CS 224n Assignment #4: NMT Systems

Koji Tadokoro

Updated May, 22 at 5:12am

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

1. Neural Machine Translation with RNNs (45 points)

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

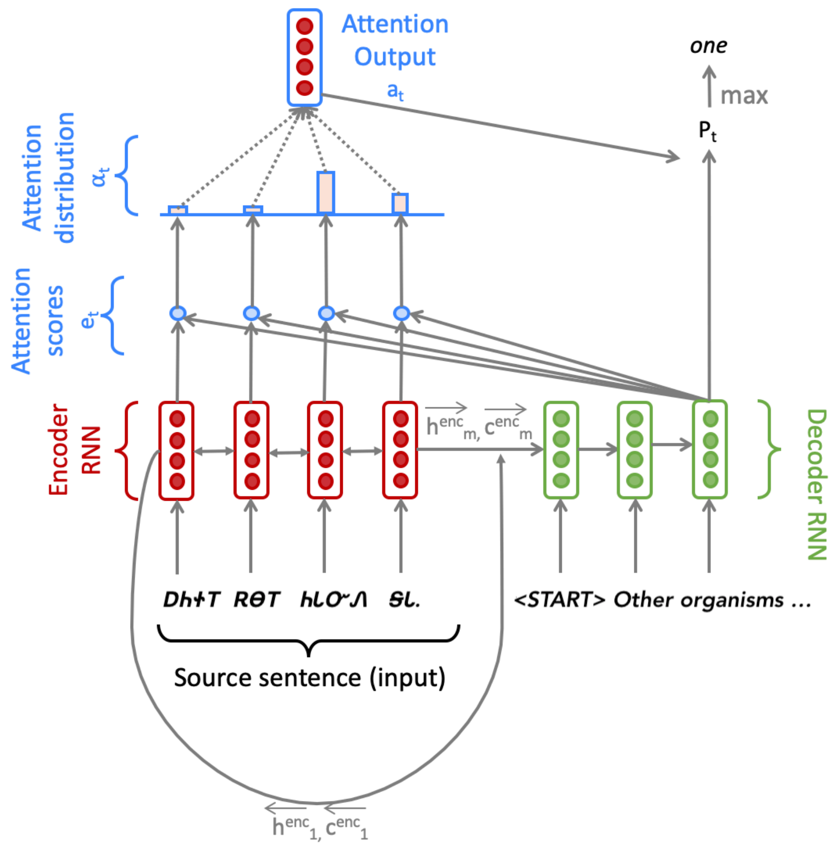

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined in the next page.

Model description (training procedure)

Given a sentence in the source language, we look up the subword embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \dots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e\times 1}$), 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.

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

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

$$\mathbf{e}_{t,i} = (\mathbf{h}_t^{\text{dec}})^T \mathbf{W}_{\text{attProj}} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{e}_t \in \mathbb{R}^{m\times 1}, \mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h} \qquad 1 \le i \le m \tag{6}$$

$$\alpha_t = \text{softmax}(\mathbf{e}_t) \text{ where } \alpha_t \in \mathbb{R}^{m\times 1} \tag{7}$$

$$\mathbf{a}_t = \sum_{i=1}^{m} \alpha_{t,i} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{a}_t \in \mathbb{R}^{2h\times 1} \tag{8}$$

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

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

$$\mathbf{u}_t = [\mathbf{a}_t; \mathbf{h}_t^{\text{dec}}] \text{ where } \mathbf{u}_t \in \mathbb{R}^{3h\times 1} \tag{9}$$

$$\mathbf{v}_t = \mathbf{W}_u \mathbf{u}_t \text{ where } \mathbf{v}_t \in \mathbb{R}^{h\times 1}, \mathbf{W}_u \in \mathbb{R}^{h\times 3h} \tag{10}$$

$$\mathbf{o}_t = \text{dropout}(\tanh(\mathbf{v}_t)) \text{ where } \mathbf{o}_t \in \mathbb{R}^{h\times 1} \tag{11}$$

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

Then, we produce a probability distribution $\mathbf{P}_t$ over target subwords at the t^{th} timestep:

$$\mathbf{P}_t = \text{softmax}(\mathbf{W}_{\text{vocab}}\mathbf{o}_t) \text{ where } \mathbf{P}_t \in \mathbb{R}^{V_t \times 1}, \mathbf{W}_{\text{vocab}} \in \mathbb{R}^{V_t \times h} \tag{12}$$

Here, V_t is the size of the target vocabulary. Finally, to train the network we then compute the cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the one-hot vector of the target subword at timestep t:

$$J_t(\theta) = \text{CrossEntropy}(\mathbf{P}_t, \mathbf{g}_t) \tag{13}$$

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!

