CS 224n: Assignment #4

This assignment is split into two sections: *Neural Machine Translation with RNNs* and *Analyzing NMT Systems*. The first is primarily coding and implementation focused, whereas the second entirely consists of written, analysis questions. If you get stuck on the first section, you can always work on the second as the two sections are independent of each other. Note that the NMT system is more complicated than the neural networks we have previously constructed within this class and takes about **4 hours to train on a GPU**. Thus, we strongly recommend you get started early with this assignment. Finally, the notation and implementation of the NMT system is a bit tricky, so if you ever get stuck along the way, please come to Office Hours so that the TAs can support you.

1. Neural Machine Translation with RNNs (45 points)

In Machine Translation, our goal is to convert a sentence from the *source* language (e.g. Cherokee) to the *target* language (e.g. English). In this assignment, we will implement a sequence-to-sequence (Seq2Seq) network with attention, to build a Neural Machine Translation (NMT) system. In this section, we describe the **training procedure** for the proposed NMT system, which uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder.

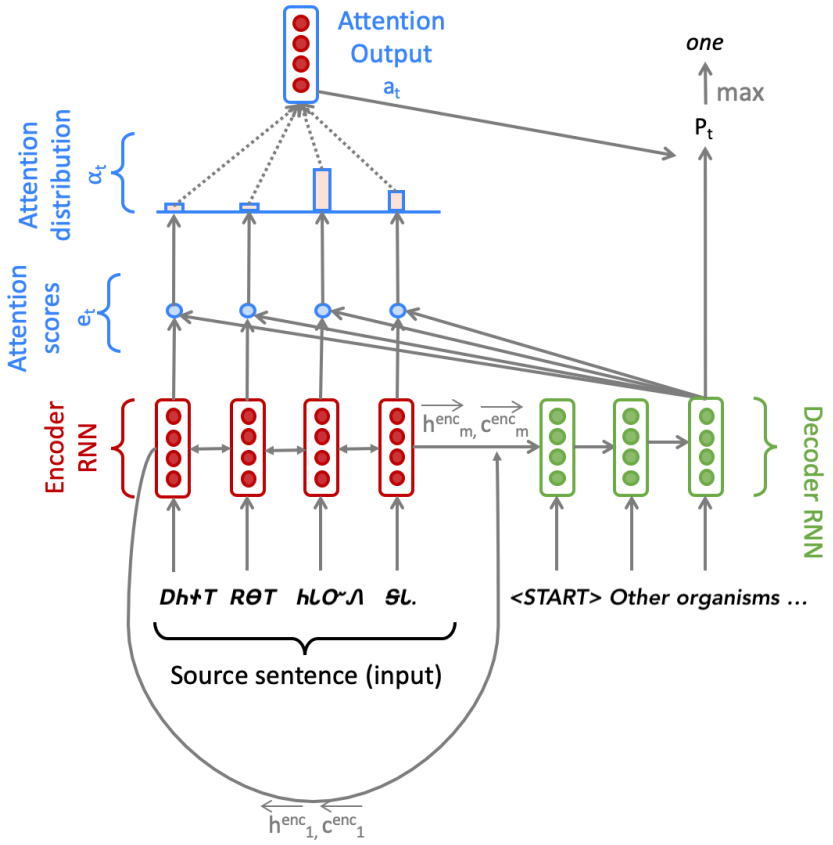


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined in the next page.

Model description (training procedure)

Given a sentence in the source language, 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 $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$:

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

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

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.[1]

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

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

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

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

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

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

$\mathbf{e}_{t,i}$ 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 $\mathbf{o}_t$.

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

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

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

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

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 Cherokee to English translation!

Setting up your Virtual Machine

Follow the instructions in the CS224n Azure Guide (link also provided on website and Ed) in order to create your VM instance. This should take you approximately 45 minutes. Though you will need the GPU to train your model, we strongly advise that you first develop the code locally and ensure that it runs, before attempting to train it on your VM. GPU time is expensive and limited. It takes approximately **30 minutes to 1 hour** to train the NMT system. We don't want you to accidentally use all your GPU time for debugging your model rather than training and evaluating it. Finally, **make sure that your VM is turned off whenever you are not using it.**

If your Azure subscription runs out of money, your VM will be temporarily locked and inaccessible. If that happens, please fill out a request form here.

