CS 224n Spring 2024: Assignment #3

Due Date: April 30th, Tuesday, 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.

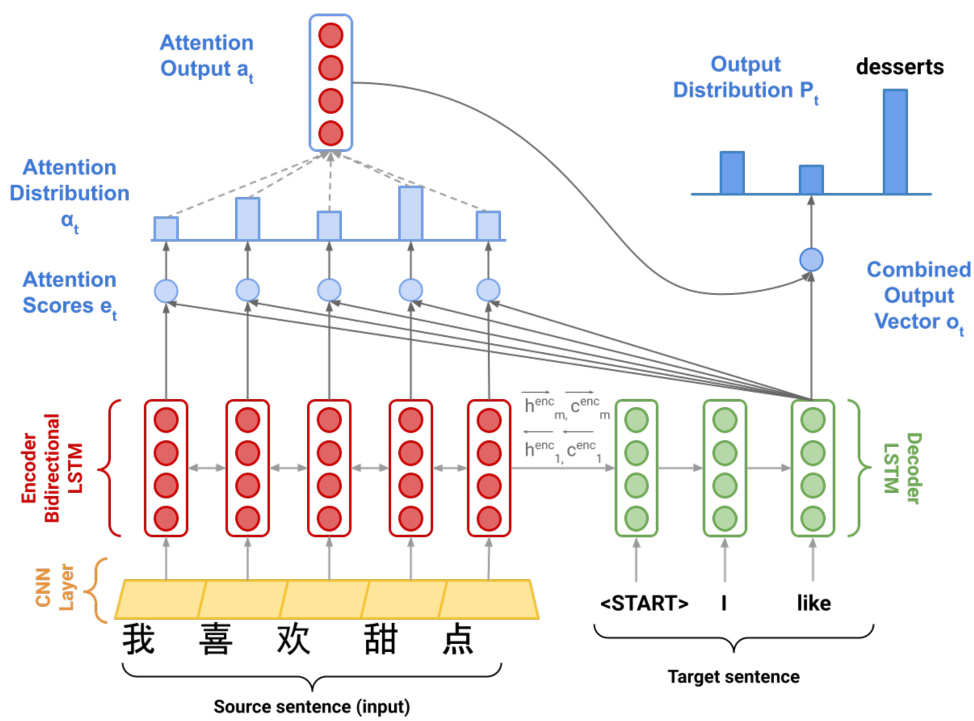

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined on the next page.

Model description (training procedure)

Given a sentence in the source language, we look up the character or word embeddings from an **embeddings matrix**, yielding $\mathbf{x}_1, \ldots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e\times 1}$), where m is the length of the source sentence and e is the embedding size. We then feed the embeddings to a **convolutional layer** [1] 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 $\mathbf{c}_i^{\text{enc}}$:

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

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

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.[2]

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

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

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

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

$$\tag{6}$$

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

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

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

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

$\mathbf{e}_{t,i}$ is a scalar, the ith element of $\mathbf{e}_t \in \mathbb{R}^{m\times 1}$, computed using the hidden state of the decoder at the tth step, $\mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h\times 1}$, the attention projection $\mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h}$, and the hidden state of the encoder at the ith step, $\mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}$.

We now concatenate the attention output $\mathbf{a}_t$ with the decoder hidden state $\mathbf{h}_t^{\text{dec}}$ and pass this through a linear layer, tanh, and dropout to attain the *combined-output* vector $\mathbf{o}_t$.

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

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

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

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

[2] If it's not obvious, think about why we regard $[\overleftarrow{\mathbf{h}_1^{\text{enc}}}, \overrightarrow{\mathbf{h}_m^{\text{enc}}}]$ as the 'final hidden state' of the Encoder.

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

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

Here, V_t is the size of the target vocabulary. Finally, to train the network we then compute the cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the one-hot vector of the target subword at timestep t:

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

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

Setting up Your Local Development Environment

Ensure that you have conda installed. Unzip the starter code, and open the resulting directory in your favorite python integrated development environment (Visual Stutio Code is a popular choice). Open a terminal and navigate (cd) to the directory that contains the env-cpu.yml and env-gpu.yml files. Create and activate the cs224n-cpu conda environment as follows:

```
conda env create --file env-cpu.yml
conda activate cs224n-cpu
```

Setting up Your Cloud GPU-powered Virtual Machine

Follow the instructions in the GCP Guide for CS224n (link also provided on website and Ed) 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 GCP subscription runs out of money, your VM will be temporarily locked and inaccessible. If that happens, please make a private Ed post describing the situation.

