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. That being said, 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. Spanish) 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.

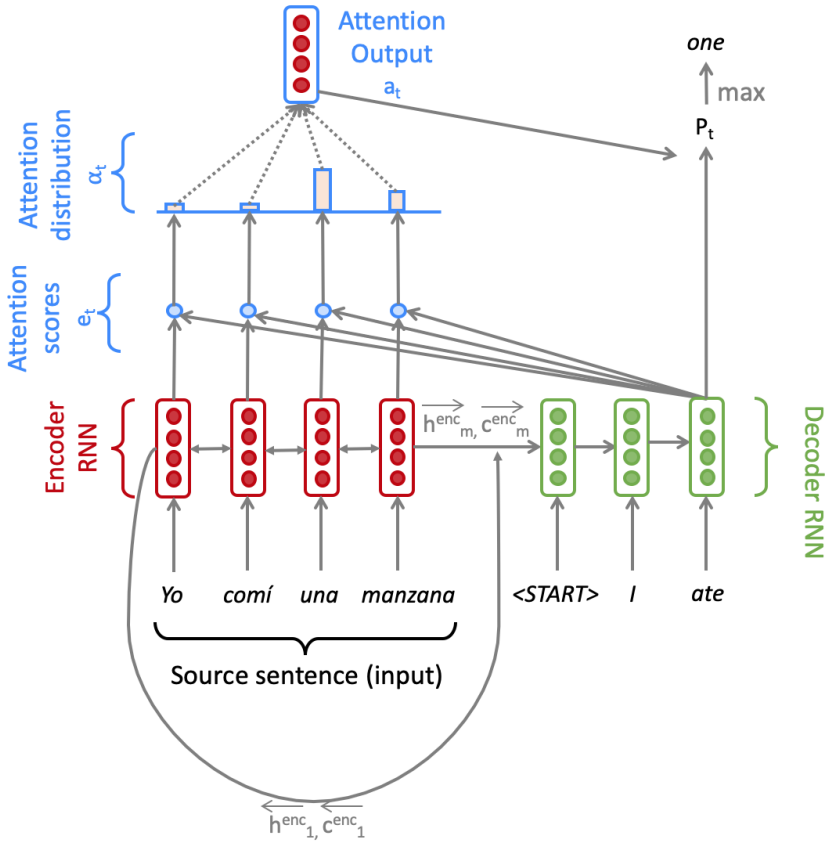

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Note that for readability, we do not picture the concatenation of the previous combined-output with the decoder input.

Given a sentence in the source language, we look up the word embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \ldots, \mathbf{x}_m \mid \mathbf{x}_i \in \mathbb{R}^{e \times 1}$, 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 matching sentence in the target language. On the t^{th} step, we look up the embedding for the t^{th} word, $\mathbf{y}_t \in \mathbb{R}^{e\times 1}$. 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 word (i.e. the start token) $\mathbf{o}_0$ is a zero-vector. We then feed $\overline{\mathbf{y}_t}$ as input to the Decoder LSTM.

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

(6)

We then use $\mathbf{h}_t^{\text{dec}}$ to compute multiplicative attention over $\mathbf{h}_0^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$:

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

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

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

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 $\mathbf{o}_t$.

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

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

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

Then, we produce a probability distribution $\mathbf{P}_t$ over target words 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 softmax cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the 1-hot vector of the target word at timestep t:

[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.

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

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 Spanish to English translation!

Follow the instructions in the CS224n Azure Guide (link also provided on website and Piazza) 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 **4 hours** to train the NMT system. We don't want you to accidentally use all your GPU time for the assignment, 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 locked and all code and data on the VM will be lost. Turn off your VM and request more Azure credits before your subscription runs out. See Piazza for instructions on requesting more credits if you are about to run out.

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.

(a) (2 points) 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) Implement the `__init__` function in `model_embeddings.py` to initialize the necessary source and target embeddings.

(c) (4 points) 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) 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) 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) 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 word $\mathbf{h}_t^{\text{dec}}$, 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.