In order to run the model code on your **local** machine, please run the following command to create the proper virtual environment:

```
conda env create --file local_env.yml
```

Note that this virtual environment **will not** be needed on the VM.

Implementation and written questions

(a) (2 points) (coding) In order to apply tensor operations, we must ensure that the sentences in a given batch are of the same length. Thus, we must identify the longest sentence in a batch and pad others to be the same length. Implement the pad_sents function in utils.py, which shall produce these padded sentences.

(b) (3 points) (coding) Implement the ___init___ function in model_embeddings.py to initialize the necessary source and target embeddings.

(c) (4 points) (coding) Implement the ___init___ function in nmt_model.py to initialize the necessary model embeddings (using the ModelEmbeddings class from model_embeddings.py) and layers (LSTM, projection, and dropout) for the NMT system.

(d) (8 points) (coding) Implement the encode function in nmt_model.py. This function converts the padded source sentences into the tensor $\mathbf{X}$, generates $\mathbf{h}_1^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$, and computes the initial state $\mathbf{h}_0^{\text{dec}}$ and initial cell $\mathbf{c}_0^{\text{dec}}$ for the Decoder. You can run a non-comprehensive sanity check by executing:

```
python sanity_check.py 1d
```

(e) (8 points) (coding) Implement the decode function in nmt_model.py. This function constructs $\bar{\mathbf{y}}$ and runs the step function over every timestep for the input. You can run a non-comprehensive sanity check by executing:

```
python sanity_check.py 1e
```

(f) (10 points) (coding) 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
```

(g) (3 points) (written) The generate_sent_masks() function in nmt_model.py produces a tensor called enc_masks. It has shape (batch size, max source sentence length) and contains 1s in positions corresponding to 'pad' tokens in the input, and 0s for non-pad tokens. Look at how the masks are used during the attention computation in the step() function (lines 295-296).

First explain (in around three sentences) what effect the masks have on the entire attention computation. Then explain (in one or two sentences) why it is necessary to use the masks in this way.

Solution: In the step() function, we set the attention scores corresponding to 'pad' tokens to $-inf$. When we convert the scores to probabilities using a Softmax() operation, these values get assigned a value of *zero* because of the $exp(-inf)$ term. This basically tells the network to not put any attention on 'pad' tokens at all. If we did not have these masking, then the 'pad' tokens and thereby the sentence length will likely start influencing the predictions.

Now it's time to get things running! Execute the following to generate the necessary vocab file:

```
sh run.sh vocab
```

Or if you are on Windows, use the following command instead. Make sure you execute this in an environment that has python in path. For example, you can run this in the terminal of your IDE or your Anaconda prompt.

```
run.bat vocab
```

As noted earlier, we recommend that you develop the code on your personal computer. Confirm that you are running in the proper conda environment and then execute the following command to train the model on your local machine:

```
sh run.sh train_local
(Windows) run.bat train_local
```

To help with monitoring and debugging, the starter code uses tensorboard to log loss and perplexity during training using TensorBoard[2]. TensorBoard provides tools for logging and visualizing training information from experiments. To open TensorBoard, run the following in your conda environment:

```
tensorboard --logdir=runs
```

You should see a significant decrease in loss during the initial iterations. Once you have ensured that your code does not crash (i.e. let it run till iter 10 or iter 20), power on your VM from the Azure Web

[2]https://pytorch.org/docs/stable/tensorboard.html

Portal. Then read the *Managing Code Deployment to a VM* section of our Practical Guide to VMs (link also given on website and Ed) for instructions on how to upload your code to the VM.

Next, install necessary packages to your VM by running:

```
pip install -r gpu_requirements.txt
```

Finally, turn to the *Managing Processes on a VM* section of the Practical Guide and follow the instructions to create a new tmux session. Concretely, run the following command to create tmux session called nmt.

