

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 **2 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. Mandarin Chinese) 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.

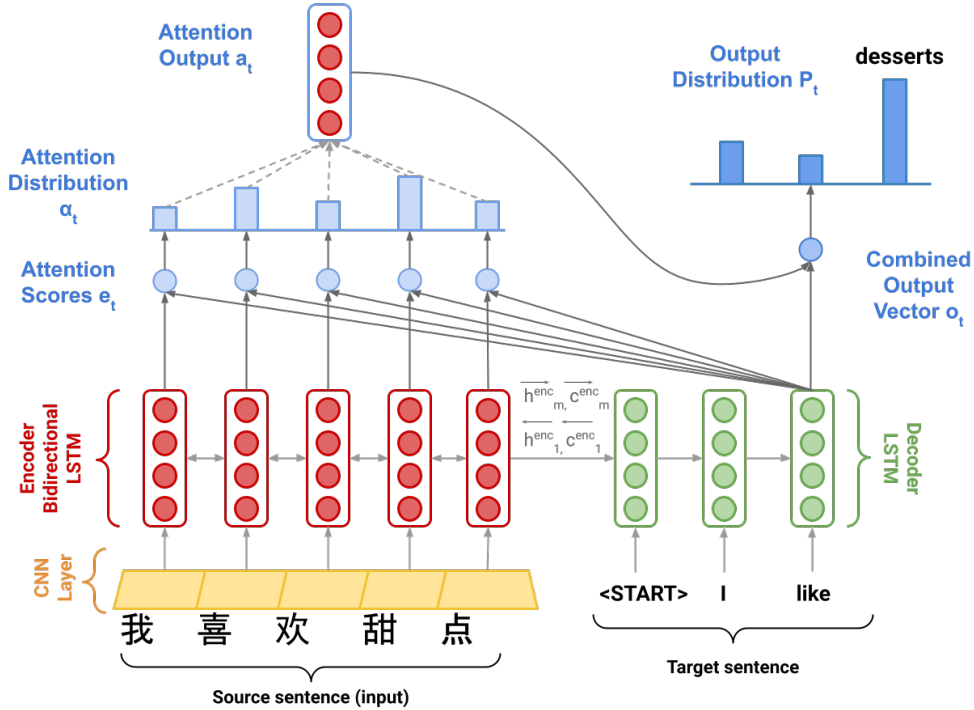


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states \vec{h}^{enc}_i and cell states \vec{c}^{enc}_i are defined on the next page.

Model description (training procedure)

Given a sentence in the source language, we look up the character or word 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 then feed the embeddings to a **convolutional layer**¹ while maintaining their shapes. We feed the convolutional layer outputs 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.²

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

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

With the decoder initialized, we must now feed it a target sentence. On the t^{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.

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

$$(6)$$

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 .

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

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

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

¹Checkout <https://cs231n.github.io/convolutional-networks> for an in-depth description for convolutional layers if you are not familiar

²If 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, 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 Mandarin Chinese 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 **1.5 to 2 hours** 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 layers (LSTM, CNN, 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 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 311-312).

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 mask marks where in the source sentence are padded with padding tokens - 1 for padded, 0 otherwise. It helps sets values of e_{ti} at the padded locations to $-\infty$ before applying softmax and calculating the attention score.

Since the values are set to $-\infty$, after softmax the value goes to zero, so in the attention calculation, it will not incorporate the encoder hidden state of the padded tokens.

This is necessary because otherwise, we would artificially considered the padded tokens in attention calculation, even though they were added for training purposes without any real meanings. Masking out the padded locations allows the model to attend to the actual part of the sentences without noises.

Now it's time to get things running! 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
```

For a faster way to debug by training on less data, you can run the following instead:

```
sh run.sh train_debug
(Windows) run.bat debug
```

To help with monitoring and debugging, the starter code uses tensorboard to log loss and perplexity during training using TensorBoard³. 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

³<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 2 hours 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 18.

Solution: Corpus BLEU: 19.519071234461872

- (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.

Solution:

- dot product attention* compared to multiplicative attention

Advantage Dot product is batch vector dot product, it is computationally fast, and does not need to train an extra weight matrix.

Disadvantage It requires s_t and h_i to have the same dimension. In the multiplicative attention case, W_t is able to reshape h_t by projection and allow attention calculation with encoder hidden states and decoder hidden states of different dimensions.

- *additive attention* compared to multiplicative attention.

Advantage It has separate projection matrix for encoder hidden states and decoder hidden states, and it is similar to feed forward neural net layer with nonlinear activation layer, so it is more expressive, and can potentially learn a more complicated representation than using a shared projection matrix.

Disadvantage It does not model explicitly the interactions between decoder hidden state and encoder hidden states since additive attention combines the two states by addition. It also requires learning two separate matrix parameters.

2. Analyzing NMT Systems (25 points)

- (a) (3 points) Look at the `src.vocab` file for some examples of phrases and words in the source language vocabulary. When encoding an input Mandarin Chinese sequence into “pieces” in the vocabulary, the tokenizer maps the sequence to a series of vocabulary items, each consisting of one or more characters (thanks to the `sentencepiece` tokenizer, we can perform this segmentation even when the original text has no white space). Given this information, how could adding a 1D Convolutional layer after the embedding layer and before passing the embeddings into the bidirectional encoder help our NMT system? **Hint:** each Mandarin Chinese character is either an entire word or a morpheme in a word. Look up the meanings of 电, 脑, and 电脑 separately for an example. The characters 电 (electricity) and 脑 (brain) when combined into the phrase 电脑 mean computer.

Solution: 1D convolution layer allows processing of several words’ embedding together (n-grams), based on the window size, rather than one word at a time. Using 1D convolution can capture the local dependencies between words in the input sequence. This is especially helpful in languages such as Chinese where there is no white space to denote separation of words.

In the case of 电脑, applying 1D convolution, the model would learn to associate these two characters together (computer), rather than see them as two separate characters (electricity + computer).

- (b) (8 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 Mandarin to answer these questions. You just need to know English! If, however, you would like some additional color on the source sentences, feel free to use a resource like https://www.archchinese.com/chinese_english_dictionary.html to look up words. Feel free to search the training data file to have a better sense of how often certain characters occur.

- i. (2 points) **Source Sentence:** 贼人其后被警方拘捕及被判处盗窃罪名成立。

Reference Translation: *the culprits were subsequently arrested and convicted.*

NMT Translation: *the culprit was subsequently arrested and sentenced to theft.*

Solution:

- ****error**** NMT did not identify the noun as plural.
- ****reason**** In Chinese, plural or singular form is not reflected in the noun, rather inferred in context.
- ****fix**** Train with more sentences that use singular and plural forms in different context. Or try to train/test with longer and several sentences, such that NMT might be able to infer singular and plural forms from additional context.

ii. (2 points) **Source Sentence:** 几乎已经没有地方容纳这些人, 资源已经用尽。

Reference Translation: *there is almost no space to accommodate these people, and resources have run out.*

NMT Translation: *the resources have been exhausted and resources have been exhausted.*

Solution:

- ****error**** NMT repeatedly translated the latter part of the sentence.
- ****reason**** Attention does not have an inherent sense of order, and in this case attention weights are overly high for the latter part of the sentence.
- ****fix**** Add position representations to the input, so that output can be aware of the orders of the input sentence, and to not overlook certain parts of the sentence.

iii. (2 points) **Source Sentence:** 当局已经宣布今天是国殇日。

Reference Translation: *authorities have announced a national mourning today.*

NMT Translation: *the administration has announced today's day.*

Solution:

- ****error**** NMT did not translate the word 国殇 at all.
- ****reason**** Zero or very low attention on these words, such that NMT did not attend at to these source words.
- ****fix**** Try a multihead attention, so that even one attention head misses the word, it might be picked up by the others. Increase the hidden layer dimension, so a larger or wider feature could capture the words without missing. Adding a coverage vector to the decoder, so that NMT learns to cover as high percentage of the source sentence as possible in translation.

iv. (2 points) **Source Sentence**⁴: 俗语有云:“唔做唔错”。

Reference Translation: *“act not, err not”, so a saying goes.*

NMT Translation: *as the saying goes, “it's not wrong.”*

Solution:

- ****error**** NMT gave a translation with wrong meaning (potentially also due to skipping the first part 唔做 or did not understand the “if-then” structure in the source sentence).
- ****reason**** 1) It could be due to low attention weights on 唔做, thus skipping translation, similar to this case above. 2) Or it could be due to the fact that the source is ancient chinese, which are very rare in modern chinese languages, so NMT has not seen enough in training to properly translate. In ancient chinese, from a syntactic perspective, the conditional structure of the original statement is implied by an appositive (A, B), instead