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

```
sh run.sh 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
```

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 Piazza) 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
```

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
```

(i) (4 points) Once your model is done training (**this should take about 4 hours 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 21.

(j) (3 points) In class, we learned about dot product attention, multiplicative attention, and additive attention. Please provide one possible advantage and disadvantage of each attention mechanism, with respect to either of the other two attention mechanisms. 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 $\mathbf{e}_{t,i} = \mathbf{v}^T (\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t)$.

2. Analyzing NMT Systems (30 points)

(a) (12 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 Spanish source sentence, reference (i.e., 'gold') English translation, and NMT (i.e., 'model') English translation, please:

1. Identify the error in the NMT translation.
2. Provide a reason why the model may have made the error (either due to a specific linguistic construct or specific model limitations).
3. Describe one possible way we might alter the NMT system to fix the observed error.

Below are the translations that you should analyze as described above. Note that out-of-vocabulary words are underlined.

i. (2 points) **Source Sentence**: *Aquí otro de mis favoritos, "La noche estrellada".*
Reference Translation: *So another one of my favorites, "The Starry Night".*
NMT Translation: *Here's another favorite of my favorites, "The Starry Night".*

ii. (2 points) **Source Sentence**: *Ustedes saben que lo que yo hago es escribir para los niños, y, de hecho, probablemente soy el autor para niños, ms ledo en los EEUU.*
Reference Translation: *You know, what I do is write for children, and I'm probably America's most widely read children's author, in fact.*
NMT Translation: *You know what I do is write for children, and in fact, I'm probably the author for children, more reading in the U.S.*

iii. (2 points) **Source Sentence:** *Un amigo me hizo eso – Richard Bolingbroke.*
Reference Translation: *A friend of mine did that – Richard Bolingbroke.*
NMT Translation: *A friend of mine did that – Richard <unk>*

iv. (2 points) **Source Sentence:** *Solo tienes que dar vuelta a la manzana para verlo como una epifanía.*
Reference Translation: *You've just got to go around the block to see it as an epiphany.*
NMT Translation: *You just have to go back to the apple to see it as a epiphany.*

v. (2 points) **Source Sentence:** *Ella salvó mi vida al permitirme entrar al baño de la sala de profesores.*
Reference Translation: *She saved my life by letting me go to the bathroom in the teachers' lounge.*
NMT Translation: *She saved my life by letting me go to the bathroom in the women's room.*

vi. (2 points) **Source Sentence:** *Eso es más de 100,000 hectáreas.*
Reference Translation: *That's more than 250 thousand acres.*
NMT Translation: *That's over 100,000 acres.*

(b) (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`. Please identify **2 examples** of errors that your model produced.[2] The two examples you find should be different error types from one another and different error types than the examples provided in the previous question. For each example you should:

1. Write the source sentence in Spanish. The source sentences are in the `en_es_data/test.es`.
2. Write the reference English translation. The reference translations are in the `en_es_data/test.en`.
3. Write your NMT model's English translation. The model-translated sentences are in the `outputs/test_outputs.txt`.
4. Identify the error in the NMT translation.
5. Provide a reason why the model may have made the error (either due to a specific linguistic construct or specific model limitations).
6. Describe one possible way we might alter the NMT system to fix the observed error.

[2] An 'error' is not just a NMT translation that doesn't match the reference translation. There must be something wrong with the NMT translation, in your opinion.

(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.[3] 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 $n = 1, 2, 3, 4$:

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

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 c be the length of $\mathbf{c}$ and let r^* be the length of the reference translation that is closest to c (in the case of two equally-close reference translation lengths, choose r^* as the shorter one).

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

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

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

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights that sum to 1.

i. (5 points) Please consider this example:

Source Sentence $\mathbf{s}$: **el amor todo lo puede**
Reference Translation $\mathbf{r}_1$: *love can always find a way*
Reference Translation $\mathbf{r}_2$: *love makes anything possible*
NMT Translation $\mathbf{c}_1$: *the love can always do*
NMT Translation $\mathbf{c}_2$: *love can make anything possible*

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, c, r^* and BP).

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?

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?

[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

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.

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