the representation generated using BERT of [s1 s2 s3 s4 s5 . . . si [MASK]]. We also have all the possibilities in Z (Textual Knowledge Corpus). With this we would find out the most similar sentence by taking a dot product to get top-k retrieved documents which are essentially nearest neighbours. This will give us the distribution of the next word (p_{knn}).

- Second component would come from our BERT and 16 layer transformer that we made in 2.2 question. First we pass the segments to generate next 5 words from the model. Once next 5 words have been generated, we feed it back into BERT layer to generate probability distribution for its segment representation s_{i+1} ($p_{transform}$).

- We then take an use a weighting parameter lambda to join (p_{knn}) and ($p_{transform}$) to derive the best final representation of s_{i+1} i.e equal to [lambda * (p_{knn}) + (1 - lambda)*($p_{transform}$)] where lambda is some percentage.

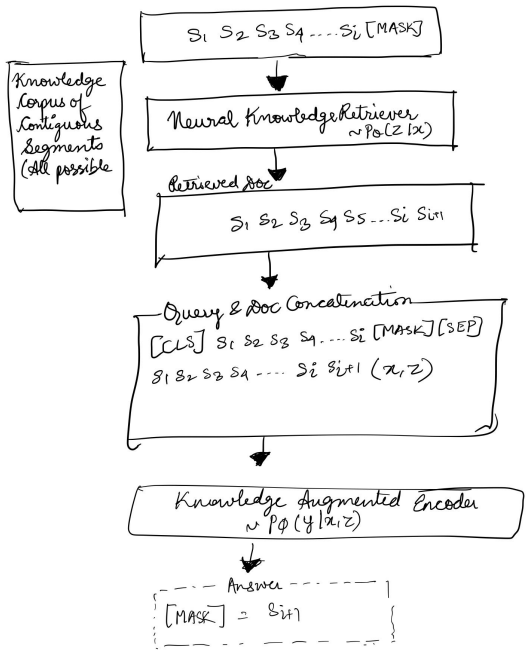


Figure 2: Diagram for REALM

- The advantage of this approach as highlighted in the lecture is that this does not require any additional training once our models are ready from question 2.2.

3 Tokenization in language models (20 pts)

Terry is growing tired of the proliferation of NLP papers about different types of tokenization. He starts wondering whether it is possible to create a single language model that can encode inputs using multiple tokenizations. In particular, he wants his model to be able to represent text as sequences of bytes, characters, subwords, and words.

Terry decides to experiment on the WikiText-103 language modeling benchmark dataset. Given an input sequence from the training data, he first chooses a random segmentation of the sequence that could mix multiple tokenization schemes. Below are two examples of different tokenizations (tokens are shown separated by commas) for the sequence *the evil warlock* that could arise from this process:

- word-level: *the*, , *evil*, , *warlock*[2]
- mixture of tokenizations: *t*, *h*, *e*, , *evil*, , *war*, *##lock*[3]

Then, he feeds these segmented training sequences to a language model whose vocabulary (both token embeddings and softmax layer) includes every byte, character, subword, and word that occurs in WikiText-103's training set. The LM is trained using the standard next-token-prediction objective and its parameters are updated through backpropagation.

Question 3.1. *[20 points]* After successfully training his model (let's call it LM-A), he wants to compare it to a baseline language model (LM-B) trained on WikiText-103 using word-level tokenization. He decides to compute the log likelihood of the sentence *Language models are stupid* according to each model. Describe how he would do this for LM-A and LM-B using words and/or math, and assume that there are no out-of-vocabulary issues associated with this sentence.

Answer:

We are given that there are no out of vocabulary words.This has a pretty important implication for LM-A. This implies that whatever word/character/sub-words or bytes that we encounter, we would definitely have an associated probability with it in the corpus.

For baseline language model (LM-B):