```
tmux new -s nmt
```

Once your VM is configured and you are in a tmux session, execute:

```
sh run.sh train
(Windows) run.bat train
```

Once you know your code is running properly, you can detach from session and close your ssh connection to the server. To detach from the session, run:

```
tmux detach
```

You can return to your training model by ssh-ing back into the server and attaching to the tmux session by running:

```
tmux a -t nmt
```

(h) (3 points) (written) Once your model is done training (**this should take under 1 hour on the VM**), execute the following command to test the model:

```
sh run.sh test
(Windows) run.bat test
```

Please report the model's corpus BLEU Score. It should be larger than 10.

Solution: Corpus BLEU: 13.214307568571083

(i) (4 points) (written) In class, we learned about dot product attention, multiplicative attention, and additive attention. As a reminder, dot product attention is $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{h}_i$, multiplicative attention is $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{W} \mathbf{h}_i$, and additive attention is $\mathbf{e}_{t,i} = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t)$.

i. (2 points) Explain one advantage and one disadvantage of *dot product attention* compared to multiplicative attention.

Solution: Advantage: dot product attention requires no additional parameters, and hence is more efficient to compute and requires less memory. Disadvantage: dot product attention is not as expressive as multiplicative attention, the latter can combine information in interesting ways. If the elements are orthogonal to each other, dot product attention will assign it a value of zero.

ii. (2 points) Explain one advantage and one disadvantage of *additive attention* compared to multiplicative attention.

Solution: Advantage: additive attention is more flexible than the multiplicative attention since each vector gets transformed individually and then combined through a non-linear transformation; this can then consequently provide a better representation. Disadvantage: additive attention requires higher number of parameters and is not as efficient to compute as the multiplicative attention.

2. Analyzing NMT Systems (33 points)

(a) (3 points) In part 1, 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 Cherokee-to-English NMT problem at the subword-level vs. the whole word-level? (Hint: Cherokee is a polysynthetic language.)

Answer: Since Cherokee is a polysynthetic language, words consist of several morphemes. This means that words can be constructed using several sub-words as building blocks. Therefore, storing sub-word embeddings would require very few entries in the embedding matrix as opposed to storing an embedding vector for every possible word in Cherokee vocabulary.

