CS 224n Winter 2025: Assignment #3

Due Date: February 4th, 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.

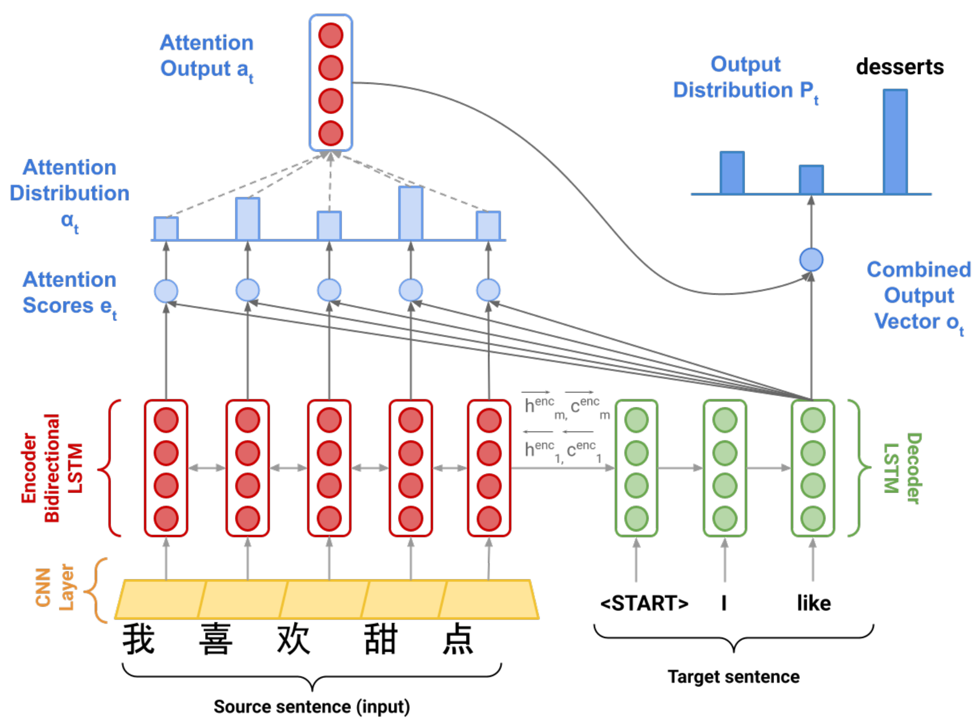

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined on the next page.

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 $\mathbf{h}_1^{\text{enc}}, \dots, \mathbf{h}_m^{\text{enc}}$:

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

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

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

$\mathbf{e}_{t,i}$ 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.