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 α_t, the attention output $\mathbf{a}_t$, and finally the combined output $\mathbf{o}_t$. You can run a non-comprehensive sanity check by executing:

```
python sanity_check.py 1f
```

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

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

Solution:

Firstly, the generate_sent_masks() function masks the pad tokens to 1s and non-pad tokens to 0s. In the step() function, pad tokens' masking are set to $-\infty$ in the attention score vector $\mathbf{e}_t$. Therefore, the corresponding dimensions become zero when applying softmax to $\mathbf{e}_t$ to yield α_t, for $\exp(-\infty) = 0$.

It is necessary since pad tokens are not actually part of the original source sequences. They are not supposed to attend the calculation of dataflow in an attention mechanism.

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

```
sh run.sh vocab
```

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

```
sh run.sh train
```

(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
```

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

Solution:

The model's corpus BLEU score is 12.730755004968334.

(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.

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

Solution:

i. **Advantage**: The *dot product attention* does not require extra model parameters and is easier to compute.
Disadvantage: In the *dot product attention*, the shape of *query* and *key* should be the same. While in the *multiplicative attention*, *query* and *key* could have different dimensions since there is a projection matrix $\mathbf{W}$ to perform linear transformation.

ii. **Advantage**: The *additive attention* introduces a non-linear transformation function(tanh) and assign two different weight matrices to *query* and *key*, so it is likely to be more powerful to capture complex connections.
Disadvantage: Obviously, the *additive attention* is more computationally expensive.

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.)

Solution:

From wiki: *Polysynthetic languages are highly synthetic languages, i.e. languages in which words are composed of many morphemes. This kind of languages typically have long sentence-words.*

Therefore, there will be a very large embedding matrix if we model the Cherokee-to-English NMT problem at the **whole word**-level and it is not practical.

Modeling at subword-level makes the embedding lookup operation more effective. On the other hand, this fine-grained processing style might capture morphosyntactic relations for polysynthetic languages.

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

Solution:

The original Cherokee characters may contain inseparable combinations of morphemes, so training on transliterated Cherokee text could make it easier to separate word into morphemes.

(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?

Solution:

Multilingual model can somehow transfer parameters from relatively languages and improve NMT performance with 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:** *ᏄᏩᏁᏲᎾ ᏕᎪᏣᎳᎩᏍᎬ, ᎯᎠ ᏄᏍᏕ ᏚᏏᎳᏛ: ᏧᏓᎴᏅᏓ ᏕᎪᎡᏍᏕ ᏧᏏᎳᏛᏙᏗ ᎠᏍᏓ ᎾᏅᏂᏍᎩ.*
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.*
Solution:
The NMT model used *"it"* and *"he"* to refer to the same person. It seems that the NMT model has not really learned the grammar and semantics of target language. Enlarging the amount of training data and making our model deeper might helps.

ii. (2 points) **Source Translation**: *ᏅᏬᏘ ᎢᏡᎬᎢ, ᏕᏙᏯᎢ? ᏅᏓᏉᏉᏁᎢ ᏅᏬᏘ ᎠᏧᏣ.*
Reference Translation: *What's wrong little tree? the boy asked.*
NMT Translation: *The little little little little little tree? asked him.*
Solution:
The word *"little"* is repeated for 5 times by the NMT decoder. It is possible that the model attends to the same parts of source sentence and calculated a similar probability distribution. It might help to introduce an self-attention mechanism.

iii. (2 points) **Source Sentence:** *"ᏅᏓᏍᏉᏗ ᏂᎨᏒᎾ," ᏅᏛᏁ ᎨᎻ.*
Reference Translation: *" 'Humble,' " said Mr. Zuckerman*
NMT Translation: *"It's not a lot," said Mr. Zuckerman.*
Solution:
The NMT model used *"It's not a lot"* as an expression of *"Humble"*. In the translations my model produced, the corresponding translation is: *"I'm not afraid," said Mr. Zuckerman.*
The reason might be that the model has not learned the representation of *"humble"* well. To fix the error, applying a character-based NMT model or a hybrid one may work (not sure).

(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:
Correct sequence of words: *"Wilbur was going to sleep."*
