

# NLPDL Assignment #2 Task 1: NMT with RNNs

## (50 Points)

Wang Keran 2100017727

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 ask TAs who 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.

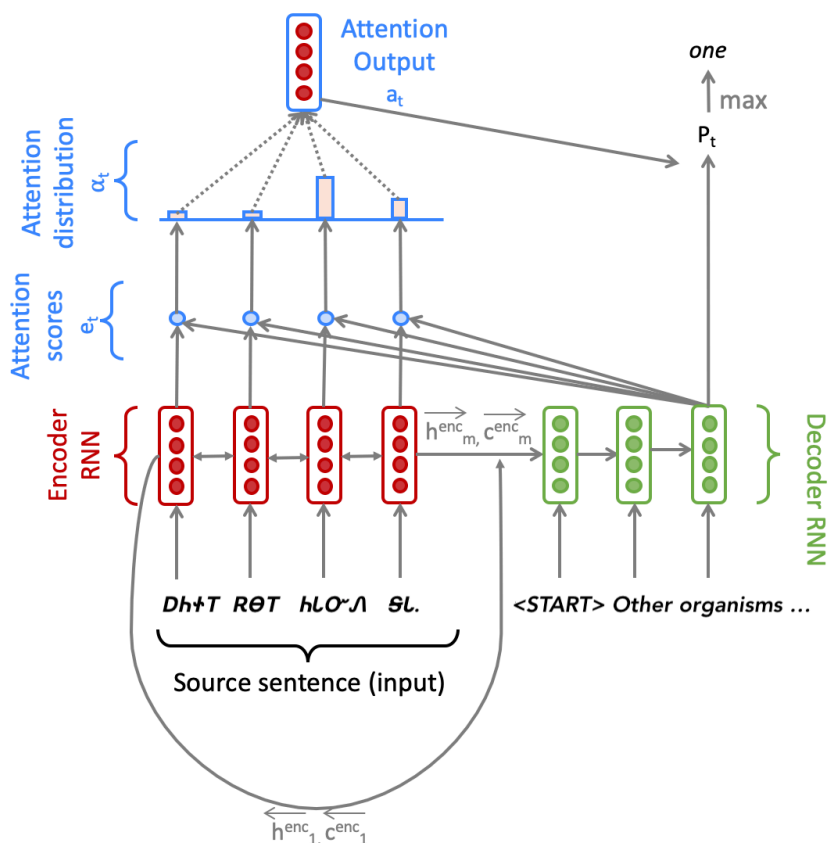


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states  $h^{enc}_i$  and cell states  $c^{enc}_i$  are defined in the next page.

## 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  $\mathbf{c}_i^{\text{enc}}$ :

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

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

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.<sup>1</sup>

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

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

With the decoder initialized, we must now feed it a target sentence. On the  $t^{\text{th}}$  step, we look up the embedding for the  $t^{\text{th}}$  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.

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

$$(6)$$

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

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

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

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

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

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

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

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

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

---

<sup>1</sup>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:

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

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  $t$ :

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

Here,  $\theta$  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

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.**

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

**Answer:**

Doing so eliminates the padding tokens, to make sure that they don't affect the model. It's necessary because if we don't do so, the vocab will be filled with padding tokens and the translation will be totally wrong.

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<sup>2</sup>. 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 (Boya No.1 or others).

Next, install necessary packages to your VM by running:

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

Finally, create a new tmux session. Concretely, run the following command to create tmux session called `nmt`.

```
tmux new -s nmt
```

---

<sup>2</sup><https://pytorch.org/docs/stable/tensorboard.html>

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.

**Answer:**

the BLEU is 12.36

- (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  $\mathbf{e}_{t,i} = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t)$ .
- (2 points) Explain one advantage and one disadvantage of *dot product attention* compared to multiplicative attention.
  - (2 points) Explain one advantage and one disadvantage of *additive attention* compared to multiplicative attention.

**Answer:**

- The advantage of *dot product attention* is that it's easy to compute, and the disadvantage is that it only considers the alignment between the query and the key vectors, which may result in loss of important contextual information.
- The advantage of *additive product attention* is that Multiplicative attention mechanisms allow the model to capture richer contextual information. The disadvantage of *additive product attention* is that they involve additional calculations and may require more computational resources to compute.

