Lab 4: Sequence-to-sequence Recurrent Network

Lab objective

In this lab, you need to implement a seq2seq encoder-decoder network with recurrent units for English spelling correction.

Important Date

1. Deadline: 4/28 (Tue.) 11:59 a.m.
2. Demo date: 4/28 (Tue)

Format

1. Experimental Report (.pdf) and Source code (.py)

Notice: zip all files in one file and name it like DLP_LAB4_your studentID_name.zip. e.g. DLP_LAB4_0756051_李仕柏.zip

Lab Description

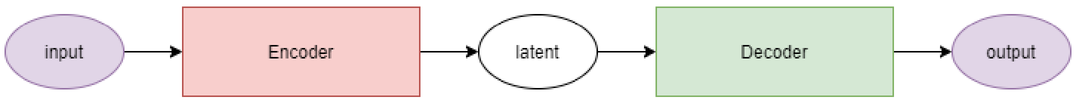
Encoder-decoder structure is well known for feature extraction and compression. We can further reconstruct the original data from those features. Basically, encoder-decoder structure is built by fully connected layers or CNNs while dealing with images input. In this lab, our input will be a word with errors and the target will be the correct one. As a result, the network units become RNNs. This sequence-to-sequence architecture [5] is widely applied to machine translation, text generation, and other tasks related to natural language processing. Overall, your model should act as a spelling corrector model so that when you feed a wrong word, it can always output the correct one. When you demo your code, you have to show your results of test.json and new_test.json which we will upload it to new e3 before demo.

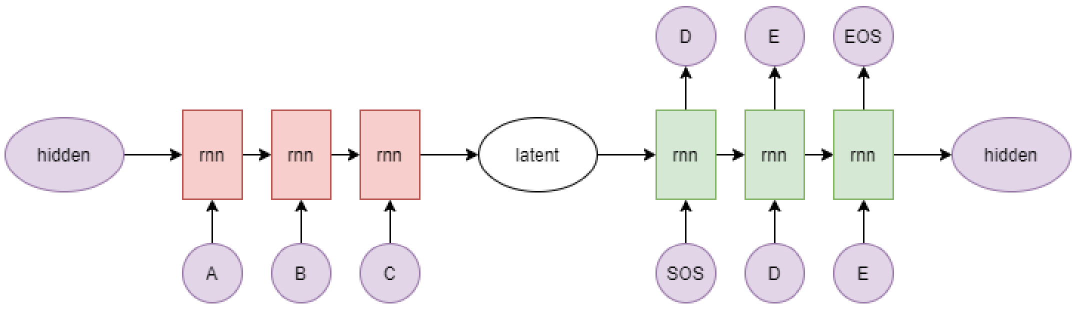
Requirements

1. Implement a seq2seq model.
   A. Modify encoder, decoder, and training functions
   B. Implement evaluation function and dataloader.
2. Plot the CrossEntropy training loss and BLEU-4 testing score curves during training.
3. Output the correction results from test.json

- **Implementation details**

1. **Seq2seq architecture**





Each character in a word can be regarded as a vector. One simple approach to convert a character to a vector is encoding a character to a number and adopting Embedding Layer (see more information in [1]). In the decoder part, you will first feed the hidden output from the encoder and a <start of string> token to the decoder and stop generation process until you receive a <end of string> token or reach the last token of the target word (the token should also be <end of string>).

2. **Embedding function**

Since we cannot directly feed words into the model, we have to encode words to specific features. The simplest way is to encode each word into one-hot vector. However, as the number of words grows, the dimension will become too large nad it is not efficient. Furthermore, one-hot vector is unable to represent the relation between words which is very important to text analysis. Therefore, we encode word with Embedding function that can be regarded as a trainable projection matrix or lookup table. Embedding function can not only compress feature dimension but also building connection between different words. You can find further research of word vector on word2vec [3] and Glove embedding [4].

3. **Teacher forcing**

In the course, we have talked about teacher forcing technique (L10, slide 25-26), which feeds the correct target $y^{(t-1)}$ into $h^{(t)}$ during training. Thus, in this part, you will need to implement teacher forcing technique. Furthermore, to enhance the robustness of the model, we can do the word dropout to weaken the decoder by randomly replacing the input character tokens with the unknown token (defined by yourself). This forces the model only relying on the latent vector z to make predictions.