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
```

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

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

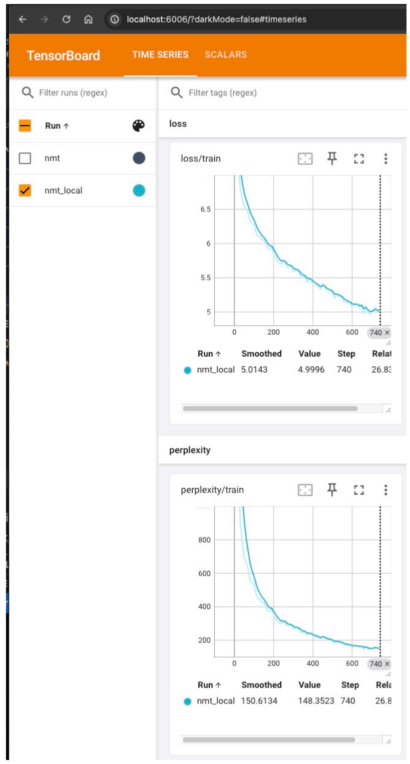

Figure 2: Tensorboard showing loss and perplexity values on your local machine

(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.

(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.

Asnwer: One advantage of *dot product attention* compared to multiplicative attention is its computational efficiency. Dot product attention directly computes the inner product $\mathbf{s}_t^T \mathbf{h}_i$ without requiring additional matrix multiplications, resulting in time complexity $O(d)$ where d is the vector dimension. In contrast, multiplicative attention requires computing $\mathbf{s}_t^T \mathbf{W} \mathbf{h}_i$, which has time complexity $O(d^2)$.

One disadvantage of *dot product attention* compared to multiplicative attention is poorer numerical stability. When vector dimensions are high, the dot product can produce very large values, potentially leading to gradient vanishing or numerical overflow in the softmax function. Multiplicative attention mitigates this by introducing a learnable weight matrix $\mathbf{W}$ that provides implicit scaling and normalization.

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

Answer: One advantage of *additive attention* compared to multiplicative attention is greater expressiveness. Additive attention uses a nonlinear transformation $\mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t)$, allowing it to learn more complex similarity functions beyond linear relationships between vectors.

One disadvantage of *additive attention* compared to multiplicative attention is higher computational complexity. Additive attention requires more parameters (two weight matrices $\mathbf{W}_1$, $\mathbf{W}_2$ and vector $\mathbf{v}$) and involves additional matrix multiplications and nonlinear activations, making it slower and more memory-intensive than multiplicative attention.

2. Analyzing NMT Systems (29 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.

Answer: Adding a 1D Convolutional layer after the embedding layer helps the NMT system better capture character composition patterns in Mandarin Chinese. Mandarin Chinese uses SentencePiece tokenization to break text into subword units, where individual characters like 电 (electricity) and 脑 (brain) may combine to form completely different semantic meanings, such as 电脑 (computer). Without the convolutional layer, the model treats each character's embedding independently, potentially missing the compositional relationships between adjacent characters.

The 1D convolutional layer with kernel size 2 slides over adjacent character embeddings, learning to combine local character pairs. For example, when processing the sequence [电, 脑], the convolution learns a feature representation that captures how "electricity" + "brain" = "computer", rather than treating them as unrelated concepts. This helps the bidirectional LSTM encoder receive more informative representations that reflect Chinese morphological composition, improving translation quality by avoiding literal character-by-character mappings and enabling better semantic understanding of compound words.

The layer uses 'padding='same'' to preserve sequence length, ensuring compatibility with subsequent LSTM processing, while maintaining computational efficiency with shared weights across the sequence. This approach is particularly beneficial for languages like Chinese where meaning emerges from character combinations rather than fixed word boundaries.

(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:** 罪犯们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.*
Error Identification: The NMT translation incorrectly uses singular "culprit" instead of plural "culprits", missing the plural marker indicated by " 们" in the source sentence.
Possible Reasons: The model may not have learned to properly handle Chinese plural markers like " 们", or it could be overly reliant on statistical patterns rather than semantic relationships. This could be due to insufficient training examples of plural constructions or limitations in the attention mechanism.
Possible Fix: Implement improved morphological awareness by adding a morphological analysis component or enhancing the attention mechanism to better track plural markers during translation.

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.*
Error Identification: The NMT translation redundantly repeats "resources have been exhausted" twice, translating the same concept from the source sentence multiple times.
Possible Reasons: This could result from attention mechanism failures where the model loses track of what has already been translated, or from decoder repetition issues. The complex coordination structure in the source sentence may confuse the model's ability to properly distribute attention across different clauses.
Possible Fix: Add a coverage mechanism to track which source tokens have been translated, or implement repetition penalties in the beam search decoding to discourage redundant translations.

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.*
Error Identification: The NMT translation fails to capture the meaning of " 国殇日" (national mourning day), instead producing the nonsensical "today's day."
Possible Reasons: " 国殇日" is a culturally specific term that may be rare in the training data, causing the model to fall back on literal word-by-word translation. The model lacks understanding of compound word semantics and cultural context.
Possible Fix: Expand the vocabulary with more culturally specific terms and their translations, or incorporate pre-trained cross-lingual embeddings that better handle rare and domain-specific vocabulary.

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. "*