⁴This is a Cantonese sentence! The data used in this assignment comes from GALE Phase 3, which is a compilation of news written in simplified Chinese from various sources scraped from the internet along with their translations. For more details, see <https://catalog.ldc.upenn.edu/LDC2017T02>.

of the usual "if-then" conjunction explicitly. The subject is also omitted from the sentence (i.e. if someone did not do something, they would not err). These are artifacts of the ancient Chinese language that the NMT model might have failed to learn.

- ****fix**** For 1) (same as above question) try multi-head attention, increase the hidden layer dimension, use a coverage vector. For 2) train with more ancient Chinese corpus so that NMT can learn the distinct syntax structure and meanings.

- (c) (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.⁵ 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:

Source Sentence \mathbf{s} : 需要有充足和可预测的资源。

Reference Translation \mathbf{r}_1 : *resources have to be sufficient and they have to be predictable*

Reference Translation \mathbf{r}_2 : *adequate and predictable resources are required*

NMT Translation \mathbf{c}_1 : there is a need for adequate and predictable resources

NMT Translation \mathbf{c}_2 : resources be sufficient and predictable to

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 work (i.e., show your computed values for p_1 , p_2 , $\text{len}(\mathbf{c})$, $\text{len}(\mathbf{r})$ 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

⁵This 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

0 to 1 scale. Please round your responses to 3 decimal places.

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?

1 gram								
	c_1	r_1	r_2		c_2	r_1	r_2	
There	1	0	0	resources	1	1	1	
is	1	0	0	be	1	2	0	
a	1	0	0	sufficient	1	1	0	
need	1	0	0	and	1	1	1	
for	1	0	0	predictable	1	1	1	
adequate	1	0	1	to	1	1	0	
and	1	1	1					
predictable	1	1	1					
resources.	1	1	1					
2 gram								
	c_1	r_1	r_2		c_2	r_1	r_2	
There is	1	0	0	resources be	1	0	0	
is a	1	0	0	be sufficient	1	1	0	
a need	1	0	0	sufficient and	1	1	0	
need for	1	0	0	And predictable	1	0	1	
for adequate	1	0	0	Predictable to	1	0	0	
adequate and	1	0	1					
and predictable	1	0	1					
predictable resources	1	0	1					

Figure 2: ngram count

Solution: For c_1 , $p_1 = \frac{0+0+0+0+0+1+1+1+1}{9} = \frac{4}{9}$

$p_2 = \frac{0+0+0+0+0+1+1+1}{8} = \frac{3}{8}$

$len(c_1) = 9, len(r_1) = 11, len(r_2) = 6$

$BP = \exp(1 - \frac{11}{9}) = 0.800$

So, $BLEU_{c_1} = 0.800 \times \exp(0.5 \times \log(\frac{4}{9}) + 0.5 \times \log(\frac{3}{8})) = 0.327$

For c_2 , $p_1 = \frac{1+1+1+1+1+1}{6} = 1$

$p_2 = \frac{0+1+1+1+0}{5} = \frac{3}{5}$

$len(c_2) = 6, len(r_1) = 11, len(r_2) = 6$

$BP = 1$ because $len(c_1) > len(r_2)$

So, $BLEU_{c_2} = 1 \times \exp(0.5 \times \log(1) + 0.5 \times \log(\frac{3}{5})) = 0.775$

In summary, $BLEU_{c_1} = 0.307$ and $BLEU_{c_2} = 0.775$. This is obviously biased. c_1 is clearly a better translation - it is a full sentence and translates all the necessary meanings, while c_2 barely constitutes a sentence. However, BLEU gives c_2 a higher score because there is more ngram overlap.

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

Solution:

For c_1 , $p_1 = \frac{4}{9}$

$p_2 = \frac{3}{8}$

$len(c_1) = 9, len(r_2) = 6$

$BP = 1$ because $len(c_1) > len(r_2)$

So, $BLEU_{c_1} = 1 \times \exp(0.5 \times \log(\frac{4}{9}) + 0.5 \times \log(\frac{3}{8})) = 0.408$

For c_2 , $p_1 = \frac{3}{6} = 0.5$

$p_2 = \frac{1}{5} = 0.2$

$len(c_2) = 9, len(r_2) = 6$

$BP = 1$ because $len(c_2) > len(r_2)$

So, $BLEU_{c_2} = 1 \times \exp(0.5 \times \log(0.5) + 0.5 \times \log(0.2)) = 0.316$

In summary, $BLEU_{c_1} = 0.408$ and $BLEU_{c_2} = 0.316$. C_1 has a higher score now. I agree that C_1 is clearly a better translation for the reasons mentioned above. However the reason why c_2 has a lower BLUE now is that it happens to have very little ngram overlap with r_1 which we omitted. This also shows how volatile BLUE score is given reference availability.

- 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.

Solution: BLEU calculated with respect to only a single reference translation is problematic because it is very susceptible to which reference chosen. Usually a sentence can be translated in different ways with no loss of meaning or grace, but switching order of words or changing words to synonyms. BLEU score can vary a lot depends on which reference is used, even though the candidate translation quality is the same.

BLEU calculated with 1 reference gives a weighted overlapping score between candidate translation and the reference translation. BLUE calculated with multiple reference works with the assumption that the candidate is good if it has a high overlap with at least 1 reference out of multiple for ngrams at different position. This is a bit more nuanced than single-reference BLEU, because it incorporated the $\max()$ function that creates some level of hashing. In this way, the BLEU score with multiple references takes into account the diversity of possible translations for a given source sentence and provides a more nuanced evaluation of the quality of the candidate translation. This alleviates a bit the single-reference issue above, but it is still susceptible to word choices, word order etc to some extent.

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

Solution: Advantages

- Fast to compute if we have sentence of reasonable length, much faster compared to human evaluation

- More objective (n gram precision defined by a set formula) is less subjective than human evaluation.

Disadvantages

- BLEU does not capture the syntax or structure of a translation well. Since it only measures ngram precision ignorant of the positions, it overlooks how the sentence is structured. It is possible that a badly structured translation that barely forms a sentence gets a high BLEU score given the right selection of ngrams.

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.