4. **Other implementation details**
   - The encoder and decoder must be implemented by LSTM, otherwise you will get no point on your report.
   - The loss function is nn.CrossEntropyLoss().
   - The optimizer is SGD.
5. **Hyper-parameters and model setting**
   - LSTM hidden size: 256 or 512
   - Teacher forcing ratio: 0~1 (>0.5)
   - Learning rate: 0.05

Derive the Back Propagation Through Time (BPTT)

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}),$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$

$$p_{\text{model}}(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}) = \prod_i \left(\hat{y}_i^{(t)}\right)^{\mathbf{1}(y_i^{(t)}=1)}$$

$$L^{(t)} = -\log p_{\text{model}}(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)})$$

$$L(\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}\}, \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \dots, \boldsymbol{y}^{(t)}\}) = \sum_t L^{(t)}$$

In the derivation part, you should compute $\nabla_W L$ step by step with clear notations. You can see more information in slides of recurrent neural network. (page 6-7)

Dataset Descriptions

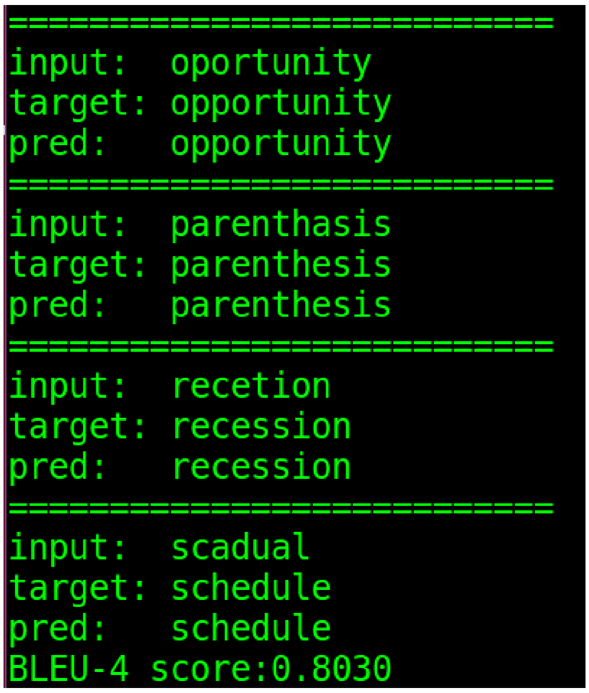
You can download the .zip file from new e3. There are three files in the .zip: readme.txt, train.json, and test.json. All the details of the dataset are in the readme.txt.

● Scoring Criteria

1. Report (60%)
    - ◆ Introduction (5%)
    - ◆ Derivation of BPTT (5%)
    - ◆ Implementation details. (30%)
        - A. Describe how you implement your model. (encoder, decoder, dataloader, etc.).
        - B. You must screen shot the code of evaluation part to prove that you do not use ground truth while testing, otherwise you will get no point at this part.
    - ◆ Results and discussion (20%)
        - A. Show your results of spelling correction and plot the training loss curve and BLUE-4 score testing curve during training. (10%)
        - B. Discussion of the results. (10%)
2. Demo (40%)
    - ◆ Average BLUE-4 scores over all predictions on test.json and new_test.json respectively. (10%+10%)
        - ■ score >= 0.8 ---- 100%
        - ■ 0.8 > score >= 0.7 ---- 90%
        - ■ 0.7 > score >= 0.6 ---- 80%
        - ■ 0.6 > score >= 0.5 ---- 70%
        - ■ score < 0.5 ---- 0%
    - ◆ Questions. (20%)

● Output examples

1. English spelling correction with BLEU-4 score(test.json)



```
===========================
input:  oportunity
target: opportunity
pred:   opportunity
===========================
input:  parenthasis
target: parenthesis
pred:   parenthesis
===========================
input:  recetion
target: recession
pred:   recession
===========================
input:  scadual
target: schedule
pred:   schedule
BLEU-4 score:0.8030
```

● Reference

1. Seq2seq reference code:
   https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
   https://github.com/pytorch/tutorials/blob/master/intermediate_source/seq2seq_translation_tutorial.py
2. Natural Language Toolkit: https://www.nltk.org/
3. Distributed Representations of Words and Phrases and their Compositionality
4. Glove: Global Vectors for Word Representation
5. Sequence to Sequence Learning with Neural Networks