机器翻译作业

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.

1. Neural Machine Translation with RNNs (100 points)

In Machine Translation, our goal is to convert a sentence from the *source* language (e.g. Braille) to the *target* language (e.g. Chinese). 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.

note: Braille is obtained based on Chinese Pinyin and Braille Word Segmentation and Concatenation Rules. The pronunciation of a Chinese character corresponds to a Braille segment, and the final Braille is obtained by combining the Braille Word Segmentation and Concatenation Rules.

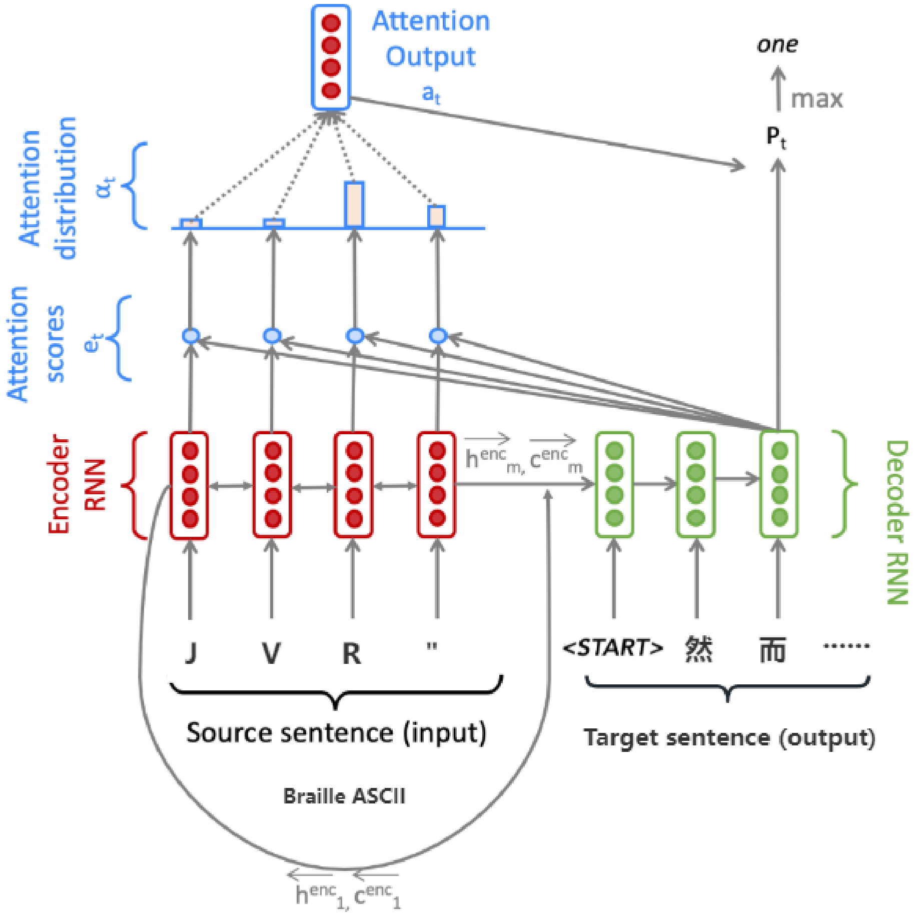

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$ are defined on the next page.

Model description (training procedure)

Given a sentence in the source language, we look up the character or word embeddings from an **embeddings matrix**, yielding $\mathbf{x}_1, \dots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e \times 1}$), where m is the length of the source sentence and e is the embedding size. We then feed the embeddings to a **convolutional layer** while maintaining their shapes. We feed the convolutional layer outputs to the **bidirectional encoder**, yielding hidden states and cell states for both the forwards (→) and backwards (←) LSTMs. The forwards and backwards versions are concatenated to give hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$:

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

We then initialize the **decoder**'s first hidden state $\mathbf{h}_0^{\text{dec}}$ and cell state $\mathbf{c}_0^{\text{dec}}$ with a linear projection of the encoder's final hidden state and final cell state.[1]

$$
\begin{aligned}
\mathbf{h}_0^{\text{dec}} &= \mathbf{W}_h [\overleftarrow{\mathbf{h}_1^{\text{enc}}}; \overrightarrow{\mathbf{h}_m^{\text{enc}}}] \text{ where } \mathbf{h}_0^{\text{dec}} \in \mathbb{R}^{h \times 1}, \mathbf{W}_h \in \mathbb{R}^{h \times 2h} \qquad (3) \\
\mathbf{c}_0^{\text{dec}} &= \mathbf{W}_c [\overleftarrow{\mathbf{c}_1^{\text{enc}}}; \overrightarrow{\mathbf{c}_m^{\text{enc}}}] \text{ where } \mathbf{c}_0^{\text{dec}} \in \mathbb{R}^{h \times 1}, \mathbf{W}_c \in \mathbb{R}^{h \times 2h} \qquad (4)
\end{aligned}
$$

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.

$$
\begin{aligned}
\mathbf{h}_t^{\text{dec}}, \mathbf{c}_t^{\text{dec}} &= \text{Decoder}(\overline{\mathbf{y}_t}, \mathbf{h}_{t-1}^{\text{dec}}, \mathbf{c}_{t-1}^{\text{dec}}) \text{ where } \mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h \times 1}, \mathbf{c}_t^{\text{dec}} \in \mathbb{R}^{h \times 1} \qquad (5) \\
& \qquad (6)
\end{aligned}
$$

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

$$
\begin{aligned}
\mathbf{e}_{t,i} &= (\mathbf{h}_t^{\text{dec}})^T \mathbf{W}_{\text{attProj}} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{e}_t \in \mathbb{R}^{m \times 1}, \mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h \times 2h} \qquad 1 \le i \le m \qquad (7) \\
\alpha_t &= \text{softmax}(\mathbf{e}_t) \text{ where } \alpha_t \in \mathbb{R}^{m \times 1} \qquad (8) \\
\mathbf{a}_t &= \sum_{i=1}^{m} \alpha_{t,i} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{a}_t \in \mathbb{R}^{2h \times 1} \qquad (9)
\end{aligned}
$$

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

$$
\begin{aligned}
\mathbf{u}_t &= [\mathbf{a}_t; \mathbf{h}_t^{\text{dec}}] \text{ where } \mathbf{u}_t \in \mathbb{R}^{3h \times 1} \qquad (10) \\
\mathbf{v}_t &= \mathbf{W}_u \mathbf{u}_t \text{ where } \mathbf{v}_t \in \mathbb{R}^{h \times 1}, \mathbf{W}_u \in \mathbb{R}^{h \times 3h} \qquad (11) \\
\mathbf{o}_t &= \text{dropout}(\tanh(\mathbf{v}_t)) \text{ where } \mathbf{o}_t \in \mathbb{R}^{h \times 1} \qquad (12)
\end{aligned}
$$

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

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

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

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:

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

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 Braille to Chinese translation!

Implementation and written questions

Note: Apart from completing the code, if necessary, the remaining parts of the code can also be modified.

(a) (5 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) (7 points) (coding) Implement the ___init___ function in model_embeddings.py to initialize the necessary source and target embeddings.

(c) (10 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) (15 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) (15 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) (20 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) (8 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.

Now it’ s time to get things running! Execute the following to generate the necessary vocab file:

```
sh run.sh vocab
```

Or if you are on Windows, use the following command instead. Make sure you execute this in an environment that has python in path. For example, you can run this in the terminal of your IDE or your Anaconda prompt.

```
sh run.sh vocab
```

run.bat vocab

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

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[2]. 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 Huawei Web Portal.

Next, install necessary packages to your VM by running:

```
pip install -r gpu_requirements.txt
```

Once your VM is configured and you are in a tmux session, execute:

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

(h) (10 points) (written) Once your model is done training (**this should take under 3 hours on the VM**), execute the following command to test the model:

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

(i) (10 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. (5 points) Explain one advantage and one disadvantage of *dot product attention* compared to multiplicative attention.

ii. (5 points) Explain one advantage and one disadvantage of *additive attention* compared to multiplicative attention.

