CS 224n: Assignment #4

1. Neural Machine Translation with RNNs (45 points)

(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 1's in positions corresponding to 'pad' tokens in the input, and 0's for non-pad tokens. Look at how the masks are used during the attention computation in the `step()` function.

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.

💡 **Answer:**

- Padding tokens are an artefact of processing data in batches which is done to speed up training. Using the `pad_sents()` function in `utils.py`, we pad the input sentences to the maximum sentence length with a token that contains no information (i.e., the padding token). Thus, a mask is used to mark out the locations of these padding tokens. The `step()` function sets the padded locations in the vector e_t to $-\infty$ (the smallest possible float) and this apportions zero probability weight to these tokens in the attention distribution, $\alpha_{t,i} = 0$ for all positions i that correspond to a 'pad' token.
    - Since the value of padding tokens is set to $-\infty$ pre-$\mathrm{softmax}$, their proabability is 0 post-$\mathrm{softmax}$ because $e^{-\infty} = 0$.
- This means that the encoder hidden state h_i^{enc} that corresponds to a 'pad' token will have no effect on the attention output, a_t. In other words, we are simply asking the model to ignore the 'pad' tokens.
- It is necessary to apply the masks in this way because we do not want the model to put any attention on the encoder hidden states that correspond to 'pad' tokens. These 'padding' hidden states were computed because we have to pad the batch of source sentences up to maximum sentence length, m, as the model expects a fixed sized input. However, the padding tokens are meaningless and thus should not affect our model. Applying the masks in this manner thus avoids skewing the attention distribution, α_t due to padding tokens.

(h) (3 points) 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.

💡 **Answer:**

$\text{BLEU Score} \approx 12.251$

(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.

💡 **Answer:**

Advantage(s):

- Dot product attention has no learnable parameters/weights compared to multiplicative attention. Therefore, it is faster to compute as it needs less memory and computationally less expensive.

Disadvantage(s):

- Since the dot production attention does not have a $\mathbf{W}$ matrix, hence it is less expressive. It can't transform the query or the value vectors; can't weigh the importance of different parts of the vectors (i.e. representational power is lower due to the lack of parameters).

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

💡 **Answer:**

Advantage(s):

- Additive attention can be viewed as a small feed forward neural network whose input is the concatenation $[\mathbf{h}_i, \mathbf{s}_t]$, followed by one linear layer with a $\tanh$non-lienarity, then another linear layer to get a scalar output. As such, it has the advantage of generality — that is, provided the internal attention dimension is large enough, it is a universal approximator.
- Furthermore, both the source and target hidden vector states get their own learned transformations $\mathbf{W}_1$ and $\mathbf{W}_2$, respectively. This mapping enables the decoder and encoder more degrees of freedom and thus flexibility in developing independent embedding spaces. This effectively allows for an affine non-linear mapping between the encoder and decoder space.

Disadvantage(s):

- Additive attention requires more paramters (i.e., $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{v}$) and can be computationally less efficient to compute than multiplicative attention. $\tanh$ is also computationally more intensive.
- Additive attention requires that you choose the attention dimension, which is another hyperparameter of the network. This could be an advantage (something
to tune to get better performance) or a disadvantage (more complex model).

2. Analyzing NMT Systems (30 points)

(a) (2 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.)

💡 **Answer:**

- Cherokee is a polysynthetic language, which means its words are composed of many morphemes (sub-word components that may have independent meanings, but which can't stand alone).
- Polysynthetic words have high morpheme-to-word ratio, meaning that single word can be composed of a bunch of morpheme. Sometimes entire sentences are composed of just a single word modified by a collection of morpheme.
- Given so much of the meaning in a single word is contained at the subword morpheme-level, we need to use a subword model to best capture that meaning — word-level models fail to capture that complexity.

(b) (2 points) Character-level and subword embeddings are often smaller than whole word embeddings. In 1-2 sentences, explain one reason why this might be the case.

