XCS224N Assignment 4 Neural Machine Translation with RNNs

Due Thrusday, April 1 at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs224n-scpd.slack.com/
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit **all files indicated in the question** to the online student portal. For further details, see Writing Code and Running the Autograder below.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be foudn at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Writing Code and Running the Autograder

All your code should be entered into the `src/submission/` directory. When editing files in `src/submission/`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files outside the `src/submission/` directory.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and that the hidden evaluation tests will be able to execute.
- `hidden`: These unit tests are the evaluated elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests. These evaluative hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

In this assignment you will write code for a Neural Machine Translation (NMT) model using RNNs. 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 post a question in the Slack workspace or contact your Course Facilitator

1 Neural Machine Translation with RNNs

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.

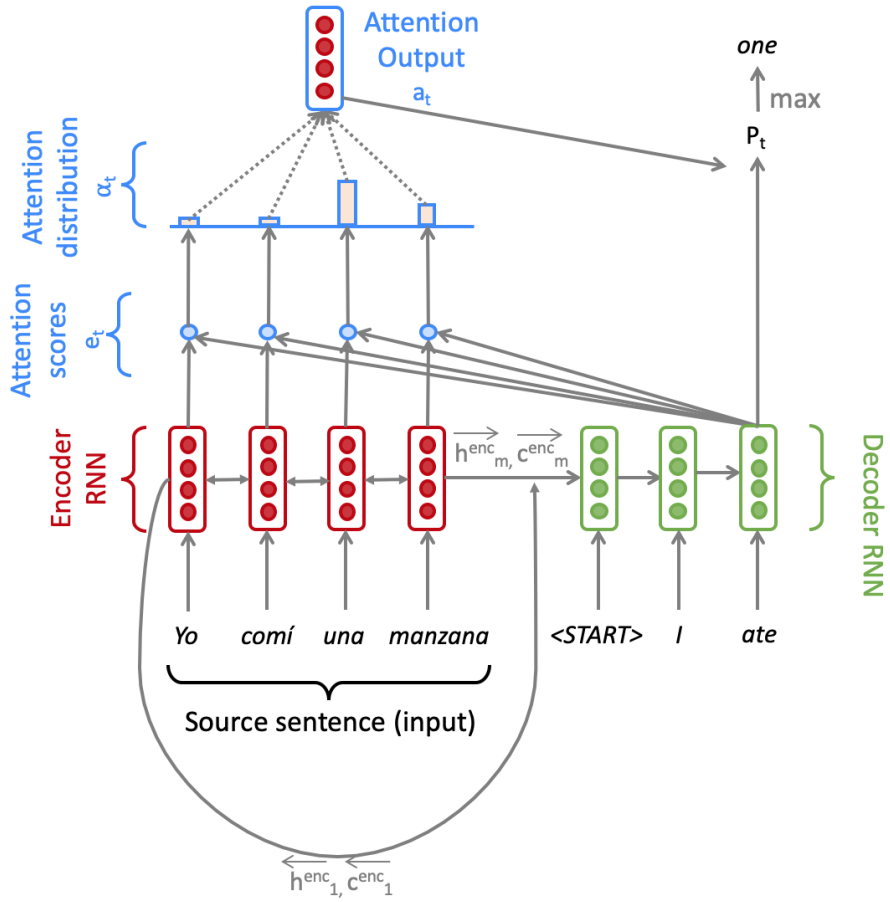

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Note that for readability, we do not picture the concatenation of the previous combined-output with the decoder input.

Model description (training procedure)

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 $\mathbf{h}^{\text{enc}}_i$ and cell states $\mathbf{c}^{\text{enc}}_i$:

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

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

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]

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

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

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.

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

$$\tag{6}$$

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:

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

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!

Setting up your Virtual Machine

Follow the instructions in the XCS224N Azure Guide in order to create your VM instance. 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.**

In order to run the model code on your VM, please run the following command to create the proper virtual environment (You did this at the beginning of the course on your local computer):

[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.

```
$ conda env create --file environment.yml
```

Next, you need to install GPU-specific dependencies on your VM. First, activate the `XCS224N` environment you just created. Then install the dependencies:

```
$ conda activate XCS224N
(XCS224N)$ conda install --file gpu_requirements.txt
```

For local development and testing, you can feel free to continue to using the same `XCS224N` environment you've used for all the assignments so far.

Implementation Assignment

(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 `submission/utils.py`, which shall produce these padded sentences.

(b) **[3 points (Coding)]** Implement the `__init__` function in `submission/model_embeddings.py` to initialize the necessary source and target embeddings.

(c) **[4 points (Coding)]** Implement the `__init__` function in `submission/nmt_model.py` to initialize the necessary layers (LSTM, projection, and dropout) for the NMT system.

(d) **[8 points (Coding)]** Implement the `encode` function in `submission/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.

(e) **[8 points (Coding)]** Implement the `decode` function in `submission/nmt_model.py`. This function constructs $\bar{\mathbf{y}}$ and runs the `step` function over every timestep for the input.

(f) **[10 points (Coding)]** Implement the `step` function in `submission/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 $\mathbf{o}_t$.

