

# CS 224n: Assignment #4

Anthony Weng

**Due:** Thursday, Feb. 9th @ 4:30 PM PST

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 **2 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. Mandarin Chinese) 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.

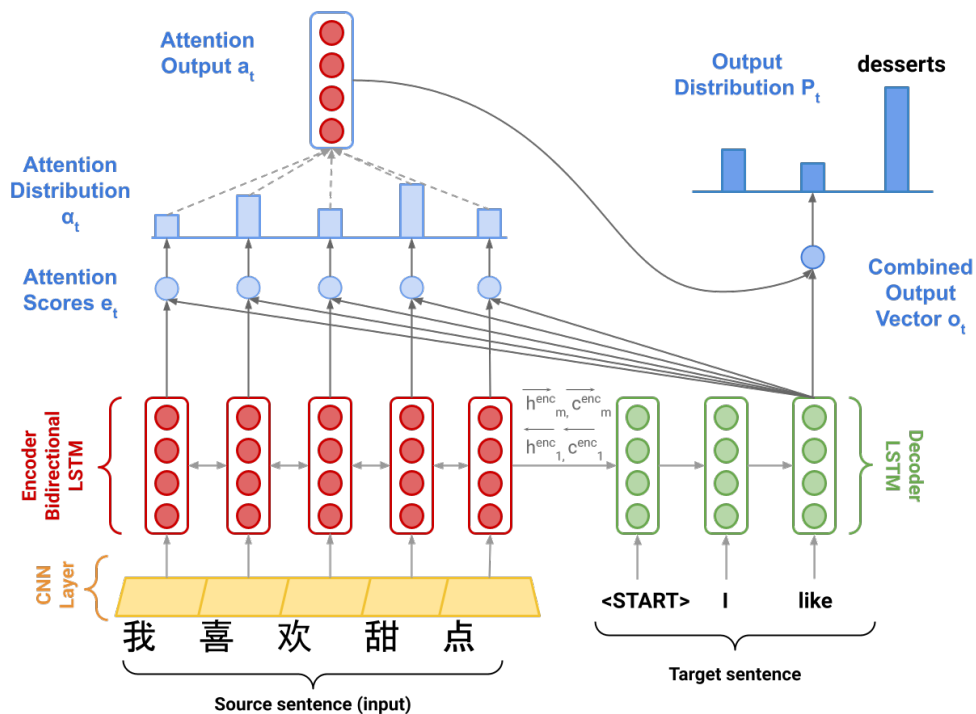


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states  $h_i^{\text{enc}}$  and cell states  $c_i^{\text{enc}}$  are defined on the next page.

## Model description (training procedure)

Given a sentence in the source language, we look up the character or word 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 then feed the embeddings to a **convolutional layer**<sup>1</sup> while maintaining their shapes. We feed the convolutional layer outputs 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  $\mathbf{c}_i^{\text{enc}}$ :

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

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

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.<sup>2</sup>

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

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

With the decoder initialized, we must now feed it a target sentence. On the  $t^{\text{th}}$  step, we look up the embedding for the  $t^{\text{th}}$  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.

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

$$(6)$$

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

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

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

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

$\mathbf{e}_{t,i}$  is a scalar, the  $i$ th element of  $\mathbf{e}_t \in \mathbb{R}^{m \times 1}$ , computed using the hidden state of the decoder at the  $t$ th 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  $i$ th 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*  $\mathbf{o}_t$ .

<sup>1</sup>Checkout <https://cs231n.github.io/convolutional-networks> for an in-depth description for convolutional layers if you are not familiar

<sup>2</sup>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.

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

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

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

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

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

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  $t$ :

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

Here,  $\theta$  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 Mandarin Chinese 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 **1.5 to 2 hours** 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.

1(a). **COMPLETED** - see code submission.

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

1(b). **COMPLETED** - see code submission.

- (c) (4 points) (coding) Implement the `__init__` function in `nmt_model.py` to initialize the necessary model layers (LSTM, CNN, projection, and dropout) for the NMT system.