💡 **Answer:**

The embedding sizes for characters and subwords are smaller than whole words because:

1. A typical character/subword vocabulary is several orders of magnitude smaller than a word vocabulary (consider the English language for example). In other words, the number of unique characters is much smaller than the number of unique words (i.e., number of unique combinations of those characters).
2. Characters or subwords do not usually encode as much information as an entire word (they usually contain more lexical and less semantic information).
3. Additionally, since individual characters/subwords typically have much less meaning compared to individual words, character dependencies turn out to be not as long-ranged compared to words, thus requiring smaller embeddings.

(c) (2 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. How does multilingual training help in improving NMT performance with low-resource languages?

💡 **Answer:**

- NMT systems are limited in translating low-resource languages owing to the challenge of having significant amounts of supervised training data to learn useful mappings between languages. Multilingual NMT tackles these challenges associated with low-resourced language translation.
- The underpinning principle of multilingual NMT is to force the creation of hidden representations of words in a shared semantic space for linguistically similar languages. This enables models to be better at generalization, and capable of capturing the representational similarity across a large body of languages.
- This is a classic example of transfer learning — the model takes insights gained by training on one language and applies it to the translation of other languages.

(d) (6 points) Here we present three examples of errors we found in the outputs of our NMT model (which is the same as the one you just trained). The errors are underlined in the NMT translation sentence. For each example of a source sentence, reference (i.e., 'gold') English translation, and NMT (i.e., 'model') English translation, please:

1. Provide possible reason(s) why the model may have made the error (either due to a specific
   linguistic construct or a specific model limitation).
2. 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 resources like https://www.cherokeedictionary.net/ to look up words.

i. (2 points) **Source Translation:** *Yona utsesdo ustiyegv anitsilvsgi digvtanv uwoduisdei.*
Reference Translation: *Fern had a crown of daisies in her hair.*
NMT Translation: *Fern had her hair with her hair.*

💡 **Answer:**

1. *Unrealized concept (crown of daisies)*

Possible reason(s):

- The model does not have a good understanding of the term *a crown of daisies*.

Potential fix(es):

- The model needs to be trained on data that conveys the meaning of *a crown of daisies*.

2. *Repetition (hair)*

Possible reason(s):

- The model paid more attention to the word *hair*, thus predicting it twice in the translation. Repetition is a common problem in the output of RNN decoders. Repetition can also be a problem with the decoding algorithm (e.g. greedy decoding/beam search).

Potential fix(es):

- Investigate the attention distribution to identify possible cases like the one above.
- Introduce an explicit penalty for word repetitions (either during training or decoding) or introduce a coverage mechanism which discourages word repetitions.
- Try a different decoding algorithm if greedy decoding or beam search is the root of the problem.

ii. (2 points) **Source Sentence:** *Ulihelisdi nigalisda.*
Reference Translation: *She is very excited.*
NMT Translation: It's *joy.*

Answer:

1. *Incomplete/Incoherent sentencing/phrasing*

Possible reason(s):

- The model is unable to infer that the subject in question is a female, which led it to predict or assign an incorrect gender token.
- The model also may not have been able to fully learn how grammar is structured in Cherokee and English which is very different between the two languages.
- One reason for this error could be the lack of context which NMT systems utilize during translation. If the model saw more of the preceding context, perhaps it would have been able to predict that the correct pronoun should be *she* (and not *it*). Therefore, the model is limited in that it is unable to attend over a wide range of dependencies.

Potential fix(es):

- Adding world knowledge to NMT systems is an ongoing research area; there are
  efforts to integrate NMT systems with knowledge bases. We could try training and testing our NMT systems on whole documents rather than single sentences to solve the context issue. To enable the model to attend to a wide range of dependencies, increase the model's complexity by say, increasing the number of hidden units or the depth of the model.

iii. (2 points) **Source Sentence:** *Tsesdi hana yitsadawoesdi usdi atsadi!*
Reference Translation: *Don't swim there, Littlefish!*
NMT Translation: *Don't know how a small fish!*

💡 **Answer:**

1. *Mistranslation of a proper name*

Possible reason(s):

- The name *Littlefish* was literally translated to a *small fish*. The model tracks the subwords and whole words based on the frequency of occurence. The word “Littlefish” might not have been common enough in the vocabularly. It might not even be present in the vocabulary of the training data at all which could lead to out-of-vocabulary error (OOV).
- The word “Littlefish” could also have been broken down the name into smaller pieces/subword components which are more common in the vocabulary like “little” and “fish”.