After setting up the cloud VM, you should turn it off while you work on the programming assignment locally.

Implementation and written questions

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

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

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

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

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

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

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

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

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

(g) (3 points) (written) The generate_sent_masks() function in nmt_model.py produces a tensor called enc_masks. It has shape (batch size, max source sentence length) and contains 1s in positions corresponding to 'pad' tokens in the input, and 0s for non-pad tokens. Look at how the masks are used during the attention computation in the step() function (lines 295-296).

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

[ANSWER] The enc_masks tensor has 1s in positions of padding tokens, and 0s in the non-padding token positions. Worth noting that it has the same shape (batch size, maximum source sentence length) as e_t, which is the attention scores. It is then used by e_t.data.masked_fill_(enc_masks.bool(), -float('inf')), where elements of e_t will be replaced by -inf if the corresponding position of enc_mask is 1. This effectively means the specific element of e_t will be excluded from the softmax computation, as values to the "-inf" power are effectively 0.

It is necessary to include masks since paddings are just used to ensure that all sentences within a batch have the same length. The padding token is thus just a placeholder, whose value does not have any meaning and thus should be excluded from attention computations.

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
```

To help with monitoring and debugging, the starter code uses tensorboard to log loss and perplexity during training using TensorBoard[3]. TensorBoard provides tools for logging and visualizing training information from experiments. To open TensorBoard and monitor the training process, run the following in a separate terminal in which you have also activated the cs224n-cpu conda environment, then access tensorboard at http://localhost:6006/.

```
tensorboard --logdir=runs --port 6006
```

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

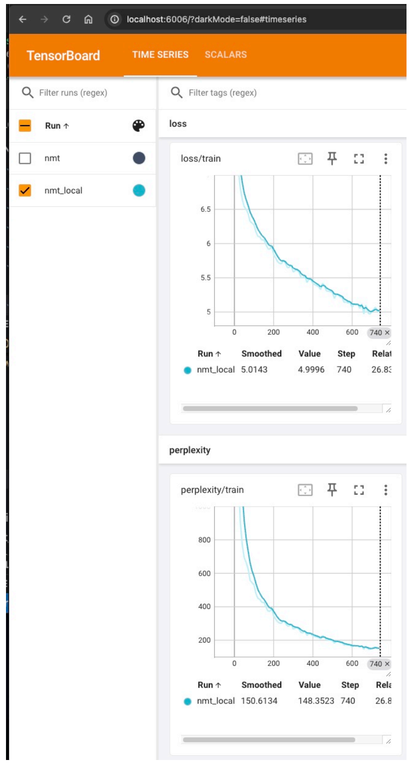

Figure 2: Tensorboard showing loss and perplexity values on your local machine

You should see a significant decrease in loss during the initial iterations (Fig 2). Once your code runs for a few hundreds of iterations without crashing, power on your VM from the GCP Console.

Train the NMT system on the VM: Refer to the GCP How-to Appendix to:

- Setup your GCP VM and Obtain SSH Connection Info
- Connect to the VM (with SSH Tunneling setup so that you can view remote tensorboard logs)
- Copy your code from your computer to the cloud VM
- Train the NMT System on the Cloud VM
- Download the Gradescope Submission Package from Your Cloud VM

(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
```

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

Solution The BLEU score obtained is 19.557919385349848, which is larger than 18. This shows that the training is successful as expected.

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

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

Solution The dot product attention requires s and h (or query and key) vectors having the same shape. Multiplicative attention is more flexible, as the matrix W can project a hidden state vector $\mathbf{h}$ of some different dimension into the same dimension of $\mathbf{s}$. The advantage of dot product attention is being computationally cheaper as no additional parameter $\mathbf{W}$ is needed.

