

# 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.

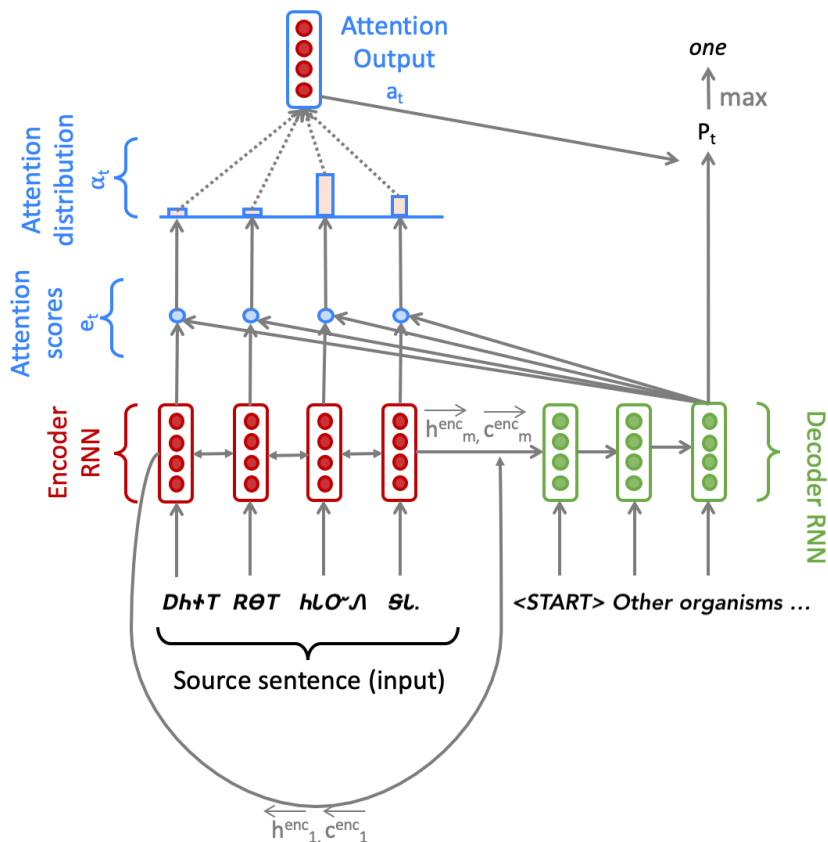


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states  $h_i^{enc}$  and cell states  $c_i^{enc}$  are defined in the next page.

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

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

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

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

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

- (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.
- (3 points) (coding) Implement the `__init__` function in `model_embeddings.py` to initialize the necessary source and target embeddings.
- (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.
- (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.

The masks zeroes out the influence of tokens corresponding to a 1 in the attention calculation. As a result, only non-pad tokens will be accounted for in attention computation. This is necessary because there is no purpose in distracting the decoder into paying attention to pad tokens. Also, masking them out will probably reduce variance in expected value of magnitude of attention layers. Its effect is akin to layer normalization, so we can expect a similar improvement from masking pad tokens.