## 2. Analyzing NMT Systems (15 points)

- (a) (1 point) 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:**

subword-level has several advantages: It can handle the rare words and unknown words better. Even if a word is not present in the pre-trained word embeddings, its subword components may still be present and can be leveraged for better translation. It also Improves sentence-level context understanding, subword-level modeling allows the model to consider sentence-level context more effectively. Each subword can be interpreted in the context of its surrounding words.

- (b) (1 point) 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 **GO·WᎠA** (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 **G** 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:**

Transliteration converts Cherokee characters to Latin script based on phonetic similarity, meaning that the resulting text still remains the original sound. By training on this transliterated text, the NMT model can learn these phonetic relationships and improve its ability to accurately convert Cherokee characters to English.

- (c) (1 point) 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:**

It can leverage knowledge and patterns learned from one language to improve its performance on another language. What's more, doing so can also increase the resource about this low-resource language.

- (d) (3 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. (1 point) **Source Sentence:** ᎠᎵᎠᎵᎠ ᎠᎵᎠᎵᎠᎵᎠ, ᎠᎵ ᎠᎵᎠ ᎠᎵᎠᎵᎠ: ᎠᎵᎠᎵ ᎠᎵᎠᎵ ᎠᎵᎠᎵᎠᎵᎠ ᎠᎵᎠ ᎠᎵᎠᎵᎠ.

**Reference Translation:** When she was finished ripping things out, her web looked something like this:

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

- ii. (1 point) **Source Translation:** ᎠᎵᎠ ᎠᎵᎠᎵᎠ, ᎠᎵᎠᎵᎠ? ᎠᎵᎠᎵᎠᎵᎠ ᎠᎵᎠ ᎠᎵᎠᎵᎠ.

**Reference Translation:** What's wrong little tree? the boy asked.

**NMT Translation:** The little little little little little tree? asked him.

- iii. (1 point) **Source Sentence:** "ᎠᎵᎠᎵᎠ ᎠᎵᎠᎵᎠ," ᎠᎵᎠ ᎠᎵᎠ.

**Reference Translation:** " 'Humble,' " said Mr. Zuckerman

**NMT Translation:** "It's not a lot," said Mr. Zuckerman.

**Answer:**

i.

error: the word "it" is not corresponding to the word "he" reason: maybe the model doesn't consider the contact between sentences. way: add position encoding

ii.

error: too many same words reason: maybe the same word has the highest probability of being behind itself. way: decrease the probability of appending to the word itself

iii.

error: can't understand the polysemy word reason: the model doesn't have the ability to distinguish different meanings of same word way: to use multilingual training

(e) (2 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. (1 point) 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?

ii. (1 point) 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:**

i.

"for as many as", it can be found in the training file ( 5 times ). This implies that maybe the model just copy the most closest word to the output

ii.

"And when he found ( I found a lot tired of the sound hills )". This implies that the model didn't learn the connection between parts of one sentence. It will decode one code into two parts of sentences.

(f) (7 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.<sup>3</sup> 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](#):

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

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  $\text{len}(\mathbf{c})$  be the length of  $\mathbf{c}$  and let  $\text{len}(\mathbf{r})$  be the length of the reference translation that is closest to  $\text{len}(\mathbf{c})$  (in the case of two equally-close reference

<sup>3</sup>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](http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleu)

translation lengths, choose  $\text{len}(r)$  as the shorter one).

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

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

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

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

- i. (3 points) Please consider this example<sup>4</sup>:

Source Sentence  $\mathbf{s}$ : **Dc' OodY Ts-Slaidlady O'fBE Sdust O'fBYZ ic vsluhvT**

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

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  $p_1$ ,  $p_2$ ,  $\text{len}(c)$ ,  $\text{len}(r)$  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.

**Answer:**

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

Counts for 1-gram in  $\mathbf{r}_1$  : 1, 2, 1, 1, 1, 2, 0, 1, 0.