Error Identification: The NMT translation misses the idiomatic meaning of the proverb " 唔做唔错" (act not, err not), instead providing a literal interpretation "it's not wrong."
Possible Reasons: Proverbs and idioms require non-literal translation that goes beyond word-by-word mapping. The model may lack sufficient examples of this specific proverb in the training data, or struggle with the semantic shift required for idiomatic expressions.

[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.

Possible Fix: Add idiom-specific training data with correct idiomatic translations, or implement a separate idiom recognition and translation module that can identify and properly handle fixed expressions.

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

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

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).

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 $\mathbf{r}_1, \ldots, \mathbf{r}_k$ is:

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

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

[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

For candidate translation $\mathbf{c}_1$: there is a need for adequate and predictable resources
$p_1 = 4/9 = 0.444$
$p_2 = 2/8 = 0.250$
$len(c) = 9$, $len(r) = 11$ (closest reference length from $\mathbf{r}_1$)
$BP = \exp(1 - 11/9) = \exp(-0.222) = 0.801$
$BLEU_1 = 0.801 \times \exp(0.5 \times \ln(0.444) + 0.5 \times \ln(0.250)) = 0.801 \times \exp(-1.099) = 0.801 \times 0.333 = 0.267$
For candidate translation $\mathbf{c}_2$: resources be sufficient and predictable to
$p_1 = 5/6 = 0.833$
$p_2 = 2/5 = 0.400$
$len(c) = 7$, $len(r) = 6$ (closest reference length from $\mathbf{r}_2$)
$BP = 1.000$
$BLEU_2 = 1.000 \times \exp(0.5 \times \ln(0.833) + 0.5 \times \ln(0.400)) = 1.000 \times \exp(-0.549) = 1.000 \times 0.578 = 0.578$
According to the BLEU score, $\mathbf{c}_2$ is considered the better translation with a score of 0.578 compared to $\mathbf{c}_1$'s 0.267. However, I disagree that $\mathbf{c}_2$ is actually the better translation. While $\mathbf{c}_2$ achieves a higher BLEU score due to greater n-gram overlap with the reference translations (particularly matching words like "resources", "sufficient", and "predictable"), $\mathbf{c}_1$ is more natural and fluent English despite using different phrasing. This highlights BLEU's limitation in preferring literal matches over semantic equivalence and grammatical fluency.

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?
With only reference translation $\mathbf{r}_2$: adequate and predictable resources are required
For candidate translation $\mathbf{c}_1$: there is a need for adequate and predictable resources
$p_1 = 4/9 = 0.444$
$p_2 = 3/8 = 0.375$
$len(c) = 9$, $len(r) = 6$
$BP = 1.000$
$BLEU_1 = 1.000 \times \exp(0.5 \times \ln(0.444) + 0.5 \times \ln(0.375)) = 1.000 \times \exp(-0.896) = 1.000 \times 0.408 = 0.408$
For candidate translation $\mathbf{c}_2$: resources be sufficient and predictable to
$p_1 = 3/6 = 0.500$
$p_2 = 1/5 = 0.200$
$len(c) = 7$, $len(r) = 6$
$BP = 1.000$
$BLEU_2 = 1.000 \times \exp(0.5 \times \ln(0.500) + 0.5 \times \ln(0.200)) = 1.000 \times \exp(-1.151) = 1.000 \times 0.316 = 0.316$
Now $\mathbf{c}_1$ receives the higher BLEU score (0.408 vs 0.316). I agree that $\mathbf{c}_1$ is the better translation, as it maintains grammatical correctness and natural phrasing while still conveying the core meaning of requiring adequate and predictable resources. The BLEU score fluctuation demonstrates how sensitive the metric is to the choice of reference translations.

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 transitions versus a single reference translation.
Using only a single reference translation for BLEU evaluation is problematic because natural language translation allows multiple valid paraphrases. A single reference cannot capture all acceptable variations, potentially penalizing high-quality translations that use different but

equivalent expressions.
When multiple references are available, BLEU computes the maximum n-gram count across all references for each candidate n-gram, allowing matches with any valid paraphrase. With a single reference, BLEU can only match that specific wording, making evaluation more restrictive and sensitive to reference translation choices.

iv. (2 points) List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.
Two advantages of BLEU compared to human evaluation:
1. **Automation and scalability:** BLEU can be computed automatically on large datasets without human intervention, making it feasible to evaluate thousands of translations quickly and consistently.
2. **Objectivity and reproducibility:** BLEU provides deterministic scores that remain consistent across different evaluators and time periods, unlike human evaluation which can vary due to subjective judgment.
Two disadvantages of BLEU compared to human evaluation:
1. **Limited semantic understanding:** BLEU only measures n-gram overlap and cannot recognize semantic equivalence, often penalizing valid paraphrases or synonyms that differ from reference translations.
2. **Insensitive to fluency and naturalness:** BLEU does not assess grammaticality, coherence, or natural language flow, potentially assigning high scores to awkward or unnatural translations that happen to match reference n-grams.

(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:

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.
I cannot answer this question because I have no access to Berkeley's TensorBoard.

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]".

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.

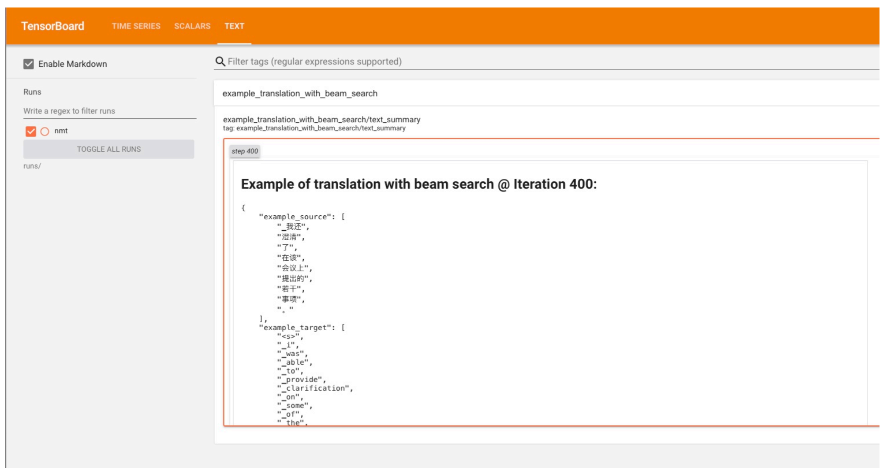

Figure 3: Translation with beam search results for an example sentence are recorded in tensorboard for various iterations. The same data is available in the outputs/beam_search_diagnostics/ folder in your working directory.

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"
```