1(c). **COMPLETED** - see code submission.

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

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

1(d). **COMPLETED** - see code submission.

- (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
```

1(e). **COMPLETED** - see code submission.

- (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  $\alpha_t$ , 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
```

1(f). **COMPLETED** - see code submission.

- (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 311-312).

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

**Solution:**

The masks ensure the attention computation does not assign any positive probability to the attention distribution in sentence positions corresponding to padding tokens. The masks achieve this effect as any position where the `enc_masks` tensor is equal to 1 (i.e., the position of a given sentence’s padding token) is assigned an attention value of  $-\infty$  in the attention distribution  $\mathbf{e}_t$ . Since the attention distribution computation employs a softmax, all padding tokens will be assigned an attention probability of 0, ensuring that the padding tokens are safely ignored and do not influence downstream calculation of the combined-output vector  $\mathbf{o}_t$  and consequent estimation of a target subword.

It is necessary to use masks in this way because all sentences in a batch of data must have the same length to enable efficient batched computations (necessitating that different sentences in a batch are padded), but padding tokens do not represent actual informative input text so we would like to ignore them in the attention computations (necessitating the use of the described masks).

Now it's time to get things running! 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
```

For a faster way to debug by training on less data, you can run the following instead:

```
sh run.sh train_debug
(Windows) run.bat train_debug
```

To help with monitoring and debugging, the starter code uses tensorboard to log loss and perplexity during training using TensorBoard<sup>3</sup>. TensorBoard provides tools for logging and visualizing training information from experiments. To open TensorBoard, run the following in your conda environment:

```
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
```

---

<sup>3</sup><https://pytorch.org/docs/stable/tensorboard.html>

- (h) (3 points) (written) Once your model is done training (**this should take under 2 hours on the VM**), execute the following command to test the model:

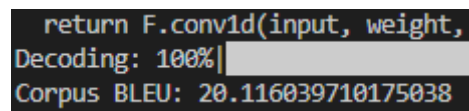
```
sh run.sh test
(Windows) run.bat test
```

Please report the model's corpus BLEU Score. It should be larger than 18.

**Solution:**

My model's corpus BLEU score was  $\approx 20.116$

*Screenshot:*



