CS 224n: Assignment #4

Sheng Li

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. 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. We also highly recommend reading Zhang et al (2020) to better understand the Cherokee-to-English translation task, which served as inspiration for this assignment.

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.

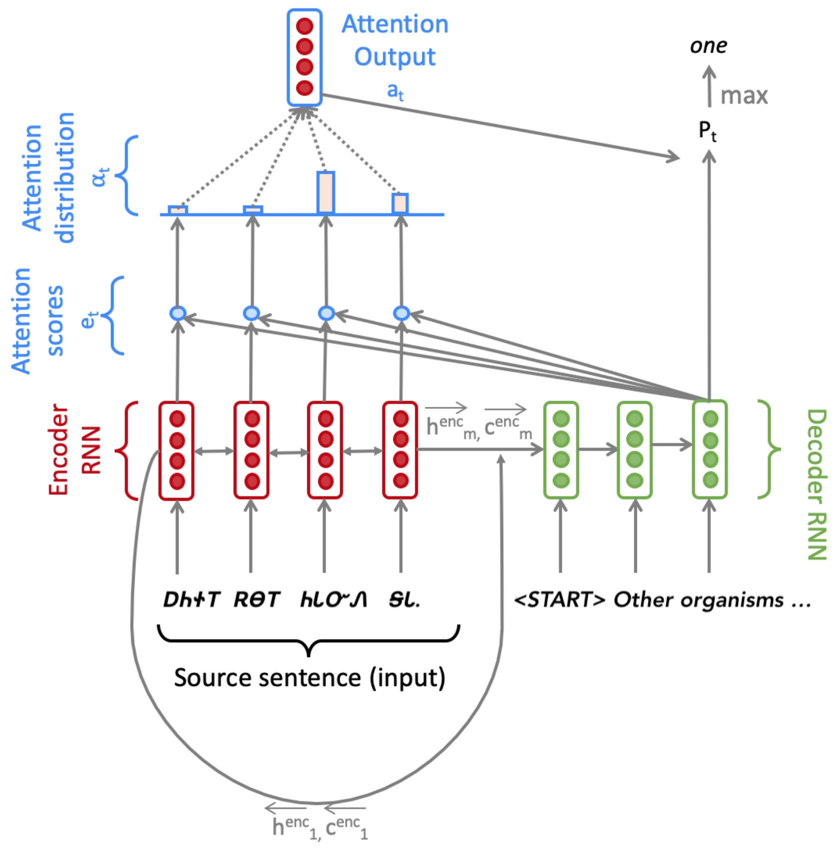

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined in the next page.

Model description (training procedure)

Given a sentence in the source language, we look up the subword embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \dots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e\times 1}$), where m is the length of the source sentence and e is the embedding size. We feed these embeddings to the bidirectional encoder, yielding hidden states and cell states for both the forwards ($\rightarrow$) and backwards ($\leftarrow$) LSTMs. The forwards and backwards versions are concatenated to give hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$:

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

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

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

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

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

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

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:

[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.

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

Here, V_t is the size of the target vocabulary. Finally, to train the network we then compute the softmax 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:

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

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 Piazza) 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, make a private post on Piazza with your Name, email used for Azure and SUNetID to request more credits.

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.

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: The `step()` function sets attention scores `e_t` to `-inf` where `enc_masks` has 1, i.e. where `e_t` is computed using 'pad' embedding. By doing so, when we compute the corresponding attention weights α_t using softmax, these scores come from 'pad' embeddings will become $\exp(-\text{inf}) = 0$.

This is necessary since there are 'pad' entries in the sequence that we do not want to use in computing the attention weights. We only want the real word entries. Therefore, by applying the `enc_masks`, the contribution of the 'pad' entries will be effectively filtered away in computing the attention weights α_t, since they will all become 0. The non-zero attention weights only will be computed for real tokens in the sequence.

