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.

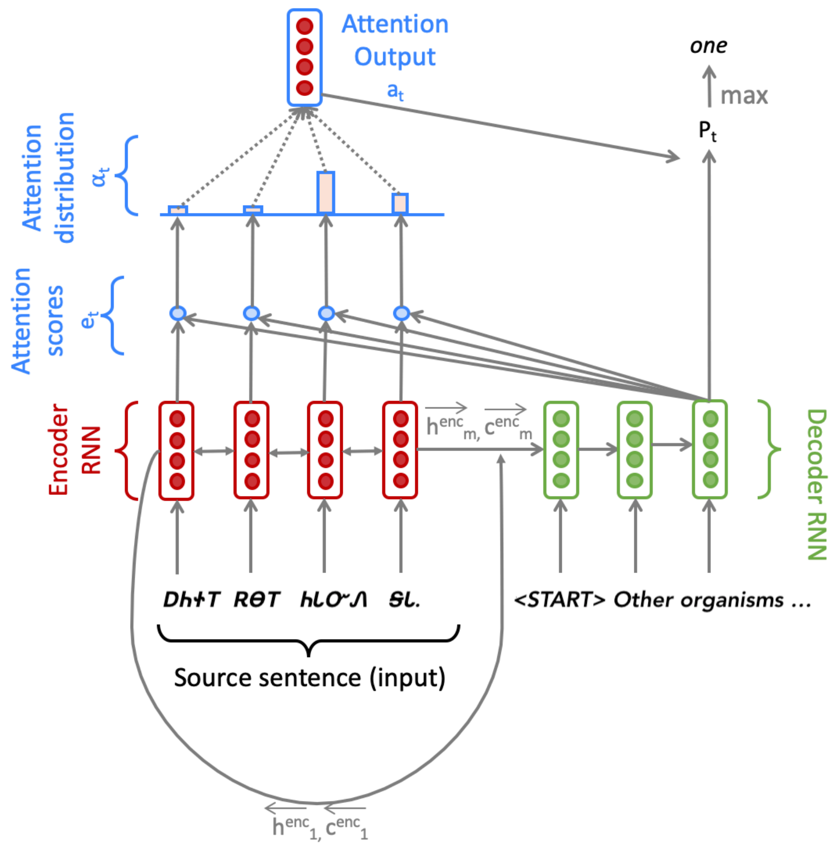

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined in the next page.

Model description (training procedure)

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

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

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

We then initialize the decoder's first hidden state $\mathbf{h}_0^{\text{dec}}$ and cell state $\mathbf{c}_0^{\text{dec}}$ with a linear projection of the encoder's final hidden state and final cell state.[1]

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

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

With the decoder initialized, we must now feed it a target sentence. On the t^{th} step, we look up the embedding for the t^{th} subword, $\mathbf{y}_t \in \mathbb{R}^{e\times 1}$. We then concatenate $\mathbf{y}_t$ with the *combined-output vector* $\mathbf{o}_{t-1} \in \mathbb{R}^{h\times 1}$ from the previous timestep (we will explain what this is later down this page!) to produce $\overline{\mathbf{y}_t} \in \mathbb{R}^{(e+h)\times 1}$. Note that for the first target subword (i.e. the start token) $\mathbf{o}_0$ is a zero-vector. We then feed $\overline{\mathbf{y}_t}$ as input to the decoder.

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

$$\tag{6}$$

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

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

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

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

$\mathbf{e}_{t,i}$ 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 $\mathbf{o}_t$.

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

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

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

[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:

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

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:

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

Here, θ represents all the parameters of the model and $J_t(\theta)$ is the loss on step t of the decoder. Now that we have described the model, let's try implementing it for Cherokee to English translation!

Setting up your Virtual Machine

Follow the instructions in the CS224n Azure Guide (link also provided on website and 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

(a) (2 points) (coding) In order to apply tensor operations, we must ensure that the sentences in a given batch are of the same length. Thus, we must identify the longest sentence in a batch and pad others to be the same length. Implement the pad_sents function in utils.py, which shall produce these padded sentences.

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

(c) (4 points) (coding) Implement the __init__ function in nmt_model.py to initialize the necessary model embeddings (using the ModelEmbeddings class from model_embeddings.py) and layers (LSTM, projection, and dropout) for the NMT system.