(b) (3 points) Transliteration is the representation of letters or words in the characters of another alphabet or script based on phonetic similarity. For example, the transliteration of ᏣᏅᏔᏍᎪ (which translates to "do you know") from Cherokee letters to Latin script is tsanvtasgo. In the Cherokee language, "ts-" is a common prefix in many words, but the Cherokee character Ꮳ is "tsa". Using this example, explain why when modeling our Cherokee-to-English NMT problem at the subword-level, training on transliterated Cherokee text may improve performance over training on original Cherokee characters.(Hint: A prefix is a morpheme.)

Answer: In the context of Transliteration, if we represent the morpheme "ts-" as a sub-word, we can essentially construct multiple words using the same prefix. On the other hand, multiple Cherokee characters with the same phonetic prefix will have to be represented using different embeddings. Hence, when modeling the Cherokee-to-English NMT problem at the subword-level, training on transliterated Cherokee text may improve the performance over training over original Cherokee characters.

(c) (3 points) One challenge of training successful NMT models is lack of language data, particularly for resource-scarce languages like Cherokee. One way of addressing this challenge is with multilingual training, where we train our NMT on multiple languages (including Cherokee). You can read more about multilingual training here:
https://ai.googleblog.com/2019/10/exploring-massively-multilingual.html.
How does multilingual training help in improving NMT performance with low-resource languages?

Answer: Multilingual training helps to capture representational similarity across various languages. Through such training, the models start to learn shared representations and is more generalizable. Such a model can then be very quickly and easily transfered towards low-resource languages.

(d) (6 points) Here we present a series of errors we found in the outputs of our NMT model (which is the same as the one you just trained). For each example of a reference (i.e., 'gold') English translation, and NMT (i.e., 'model') English translation, please:

1. Identify the error in the NMT translation.
2. Provide possible reason(s) why the model may have made the error (either due to a specific linguistic construct or a specific model limitation).
3. Describe one possible way we might alter the NMT system to fix the observed error. There are more than one possible fixes for an error. For example, it could be tweaking the size of the hidden layers or changing the attention mechanism.

Below are the translations that you should analyze as described above. Only analyze the underlined error in each sentence. Rest assured that you don't need to know Cherokee to answer these questions. You just need to know English! If, however, you would like additional color on the source sentences, feel free to use a resource like https://www.cherokeedictionary.net/ to look up words.

i. (2 points) **Source Sentence:** *ᏄᏩᏁᎰᎾ ᏕᎪᏣᎳᎩᏍᎬ, ᎯᎠ ᏄᏍᏕ ᏚᏏᎳᏛ: ᏧᏓᏧᏅᏓ ᏕᎪᏒᏍᏕ ᏧᏏᎳᏛᏙᏗ ᎠᏍᏓ ᎾᏅᏂᏍᎩ.*