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

Solution Additive attention uses matrix $\mathbf{W}_1$ and $\mathbf{W}_2$ to project and $\mathbf{s}$ into the same dimension, where their sum is passed into a non-linearity. Also, an additional "value vector' $\mathbf{v}$ is learned ($\mathbf{V}$ matrix learned more precisely) to score attention. Therefore, additive attention is more expressive, with a nonlinear layer and a value vector to represent a richer scoring function than linear/bilinear correlations. The disadvantage is more parameters and nonlinear operators, which give a higher computational cost.

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 In the source language Chinese, a sentence might be tokenized into either single-character subwords or multi-character words. Just like the case of electricity + brain = computer, the meaning of words often emerges from a local combination of characters. 1D convolution has a convolution window that slides across the embedded sentence, combining neighboring embeddings into higher-level features. Thus, even if words are split into characters, the combined word-level meaning can be captured. The encoder thus receives representations that has already integrated word-level meanings, so LSTM can better process and translate the sentence.

(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 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 The error is that the plural noun and verb culprits were being translated as a single phrase culprit was. Chinese nouns do not mark plurality explicitly, so the plurality has to be

inferred from neighboring words and context. A possible solution will be to use a larger context or to include more ambiguous cases in training data.

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 The error is repetition, where the part "resources have been exhausted" has been repetitively translated, leaving the other half of the sentence untranslated. One of the inputs to each decoder LSTM cell is the previous hidden state, where all information about past states is stored. Over long sequences, the hidden state will lose track of what parts of the source sentence have been generated or not, leading to repeated phrases. Introducing some self-attention mechanics, like Transformers, gives a better context of the entire decoder. Can also include a repetition penalty.

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 The error is failing to translate "national mourning today". This is likely due to the uncommon word " 国殇日" that is unrecognized. Being tokenized as an unknown token, its meaning can only be inferred from neighboring tokens. The meaning of "national mourning" is thus unlikely to be obtained, and only some vague concept of "day" is inferred. This can be fixed by increasing the training corpus and vocab size to include more less-common words.

iv. (2 points) **Source Sentence[4]:** 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 The error is the idiom being translated literally instead of being translated into its intended proverb. The model does not explicitly distinguish idioms from a sequence of words, so idioms will also be broken into words and translated normally. Especially when the idiom is in Cantonese, which is less common in Chinese text corpora. To fix, include more idiomatic data, or include more hidden state layers to include.

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

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

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

[4]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.

[5]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

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

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

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

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

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

i. (5 points) Please consider this example:
Source Sentence $\mathbf{s}$: **需要有充足和可预测的资源。**
Reference Translation $\mathbf{r}_1$: *resources have to be sufficient and they have to be predictable*
Reference Translation $\mathbf{r}_2$: *adequate and predictable resources are required*
NMT Translation $\mathbf{c}_1$: there is a need for adequate and predictable resources
NMT Translation $\mathbf{c}_2$: resources be sufficient and predictable to
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 work (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. 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 For NMT translation $\mathbf{c}_1$, we have $p_1 = \frac{4}{9}$ and $p_2 = \frac{3}{8}$. len($\mathbf{c}_1$)=9, len($\mathbf{r}_1$)=11, and len($\mathbf{r}_2$)=6, leading to $BP = \exp(1 - \frac{11}{9}) = \exp(-\frac{2}{9})$. BLEU is thus $\boxed{0.327}$. For $\mathbf{c}_2$, we have $p_1 = 1$ and $p_2 = \frac{3}{5}$. len($\mathbf{c}_2$)=6, thus BP=1. BLEU is thus $\boxed{0.775}$. The second translation is considered the better translation according to the BLEU score, which I don't agree with. It happens to have more common 1-gram and 2-gram words compared to the reference translation. However, the translation overall is not grammatically correct.

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 For $\mathbf{c}_1$, we have $p_1 = \frac{4}{9}$, $p_2 = \frac{3}{8}$, BP=1, giving BLEU=$\boxed{0.408}$. For $\mathbf{c}_2$, we have $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{5}$, BP=1, giving BLUE=$\boxed{0.316}$. First translation is preferred, and I agree with it.

