Lab2 - Sequence-to-Sequence Recurrent Neural Network

Hong-Sheng, Xie

- Deadline: November 17, 2021 11:59 a.m.
- Demo Date: November 17, 2021
- Format: Experimental report (.pdf) and source code (.py). Zip all files in one file and name it like DLP_LAB2_yourID_name.zip.
- Email: hongsheng.cs10g@nctu.edu.tw

1 Lab Description

In this lab, you need to implement a seq2seq encoder-decoder network with recurrent units for English spelling correction. Encoder-decoder structure is well known for feature extraction and reconstruction. Given an input image, the encoder extracts downscaled features and output affluent features through a decoder. Basically, encoder-decoder structure (see in Fig. 1) 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. As showed in Fig. 2, the sequence-to-sequence architecture [6] 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 that it can always correct the wrong words.

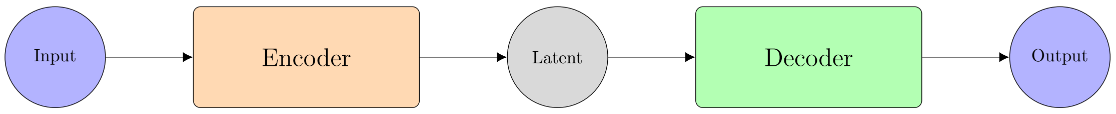


Figure 1: The illustration of auto-encoder.

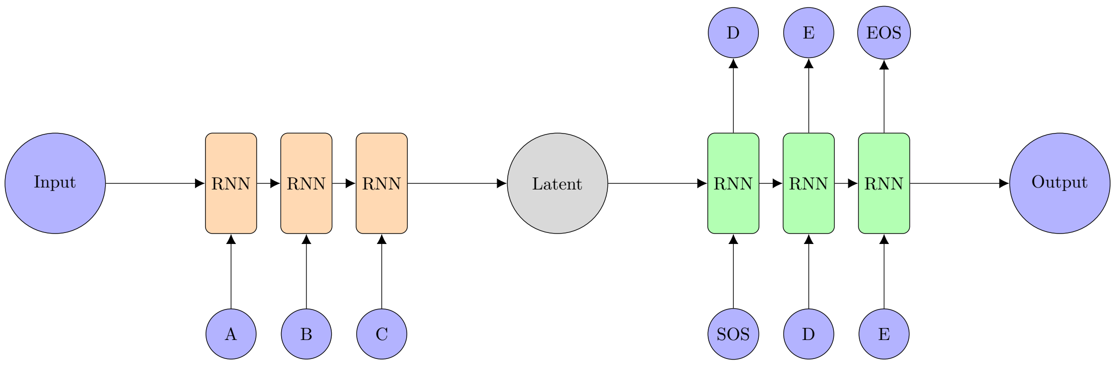


Figure 2: The illustration of sequence-to-sequence architecture.

2 Requirements

- **Implement a seq2seq model**
  - Modify encoder, decoder, and training functions.
  - Implement evaluation function and dataloader.
- **Plot the CrossEntropy training loss and BLEU-4 testing score curves during training.**
- **Output the correction results from test.json and new_test.json**
- **You need to implement GRU by yourself. (Don't use nn.GRU.)**

3 Implementation Details

3.1 Seq2seq architecture

As illustrated in Fig. 2, 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 layers. In the decoder part, you will first input 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>).

3.2 Embedding function

Since we cannot directly input 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 and 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 [4] and Glove embedding [5].

3.3 Teacher forcing

In the course, we have talked about teacher forcing technique (L10, slide 25-26), which input 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.

3.4 Other implementation details

- The encoder and decoder must be implemented by nn.GRU, otherwise you will get no point on your report.
- You cannot use *attention mechanism* in your implementation.
- The loss function is nn.CrossEntropyLoss() and the optimizer is SGD.
- You can refer to the official tutorial website [2, 3].
- Adopt BLEU-4 score function in NLTK [1] (provided in the sample code). The final score should average over all testing scores.
- Hyper-parameters and model setting.
  - RNN hidden size: 256 or 512

  - Teacher forcing ratio: $0 \sim 1$
  - Learning rate:: 0.05

4 Derivation of Back-Propagation Through Time (BPTT)

Given BPTT forward pass as follow,

$$\begin{aligned}
a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)}, \\
h^{(t)} &= \tanh(a^{(t)}), \\
o^{(t)} &= c + Vh^{(t)}, \\
\hat{y}^{(t)} &= \mathrm{softmax}(o^{(t)}), \\
L &= -\log p_{\mathrm{model}}(y^{(t)} | x^{(1)}, x^{(2)}, \cdots, x^{(t)})
\end{aligned} \tag{1}$$

derive $\frac{\partial L}{\partial w}$ step-by-step with clear notations. You can see more information in slides of recurrent neural network.(page 6-7)

5 Dataset Descriptions

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.

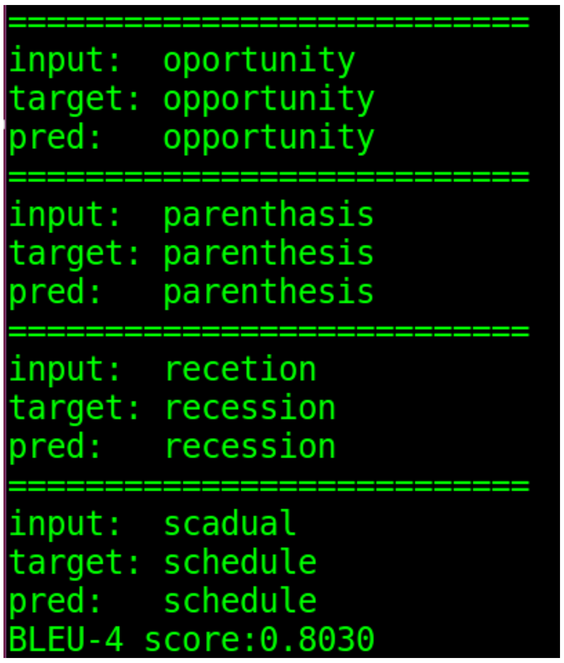
6 Scoring Criteria

1. Report (60%)
   - Introduction (5%)
   - Derivation of BPTT (5%)
   - Implementation details (30%)
     - Describe how you implement your model. (e.g. encoder, decoder, dataloader, etc.)
     - 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%)
     - Show your results of spelling correction and plot the training loss curve and BLUE-4 score testing curve during training. (10%)
     - Discuss your results or observation based on your model design or experimental settings.
2. Demo (40%)
   - Classification accuracy on test.json and new_test.json. (10% + 10%)

Accuracy	Grade
score $\geq$ 0.8	100%
0.8 > score $\geq$ 0.7	90%
0.7 > score $\geq$ 0.6	80%
0.6 > score $\geq$ 0.5	70%
0.5 > score $\geq$ 0.4	60%
score < 0.4	0%

   - Questions (20%)

7 Output Example



```
===========================
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
```

Figure 3: The output example of spelling correction

References

[1] Natural language toolkit. `https://www.nltk.org/`.

[2] Pytorch sequence-to-sequence network tutorial. `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`.

[3] Seq2seq github. `https://github.com/pytorch/tutorials/blob/master/intermediate_source/seq2seq_translation_tutorial.py`.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[5] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.