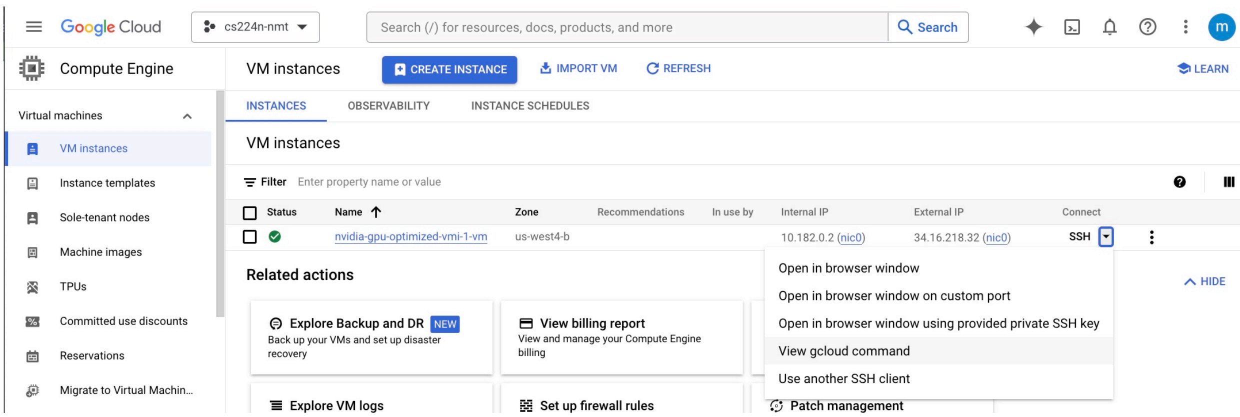

Figure 4: The cloud VM you create should be based on the nvidia-gpu-optimized-vmi virtual machine image (VMI). After deploying the instance, click on the menu near SSH and select *View gcloud command* to obtain the SSH connection information.

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.

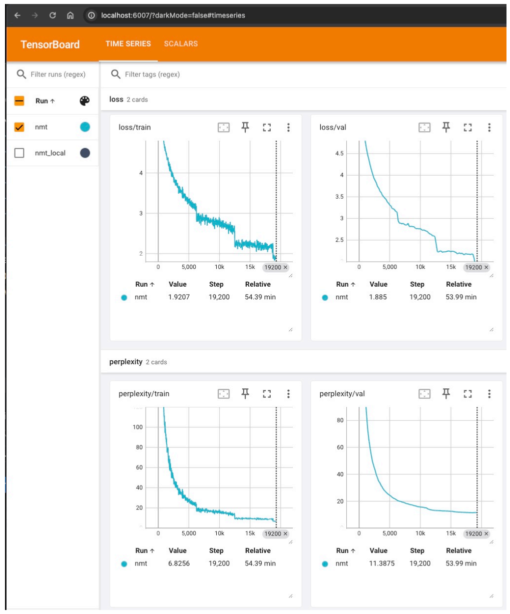

Figure 5: Tensorboard running on your local machine and showing loss and perplexity values on your cloud virtual machine.

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
  ```