```
return F.conv1d(input, weight,
Decoding: 100%|██████████
Corpus BLEU: 20.116039710175038
```

- (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  $\mathbf{e}_{t,i} = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t)$ .
- (2 points) Explain one advantage and one disadvantage of *dot product attention* compared to multiplicative attention.
  - (2 points) Explain one advantage and one disadvantage of *additive attention* compared to multiplicative attention.

**Solution:**

- i. One *advantage* of using dot product attention compared to multiplicative attention is the decreased computational demands of dot product attention; using dot product attention eliminates the need to train the  $\mathbf{W}$  matrix and the associated matrix multiplication (i.e.  $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{W} \mathbf{h}_i$ ) in each attention computation, thereby reducing the necessary computations at both training and evaluation time.

One *disadvantage* of using dot product attention compared to multiplicative attention is the enforced equality and resulting position-invariance in the attention score computations. That is, in multiplicative attention, the presence and training of the weight matrix  $\mathbf{W}$  enables a model using multiplicative attention to generate an attention distribution which consistently emphasizes certain sentence positions  $i$ . This may be useful as, for a given test example  $\mathbf{s}_t^T \mathbf{h}_i$  may be low, but the model may still benefit from applying attention to this position  $i$  as such positions have been previously learned to provide outsized utility in the downstream translation task. As such, use of multiplicative attention can provide a more human-like model of attention where learned precedent regarding useful sentence positions for attention can be incorporated into the attention distribution generation, meaning the model can emphasize sentence positions it expects and has learned to be useful rather than just those the surrounding text and associated encoder/decoder hidden states dictate to be (as use of dot product attention would compel).

- ii. One *advantage* of using additive attention compared to multiplicative attention is the increased selective capacity and associated learning ability of the translation model to determine what information from the encoder and decoder hidden states is most relevant to a given translation timestep. This capacity arises from the decoupling of the weight matrix  $\mathbf{W}$  in multiplicative attention into encoder/decoder-specific weight matrices  $\mathbf{W}_1, \mathbf{W}_2$  in additive attention. Allowing for encoder/decoder-specific weight matrices enables models using additive attention to learn to extract the most information possible from *each type of hidden state separately* before combining this information (i.e., through  $\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t$  and the calculations upon this quantity) to best inform the attention distribution. In this way, additive attention further approximates a human-like model of attention where information from both the source and target text can be separately considered before being intelligently combined to determine where one's attention should be directed to inform the next discrete translation (i.e., one translation timestep in an NMT).

One *disadvantage* of using additive attention compared to multiplicative attention is the highly increased computation demand—additive attention essentially constructs a feed-forward neural net layer for the attention computation, and training and repeatedly evaluating this sub-network demands a much higher computational load compared to the relatively simple and highly parallel-izable matrix multiplications involved in multiplicative attention.



## 2. Analyzing NMT Systems (25 points)

- (a) (3 points) Look at the `src.vocab` file for some examples of phrases and words in the source language vocabulary. When encoding an input Mandarin Chinese sequence into “pieces” in the vocabulary, the tokenizer maps the sequence to a series of vocabulary items, each consisting of one or more characters (thanks to the `sentencepiece` tokenizer, we can perform this segmentation even when the original text has no white space). Given this information, how could adding a 1D Convolutional layer after the embedding layer and before passing the embeddings into the bidirectional encoder help our NMT system? **Hint:** each Mandarin Chinese character is either an entire word or a morpheme in a word. Look up the meanings of 电, 脑, and 电脑 separately for an example. The characters 电 (electricity) and 脑 (brain) when combined into the phrase 电脑 mean computer.

### Solution:

Adding a 1D convolutional layer after the embedding layer and prior to the bidirectional encoder could help our NMT system by **convolving and amplifying contiguous word embeddings whose source characters combine to form a multi-character word, improving the NMT’s ability to process and translate such words as a single, discrete unit.**