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
```

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 Portal. Then read the *Managing Code Deployment to a VM* section of our Practical Guide to VMs (link also given on website and Piazza) 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) 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: BLEU Score $\approx$ 12.251

(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: One advantage of dot product attention over multiplicative attention is that it is faster to compute and requires less parameters. One disadvantage is that it is less expressive, i.e. the representational power is lower due to the lack of parameters.

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

Solution: One advantage of additive attention over multiplicative attention is that it can freely control the dimension of the attention scores, making it more versatile in certain applications. One disadvantage is that it can require more parameters and be less efficient to compute than multiplicative attention.

2. Analyzing NMT Systems (30 points)

(a) (2 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.)

Solution: Since Cherokee is a polysynthetic language, its words are composed of many morphemes. The lengths of Cherokee words can vary a lot, and its vocabulary can be extremely large due to unlimited combinations and permutations of morphemes. Therefore, model the Cherokee-to-English NMT problem at the morpheme-level or subword-level makes more sense.

(b) (2 points) Character-level and subword embeddings are often smaller than whole word embeddings. In 1-2 sentences, explain one reason why this might be the case.

Solution: The amount of characters or subwords are much smaller than that of all the words.

(c) (2 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. How does multilingual training help in improving NMT performance with low-resource languages?

Solution: Multilingual training can provide the model with much more data a single data-scarce language can give. Multilingual models are effective at generalization, and capable of capturing the representational similarity across a large body of languages. Languages can transfer knowledge from each other.

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

1. Provide possible reason(s) why the model may have made the error (either due to a specific linguistic construct or a specific model limitation).
2. 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 resources like `https://www.cherokeedictionary.net/` to look up words.

i. (2 points) **Source Translation**: *Yona utsesdo ustiyegv anitsilvsgi digvtanv uwoduisdei.*
Reference Translation: *Fern had a crown of daisies in her hair.*
NMT Translation: *Fern had <u>her hair</u> with her hair.*

Solution: The NMT system repeated the translation of a latter part of the sentence in the middle of the translation. This error might be caused by putting overly high attention to the latter part of the sentence. A possible fix is adjusting attention weights according to location of the words. Or adding more recurrent layers for encoder to further encode inter-word relationship.

ii. (2 points) **Source Sentence:** *Ulihelisdi nigalisda.*
Reference Translation: *She is very excited.*
NMT Translation: *It's joy.*

Solution: The NMT system picked a wrong pronoun. This error is likely caused by the final softmax output with vocabulary projection. A possible fix could be adding more layers to the final vocabulary projection.

iii. (2 points) **Source Sentence:** *Tsesdi hana yitsadawoesdi usdi atsadi!*
Reference Translation: *Don't swim there, Littlefish!*
NMT Translation: *Don't know how a small fish!*

Solution: The NMT system got the meaning right but failed to express a special word. This error is likely caused by the low memory power or representational capacity of the model. A possible fix could be adding more layers to the projection layers of the encoder and decoder, which are currently single-layer.

(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?

Solution:
Line 13 in `outputs/test_outputs.txt`:
"*But if we cast out demons by demons, not the kingdom of God is come.*"
Line 13 in the test target file:
"*But if I by the finger of God cast out demons, then is the kingdom of God come upon you.*"
The underlined string exits verbatim in line 95 of the training target file:
"*for I say unto you, I shall not eat it, until it be fulfilled in the kingdom of God.*"

This means the MT system learned to memorize some fixed phrases/combinations in a supervised learning way.

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?

Solution:
Line 41 in `outputs/test_outputs.txt`:
"*And he said unto him, Sir, why do thou that thou hast done to be a feast to us? And they were afraid.*"
