CE/CZ4045: Natural Language Processing

Assignment # 2

Assignment Logistics

- This is a **group assignment** with each group having 4 to 5 students.
- **One report** (only softcopy) is to be submitted by each group and all members in the same group will receive the same grade. However, contributions of each individual member to the assignment should be clearly indicated in the report.
- Softcopy submission: A CECZ4045A2-StudentID.zip file containing the following files and folder should be submitted: (i) Report.PDF, (ii) Readme.txt, (ii) SourceCode.
  - (i) Report.PDF should contain the written part.
  - (ii) Readme.txt should include instructions to run the code and explanations of sample output obtained from your code.
  - (iii) SourceCode folder should contain all your source code. The libraries shall NOT be included in the softcopy submission to minimize the file size.
- **Submission system:** Via NTULearn with subject "CECZ4045A2-StudentID".
- **Submission Due:** 6 Nov, 2020, before 6 pm (SGT). Late submissions are allowed until Nov 9 (6 pm), but will be penalized by 5% every calendar day. The softcopy can be submitted for at most three times, only the last submission will be graded and time-stamped.

1 Question One [30 marks]

A language model can predict the probability of the next word in the sequence based on the words already observed in the sequence. Neural networks are a preferred method for developing such language models because they can use a large context of observed words and use a distributed representation (as opposed to symbolic representation) of words. In this exercise, you will learn how to develop a neural language model in Pytorch. In particular, you will learn

(a) How to prepare texts for developing a word-based language model.

(b) How to design and fit/train a basic neural language model.

(c) How to use the learned language model to generate new texts.

The dataset for this exercise will be the wikitext-2 dataset. You can use this code as your codebase and build on top of it. Please do the following:

(i) Download the dataset and the code. The dataset should have three files: *train*, *test*, and *valid*. The code should have basic prepossessing (see data.py) and data loader (see main.py) that you can use for your work. Try to run the code.

(ii) You should understand the preprocessing and data laoding functions.

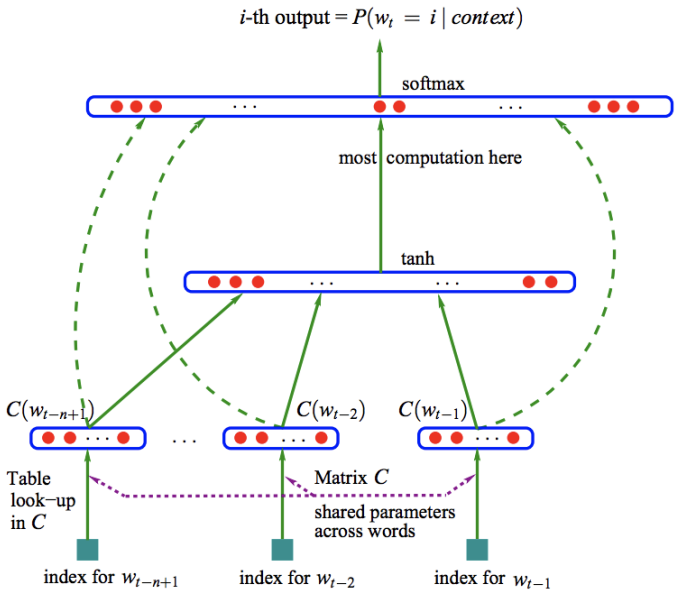

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Figure 1: A Neural Probabilistic Language Model by [1]

(iii) Write a class *class FNNModel(nn.Module)* similar to *class RNNModel(nn.Module)*. The FNNModel class should implement a language model with a feed-forward network architecture. For your reference, RNNModel implements a recurrent network architecture, more specifically, Long Short-Term Memory (LSTM) that you will learn later in the course.

The FNN model should have an architecture as shown in Figure 1. This is indeed the first (historically) neural language model [1].[1] The neural model learns the distributed representation of each word (embedding matrix **C**) and the probability function of a word sequence as a function of their distributed representations. It has a hidden layer with tanh activation and the output layer is a Softmax layer. The output of the model for each input of $(n-1)$ previous words are the probabilities over the $|V|$ words in the vocabulary for the next word.