For instance, consider the multi-character word provided in the hint, 电脑. When tokenizing an input sequence containing 电脑, the `sentencepiece` tokenizer may wrongfully process the 电 and 脑 characters separately, resulting in our embedding layer containing the word embeddings for “electricity” and “brain” rather than the desired single word embedding for “computer.” Left alone, the NMT’s bidirectional encoder may process these embeddings as distinct units, leading to an eventual sub-optimal translation of “electricity brain” instead of “computer.”

However, inserting the 1D convolutional layer provides the effect of convolving these embeddings such that the separate embeddings of each character in multi-character words are combined to form an embedding closer to that of the multi-character word they constitute.

Because the strength of the convolution (and here, the convolved embedding’s difference from the constituting **kernel size** word embeddings) is a function of the cross-correlation operator, the embeddings of source characters which form multi-character words will experience a stronger convolving effect (since, as parts of the same multi-character word, their embeddings will be more similar and their cross-correlation of larger magnitude). This stronger convolving effect will result in a convolved embedding which is more like that of the multi-character word the **kernel size** characters constitute, which ultimately implies an embedding which is more reflective of the source text will be fed to the NMT’s bidirectional encoder, improving translation accuracy downstream (permitting that the **kernel size** is chosen and/or tuned properly).

(b) (8 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 Mandarin to answer these questions. You just need to know English! If, however, you would like some additional color on the source sentences, feel free to use a resource like [https://www.archchinese.com/chinese\\_english\\_dictionary.html](https://www.archchinese.com/chinese_english_dictionary.html) to look up words. Feel free to search the training data file to have a better sense of how often certain characters occur.

- i. (2 points) **Source Sentence:** 贼人其后被警方拘捕及被判处盗窃罪名成立。  
**Reference Translation:** *the culprits were subsequently arrested and convicted.*  
**NMT Translation:** *the culprit was subsequently arrested and sentenced to theft.*

**Solution:**

- **Error:** Subject plurality error - the NMT translates the source sentence as having a singular subject (“culprit”) while the reference translation notes a plural subject (“culprits”).
- **Possible reason for model error:** In Mandarin, the plurality of subjects is often not indicated by the character(s) used to denote said subject but rather by the context surrounding it. For instance, in the source sentence, 贼人 can mean both “thief” or “thieves” (or “culprit” or “culprits” as translated here).
- **Potential fix:** Decoding longer inputs (e.g., inputs corresponding to multiple sentences instead of a single sentence) and altering the attention mechanism to compute an attention distribution over this entire longer input could allow for attention being directed toward context clue-type words in earlier parts of the input which indicate what plurality the word to be decoded should be.

- ii. (2 points) **Source Sentence:** 几乎已经没有地方容纳这些人, 资源已经用尽。  
**Reference Translation:** *there is almost no space to accommodate these people, and resources have run out.*  
**NMT Translation:** *the resources have been exhausted and resources have been exhausted.*

**Solution:**

- **Error:** Idiomatic error - the NMT translates the (part) of the source sentence using the word “exhausted” while the reference translation uses the idiom “ran out”.
- **Possible reason for model error:** The NMT may have treated the pertinent characters “用尽,” as separate compositional units rather than a single idiomatic unit, resulting in the more literal translation of 用 (to use) and 尽 (to the furthest extent) as “exhausted.”
- **Potential fix:** Manual adjustments to the NMT’s 1D convolutional layer computations such that the word embeddings of consecutive characters are convolved more strongly (e.g., through increasing the convolution’s bias term) could result in convolved embeddings that are more akin to those of multi-character words, enabling a term like “用尽” to be embedded and translated as a single unit, improving the likelihood of a more idiomatic rather than literal translation.

- iii. (2 points) **Source Sentence:** 当局已经宣布今天是国殇日。

**Reference Translation:** *authorities have announced a national mourning today.*

**NMT Translation:** *the administration has announced today's day.*

**Solution:**

- **Error:** The NMT omitted part of the source sentence in its translation (“国殇”) which the reference translation captures as “national mourning.”
- **Possible reason for model error:** The relevant characters of the omitted part of the source sentence (“国殇”) do not appear in the `src.vocab` file, meaning the characters could not be represented in the embedding layer and were thus omitted from the NMT’s translation.
- **Potential fix:** Given that this an omission error arose from a lack of presence in the source vocab, we can simply augment/expand the source vocab with more texts, words, and their constituting characters to decrease the probability our NMT encounters a character not present in its source vocab again.