Reference Translation: *When <u>she</u> was finished ripping things out, <u>her</u> web looked something*

like this:
NMT Translation: *When it was gone out of the web, he said the web in the web.*

error: The NMT translation does not understand the context at all. Furthermore, the translated gender pronoun is incorrect.
reason: The translated gender pronoun could be incorrect if the language is gender-neutral. Context misunderstanding could be caused if the complete context were not summarized through hidden states.
solution: To overcome the context misunderstanding issue, one could increase the hidden states size of RNN to summarize the context better.

ii. (2 points) **Source Translation**: *O'ϑJ TPET, SVϑT? O'LP'P'ΛT O'ϑJ DdG.*
Reference Translation: *What's wrong little tree? the boy asked.*
NMT Translation: *The little little little little little tree? asked him.*

error: The NMT translation model repeats the same word multiple times while generating the target sentence.
reason: One reason for this behaviour could be that the next generated word might not be conditioned on previous word.
solution: This could be remedied by feeding the previous word embedding as a part of the input to the next cell for predicting next word.

iii. (2 points) **Source Sentence:** *"O'L3Ψ'J hFRΘ," O'P'Λ FH.*
Reference Translation: *" 'Humble,' " said Mr. Zuckerman*
NMT Translation: *"It's not a lot," said Mr. Zuckerman.*

error: The NMT translation model does not understand the accurate meaning of the word 'humble' in the given context, and incorrectly generates words that means something different.
reason: The reason for this could be that the network has not seen the word 'humble' in many different contexts.
solution: One solution for this would be to train this model on a large amount of dataset where the network can observe and learn from various contexts.

(e) (4 points) Now it is time to explore the outputs of the model that you have trained! The test-set translations your model produced in question 1-i should be located in outputs/test_outputs.txt.

i. (2 points) Find a line where the predicted translation is correct for a long (4 or 5 word) sequence of words. Check the training target file (English); does the training file contain that string (almost) verbatim? If so or if not, what does this say about what the MT system learned to do?
Answer: Correctly predicted translation: "Yes, it is," said Wilbur.
Similar string in training target file: "Oh, yes indeed," said Wilbur.

Looking at the similarities in target translations training data and predicted translation, it seems like the model has learnt to produce similar distribution of words, and produce the words that makes sense within a given context, to some extent the model also memorizes phrases in the training set.

ii. (2 points) Find a line where the predicted translation starts off correct for a long (4 or 5 word) sequence of words, but then diverges (where the latter part of the sentence seems totally unrelated). What does this say about the model's decoding behavior?

