Lab 3: 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: 12/14 (Mon) 11:55 pm
2. Late submission: 12/17 (Thu) 11:55 pm

● Format

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

Notice: zip all files to lab3_studentID_name.zip

e.g., lab3_0858603_曾于耘.zip

● Lab Description

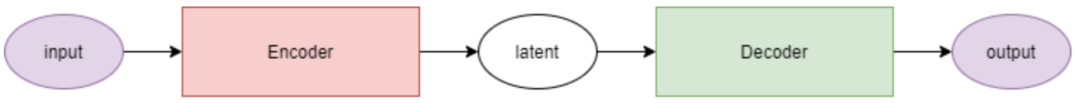
Encoder-decoder structure is well known for feature extraction and compression. We can further reconstruct the original data from those features. Basically, encoder-decoder structure is built by fully connected layers or CNNs while dealing with images input. In this lab, our input will be a word with 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. For evaluation, predict both test.json and new_test.json using BLEU-4 score (you may use nltk toolkit [2]).

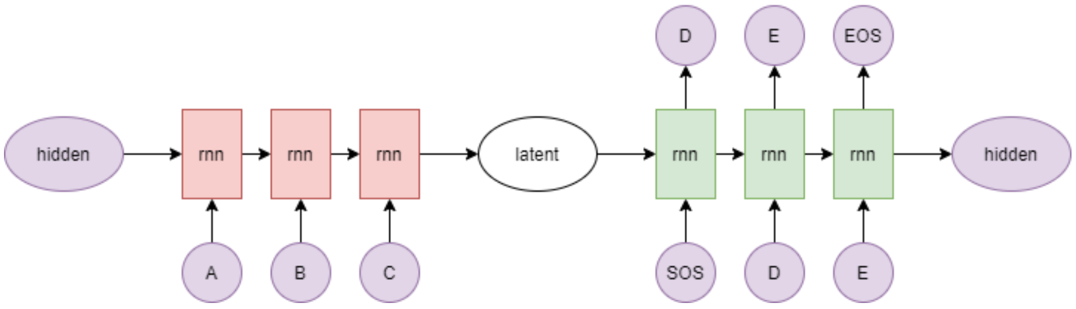
● Requirements

1. Implement a seq2seq model: select either track
   A. Track 1: Write your own code!
   B. Track 2: Use sample code (sample.py)
      - Modify encoder, decoder, and training functions
      - Implement the evaluation function and the dataloader
      - Note: This sample code is provided for those who have problem constructing their own project. The model structure in sample code may not bring out an optimal result. If you wish to improve the performance, we strongly recommend Track 1 for you!
2. Plot the CrossEntropy training loss and BLEU-4 testing score curves during training.
3. Output the correction results from test.json and new_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. Word encoding/embedding function

Since we cannot directly feed 'characters' into the model, so encoding/embedding techniques are required here. The simplest way is to encode each word into one-hot vector. However, one-hot vector is unable to represent the **relation between words** which is very important to NLP, and **padding** is also needed because of the uneven length of words. Therefore, you can further encode words 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. If you prefer using embeddings instead of one-hot encoding, you can look up word2vec [3], Glove embedding [4], or other tools.

3. Teacher forcing

In the course, we have talked about teacher forcing technique, 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.
   - Evaluate your model performance with BLEU-4 score. You may use NLTK toolkit to help you get this score (see reference [2]).
5. **Parameters settings (For reference only. Try different settings and find out which combinations is better!)**
   - Recommended loss function: nn.CrossEntropyLoss().
   - Recommended optimizer: SGD.
   - Recommended LSTM hidden size: 256 or 512
   - Learning rate: 0.05

Derive the Back Propagation Through Time (BPTT)

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

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

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

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

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

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

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

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.

Train your model with train.json, and test your model with both test.json and new_test.json. Details of the dataset can be found in readme.txt.

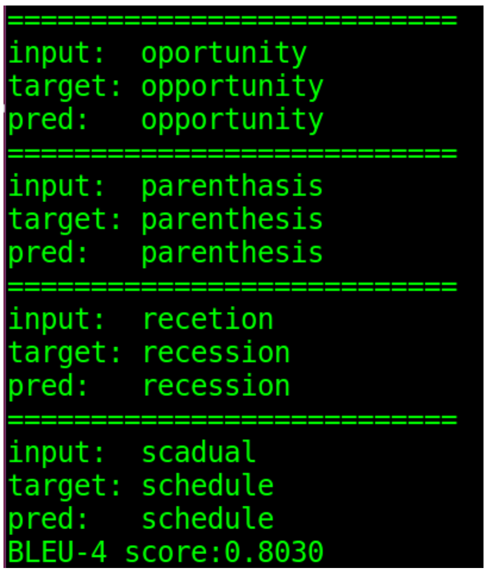
- **Scoring Criteria**

1. Report (100%)
   - ◆ Introduction (5%)
   - ◆ Derivation of BPTT (10%)
   - ◆ Implementation details. (35%)
     - A. Describe how you encode or embed words into vectors
     - B. Describe how you implement your LSTM model. (encoder, decoder, dataloader, etc.).
     - C. You must screen shot the code of evaluation part to prove that you do not use ground truth while testing.
   - ◆ Results and discussion (40%)
     - A. Show your training of spelling correction and plot the training loss curve and evaluation curve during training. (10%)
     - B. Model performance: BLUE-4 score (see below for performance scoring details) (30%)
     - C. Discussion of the results. (10%)

2. Model performance:
   - ◆ Average BLUE-4 scores over all predictions on test.json and new_test.json respectively. (each file includes 50 testing data)
     - ■ score >= 0.8 -------------- 100%
     - ■ 0.8 > score >= 0.7 ------- 90%
     - ■ 0.7 > score >= 0.6 ------- 80%
     - ■ 0.6 > score >= 0.5 ------- 70%
     - ■ 0.5 > score >= 0.4 ------- 60%
     - ■ score < 0.4 ---------------- < 50%

- **Output examples**

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