Reference translation: *"Wilbur was pleased to receive so much attention."*
The reference translation contains the string *"Wilbur was"* verbatim. It indicates that the MT system somehow learned to build a sequence according to a language model on the target data.

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?
Predicted sequence: *Everyone of peace was happy and joy in his life and a world in the world.*
Reference translation: *All these sounds made him feel comfortable and happy, for he loved life and loved to be a part of the world on a summer evening.*
It seems that the decoder cannot be aware of what it generated in previous time-steps.

(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 $\mathbf{r}_1, \ldots, \mathbf{r}_k$, and a candidate translation $\mathbf{c}$. To compute the BLEU score of $\mathbf{c}$, we first compute the *modified n-gram precision* p_n of $\mathbf{c}$, for each of $n = 1, 2, 3, 4$, where n is the n in n-gram:

$$p_n = \frac{\displaystyle\sum_{\text{ngram}\in\mathbf{c}} \min\left(\max_{i=1,\ldots,k} \text{Count}_{\mathbf{r}_i}(\text{ngram}),\ \ \text{Count}_{\mathbf{c}}(\text{ngram})\right)}{\displaystyle\sum_{\text{ngram}\in\mathbf{c}} \text{Count}_{\mathbf{c}}(\text{ngram})} \tag{14}$$

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).

$$BP = \begin{cases} 1 & \text{if } len(c) \geq len(r) \\ \exp\left(1 - \frac{len(r)}{len(c)}\right) & \text{otherwise} \end{cases} \tag{15}$$

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

$$BLEU = BP \times \exp\left(\sum_{n=1}^{4} \lambda_n \log p_n\right) \tag{16}$$

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[3]:
Source Sentence $\mathbf{s}$: **ᎠᏯ ᎾᏍᎩ ᎢᏕ-ᏕᏗᏍᏗᏍᎩ ᎤᎵᏏᎬ ᎦᏎᏌᏕᎢ ᎤᎵᏏᎩᏃ ᎥᏝ ᏬᏚᏓᏂᏎᏙᎢ**
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, $len(c)$, $len(r)$ and BP). Note that the BLEU scores can be expressed between 0 and 1 or between 0 and 100. The code is using the 0 to 100 scale while in this question we are using the **0 to 1** scale.

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

Solution:

BLEU Score of $\mathbf{c}_1$ and $\mathbf{c}_2$ is computed below.

[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

[3]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 $\mathbf{c}_1$: $p_1 = 12/13, p_2 = 10/12, len(c) = 13, len(r) = 13, BP = 1$, then

$$BLEU = BP \times \exp\left(0.5 \log p_1 + 0.5 \log p_2\right) = 0.8771$$

NMT Translation $\mathbf{c}_2$: $p_1 = 11/13, p_2 = 9/12, len(c) = 13, len(r) = 13, BP = 1$, then

$$BLEU = BP \times \exp\left(0.5 \log p_1 + 0.5 \log p_2\right) = 0.7966$$

Translation $\mathbf{c}_1$ is considered the better translation according to the BLEU Score. I also agree that $\mathbf{c}_1$ is better than $\mathbf{c}_2$.

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:
NMT Translation $\mathbf{c}_1$: $p_1 = 10/13, p_2 = 8/12, len(c) = 13, len(r) = 13, BP = 1$, then

$$BLEU = BP \times \exp\left(0.5 \log p_1 + 0.5 \log p_2\right) = 0.7161$$

NMT Translation $\mathbf{c}_2$: $p_1 = 11/13, p_2 = 9/12, len(c) = 13, len(r) = 13, BP = 1$, then

$$BLEU = BP \times \exp\left(0.5 \log p_1 + 0.5 \log p_2\right) = 0.7966$$

Now, $\mathbf{c}_2$ receives the higher BLEU Score. I do not agree it is the better translation.

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.
Solution:
When a candidate translation is compared to a single reference translation, the representation of the relevant n-grams should be matched to obtain a high BLEU Score and this kind of representation is noisy. As a result, it is obviously possible that an appropriate translation also receives a relatively low BLEU Score.
If there are multiple reference translations, good translations are more likely to get fair BLEU Scores. Therefore, the NMT systems are more flexible to interpret and decode the text from source language to target language.

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

- BLEU makes NMT evaluation fast and economical.
- It is language independent and easy to understand.

Disadvantages

- BLEU compares overlap in tokens from the predictions and references, instead of comparing meaning. This can lead to discrepancies between BLEU scores and human ratings.
- BLEU scores are not comparable across different datasets, nor are they comparable across different languages.
- BLEU scores can vary greatly depending on which parameters are used to generate the scores, especially when different tokenization and normalization techniques are used. It is therefore not possible to compare BLEU scores generated using different parameters, or when these parameters are unknown.