Answer: Predicted sentence: And when they saw him, they were astonished; and his mother said unto him, Son, why hast thou thus dealt with us? behold, thy father and I sought thee sorrowing.

The predicted sentence starts off well but then diverges completely. The later part of the sentence is not related at all. This means that the network, during decoding phase is not able to keep track of and *attend* to far enough words, and lacks the ability to comprehend complete context.

(f) (14 points) 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.[3] 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. (5 points) Please consider this example[4]:
Source Sentence $\mathbf{s}$: **Dᏼ ᎾᏍᎩ ᎢᎦ-ᎦᏓᏍᏓᏩᏍᎩ ᎤᏲᏏᎬ ᏚᏆᏐᎢ ᎤᏲᏏᎽᏃ ᎥᏝ ᏯᏚᏗᏂᏙᎢ**
Reference Translation $\mathbf{r}_1$: *the light shines in the darkness and the darkness has not overcome it*
Reference Translation $\mathbf{r}_2$: *and the light shines in the darkness and the darkness did not comprehend it*
NMT Translation $\mathbf{c}_1$: and the light shines in the darkness and the darkness can not comprehend
NMT Translation $\mathbf{c}_2$: the light shines the darkness has not in the darkness and the trials

[3] 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

[4] Due to data availability, many Cherokee sentences with English reference translations are from the Bible. This example is John 1:5. The two reference translations are from the New International Version and the New King James Version translations of the Bible.

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.

Which of the two NMT translations is considered the better translation according to the BLEU Score? Do you agree that it is the better translation?
Solution:

2) f) i)

r_1 = the light shines in the darkness and the darkness has not overcome it

$len(r_1) = 13$

r_2 = and the light shines in the darkness and the darkness did not comprehend it

$len(r_2) = 14$

c_1 = and the light shines in the darkness and the darkness can not comprehend it

$len(c_1) = 14$

c_2 = the light shines the darkness has not in the darkness and the trials

$len(c_2) = 13$

for c_1: $len(r) = len(r_2) = 14 \rightarrow$ closest to $len(c_1)$

for c_2 $len(r) = len(r_1) = 13 \rightarrow$ closest to $len(c_2)$

1-gram	$Count_{r_1}$	$Count_{r_2}$	$Count_{c_1}$	$Count_{c_2}$
the	3	3	3	4
light	1	1	1	1
shines	1	1	1	1
in	1	1	1	1
darkness	2	2	2	2
and	1	2	2	1
has	1	0	0	1
not	1	1	1	1
overcome	1	0	0	0
it	1	1	1	0
did	0	1	0	0
Comprehend	0	1	1	0
Trials	0	0	0	1
Can	0	0	1	0

p_1 for C_1:

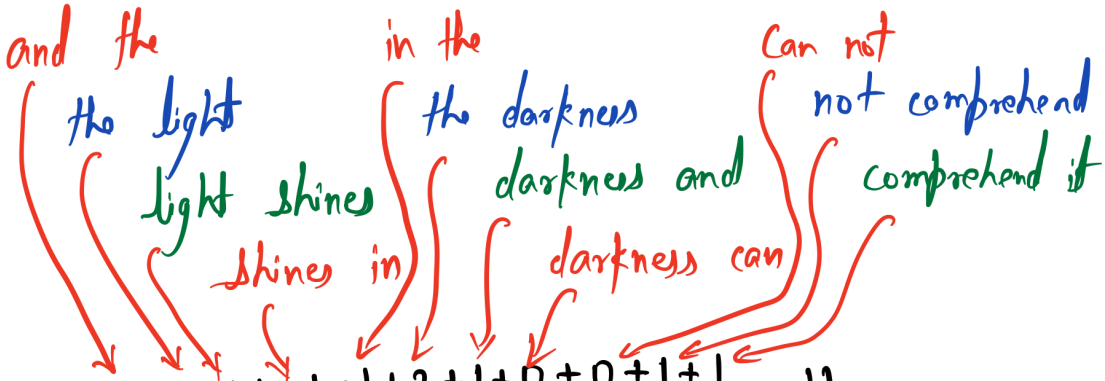
$$p_1 = \frac{\sum_{ngram \in C_1} \min\left(\max_{i=1,2} Count_{r_i}(ngram), Count_{c_1}(ngram)\right)}{\sum_{ngram \in C_1} Count_{c_1}(ngram)}$$

C_1 = and the light shines in darkness can not comprehend it

$$p_1 = \frac{2+3+1+1+1+2+0+1+1+1}{2+3+1+1+1+2+1+1+1+1} = \frac{13}{14} = 0.9286$$

2-gram	$Count_{r_1}$	$Count_{r_2}$	$Count_{c_1}$	$Count_{c_2}$