Line 41 in the test target file:

"*and he saith unto him, Friend, how camest thou in hither not having a wedding-garment? And he was speechless.*"

The starting 5 words of translation are correct, but the latter part of the sentence seems unrelated. This shows the model has a limited field of view when decoding. The correct 5-word sequence can be found verbatim in training file. But not the subsequent part. The model cuts off the relationship between the two parts according to its memory.

(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.[2] Suppose we have a source sentence $\mathbf{s}$, a set of k reference translations $\mathbf{r}_1, \dots, \mathbf{r}_k$, and a candidate translation $\mathbf{c}$. To compute the BLEU score of $\mathbf{c}$, we first compute the *modified n-gram precision* p_n of $\mathbf{c}$, for each of $n = 1, 2, 3, 4$, where n is the n in n-gram:

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

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 $\mathbf{r}_1, \dots, \mathbf{r}_k$ 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 from Spanish:

Source Sentence $\mathbf{s}$: **el amor todo lo puede**
Reference Translation $\mathbf{r}_1$: *love can always find a way*
Reference Translation $\mathbf{r}_2$: *love makes anything possible*
NMT Translation $\mathbf{c}_1$: *the love can always do*
NMT Translation $\mathbf{c}_2$: *love can make anything possible*

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).

[2] 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

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.

Solution: BLEU score for $\mathbf{c}_1$:

$$\begin{aligned} p_1 &= \frac{\min(\max(0,0),1)+\min(\max(1,1),1)+\min(\max(1,0),1)+\min(\max(1,0),1)+\min(\max(0,0),1)}{1+1+1+1+1} \\ &= \frac{0+1+1+1+0}{5} = 0.6 \end{aligned}$$

$$\begin{aligned} p_2 &= \frac{\min(\max(0,0),1)+\min(\max(1,0),1)+\min(\max(1,0),1)+\min(\max(0,0),1)}{1+1+1+1} \\ &= \frac{0+1+1+0}{4} = 0.5 \end{aligned}$$

$BP_{\mathbf{c}_1} = 1$
$\text{BLEU}_{\mathbf{c}_1} = 1 \times \exp(0.5\log(0.6) + 0.5\log(0.5)) = 0.548$

BLEU score for $\mathbf{c}_2$:

$$\begin{aligned} p_1 &= \frac{\min(\max(1,1),1)+\min(\max(1,0),1)+\min(\max(0,0),1)+\min(\max(0,1),1)+\min(\max(0,1),1)}{1+1+1+1+1} \\ &= \frac{1+1+0+1+1}{5} = 0.8 \end{aligned}$$

$$\begin{aligned} p_2 &= \frac{\min(\max(1,0),1)+\min(\max(0,0),1)+\min(\max(0,0),1)+\min(\max(0,1),1)}{1+1+1+1} \\ &= \frac{1+0+0+1}{4} = 0.5 \end{aligned}$$

$BP_{\mathbf{c}_2} = 1$
$\text{BLEU}_{\mathbf{c}_2} = 1 \times \exp(0.5\log(0.8) + 0.5\log(0.5)) = 0.632$

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: The second NMT translation $\mathbf{c}_2$ is considered a better translation according to the BLEU score. I agree with the score.

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: BLEU score for $\mathbf{c}_1$:

$$\begin{aligned} p_1 &= \frac{\min(0,1)+\min(1,1)+\min(1,1)+\min(1,1)+\min(0,1)}{1+1+1+1+1} \\ &= \frac{0+1+1+1+0}{5} = 0.6 \end{aligned}$$

$$\begin{aligned} p_2 &= \frac{\min(0,1)+\min(1,1)+\min(1,1)+\min(0,1)}{1+1+1+1} \\ &= \frac{0+1+1+0}{4} = 0.5 \end{aligned}$$

$BP_{\mathbf{c}_1} = \exp(1 - 6/5) = 0.819$
$\text{BLEU}_{\mathbf{c}_1} = 0.819 \times \exp(0.5\log(0.6) + 0.5\log(0.5)) = 0.448$

BLEU score for $\mathbf{c}_2$:

$$\begin{aligned} p_1 &= \frac{\min(1,1)+\min(1,1)+\min(0,1)+\min(0,1)+\min(0,1)}{1+1+1+1+1} \\ &= \frac{1+1+0+0+0}{5} = 0.4 \end{aligned}$$

$$\begin{aligned} p_2 &= \frac{\min(1,1)+\min(0,1)+\min(0,1)+\min(0,1)}{1+1+1+1} \\ &= \frac{1+0+0+0}{4} = 0.25 \end{aligned}$$

$BP_{\mathbf{c}_2} = \exp(1 - 6/5) = 0.819$
$\text{BLEU}_{\mathbf{c}_2} = 0.819 \times \exp(0.5\log(0.4) + 0.5\log(0.25)) = 0.259$

The first NMT translation $\mathbf{c}_1$ is considered a better translation according to the BLEU score. I do not agree with the score.

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.

Solution: A single reference translation is often biased towards a single human translator's preference or habit. It is not an unbiased or fair representative of a good translation. The BLEU score measured by a single reference translation is therefore often biased and not comprehensive.

iv. (2 points) List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.

Solution:
Advantages:
- Can be computed automatically, a lot faster than human evaluation.
- Totally quantitative and non-subjective.

Disadvantages:
- Too mechanical, cannot give fair scores to some subtle translations.
- Hunger for data and can be biased if data are not enough. It relies on multiple reference translations to give better judgment.

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.
   If you are on Windows, run `collect_submission.bat`. You can either double-click on the script file or execute it in command line.
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.