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

2. Analyzing NMT Systems (100 points)

(a) (15 points) In part 1, we modeled our NMT problem at a subword-level. That is, given a sentence in the source language, we looked up subword components from an embeddings matrix. Alternatively, we could have modeled the NMT problem at the word-level or char-level, by looking up whole words or the single char from the embeddings matrix. Try to analyze the advantages and disadvantages of these three methods, and explain which method you think is better, and provide reasons. (Hint: Consider dictionary size and sequence length.)

(b) (15 points) What is the biggest difficulty in translating Braille into Chinese? (Hint: Braille is derived from Chinese pronunciation.)

(c) (15 points) One challenge of training successful NMT models is lack of parallel corpus, particularly for resource-scarce languages like Braille. One way to address this challenge is to pre-train the model using monolingual data. Train two language models of two source and target languages to initialize the NMT model, and then fine-tune the model using parallel data.

How does monolingual training help in improving NMT performance with low-resource languages?

(d) (16 points) Here we present a series of errors we found in the outputs of our NMT model. For each example of a reference (i.e., '字') Chinese translation, and NMT (i.e., '自') 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 Braille to answer these questions. You just need to know Chinese! If you want to learn about some braille corresponding to Chinese, you can use the translation function on this website.[3]

i. (4 points) **Source Sentence:** GO1GI T4A? D D%' Z2\1 L\G5 TUAKI' D &1D%' W GIAB0' GE1G(ZU'Q#"2
Reference Translation: 国际通用的点字 由六个凸 起的圆 点为基本结构组成。
NMT Translation: 国际通用的点自 由六个突 起的原 点为基本结构组成。

ii. (4 points) **Source Sentence**: H]2 G5 SALU /=A G]'LI'"2
Reference Translation: 换个思路抓 管理。
NMT Translation: 换个思路抓抓抓抓 管理。

iii. (4 points) **Source Sentence**: FVI2 J0&1 H5 G%D* J0&1 D HWBI"2
Reference Translation: 翻译 人员和鉴定人员的回避。
NMT Translation: 反映 人员和鉴定人员的回避。

iv. (4 points) **Source Sentence:** JVH(2" T K>AK>A DI K[A /O1 Q5A Z('L5"2
Reference Translation: 然后，他悄悄地开着车走了。
NMT Translation: 然后，

(e) (40 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.[4] Suppose we have a source sentence $\mathbf{s}$, a set of k reference translations $\mathbf{r}_1, \ldots, \mathbf{r}_k$, and a

[3] http://www.braille.org.cn:8080/braille-web/blinds/index.html?userId=0&username=

[4] 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

candidate translation **c**. To compute the BLEU score of **c**, we first compute the *modified n-gram precision* p_n of **c**, for each of $n = 1, 2, 3, 4$, where n is the n in n-gram:

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

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

Next, we compute the *brevity penalty* BP. Let $len(c)$ be the length of **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 **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. (10 points) Please consider this example:
Source Sentence **s**: GO1GI T4A? D D%' Z2\1 L\G5 TUAKI' D &1D%' W GIAB0' GE1G(ZU'Q#"2
Reference Translation **r**: 国际通用的点字由六个凸起的圆点为基本结构组成。
NMT Translation $\mathbf{c}_1$: 国际通用的点自由六个突起的原点为基本结构组成。
NMT Translation $\mathbf{c}_2$: 国际通用的点字由由由六个凸起的点为基本结构组成。
Please compute the BLEU scores for $\mathbf{c}_1$ and $\mathbf{c}_2$ (Using each char as the basic unit). 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?

ii. (10 points) Translation tasks in other languages can have multiple translations of a source language sentence, while there is only one correct translation for Braille to Chinese translation. Is BLEU suitable for evaluating Braille to Chinese translation tasks. Why?

iii. (10 points) Calculate BLEU based on chars or BLEU based on words to evaluate the translation task from Braille to Chinese. Which is better and explain your reasons.

iv. (10 points) List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.

Submission Instructions

You shall submit this assignment on 学习通 as two submissions –one for “MT Assignment [coding]” and another for ‘MT Assignment [written]”：