Counts for 1-gram in  $\mathbf{r}_2$  : 2, 3, 1, 1, 1, 2, 0, 1, 1.

Therefore, we obtain

$$p_1 = \frac{2 + 3 + 1 + 1 + 1 + 2 + 0 + 1 + 1}{2 + 3 + 1 + 1 + 1 + 2 + 1 + 1 + 1} = \frac{12}{13}$$

Counts for 2-gram in  $\mathbf{r}_1$  : 1, 1, 1, 1, 1, 2, 1, 0, 0, 0.

Counts for 2-gram in  $\mathbf{r}_2$  : 2, 1, 1, 1, 1, 2, 1, 0, 0, 0.

Therefore, we obtain

$$p_2 = \frac{2 + 1 + 1 + 1 + 1 + 2 + 1 + 0 + 0 + 1}{2 + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 1} = \frac{5}{6}$$

Finally, we have

$$BLEU = 1 \times \exp\left(0.5 \times \log \frac{12}{13} + 0.5 \times \log \frac{5}{6}\right) = 0.88$$

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

Counts for 1-gram in  $\mathbf{r}_1$  : 3, 1, 1, 2, 1, 1, 1, 1, 0

Counts for 1-gram in  $\mathbf{r}_2$  : 3, 1, 1, 2, 0, 1, 1, 2, 0

Therefore, we obtain

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

<sup>4</sup>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.



Counts for 2-gram in  $\mathbf{r}_1$  : 1, 1, 0, 2, 1, 1, 0, 1, 1, 1, 0.

Counts for 2-gram in  $\mathbf{r}_2$  : 1, 1, 0, 2, 0, 0, 0, 1, 1, 1, 0.

Therefore, we obtain

$$p_2 = \frac{1+1+0+2+1+1+0+1+1+1+0}{1+1+1+2+1+1+1+1+1+1+1} = \frac{3}{4}$$

Finally, we have

$$BLEU = 1 \times \exp(0.5 \times \log \frac{11}{13} + 0.5 \times \log \frac{3}{4}) = 0.80$$

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?

**Answer:**

According to them,  $c_1$  is considered better. I agree with the evaluation.

- ii. (2 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?

**Answer:**

BLEU score for  $c_1$  :

$$p_1 = \frac{10}{13}$$

$$p_2 = \frac{2}{3}$$

therefore,

$$BLEU = 0.865$$

BLEU score for  $c_2$  :

$$p_1 = \frac{11}{13}$$

$$p_2 = \frac{3}{4}$$

therefore,

$$BLEU = 0.797 \text{ So, I disagree with the evaluation.}$$

- iii. (1 point) Due to data availability, NMT systems are often evaluated with respect to only a single reference translation. Please explain (in a few sentences) why this may be problematic. In your explanation, discuss how the BLEU score metric assesses the quality of NMT translations when there are multiple reference translations versus a single reference translation.

**Answer:**

When there is only a single reference translation, we simply compute  $\text{Count}_r(\text{ngram})$  and  $\text{Count}_c(\text{ngram})$  for any n-gram appearing in  $c$ , take the smaller one between them and sum those minima together. This summation term represents the similarity between the machine translation and the reference translation based on the counts of n-grams. When we enlarge the number of the reference translations, for any n-gram appearing in  $c$ , we will select the reference sentence  $r^*$  where the n-gram appears the most times, representing that the sentence is to some extent the most similar to  $c$ . If we only use a single reference translation, the evaluation criterion of machine translation will be biased and consequently unconvincing, because the reasonable translations for a sentence may be diverse and this is exactly the beauty of translations.

- iv. (1 point) List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.

**Answer:**

Advantages: (1) BLEU is a quantitative metric to automatically evaluate the quality of machine translation, so it can greatly reduce the human efforts and improve the efficiency; (2) The method of BLEU is language-independent, so we can easily obtain the translation quality of various languages given the human translations. Disadvantages: (1) To better evaluate the results, we need to collect multiple reference translations for one sentence, which is definitely

time-consuming; (2) The translation work involves a plethora of diversity, and simply employing the similarity between machine translations and human translations cannot sufficiently illustrate that the model definitely does a good or bad job in translating sentences.