Potential fix(es):

- Either more training data where the proper name were used or learn to "copy" from source text. The source word can be identified using the argmax of the attention weights vector.

(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?

💡 **Answer:**

Using output from `outputs/test_outputs_v2.txt`.

NMT Translation: Line 220: “angel said unto him”
Training Sample: Line 4636: “angel said unto him”
Reason(s):

- It seems like the MT system has learned to memorized some fixed phrases/combinations. In this case, that would be the phrase “said unto him” which appeared 196 times in the 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?

💡 Answer:

Using output from `outputs/test_outputs_v1.txt`.

NMT Translation: Line 675: "He climbed up into the straw, in the manure pile."
Reference: Line 675: "So he pushed the straw to one side and stretched out in the manure."
Reason(s):

- The model struggles at longer range dependencies. The model is conditioned on its own predictions during testing, unlike in training it always sees the ground truth, so errors could accumulate to further distort longer predictions.
- The model needs to be trained on more data for it to understand the concept of *straw* and the fact it can mean more than one things (i.e., the straw you put in your drink or the straw as in animal feed). There are 47 *straw* words in the training data where most of them mean the latter. It is apparent this is not enough to help the NMT better understand the context of *straw*.

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

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

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).

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

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

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

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 from Spanish:

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 $p_1, p_2, len(c), len(r)$ 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?

💡 **Answer:**

For $\mathbf{c}_1$,

Unigram	**Numerator Term**
the	0
love	1
can	1
always	1
do	0

Bigram	**Numerator Term**
the love	0
love can	1
can always	1
always do	0

$$p_1 = \frac{3}{5} = 0.6$$

$$p_2 = \frac{2}{4} = 0.5$$

Since `len(` $\mathbf{r}_1$ `)` $= 6$ and `len(` $\mathbf{r}_2$ `)` $= 4$, we choose `len(` $\mathbf{r}_2$ `)` (i.e. the shorter one) because 6 and 4 are equally close to 5 which is the length of the candidate translation, $\mathbf{c}_1$.

$$BP = 1 \text{ since } \text{len}(\mathbf{c}_1) > \text{len}(\mathbf{r}_2)$$

Therefore the BLEU score for $\mathbf{c}_1$ is,

$$\text{BLEU}_{\mathbf{c}_1} = 1 \times \exp(0.5 \log(0.6) + 0.5 \log(0.5)) = \boxed{0.5477}$$

For $\mathbf{c}_2$,

Unigram	**Numerator Term**
love	1
can	1
make	0

Bigram	**Numerator Term**
love can	1
can make	0
make anything	1

Unigram	**Numerator Term**
anything	1
possible	1

Bigram	**Numerator Term**
anything possible	1

$$p_1 = \frac{4}{5} = 0.8$$

$$p_2 = \frac{3}{4} = 0.75$$

Since `len(` $\mathbf{r}_1$ `)` $= 6$ and `len(` $\mathbf{r}_2$ `)` $= 4$, we choose `len(` $\mathbf{r}_2$ `)` (i.e. the shorter one) because 6 and 4 are equally close to 5 which is the length of the candidate translation, $\mathbf{c}_2$.