- iv. (2 points) **Source Sentence**<sup>4</sup>: 俗语有云:“唔做唔错”。

**Reference Translation:** *“act not, err not”, so a saying goes.*

**NMT Translation:** *as the saying goes, “it's not wrong.”*

**Solution:**

- **Error:** Idiomatic error - the source sentence contains a Cantonese idiom (converted to simplified Chinese) that the reference translation writes as “act not, err not” which the NMT translation (egregiously) mistranslates as “it’s not wrong.”
- **Possible reason for model error:** The relevant part of the source sentence derives from a Cantonese saying, and the model training data may lack many Cantonese-derived examples, leading to few learning opportunities and an overall poor quality of translation on this subset of input types.
- **Potential fix:** Fine-tuning the NMT on a smaller dataset consisting of many Cantonese idioms (converted to simplified Chinese) could increase learning opportunities and resultingly, NMT translation quality on this type of input. Also, varying/increasing the kernel size of the NMT’s 1D convolutional layer (note that the idiom in question is represented as 4 Chinese characters, “唔做唔错,” whereas our model’s 1D convolutional layer uses kernel size of 2) could result in more accurate embeddings and downstream translations of multi-character idioms.

---

<sup>4</sup>This is a Cantonese sentence! The data used in this assignment comes from GALE Phase 3, which is a compilation of news written in simplified Chinese from various sources scraped from the internet along with their translations. For more details, see <https://catalog.ldc.upenn.edu/LDC2017T02>.

- (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.<sup>5</sup> 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**:

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

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

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

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

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

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

---

<sup>5</sup>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](http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleu)

- i. (5 points) Please consider this example:

Source Sentence **s**: 需要有充足和可预测的资源。

Reference Translation **r**<sub>1</sub>: *resources have to be sufficient and they have to be predictable*

Reference Translation **r**<sub>2</sub>: *adequate and predictable resources are required*

NMT Translation **c**<sub>1</sub>: there is a need for adequate and predictable resources

NMT Translation **c**<sub>2</sub>: resources be sufficient and predictable to

Please compute the BLEU scores for **c**<sub>1</sub> and **c**<sub>2</sub>. 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 work (i.e., show your computed values for  $p_1$ ,  $p_2$ ,  $\text{len}(c)$ ,  $\text{len}(r)$  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. Please round your responses to 3 decimal places.

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

**Solution:**

*BLEU* score for **c**<sub>1</sub>:

- $p_1 = \frac{4}{9}, p_2 = \frac{3}{8}, \text{len}(\mathbf{c}_1) = 9, \text{len}(r) = 11$
- $BP = e^{(1 - \frac{11}{9})} \approx 0.801$
- $\text{BLEU} = BP \times \exp\left(\sum_{n=1}^4 \lambda_n \log p_n\right) = 0.801 \times e^{(0.5 * \ln(4/9) + 0.5 * \ln(3/8))} \approx \mathbf{0.327}$

*BLEU* score for **c**<sub>2</sub>:

- $p_1 = 1, p_2 = \frac{3}{5}, \text{len}(\mathbf{c}_2) = 6, \text{len}(r) = 6$
- $BP = 1$
- $\text{BLEU} = BP \times \exp\left(\sum_{n=1}^4 \lambda_n \log p_n\right) = 1 \times e^{(0.5 * \ln(1) + 0.5 * \ln(3/5))} \approx \mathbf{0.775}$

According to the BLEU score, NMT translation **c**<sub>2</sub> is the better translation. However, I would **not** agree that it is the better translation; although translation **c**<sub>1</sub> may obtain a lower BLEU score, it is (to me, in a subjective sense) more human-like in terms of grammatical/syntactic structuring, and it also captures the meaning of the source sentence more clearly than translation **c**<sub>2</sub>.

- ii. (5 points) Our hard drive was corrupted and we lost Reference Translation  $\mathbf{r}_1$ . Please recompute BLEU scores for  $\mathbf{c}_1$  and  $\mathbf{c}_2$ , this time with respect to  $\mathbf{r}_2$  only. Which of the two NMT translations now receives the higher BLEU score? Do you agree that it is the better translation?

**Solution:**

BLEU score for  $\mathbf{c}_1$ :

- $p_1 = \frac{4}{9}, p_2 = \frac{3}{8}, \text{len}(\mathbf{c}_1) = 9, \text{len}(r) = 6$
- $BP = 1$
- $\text{BLEU} = BP \times \exp\left(\sum_{n=1}^4 \lambda_n \log p_n\right) = 1 \times e^{(0.5 * \ln(4/9) + 0.5 * \ln(3/8))} \approx \mathbf{0.408}$

BLEU score for  $\mathbf{c}_2$ :

- $p_1 = \frac{1}{2}, p_2 = \frac{1}{5}, \text{len}(\mathbf{c}_1) = 6, \text{len}(r) = 6$
- $BP = 1$
- $\text{BLEU} = BP \times \exp\left(\sum_{n=1}^4 \lambda_n \log p_n\right) = 1 \times e^{(0.5 * \ln(1/2) + 0.5 * \ln(1/5))} \approx \mathbf{0.316}$

Under the new conditions, NMT translation  $\mathbf{c}_1$  now achieves a higher BLEU score. For the reasons described in my answer to 2(c)i., I agree that  $\mathbf{c}_1$  is the better translation.