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
```

---

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

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 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.

**BLEU Score: 12.579392661359114**

- (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.

**Dot product attention has lower dimensionality compared to multiplicative attention; dot product is linear in dimensions, while multiplicative is quadratic. Disadvantage of dot product attention is that encoder and decoder and hidden states may be encoded differently, so similarity of hidden states does not imply the corresponding vector contains useful information.**

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

**The main problem with additive attention is that there's no interaction between hidden and current time step state, since they're just being added together. Due to the properties of tanh, one can easily dominate the other. However, it's less computationally compared to multiplicative attention; since  $W_1 h_i$  and  $W_2 s_t$  can both either be precomputed or reused, there is only**

a linear number of easily parallelizable add operations to worry about, which are a lot more efficient to compute than a linear number of dot products.

## 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.)

It's important to model at subword-level because the meaning of each morpheme is shared between words. If we modeled at word-level instead, we'll struggle due to high dimensionality since we need to learn an embedding for every word in vocab and we'll have no information about words not present in training set.

- (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 ᎠᎵᎦᎵᎦᎵᎦ (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 Ꭰ 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.)

Since the morpheme is "ts" and not the Cherokee character itself, while there may be many Cherokee characters that start with "ts" when transliterated, it would be better to learn an embedding for "ts" directly instead of learning each one separately. The reason for this improvement of performance is similar to training at subword-level instead of whole word-level: we want to learn embeddings for the most basic units of meaning.

- (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?

Multilingual training is essentially a type of transfer learning. It helps improve NMT performance because even though the meaning of each word is not transferable, there are similarities in syntactic structures and grammatical concepts across languages, which is a good starting point for 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:** ᐱᐱᐱᐱ ᐱᐱᐱᐱᐱᐱ, ᐱᐱ ᐱᐱᐱ ᐱᐱᐱᐱ: ᐱᐱᐱᐱ ᐱᐱᐱᐱ ᐱᐱᐱᐱᐱᐱ ᐱᐱᐱ ᐱᐱᐱᐱᐱ.

**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.

**Error:** Incorrect pronoun

**Possible Reason:** The model did not record the gender of the personal pronoun. This error can be attributed to a linguistic construct since the inferred meaning behind the translation is essentially the same regardless of gender.

**Fix:** Increase size of hidden state (so it's likely to encode gender) and Change the attention mechanism so that it pays attention to subject gender (such as K, Q, V attention mechanism)

- ii. (2 points) **Source Translation:** ᐱᐱᐱ ᐱᐱᐱᐱ, ᐱᐱᐱᐱ ᐱᐱᐱᐱᐱᐱ ᐱᐱᐱ ᐱᐱᐱ.

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

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

**Error:** Word gets repeated

**Possible Reason:** The hidden and cell state barely react to outputting the word "little" and knowing "little" to be the previous combined-output, causing the decoder to continually output "little."

**Fix:** Change the attention mechanism so that it plays more of a role in calculating the new hidden state.

- iii. (2 points) **Source Sentence:** "ᐱᐱᐱᐱ ᐱᐱᐱᐱ," ᐱᐱᐱ ᐱᐱ.

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

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

**Error:** Word gets replaced by meaning

**Possible reason:** The hidden state encodes the meaning, but it translates it into a longer sequence instead of the target word, which may be because of the brevity penalty. **Fix:** Use K, Q, V attention, so that if the context is something like "What quality..." or "What word..." for which "Humble" is an appropriate answer, the decoder can make use not only of the meaning (which is what is most often encoded by hidden state) but also be able to learn that the target should be a single word.

- (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?

On line 9 of test output, the phrase "And they were all amazed" is matched word for word. In the training set, this exact phrase appeared twice. The system likely overfitted to the training data's writing style, which will probably result in poor performance when applied to a different

data source.

- 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?

On line 9 of test output, the output diverges in what they're saying. The model outputs that there are things of Galilee, while the reference says that these people are all Galilæans. This illustrates the fact that while the decoder is still able to remember the gist of the text, it forgets that the "things" are people; while attention helps, the hidden and cell states are still too heavily dependent on the LSTM for updates.

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

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

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. (5 points) Please consider this example<sup>4</sup>:

Source Sentence  $\mathbf{s}$ : **Dɔ̃ ʔəɔ̃ Tʃ-ʃɪəɔ̃ ʔɪbɛ ʃəʊʃt ʔɪbɪz ɪɫ ʋʃlɪʔɪʔ**

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*

<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)

<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.



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.

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?

BLEU 1: 0.8121215895894301, BLEU 2: 0.7376460654871967

According to BLEU score, the first NMT translation is the better translation. I agree that it is the better translation.

NMT Translation 1:

$p_1 = 0.9230769230769231$

$p_2 = 0.8333333333333334$

$\text{len}(c) = 13$

$\text{len}(r) = 14$

$BP = 0.925961078642316$

NMT Translation 2:

$p_1 = 0.8461538461538461$

$p_2 = 0.75$

$\text{len}(c) = 13$

$\text{len}(r) = 14$

$BP = 0.925961078642316$

- 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?

BLEU 1: 0.7161148740394329, BLEU 2: 0.7966275068156915

The second NMT translation now receives the higher BLEU score. The second translation is clearly worse compared to the first one.

NMT Translation 1:

$p_1 = 0.7692307692307693$

$p_2 = 0.6666666666666666$

$\text{len}(c) = 13$

$\text{len}(r) = 13$

$BP = 1$

NMT Translation 2:

$p_1 = 0.8461538461538461$

$p_2 = 0.75$

$\text{len}(c) = 13$

$\text{len}(r) = 13$

$BP = 1$

- 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 translations versus a single reference translation.

Having only a single reference translation is problematic because it requires NMT translations to be verbatim in order to achieve high scores, when the meaning behind the NMT translation could be the same as the reference's, but paraphrased. When multiple reference translations are available, BLEU score is calculated with n-gram counts from the NMT translation with the highest count for that n-gram, giving "benefit of the doubt" to the NMT translation. Having multiple reference translations better accounts for synonyms and paraphrases, making it better than single reference translation by assessing quality of meaning instead of assessing whether NMT translation is a verbatim replica of reference translation.

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

Advantages:

- Objective
- Requires no human effort

Disadvantages:

- BLEU penalizes brevity, but concise is often better than verbose.
- BLEU doesn't actually assess the meaning behind the translation. If a word is similar in meaning but not close enough in meaning to be present in the provided reference translations, BLEU fails to account for this closeness in meaning.

## 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.