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 BLEU score metric calculates the "reoccurring" n-grams that exist in both the reference and NMT translations. With multiple reference transitions, the BLEU score assesses the quality of translation more systematically, as each word, phrase, and sentence is not unique and can be translated differently. This increases the chances of correct translations recurring in one of the references. On the other hand, having a single reference is simply assessing the

similarity between the NMT translation with the reference translation. With a different word choice or sentence structure, a good quality translation will be assigned a low BLEU score.

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

Solution Compared to human evaluation, BLEU has the advantage of (1) BLEU can be calculated automatically and is much more scalable without the need for human evaluation, and (2) BLEU quantifies the translation quality, generating an objective and numeric result, which is not possible for humans to do. However, BLEU also has disadvantages (1) Translations are not unique, so it will be hard to cover all possibilities in reference translations. Unseen or dissimilar but correct translations are penalized. (2) BLEU only measures the n-gram overlap with the reference, and not the adequacy and fluency. It will also over-reward wordy translations.

(d) (4 points) *Beam search* is often employed to improve the quality of machine translation systems. While you were training the model, beam search results for the same example sentence at different iterations were also recorded in TensorBoard, and accessible in the *TEXT* tab (Fig 3).

The recorded diagnostic information includes json documents with the following fields: example_source (the source sentence tokens), example_target (the ground truth target sentence tokens), and hypotheses (10 hypotheses corresponding to the search result with beam size 10). Note that a predicted translation is often called *hypothesis* in the neural machine translation jargon.

i. (2 points) Did the translation quality improve over the training iterations for the model? Give three examples of translations of the example sentence at iterations 200, 3000, and the last iteration to illustrate your answer. For each iteration, pick the first beam search hypothesis as an example:

Solution The example source sentence is " 我还澄清了在该会议上提出的若干事项。" and the example target sentence is "<s> i was able to provide clarification on some of the matters which were raised at that meeting. </s>".

At iteration 200, the first hypothesis is "it is not that the united nations of the united nations of the united nations." (score=-31.028). At iteration 3000, the first hypothesis is "i also wish to clarify the number of cases in the conference." (score=-13.374). At last iteration 18800, the first hypothesis is "i have also clarified a number of matters raised at the meeting." (score=-7.010). The translation quality does improve over training iterations, from repetition of phrases to coherent but inaccurate sentences to good translations.

ii. (2 points) How do various hypotheses resulting from beam search qualitatively compare? Give three other examples of hypotheses proposed by beam search at the last iteration to illustrate your answer.

Solution The first three hypotheses from the last iteration are "i have also clarified a number of matters raised at the meeting.", "i have also clarified a number of issues raised at the meeting.", and "i have also clarified a number of matters raised at this meeting." The three translations were very close, and between hypotheses, only single word choices vary. These are of very similar qualities.

Submission Instructions

You shall submit this assignment on GradeScope as two submissions – one for "Assignment 3 [coding]" and another for 'Assignment 3 [written]":

1. Run the collect_submission.sh script on the cloud VM to produce your assignment3.zip file. See *How to Download the Gradescope Submission Package from Your Cloud VM* in the GCP How-to Appendix.

2. Upload your assignment3.zip file to GradeScope to "Assignment 3 [coding]".

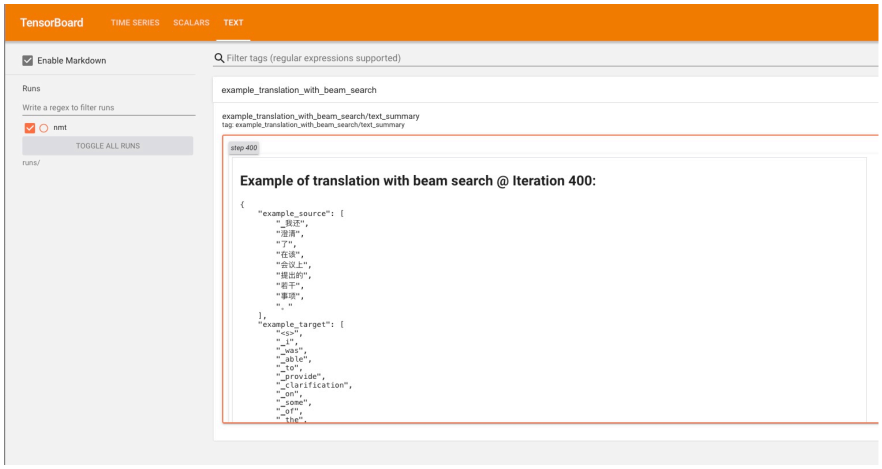

Figure 3: Translation with beam search results for an example sentence are recorded in tensorboard for various iterations. The same data is available in the outputs/beam_search_diagnostics/ folder in your working directory.