Now it's time to get things running! Execute the following to generate the necessary vocab file (you can do this on your local computer):

```
(XCS224N)$ 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:

```
(XCS224N)$ sh run.sh train_local
```

Once you have ensured that your code does not crash (i.e. let it run until `iter 10` or `iter 20`), power on your VM from the Azure Web Portal. Then read the *Practical Guide for Using the VM* section of the XCS224N Azure Guide for instructions on how to upload your code to the VM. Next, 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`.

```
(XCS224N)$ tmux new -s nmt
```

Once your VM is configured and you are in a tmux session, reactivate your `XCS224N` environment and execute:

```
$ conda activate XCS224N
(XCS224N)$ 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:

```
(XCS224N)$ tmux detach
```

You can return to your training model by ssh-ing back into the server and attaching to the tmux session by running:

```
(XCS224N)$ tmux a -t nmt
```

(g) **[3 points (Coding)]** Once your model is done training (**this should take about 4 hours on the VM**), execute the following command to test the model:

```
(XCS224N)$ sh run.sh test
```

To achieve credit for this portion of the assignment (i.e., training a large NMT model using a GPU), you must use your trained model to translate a Spanish test set into English. Your results will be compared to the correct translation and a BLEU score of 21 will be required to achieve full credit. To generate the gradescope test data, execute the following (local computer or VM):

```
(XCS224N)$ python evaluation_output.py
```

Deliverables

For this assignment, please submit all files within the `src/submission` subdirectory. This includes:

- `src/submission/__init__.py`
- `src/submission/model_embeddings.py`
- `src/submission/nmt_model.py`
- `src/submission/utils.py`
- `src/submission/gradescope_test_outputs_(soln).txt`

2 Quiz

This remainder of this homework is a series of multiple choice questions related to the word2vec algorithm.

How to submit: Even though these are not coding questions, you will submit your response to each question in the `src-quiz/submission.py` file. This file will act as your 'bubble sheet' for multiple choice questions in this course. A sample response might look like this:

```
def multiple_choice_1a():
  """
  # Return a python collection with the option(s) that you believe are correct
  # like this:
  # `return ['a']`
  # or
  # `return ['a', 'd']`
  response = []
  ### START CODE HERE ###
  ### END CODE HERE ###
  return response
```

If you believe that `a` and `b` are the correct responses to this question, you will type `response = ['a', 'b']` between the indicated lines like this:

```
def multiple_choice_1a():
  """
  # Return a python collection with the option(s) that you believe are correct
  # like this:
  # `return ['a']`
  # or
  # `return ['a', 'd']`
  response = []
  ### START CODE HERE ###
  response = ['a', 'b']
  ### END CODE HERE ###
  return response
```

How to verify your submission: You can run the student version of the autograder locally like all coding problem sets. In the case of this problem set, the helper tests will verify that your responses are within the set of possible choices for each question (e.g. the helper functions will flag if you forget to answer a question of if you respond with `['a', 'd']` when the choices are `['a', 'b', 'c']`.) See the front pages of this assignment for instructions to run the autograder.

1. **[2 points]**

   Note: For Question 1, reference the Python file `nmt_model.py` in your Assignment 4 coding files folder.

   The `generate_sent_masks()` function in `nmt_model.py` produces a tensor called `enc_masks`. It has a 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 231-320). Which among the following options do you think explains why the masks are required? **Select all that apply**.

   (a) The masks are used to set the attention scores $e_{t,i}$ to $-\text{inf}$ for all the positions i that correspond to `pad` tokens in the source sentence. This means the encoder hidden states h_i^{enc} that correspond to `pad` tokens have no effect on the attention output a_t.

   (b) The masks are used to set the attention scores $e_{t,i}$ to 0 for all the positions i that correspond to `pad` tokens in the source sentence. This means the encoder hidden states h_i^{enc} that correspond to `pad` tokens have no effect on the attention output a_t.

   (c) It is necessary to apply the masks to ensure we don't apply attention on the encoder hidden states that correspond to `pad` tokens.

2. **[1 point]**

   The example below contains a Spanish source sentence, reference English translation, NMT English translation, and error type. Choose the options that articulate a viable **reason(s)** for the observed error that you might explore further. ***Select all that apply.***

   Source Sentence: Aqui 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." **Error:** Repetition ('favorite of my favorites')

   (a) A possible reason is that the model attended to ***favoritos*** twice, thus producing both ***favorite*** and ***favorites***.

   (b) The reference translation says ***another one***, where the word ***one*** has no direct counterpart in the source sentence. These types of cases can be difficult for "sequence-to-sequence + attention" systems to produce.

   (c) Repetition can be a problem with the decoding algorithm (e.g. greedy decoding / beam search).

3. The examples below contain a Spanish source sentence, reference English translation, NMT English translation, and error type. For each example, analyze the error and choose the options that represent reasonable approaches to try to fix the error observed. ***Select all that apply.***

   3a. **[1 point]** **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¿* **Error:** Out of vocabulary words (¡unk¿)

   (a) We could add a neural copy/pointer mechanism to copy words from the source sentence (e.g., names).