(iv) Train the model with any of SGD variants (Adam, RMSProp) for $n = 8$ to train an 8-gram language model.

(v) Show the perplexity score on the test set. You should select your best model based on the perplexity score on the *valid* set.

(vi) Do steps (iv)-(v) again, but now with sharing the input (look-up matrix) and output layer embeddings (final layer weights).

(vii) Adapt generate.py so that you can generate texts using your language model (FNNModel).

[1] You do not need to read the paper to do this assignment.

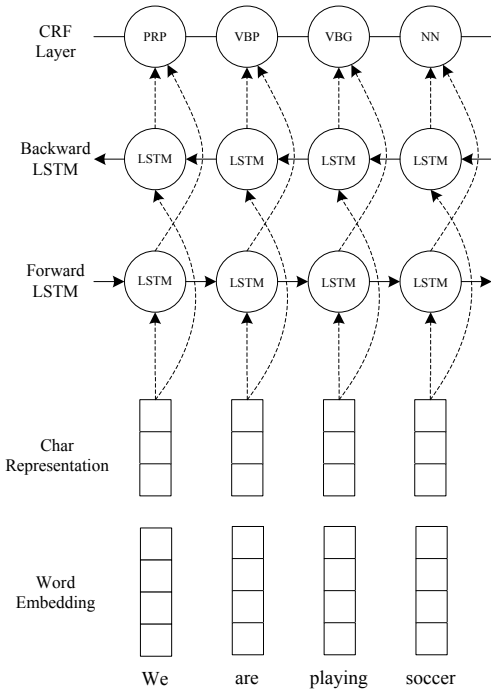

Figure 2: The CNN-LSTM-CRF Model for NER by [2]

2 Question Two [20 marks]

Named Entity Recognition (NER) is an important information extraction task that requires to *identify* and *classify* named entities in a given text. These entity types are usually predefined like *location*, *organization*, *person*, and *time*. In this exercise, you will learn how to develop a neural NER model in Pytorch. In particular, you will learn

(a) How to prepare data (input and output) for developing a NER model.

(b) How to design and fit/train a neural NER model.

(c) How to use the trained NER model to predict named entity types for a new text.

You can use this code as your codebase and build on top of it. This code implements the CNN-LSTM-CRF NER model described in [2]. The model has an architecture as shown in Figure 2.

The dataset for this exercise will be the standard CoNLL NER dataset, which is also available in the codebase (inside 'data' directory). Please do the following:

(i) Download the code repository. The dataset should have three files: *eng.train*, *eng.testb*, and *eng.testa* to be respectively used for training, testing, and validation. The dataset contains four different types of named entities: PERSON, LOCATION, ORGANIZATION, and MISC; it uses the BIO tagging scheme introduced in the lecture. Try to be familiar with the data and the BIO tagging scheme used in the data.

(ii) Run the code and see the results. The code performs basic prepossessing steps to generate tag mapping, word mapping and character mapping that you should use. You should understand the preprocessing and data laoding functions.

(iii) Go through the model part (get_lstm_features (..), class BiLSTM_CRF(nn.Module)). Notice that it implements a **charater-level encoder** with a *convolutional* neural network (CNN) and an LSTM recurrent neural network. The default setting uses a CNN for character-level encoding. Please read these implementations carefully.

(iv) The code implements a **word-level encoder** with an LSTM network. In its default setting, the output layer of the network has a CRF layer. You can change it to a regular Softmax layer. For now, leave it as it is (i.e., use CRF). Your job is to replace the LSTM-based word-level encoder with a CNN layer (convolutional layer followed by an optional max pooling layer). The CNN layer should have the same output dimensions (out_channels) as the LSTM.

(v) Report the testset results when you use only one such CNN layer in your network.

(vi) Now increase the number of CNN layers in your network and see the impact on your results. In each case, report the results on the testset (use the validation set to select the best model).

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[2] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th ACL*, pages 1064–1074, Berlin, Germany, August 2016. ACL.