(d) (8 points) (coding) Implement the encode function in nmt_model.py. This function converts the padded source sentences into the tensor $\mathbf{X}$, generates $\mathbf{h}_1^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$, and computes the initial state $\mathbf{h}_0^{\text{dec}}$ and initial cell $\mathbf{c}_0^{\text{dec}}$ for the Decoder. You can run a non-comprehensive sanity check by executing:

```
python sanity_check.py 1d
```

(e) (8 points) (coding) Implement the decode function in nmt_model.py. This function constructs $\bar{\mathbf{y}}$ and runs the step function over every timestep for the input. You can run a non-comprehensive sanity check by executing:

```
python sanity_check.py 1e
```

(f) (10 points) (coding) Implement the step function in nmt_model.py. This function applies the Decoder's LSTM cell for a single timestep, computing the encoding of the target subword $\mathbf{h}_t^{\text{dec}}$, the attention scores $\mathbf{e}_t$, attention distribution α_t, the attention output $\mathbf{a}_t$, and finally the combined output $\mathbf{o}_t$. You can run a non-comprehensive sanity check by executing:

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

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

Encoder Masks (enc_masks) generated by the 'generate_sent_masks()' helps fill the attention scalar vector (e_t) by $-\inf$ at padded-word locations in the encoded source sentence. By doing so, when we estimate attention-weights α_t using softmax, attention-weights corresponding to these padded-words would become $\exp(-\inf) = 0$.

The above process will negate the influence of 'pad' embeddings in estimating the final Attention output vector a_t. This is necessary since there are 'pad' entries in the source sequence that we do not want to be used in computing the attention weights. We only want real-word entries. Therefore, by applying enc_masks, the contribution of 'pad' words will effectively be filtered away in computing the attention weights α_t, since they will all become 0.

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[2]. TensorBoard provides tools for logging and visualizing training information from experiments. To open TensorBoard, run the following in your conda environment:

[2]https://pytorch.org/docs/stable/tensorboard.html

```
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 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.

Solution: BLEU Score $\approx$ 12.472

(i) (4 points) (written) In class, we learned about dot product attention, multiplicative attention, and additive attention. As a reminder, dot product attention is $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{h}_i$, multiplicative attention is $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{W} \mathbf{h}_i$, and additive attention is $\mathbf{e}_{t,i} = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t)$.

i. (2 points) Explain one advantage and one disadvantage of *dot product attention* compared to multiplicative attention.

Solution:

(Adv.) *dot product attention* is simple and faster to compute as it does not require any parameter compared to *multiplicative attention.*

(Disadv.) The *dot product* is a good measure of similarity, but it can not capture what parts of $\mathbf{s}_t$ to pay attention to by what parts of $\mathbf{h}_t$ for estimating the attention scores. Additionally, the encoder/decoder's hidden vector contains multiple info, e.g. past condition, next output and other information. We want to use only some part of these informations to calculate attention scores. This can easily be done with multiplicative attention using a projective $\mathbf{W}$ matrix.

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

Solution:

(Adv.) *Additive attention* has control over the number of parameters used for the attention. It can freely control the dimension of attention scores, making it versatile for many applications. It's really a separate neural network layer for estimating the attention score.

(Disadv.) It can require comparatively more parameters than *multiplicative attention* and be less efficient in estimating the attention scores.

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: Being a polysynthetic language, *Cherokee* has words formed of many morphemes, where the symbols directly represent a syllable, i.e. CV unit. Hence, the vocabulary size can be extremely large due to exponential combinations and permutations of those morphemes. In addition, the word length can also vary a lot. Therefore, it is beneficial to model Cherokee-to-English NMT at the morpheme-level or word-level.

(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 transliterated Cherokee words are easier to split into morphemes (subwords) because the original Cherokee characters may contain inseparable combinations of morphemes. For example, many words in Cherokee start with a common prefix "ts-", can easily be segregated into subwords, if their transliterated (Latinized) version is used. It would be easier for the tokenizer to manage vocabulary efficiently as there are fewer subwords (morphemes) now.

(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?

Answer: Recently, the limits of multilingual NMT have been pushed by training a very large network on 25+ billion sentence pairs, from 100+ languages to and from English. Big multilingual models are trained on many languages and are found to be generalizing well on new languages. This is possible because of the inherent inductive bias of transfer learning which helps to learn the linguistic similarities across linguistic families and thus able to acquire insights into any new language that falls within the such family.

(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:** *ᏄᏣᏁᎨᎾ ᎦᎪᏣᏔᎩᏍᎬ, ᎯᎠ ᏄᏍᎦ ᏚᏏᏔᏛ: ᏧᏓᎴᏅᏓ ᎦᎪᎡᏍᎦ ᏧᏏᏔᏛᏙᏗ ᎠᏍᏓ ᎧᎤᏂᏍᎩ.*
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:
1. The NMT model is not able to pick the pronouns correctly for the target sentence in the translation.
2. The most likely reason for the error might be caused by the final softmax output with vocabulary projection.
3. Adding more layers to the final projection layer could be a possible fix. Or, the NMT has not learned the target language grammar well enough. This would be solved by the combination of incorporating more data, running more epochs, adding more layers, and/or increasing the size of hidden layers.

ii. (2 points) **Source Translation**: *ᎤᏍᏗ ᎢᏍᎬᎢ, ᎦᏙᎲᎢ? ᎤᏓᏛᏛᏁᎢ ᎤᏍᏗ ᎠᏧᏣ.*
Reference Translation: *What's wrong little tree? the boy asked.*
NMT Translation: *The little little little little little tree? asked him.*
Solution:
1. The NMT model repeatedly predicts the same word multiple times in the translation.
2. The error might be caused by paying overly attention to the same part of the source sentence, leading to a similar probability distribution at each decoding step.
3. The concept of self-attention on the decoder might be useful to overcome the issue. Alternatively, as another solution, more recurrent layers to the encoder can be added to better encode the inter-word relationship present in the source sentence.