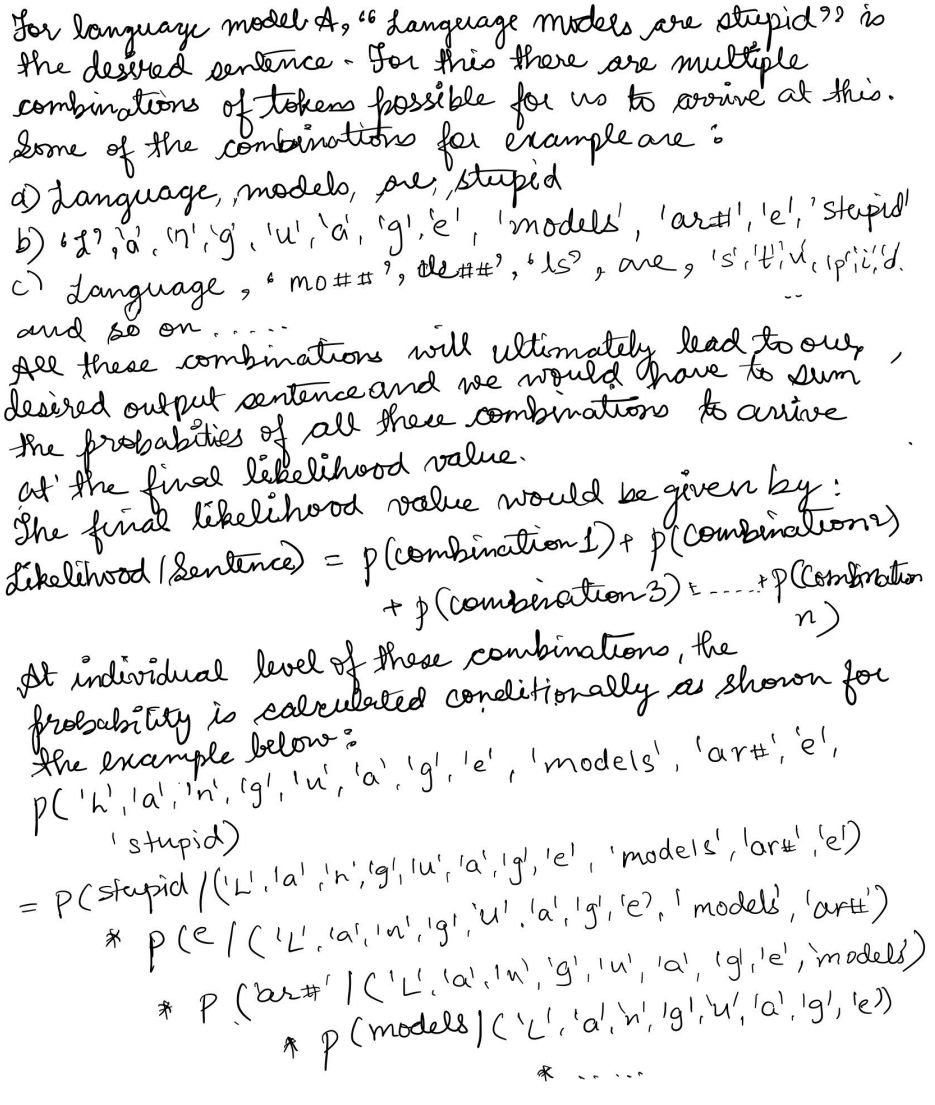
- For "Language models are stupid": Likelihood = P(Language)*P(models | Language)*P(are | Language models)*P(stupid | Language models are)
- For log-likelihood, we take log on both sides above

For language model that Terry developed (LM-A):

[2]For simplicity, we'll always treat spaces between words as individual tokens regardless of choice of segmentation.

[3]The ## in front of *lock* indicates that it is a subword.

- It is given that input can have multiple ways tokenizations i.e. either one of bytes, characters, subwords, and words. Since we are doing random segmentation of the sequence, there are multiple ways of the segmentation strategy.
- We take valid representations from each of the given tokens and all their possible set of combinations and use sum rule of probability to compute the final value of likelihood calculated from each of the tokenization schemes.



We can generalize the above example as :

$$\text{Likelihood of sequence} = \sum_{\substack{\text{probabilities} \\ \text{of all} \\ \text{combinations}}} \prod_{i=0}^{i} P\left(i \mid i-1\right)$$

Thus, log-likelihood will be obtained by taking log on both sides.

$$\text{Log likelihood} = \log\left(\sum_{\substack{\text{prob} \\ \text{of} \\ \text{all} \\ \text{combinations}}} \prod_{i=0}^{i} \left(P\left(i \mid i-1\right)\right)\right)$$

where $P\left(i \mid i-1\right)$ means the conditional probability of observing sequence upto 'i' given sequence till 'i-1' has been observed.

4 Machine translation (25 pts)

Your friend Sophia recommends you a novel written in Korean, claiming it's the best novel ever written. Unfortunately, you don't speak or read Korean, and you don't have money to hire a professional Korean-to-English translator. Thus, you purchase an ebook of the novel and shamefully resort to machine translation, copying chunks of the text and pasting it into Google Translate. However, you quickly discover that you cannot follow the story at all due to many jarring translation errors.

Since you trust Sophia's book recommendations, you decide not to give up so easily. You wonder if it's possible to build a model to perform automatic *post-editing* on top of Google Translate's output. In particular, given a Korean sentence and an English translation of this sentence produced by Google Translate, your model should produce a corrected English translation. To enable this, you collect a large parallel dataset of classic Korean novels paired with human-written English translations.