the light	1	1	1	1
light shines	1	1	1	1
shines in	1	1	1	0
in the	1	1	1	1
the darkness	2	2	2	2
darkness and	1	1	1	1
and the	1	2	2	1
darkness has	1	0	0	1
has not	1	0	0	1
not overcome	1	0	0	0
overcome it	1	0	0	0
darkness did	0	1	0	0
did not	0	1	0	0
not comprehend	0	1	1	0
comprehend it	0	1	1	0
darkness can	0	0	1	0
can not	0	0	1	0
shines the	0	0	0	1
not in	0	0	0	1
the trials	0	0	0	1

p_2 for C_1 :



$$p_2 = \frac{2+1+1+1+1+2+1+0+0+1+1}{2+1+1+1+1+2+1+1+1+1+1} = \frac{11}{13} = 0.8461$$

$$len(C_1) = 14 \; ; \; len(r) = 14 \Rightarrow BP = 1$$

$$\lambda_1 = 0.5 \; ; \; \lambda_2 = 0.5 \; ; \; \lambda_3 = 0 \; ; \; \lambda_4 = 0$$

$$BLEU = BP \times \exp\left(\sum_{n=1}^{n} \lambda_i \log p_n\right)$$

$$= \exp\left(0.5 \log(0.9286) + 0.5 \log(0.8461)\right)$$

$$= \exp(-0.037 - 0.08355)$$

$$= \exp(-0.12055)$$

$$\Rightarrow BLEU = 0.8864$$

p_1 for c_2:

$$p_1 = \frac{\sum_{ngram \in c_2} \min\left(\max_{i=1,2} Count_{r_i}(ngram), Count_{c_2}(ngram)\right)}{\sum_{ngram \in c_2} Count_{c_2}(ngram)}$$

the light shines darkness has not in and trials

$$p_1 = \frac{3+1+1+2+1+1+1+1+0}{4+1+1+2+1+1+1+1+1} = \frac{11}{13} = 0.8461$$

p_2 for c_2:

$$p_2 = \frac{\sum_{ngram \in c_2} \min\left(\max_{i=1,2} Count_{r_i}(ngram), Count_{c_2}(ngram)\right)}{\sum_{ngram \in c_2} Count_{c_2}(ngram)}$$

the light
light shines
shines the
the darkness
darkness has
has not
not in
in the
darkness and
and the
the trials

$$p_2 = \frac{1+1+0+2+1+1+0+1+1+1+0}{1+1+1+2+1+1+1+1+1+1+1} = \frac{9}{12} = 0.75$$

$len(c_2) = 13$; $len(r) = 13 \Rightarrow BP = 1$

$\lambda_1 = 0.5$; $\lambda_2 = 0.5$; $\lambda_3 = 0$; $\lambda_4 = 0$

$$BLEU = BP \times \exp\left(\sum_{n=1}^{n} \lambda_i \log p_n\right)$$

$$= \exp\left(0.5 \log(0.8461) + 0.5 \log(0.75)\right)$$

$$= \exp\left(-0.08355 - 0.1438\right)$$

$$= \exp(-0.22735)$$

$$\Rightarrow BLEU = 0.79664$$

BLUE score for $\mathbf{c}_1$: 0.8864
BLUE score for $\mathbf{c}_2$: 0.79664

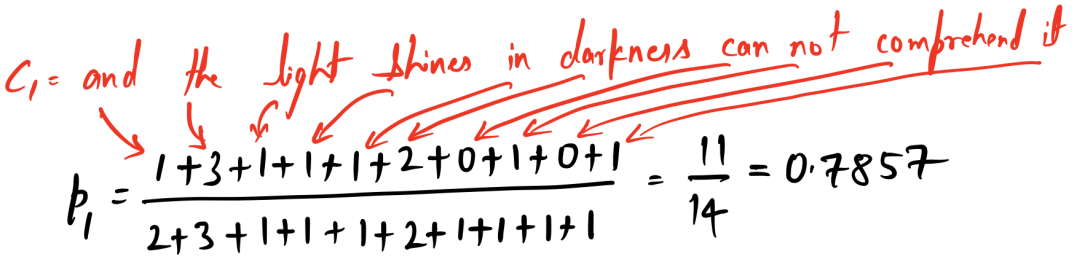
According to the BLUE scores, NMT translation for $\mathbf{c}_1$ is better than $\mathbf{c}_2$, and I agree with the evaluation.

ii. (5 points) Our hard drive was corrupted and we lost Reference Translation $\mathbf{r}_2$. Please recompute BLEU scores for $\mathbf{c}_1$ and $\mathbf{c}_2$, this time with respect to $\mathbf{r}_1$ only. Which of the two NMT translations now receives the higher BLEU score? Do you agree that it is the better translation?
Solution:

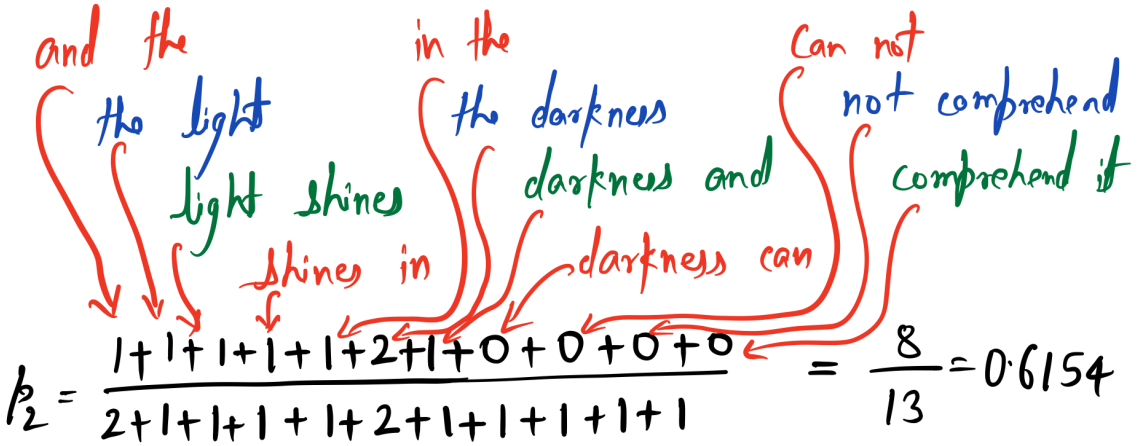
2) f) ii) We will use the same table derived for the previous problem.

p_1 for c_1:

$$p_1 = \frac{\sum_{ngram \in c_1} \min\left(Count_{r_1}(ngram), Count_{c_1}(ngram)\right)}{\sum_{ngram \in c_1} Count_{c_1}(ngram)}$$



p_2 for c_1:



$len(c_1) = 14$; $len(r_1) = 13 \Rightarrow BP = 1$

$\lambda_1 = 0.5$; $\lambda_2 = 0.5$; $\lambda_3 = 0$; $\lambda_4 = 0$

$$BLEU = BP \times \exp\left(\sum_{n=1}^{n} \lambda_i \log p_n\right)$$