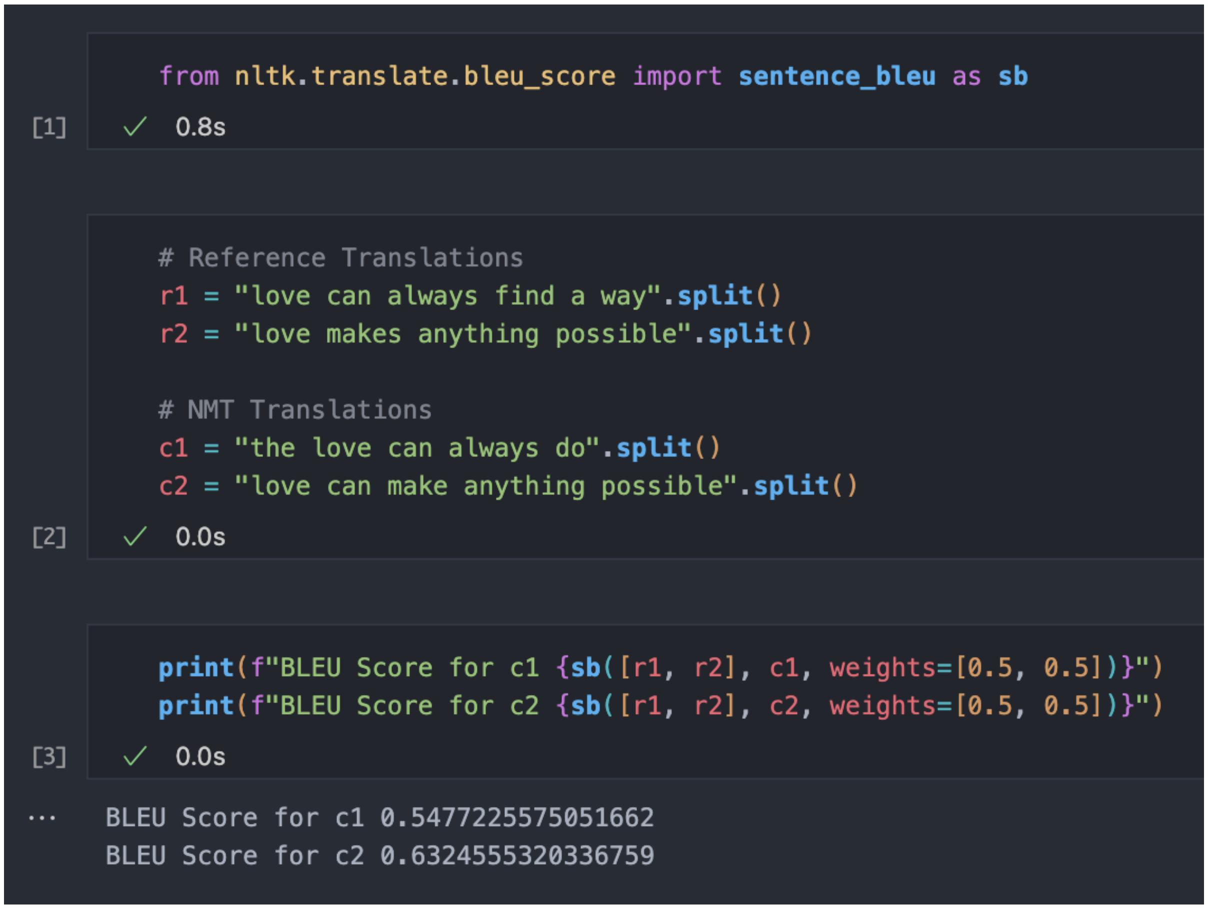
$$BP = 1 \text{ since } \mathrm{len}(\mathbf{c}_2) > \mathrm{len}(\mathbf{r}_2)$$

Therefore the BLEU score for $\mathbf{c}_2$ is,

$$\mathrm{BLEU}_{\mathbf{c}_2} = 1 \times \exp(0.5 \log(0.8) + 0.5 \log(0.75)) = \boxed{0.6324}$$

According to these BLEU scores, NMT translation $\mathbf{c}_2$ is the better one. $\mathbf{c}_2$ is indeed
the better translation - it has the correct meaning and makes intuitive sense, whereas $\mathbf{c}_1$ translates the Spanish phrase too literally, leading to unnatural and nonsensical English.