Question 4.1. *[15 points]* Describe an approach for this post-editing task that leverages the ByT5 model we discussed in class. Make sure to specify the format of the inputs and outputs to this model, as well as any pre-processing you would need to do on your dataset as well as the output of Google Translate. Feel free to introduce any notation (or figures) to make your answer more clear.

Answer:
Basic Architecture and Approach can be found below:

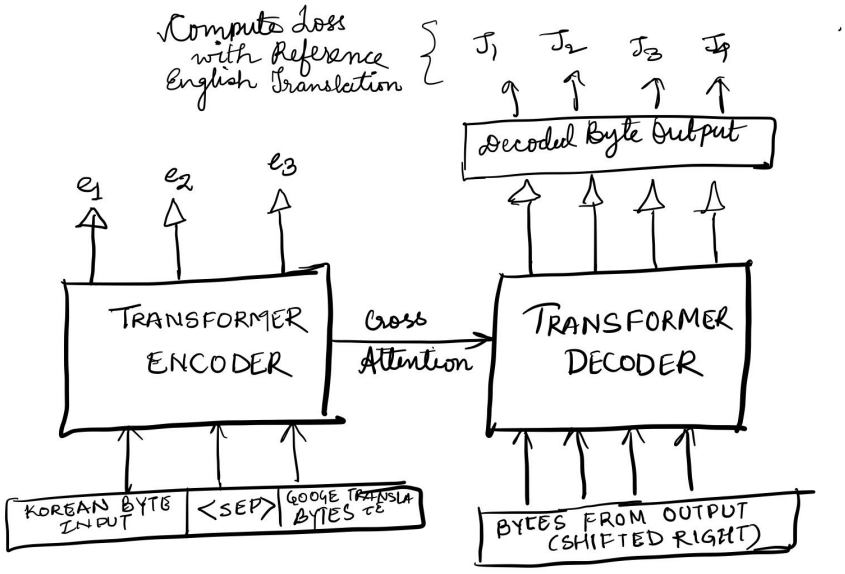
- We can use the large parallel dataset of classic Korean novels paired with human-written English translation to fine tune our model. In general, for the fine-tuning task, the reference English translation bytes would serve as ground truth to be used for back-propagation.
- In order to perform fine-tuning, we use a transformer based encoder-decoder architecture similar to one outlined in ByT5 paper.
- Note that we don't need two separate encoders for Encoding Korean and Reference Google Translate sentence and a single encoder would work as we are using Byte Level representation which has 256 unique values and is not affected by language.

Format of Input and Outputs:

- In the input side, we will concatenate the Korean text and the translation from Google translate system so that the decoder side of the model could accurately predict the output.
- Input to encoder would be of form: "Korean Language Bytes <sep> Corresponding Google Translate Bytes"
- Output Would be a sequence of bytes corresponding to the post corrected English translation which can be compared to the ground truth bytes

Pre-processing Requirement:

- As for the pre-processing, we will need to translate the Korean texts using Google translate. Then we will need to generate byte level representation of input to the encoder and byte level representation of the reference ground truth. Since the max sequence length is 1024 in ByT5 model, we will use smaller and medium sentence lengths so that the max sequence length is within this value.



Question 4.2. *[10 points]* Describe two ways that you could evaluate whether your post-editing model is producing superior translations than Google Translate.

Answer: Sharma et. al. describe the two major approaches to evaluate post-editing tasks. These two major ways to evaluate these kinds of post editing models are:

- TER (Translational Error Rate): It is a method used by Machine Translation specialists to determine the amount of post editing required for Machine Translation jobs and is defined as number of edits a translated segment requires with respect to one of the reference translations. A lower value of TER is preferred.
- BLEU (Bilingual Evaluation Under Study): One of the most import metric to evaluate any machine translation system is BLEU score and it is a number between zero to one. The idea is "The closer a machine translation is to a professional human reference translation, the better it is". BLEU scores have till date correlated well with human judgement and hence using them would be a better idea (Definition and formula over here: https://en.wikipedia.org/wiki/BLEU)
- One another approach could be to use the help of Mechanical Turk (Human Evaluation) and use maybe a Likert Scale on an average (1 to 5) for each of the sentences in test set with Google Translate and our model prediction and take an average to see which is performing better.Other metrics like ROUGE or COMET can be used too.

One thing to note here that individual scores would have no meaning in our case. At test time, first we need to compute BLEU score using the existing machine translation system (Google Translate) and then evaluate these metrics over our final predictions from model. If we see an increase in BLEU score, we can say that our system is performing well. Also, a lower TER is preferred.