$$= \exp\left(0.5 \log(0.7857) + 0.5 \log(0.6154)\right)$$

$$= \exp(-0.15779)$$

$$\Rightarrow BLEU = 0.85402$$

P_1 for C_2:

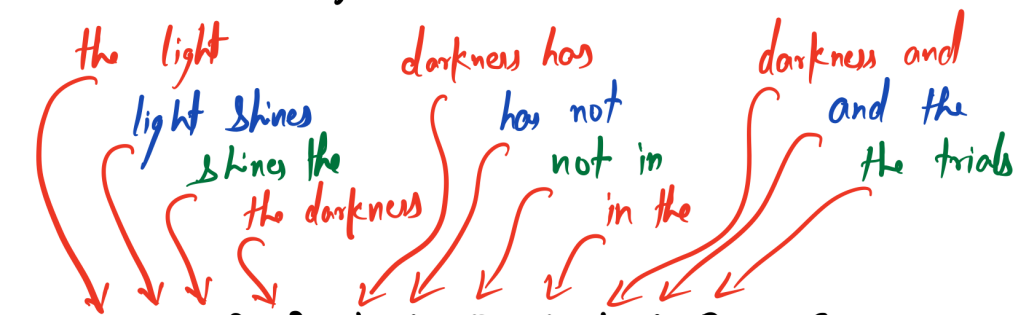
$$p_1 = \frac{\sum_{ngram \in C_2} \min\left(Count_{r_2}(ngram), Count_{c_2}(ngram)\right)}{\sum_{ngram \in C_2} Count_{c_2}(ngram)}$$

the light shines darkness has not in and trials

$$p_1 = \frac{3+1+1+2+1+1+1+1+0}{4+1+1+2+1+1+1+1+1} = \frac{11}{13} = 0.84615$$

p_2 for c_2:

$$p_2 = \frac{\sum_{ngram \in c_2} \min\left(Count_{r_1}(ngram), Count_{c_2}(ngram)\right)}{\sum_{ngram \in c_2} Count_{c_2}(ngram)}$$



$$p_2 = \frac{1+1+0+2+1+1+0+1+1+1+0}{1+1+1+2+1+1+1+1+1+1+1} = \frac{9}{12} = 0.75$$

$len(c_2) = 13$; $len(r_1) = 13 \Rightarrow BP = 1$

$\lambda_1 = 0.5$; $\lambda_2 = 0.5$; $\lambda_3 = 0$; $\lambda_4 = 0$

$$BLEU = BP \times \exp\left(\sum_{n=1}^{n} \lambda_i \log p_n\right)$$

$$= \exp\left(0.5 \log(0.84615) + 0.5 \log(0.75)\right)$$

$$= \exp(-0.098745)$$

$$\Rightarrow BLEU = 0.90597$$

BLUE score for $\mathbf{c}_1$: 0.85402
BLUE score for $\mathbf{c}_2$: 0.90597

The second NMT translation, $\mathbf{c}_2$, now receives the higher BLEU score. I do not agree at all

with the evaluation here, first translation is definitely better than the second.

iii. (2 points) 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.
Answer: The reference translation in available dataset is subjective to the person translating it. With n-gram based evaluations such as the BLUE score metric, we want the predicted translation to have roughly the same word organization and n-gram, this might be problematic. In contrast, with multiple reference translations available, we take the maximum of each n-gram scores across reference translations which simply reflects various ways to express the same sentiment and thus provides us with a better metric of NMT translation quality.

iv. (2 points) List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.
Advantages:
- It is easy to compute; a computer can easily be programmed to compute the BLUE score in an automated way.
- It is not subjective to human opinions.

Disadvantages:.
- The BLUE metrics computation does not take into account the meaning or semantics of the sentence.
- The position of words and n-gram has no effect on the BLUE score. This effect is even worse for 1-gram.

Submission Instructions

You shall submit this assignment on GradeScope as two submissions – one for "Assignment 4 [coding]" and another for 'Assignment 4 [written]":

1. Run the collect_submission.sh script on Azure to produce your assignment4.zip file. You can use scp to transfer files between Azure and your local computer.
2. Upload your assignment4.zip file to GradeScope to "Assignment 4 [coding]".
3. Upload your written solutions to GradeScope to "Assignment 4 [written]". When you submit your assignment, make sure to tag all the pages for each problem according to Gradescope's submission directions. Points will be deducted if the submission is not correctly tagged.