Note that the numbers in both this part and the next can be validated using `nltk` like so:



```
[1]  from nltk.translate.bleu_score import sentence_bleu as sb
     ✓ 0.8s
```

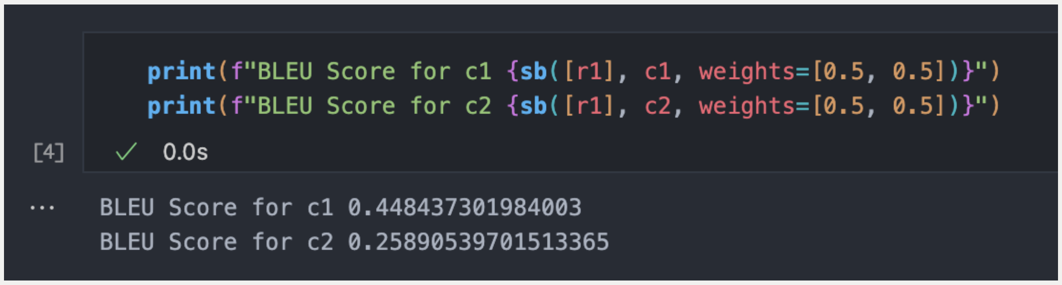
```
     # Reference Translations
     r1 = "love can always find a way".split()
     r2 = "love makes anything possible".split()

     # NMT Translations
     c1 = "the love can always do".split()
     c2 = "love can make anything possible".split()
[2]  ✓ 0.0s
```

```
     print(f"BLEU Score for c1 {sb([r1, r2], c1, weights=[0.5, 0.5])}")
     print(f"BLEU Score for c2 {sb([r1, r2], c2, weights=[0.5, 0.5])}")
[3]  ✓ 0.0s
...  BLEU Score for c1 0.5477225575051662
     BLEU Score for c2 0.6324555320336759
```

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?

💡 **Answer:**



```
print(f"BLEU Score for c1 {sb([r1], c1, weights=[0.5, 0.5])}")
print(f"BLEU Score for c2 {sb([r1], c2, weights=[0.5, 0.5])}")
[4]   ✓  0.0s
...   BLEU Score for c1 0.448437301984003
      BLEU Score for c2 0.25890539701513365
```

According to these BLEU scores, NMT translation $\mathbf{c}_1$ is the better translation. I disagree because $\mathbf{c}_1$ doesn't make intuitive sense.

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.

💡 **Answer:**

- The task of sentence translation is somewhat subjective especially in certain scenarios and usually doesn't have a single correct answer. In other words, often there are many valid ways to translate a source sentence. This is particularly true for idiomatic phrases such as the previous example. The BLEU metric is designed to accommodate this flexibility: a n-gram in $\mathbf{c}$ is rewarded if it appears in any one of the reference translations.
- With only a single reference translation, the BLEU metric only recognizes similarity to that particular translation potentially penalizing other valid candidate translations. As a result, even good translations might receive a low BLEU score due to little n-gram overlaps. By providing multiple independent translations, the NMT system has a chance to learn patterns from multiple different answers for the same input sentence. With more reference sentences, the target space increases with more n-grams available for the model to generate and receive a good BLEU score, as was demonstrated in the above question. This leads to the BLEU metric rewarding similarity to any of the several valid translations.

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

💡 **Answer:**

Advantages:

- Single quantitative score that aims to remove subjectiveness from the evaluation process. Owing to its objectiveness (unlike human evaluation, which is hard to define and varies depending on the human judge), researchers can reproduce each others' BLEU results and use BLEU to compare different systems.
- Simple and easy implementation, which implies that it is scalable and it's possible to calculate it relatively cheaply for large datasets.
- Fully automated evaluation, faster than humans. Human evaluators would have to understand both the source and target languages.

Disadvantages:

- BLEU requires a reference translation (whereas human evaluation doesn't, assuming the human judges are bilingual), and optimally requires multiple reference translations.
- BLEU only measures n-gram overlaps, not the quality of the sentences produced by evaluating sentence/corpus semantics, grammar, etc. It doesn't reward synonyms, paraphrases, or different inflections of the same word (e.g., make and makes).
- Human translation can utilize common sense and general world understanding to render a good translation. Since BLEU lacks this understanding, it can fail to fully capture the true notion of a good translation in some cases. It's an extremely difficult task for an NLP system to infer how the world functions using only individual sentences without broader context, to know idioms, and recognizing what 'sounds good' and what doesn't, etc. In these scenarios, the human evaluation of translations is vital and can't be replaced with a BLEU score.