   (b) We could initialize the decoder part with weights of pre-trained language models (trained on a large English corpus) instead of initializing them with random weights.

   (c) We could switch to a subword-based NMT model (e.g. one using characters, BPE or word-pieces); this would enable the decoder to produce new (out-of-vocabulary) words.

   3b. **[1 point]** **Source Sentence:** *Eso es mas de 100,000 hectareas.* **Reference Translation:** *That's more than 250 thousand acres.* **NMT Translation:** *That's over 100,000 acres.* **Error:** Incorrect numeric conversions (100,000 hectares = 250,000 acres, not 100,000 acres) Acre and Hectare numeric conversions.

   (a) We could collect more Spanish to English translation pairs that contain examples with metric to imperial conversions.

   (b) We could supply the NMT system with a knowledge base of units of measurement and their conversion rates, and train a system to convert from metric to imperial (imagine such a system already exists outside of NMT, as a post-processing step).

(c) We could implement a subword based NMT model (e.g. one using characters, BPE or word-pieces).

4. 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 Suppose we have a source sentence s, a set of k reference translations $r_1, ..., r_k$ and a candidate translation c. To compute the BLEU score of c, we first compute the *modified n-gram precision* p_n of b_c, for each of $n = 1, 2, 3, 4$:

$$p_n = \frac{\sum_{\text{ngram} \in c} \min\left(\max_{i=1,...,k} \text{Count}_{r_i}(\text{ngram}),\ \text{Count}_c(\text{ngram}) \right)}{\sum_{\text{ngram} \in c} \text{Count}_c(\text{ngram})}$$

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

Next, we compute the *brevity penalty* **BP**. Let c be the length of 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)

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

Lastly, the BLEU score for candidate c with respect to $r_1, ..., r_k$ is:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{4} \lambda_n \log p_n \right) \tag{15}$$

where λ_1, λ_2, λ_3, λ_4 are weights that sum to 1.

4a. **[1 point]** Consider this example:

Source Sentence s**:** el amor todo lo puede **Reference Translation** r_1**:** love can always find a way **Reference Translation** r_2**:** love makes anything possible **NMT Translation** c_1**:** the love can always do **NMT Translation** c_2**:** love can make anything possible

Compute the BLEU scores for c_1 and 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). Which of the two NMT translations (among c_1 and c_2) is considered the better translation according to the BLEU Score?

(a) c_1 has a higher BLEU score than c_2

(b) c_2 has a higher BLEU score than c_1

(c) Both translations, c_1 and c_2 are equally good

4b. **[1 point]**

The hard drive was corrupted and we lost Reference Translation r_2. Recompute BLEU scores for c_1 and c_2, this time with respect to r_1 only. Which of the two NMT translations has a higher BLEU Score?

(a) c_1 has a higher BLEU score than c_2

(b) c_2 has a higher BLEU score than c_1

(c) Both translations, c_1 and c_2 are equally good

Footnote for Q4.1 and Q4.2 *Why are multiple reference translations required?*

- *Often, there are many valid ways to translate a source sentence. This is particularly true for idiomatic phrases such as the above example. The BLEU metric is designed to accommodate this flexibility: an n-gram in c is rewarded if it appears in any one of the reference translations.*

- *If we have multiple reference translations, the BLEU metric will thus reward similarity to any of the several valid translations.*

4c. **[1 point]**

Which of the below statements are true regarding the BLEU scoring metric?

(a) BLEU can be calculated programmatically and is therefore fast to compute relative to human evaluation.

(b) BLEU is based on absolute n-gram matching, so it doesn't reward synonyms, paraphrases, or different inflections of the same word.

(c) BLEU uses multiple reference translations and hence does a better job than a human evaluator (assuming the evaluators are bilingual and do not use reference translations).

5. In the lectures you learned about dot product attention, multiplicative attention, and additive attention. As a reminder, dot product attention is $e_{t,i} = s_t h_i^\intercal$, multiplicative attention is $e_{t,i} = s_t^\intercal W h_i$, and additive attention is $e_{t,i} = v^\intercal \tanh(W_1 h_i + W_2 s_t)$ (v, W, W_1 and W_2 are weights to be learned).

6. **[1 point]**

Choose all options that accurately describe Dot Product attention:

(a) Encoder and decoder hidden states can be of different dimensions.

(b) Requires additional weights in the model.

(c) More computationally efficient compared to multiplicative and additive attention mechanisms.

7. **[1 point]**

Choose all options that accurately describe Additive attention:

(a) Encoder and decoder hidden states can be of different dimensions.

(b) Requires additional weights in the model.

(c) More computationally efficient compared to multiplicative attention mechanisms.

8. **[1 point]**

Choose all options that accurately describe Multiplicative attention:

(a) Encoder and decoder hidden states can be of different dimensions.

(b) Requires additional weights in the model.

(c) More computationally efficient compared to additive attention mechanisms.

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

THERE IS NO WRITTEN SUBMISSION FOR THIS ASSIGNMENT.

YOU ARE NOT REQUIRED TO SUBMIT ANYTHING.