iii. (2 points) **Source Sentence:** *"ᎤᏓᏄᏬᏗ ᏂᎨᏒᎾ," ᎤᏛᏁ ᎨᎻ.*
Reference Translation: *" 'Humble,' " said Mr. Zuckerman*
NMT Translation: *"It's not a lot," said Mr. Zuckerman.*
Solution:
1. The NMT model got the meaning right but failed to manifest the desired word.
2. The error is likely caused by the model's low memory power or representation capability.
3. One solution could change the architecture, i.e. add more layers to the encoder/decoder, and apply self-attention. Another reason is that possibly the desired word or its representation was not in the training corpus. So adding more data to the training may resolve such errors.

(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: There are many examples where a similar sequence of words present in both the NMT output and the target training file. For example, *"they that were with him"* exists at line no. 28 and 334 in outputs/test_outputs.txt and also repeats 9 times in the target training file chr_en_data/train.en. This shows that the NMT model has memorised some sequences of words while learning a language model on the target training 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?

Solution:

Line no. 41 in outputs/test_outputs.txt

"And he said unto him, Friend, why do ye that thou art in the temple? And they were healed."

Line no. 41 in chr_en_data/test.en

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

In the above translation, we can see that the starting five words are similar; after that, test output diverges completely. It shows the inability of the decoder module to maintain the context after translating a few words.

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

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

Here, for each of the n-grams that appear in the candidate translation $\mathbf{c}$, we count the maximum number of times it appears in any one reference translation, capped by the number of times it appears in $\mathbf{c}$ (this is the numerator). We divide this by the number of n-grams in $\mathbf{c}$ (denominator).

Next, we compute the *brevity penalty* BP. Let $len(c)$ be the length of $\mathbf{c}$ and let $len(r)$ be the length of the reference translation that is closest to $len(c)$ (in the case of two equally-close reference translation lengths, choose $len(r)$ as the shorter one).

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

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

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

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

[3]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

i. (5 points) Please consider this example[4]:
Source Sentence **s**: **ᎠᏛ ᎾᏍᎩ ᎢᎦ-ᎦᏘᏍᏗᏍᎩ ᎤᎵᏏᎬ ᏚᏩᏌᎦᎢ ᎤᎵᏏᎩᏃ ᎥᏝ ᏱᏚᏓᏂᏩᎲᎢ**
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:

Step-1: Estimate the modified n-gram precision for each candidate c:

We will consider only **1-gram** and **2-gram** during the BLEU estimation as 3-gram and 4-gram are ignored.

1-grams for candidate and reference sentences:

1-grams	and	the	light	shines	in	darkness	can	not	comprehend
$\text{Count}_{\mathbf{c}_1}$	2	3	1	1	1	2	1	1	1
$\text{Count}_{\mathbf{r}_1}$	1	3	1	1	1	2	0	1	0
$\text{Count}_{\mathbf{r}_2}$	2	3	1	1	1	2	0	1	1

1-grams	the	light	shines	darkness	has	not	in	and	trials
$\text{Count}_{\mathbf{c}_2}$	4	1	1	2	1	1	1	1	1
$\text{Count}_{\mathbf{r}_1}$	3	1	1	2	1	1	1	1	0
$\text{Count}_{\mathbf{r}_2}$	3	1	1	2	0	1	1	2	0

2-grams for candidate and reference sentences:

2-grams	and the	the light	light shines	shines in	in the	the darkness	darkness and	darkness can	can not	not comprehend
$\text{Count}_{\mathbf{c}_1}$	2	1	1	1	1	2	1	1	1	1
$\text{Count}_{\mathbf{r}_1}$	1	1	1	1	1	2	1	0	0	0
$\text{Count}_{\mathbf{r}_2}$	2	1	1	1	1	2	1	0	0	1

p_1 and p_2 can not easily be computed for both $\mathbf{c}_1$ and $\mathbf{c}_2$:

$$p_{1,\mathbf{c}_1} = \frac{2+3+1+1+1+2+0+1+1}{2+3+1+1+1+2+1+1+1} = \frac{12}{13} \approx 0.9231$$

$$p_{1,\mathbf{c}_2} = \frac{3+1+1+2+1+1+1+1+0}{4+1+1+2+1+1+1+1+1} = \frac{11}{13} \approx 0.8462$$

[4]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.

2-grams	the light	light shines	shines the	the darkness	darkness has	has not	not in	in the	darkness and	and the	the trials
$\text{Count}_{\mathbf{c}_2}$	1	1	1	2	1	1	1	1	1	1	1
$\text{Count}_{\mathbf{r}_1}$	1	1	0	2	1	1	0	1	1	1	0
$\text{Count}_{\mathbf{r}_2}$	1	1	0	2	0	0	0	1	1	2	0

$$p_{2,\mathbf{c}_1} = \frac{2+1+1+1+1+2+1+0+0+1}{2+1+1+1+1+2+1+1+1+1} = \frac{10}{12} \approx 0.8333$$
$$p_{2,\mathbf{c}_2} = \frac{1+1+0+2+1+1+0+1+1+1+0}{1+1+1+2+1+1+1+1+1+1+1} = \frac{9}{12} \approx 0.7500$$

Step-2: Estimate the *brevity penalty* (BP) for each candidate c:
Compute the length of both candidates and reference sentences:

$$len(\mathbf{c}_1) = 13; \;\; len(\mathbf{c}_2) = 13$$

$$len(r_1) = 13; \;\; len(r_2) = 14$$

As $len(r_1)$ is closer to both $len(\mathbf{c}_1)$ and $len(\mathbf{c}_2)$, hence:

$$BP_{\mathbf{c}_1} = 1; \;\; BP_{\mathbf{c}_2} = 1$$

Step-3: Now, calculate the BLEU score of each candidate c:

$$BLEU_{\mathbf{c}_1} = BP_{\mathbf{c}_1} \times \exp\left(\lambda_1 \ln p_{1,\mathbf{c}_1} + \lambda_2 \ln p_{2,\mathbf{c}_1}\right) = \exp(0.5 \ln 0.9231 + 0.5 \ln 0.8333) \approx 0.8771$$
$$BLEU_{\mathbf{c}_2} = BP_{\mathbf{c}_2} \times \exp\left(\lambda_1 \ln p_{1,\mathbf{c}_2} + \lambda_2 \ln p_{2,\mathbf{c}_2}\right) = \exp(0.5 \ln 0.8462 + 0.5 \ln 0.7500) \approx 0.7966$$

Since $BLEU_{\mathbf{c}_1} > BLEU_{\mathbf{c}_2}$, candidate $\mathbf{c}_1$ is the better translation. This is indeed reflected in the translation also as it is more similar to the reference sentences.

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:
Step-1: Estimate the modified the n-gram precision for each candidate c:
Similar to the previous part, we will consider only **1-gram** and **2-gram** during the BLEU estimation as 3-gram and 4-gram are ignored. Additionally, $\mathbf{r}_2$ is also going to be ignored, as mentioned in the question.
1-grams for candidates and reference sentence:

1-grams	and	the	light	shines	in	darkness	can	not	comprehend
$\text{Count}_{\mathbf{c}_1}$	2	3	1	1	1	2	1	1	1
$\text{Count}_{\mathbf{r}_1}$	1	3	1	1	1	2	0	1	0

1-grams	the	light	shines	darkness	has	not	in	and	trials
$\text{Count}_{\mathbf{c}_2}$	4	1	1	2	1	1	1	1	1
$\text{Count}_{\mathbf{r}_1}$	3	1	1	2	1	1	1	1	0

2-grams for candidate and reference sentences:

2-grams	and the	the light	light shines	shines in	in the	the darkness	darkness and	darkness can	can not	not comprehend
$\text{Count}_{\mathbf{c}_1}$	2	1	1	1	1	2	1	1	1	1
$\text{Count}_{\mathbf{r}_1}$	1	1	1	1	1	2	1	0	0	0

2-grams	the light	light shines	shines the	the darkness	darkness has	has not	not in	in the	darkness and	and the	the trials
$\text{Count}_{\mathbf{c}_2}$	1	1	1	2	1	1	1	1	1	1	1
$\text{Count}_{\mathbf{r}_1}$	1	1	0	2	1	1	0	1	1	1	0

p_1 and p_2 can not easily be computed for both $\mathbf{c}_1$ and $\mathbf{c}_2$:

$$p_{1,\mathbf{c}_1} = \frac{1+3+1+1+1+2+0+1+0}{2+3+1+1+1+2+1+1+1} = \frac{10}{13} \approx 0.7692$$
$$p_{1,\mathbf{c}_2} = \frac{3+1+1+2+1+1+1+1+0}{4+1+1+2+1+1+1+1+1} = \frac{11}{13} \approx 0.8462$$
$$p_{2,\mathbf{c}_1} = \frac{1+1+1+1+1+2+1+0+0+0}{2+1+1+1+1+2+1+1+1+1} = \frac{8}{12} \approx 0.6667$$
$$p_{2,\mathbf{c}_2} = \frac{1+1+0+2+1+1+0+1+1+1+0}{1+1+1+2+1+1+1+1+1+1+1} = \frac{9}{12} \approx 0.7500$$

Step-2: Estimate the *brevity penalty* (BP) for each candidate c:
Compute the length of both candidates and reference sentences:

$$len(\mathbf{c}_1) = 13; \;\; len(\mathbf{c}_2) = 13$$
$$len(r_1) = 13; \;\; len(r_2) = 14$$

As $len(r_1)$ is closer to both $len(\mathbf{c}_1)$ and $len(\mathbf{c}_2)$, hence:

$$BP_{\mathbf{c}_1} = 1; \;\; BP_{\mathbf{c}_2} = 1$$

Step-3: Now, calculate the BLEU score of each candidate c:

$$BLEU_{\mathbf{c}_1} = BP_{\mathbf{c}_1} \times \exp\left(\lambda_1 \ln p_{1,\mathbf{c}_1} + \lambda_2 \ln p_{2,\mathbf{c}_1}\right) = \exp(0.5 \ln 0.7692 + 0.5 \ln 0.6667) \approx 0.7161$$
$$BLEU_{\mathbf{c}_2} = BP_{\mathbf{c}_2} \times \exp\left(\lambda_1 \ln p_{1,\mathbf{c}_2} + \lambda_2 \ln p_{2,\mathbf{c}_2}\right) = \exp(0.5 \ln 0.8462 + 0.5 \ln 0.7500) \approx 0.7966$$

Since $BLEU_{\mathbf{c}_1} < BLEU_{\mathbf{c}_2}$, candidate $\mathbf{c}_2$ is the better translation. The reason is that the n-gram overlapping between $\mathbf{c}_2$,$\mathbf{r}_1$ is more than the $\mathbf{c}_1$,$\mathbf{r}_1$. Some overlapping n-grams are not judged correctly, i.e. "**the darkness** has not in **the darkness**". Considering higher n-grams, i.e. 3-gram, 4-gram, could be helpful in solving this problem. Thus, the better score does not justify the 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 transitions versus a single reference translation. **Solution:**

Multiple references give a better chance of generalization in evaluating the candidate translations. It leads to obtaining a fair BLEU score. The chances are high that the single reference may be noisy or syntactically different and unable to provide relevant n-grams for estimating the BLEU score. Hence, the BLUE score will also be noisy.

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

Solution:

Advantages:

1. Based on a simple statistical method, it is **very fast**.
2. It is also **language-independent**. Same metric for every language.

Disadvantages:

1. It is unable to capture the **semantics or structure of the sentences**.
2. **It may not be cost-effective** as it requires large human resources for annotating and writing reference translations.

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.