3. Upload your written solutions to GradeScope to "Assignment 3 [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.

Appendix: GCP How-to

How to Obtain SSH connection info After following the instructions in the GCP Guide for CS224n, your Gloogle Cloud Compute Engine dashboard should look like Fig 4. Click on the arrow button near SSH and select *View gcloud command* as shown in the figure. What you will get looks like the following. Make a note of the zone, the machine name, and the project name which in the example are respectively us-west4-b, nvidia-gpu-optimized-vmi-1-vm and grand-hangar-420500. **Yours will be different!**

```
gcloud compute ssh --zone "us-west4-b" "nvidia-gpu-optimized-vmi-1-vm" --
    ↪ project "grand-hangar-420500"
```

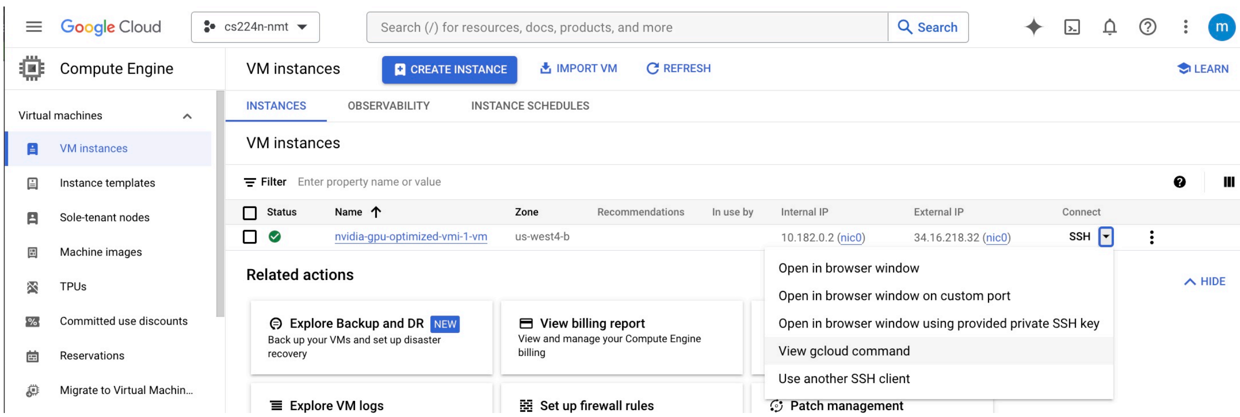

Figure 4: The cloud VM you create should be based on the nvidia-gpu-optimized-vmi virtual machine image (VMI). After deploying the instance, click on the menu near SSH and select *View gcloud command* to obtain the SSH connection information.

How to Connect to the Cloud VM with SSH Tunelling The following ssh connection command, which includes the –ssh-flag "-L 6007:localhost:6007" option, will also forward port 6007 of the cloud machine to the same port number on your local machine (ssh tunneling). Therefore, if the tensorboard server is running on the cloud VM and listening to port 6007, after establishing an ssh connection as follows, you will be able to access at http://localhost:6007/ in a web browser on your local machine. Run the following ssh command to connect to the cloud machine. **be sure to use your own zone, project name and machine name**

```
gcloud compute ssh --zone "us-west4-b" "nvidia-gpu-optimized-vmi-1-vm" --
    ↪ project "grand-hangar-420500" --ssh-flag "-L␣6007:localhost:6007"
```

How to Copy Your Code From Your Computer to the Cloud VM

- zip current directory, which should contain the env-cpu.yml and env-gpu.yml files.

```
zip -r student.zip *
```