- iii. (2 points) Due to data availability, NMT systems are often evaluated with respect to only a single reference translation. Please explain (in a few sentences) why this may be problematic. In your explanation, discuss how the BLEU score metric assesses the quality of NMT translations when there are multiple reference translations versus a single reference translation.

**Solution:**

The paradigm of evaluating NMT systems with respect to only a single reference translation may be problematic because doing so fails to capture the NMT's ability provide a translation which accords well with each and all of the multiple valid translations of some source text, a capacity which can be considered a proxy for the NMT's ability to generate "robust" translations (i.e., those which capture the true underlying meaning of the source text as opposed to surface-level translations which function as minimalist vocabulary look-ups and fuzzy matches).

To understand this better, note how the BLEU score metric assesses the quality of NMT translations when there is a single versus multiple reference translations:

- When there is a single reference translation, the BLEU score can be heavily influenced by individual human preferences for syntax, word choice, etc. expressed in the reference translation; in such contexts, the BLEU score assesses the quality of the NMT translation on its brevity and  $n$ -gram similarity to the reference translation, but the latter component (i.e.,  $n$ -gram similarity) can be heavily influenced by the NMT's translation's matching (or lack thereof) of some translator's stylistic preferences.
- When there are multiple reference translations, the BLEU score metric assesses the quality of NMT translations again by its brevity and  $n$ -gram similarity to the reference translations, but because there are multiple reference translations, the  $n$ -gram similarity component is less likely to be heavily biased in favor of one particular NMT translation across all reference translations because translator stylistic preferences are likely to be varied.

In short, if a metric like the BLEU score is used to evaluate a NMT system, using only one reference translation can provide a poor assessment of translation quality since the matching of stylistic preferences of the translator providing the reference can influence the BLEU score heavily, which is a non-desirable criterion for translation evaluation since many texts have multiple valid translations and a more important/desirable criterion is meaning preservation (which may be better approximated by  $n$ -gram similarity when there are many reference translations).



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

**Solution:**

*Advantages of BLEU:*

1. BLEU is cheaper and faster to implement than human evaluation – BLEU essentially consists of a bunch of count and sum operations with a dusting of some slightly more involved operations (log, multiplication, etc.). These operations are all very easy to implement and conduct via a computer program, meaning computing the BLEU score for some NMT translation for a huge set of reference translations is likely much cheaper and faster than having a human translator attempt to compare and compute some similarity metric for each reference translation and the NMT translation.
2. BLEU is consistent unlike human evaluation – given the same NMT translation and reference translation(s), the BLEU score for said NMT translation will always be the same fixed number. However, two human translators may judge the NMT translation to approximate the reference translation(s) to differing degrees of goodness. This issue with human evaluation is problematic for the evaluation and improvement of NMTs as it is not immediately clear which reference translation(s) we should try to push the NMT toward approximating, whereas with the BLEU score, we have a consistent metric through which we can measure and attempt to improve NMT performance.

*Disadvantages of BLEU:*

1. Unlike human evaluation, BLEU is poor at capturing semantic similarity – human evaluators can evaluate machine translations on the basis of their meaning, whereas BLEU is based on word count and  $n$ -gram overlap which is (at best) a poor approximation of semantic similarity. Since meaning preservation is a desirable goal for translation tasks, human evaluation can be better than BLEU for evaluating machine translations with respect to this criterion.
2. Unlike human evaluation, BLEU fails to account for high-level criterion like translation fluency, coherence, etc. – human evaluators can understand and assess machine translations according to high-level language abstractions like sentence fluency, coherence, etc. which BLEU fails to capture. Owing to this, human evaluators can disambiguate and choose a single “best” translation between multiple machine translations which may be equally “good” according to a metric like the BLEU score but which vary in terms of their approximation of these more sophisticated language characteristics—ultimately leading to more robust evaluations and improvements of machine 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.