- copy the obtained zip file to the cloud machine

```
gcloud compute scp student.zip nvidia-gpu-optimized-vmi-1-vm:~/ --zone "
    ↪ us-west4-b" --project "grand-hangar-420500"
```

How to Train the NMT System on the Cloud VM

- If zip is not installed on your cloud machine. Install it as follows and try again.

```
sudo apt-get install zip
```

- Unzip student folder on the cloud machine

```
unzip student.zip -d student
cd student
```

- Create conda GPU environment

```
conda env create --file env-gpu.yml
```

- Activate conda GPU environment

```
conda activate cs224n-nmt-gpu
```

- create a tmux session in which you start training the machine translation model

```
tmux new -s s-nmt

# start the training (in the tmux session)
sh run.sh train

# detach from the tmux session
CTRL+B D
```

- create a tmux session in which you start TensorBoard

```
tmux new -s s-tboard

# start the tensorboard (in the tmux session)
tensorboard --logdir runs/ --port 6007

# detach from the tmux session
CTRL+B D
```

- You can attach/detach from the two tmux sessions as needed. See the appendix on Tmux for basic use cases.
- If you established the SSH connection with the appropriate port forwarding options, you can view tensorboard in a web browser on your local machine at this address http://localhost:6007/ (Fig. 5)
- when the training is complete, run the test phase

```
tmux attach -t s-nmt
sh run.sh test
```

- The next section shows how to make a grade scope submission with the artifacts generated on the cloud VM.

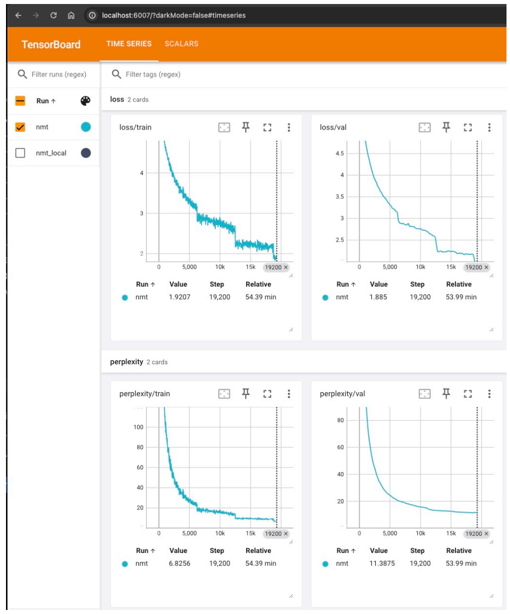

Figure 5: Tensorboard running on your local machine and showing loss and perplexity values on your cloud virtual machine.

How to Download the Gradescope Submission Package from Your Cloud VM

- Connect to the cloud VM as specified above
- On the Cloud VM, run the following command to create the gradescope submission package (assignment3.zip)

```
sh collect_submission.sh
```

- Disconnect from the cloud VM (by running **exit**), or open a terminal on your local machine.
- Download the GradeScope submission package to your local machine

```
gcloud compute scp nvidia-gpu-optimized-vmi-1-vm:~/student/assignment3.zip
    . --zone "us-west4-b" --project "grand-hangar-420500"
```

- Submit assignment3.zip to GradeScope

Appendix: Tmux How-to

tmux (terminal multiplexer) is a tool that lets you create and toggle between multiple terminals that run in the background, and keep then running even after you disconnect your ssh session. The following assumes that you are already connected (via ssh) to your cloud virtual machine instance, and shows you basic use cases of tmux. Learn more about tmux at https://github.com/tmux/tmux/wiki

- Create and attach to a new tmux session (named *s-tboard*)

  ```
  tmux new -s s-tboard
  ```

- Run a program in the new session. We will run tensorboard

  ```
  tensorboard --logdir runs/ --port 6007
  ```

- (From within the tmux session), Detach from a tmux session, and leave the program you started running in the session.

  ```
  CTRL+b d
  ```

- (After detaching from tmux), view all tmux sessions

  ```
  tmux ls
  ```

- attach to a particular tmux session (named s-nmt)

  ```
  tmux attach -t s-nmt
  ```

- to terminate a tmux session (from within the session)

  ```
  exit
  ```