CE7455: DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING - ASSIGNMENT # 2

Li, Yang
G2104961C - SCSE NTU
YANG047@e.ntu.edu.sg
Mar 15, 2022

QUESTION ONE

(i) Download the code repository. Try to be familiar with the data and the BIO tagging scheme used in the data.

The dataset we work with is English sentences with the NER tag information. The preprocessing step replaces all digits in the text by "0" to make the model concentrate on alphabets. The alphabets converts to lower cases to eliminate the difference of upper and lower. Two test files eng.testa and eng.testb are used as dev and test sets respectively.

The dataset's tagging scheme is BIO, in this study we are using BIOES tagging scheme. We need to convert the tag scheme to BIOES. This step is done in the update_tag_scheme function. We replace the tags for alphabets in specific location with E and S. Compare with BIO, BIOES scheme could potentially provide us more information.

After the conversion, our train, dev and test dataset are all in BIOES tag scheme.

(ii) Run the code and see the results. You should understand the preprocessing and data loading functions.

The prepare_dataset function convert the sentence in the dataset to a dictionary. This dictionary includes the processed list of test string, the word ids, the character ids and the tags ids. In this way, the information in strings is converted to integer ids and ready for further model training and testing.

We also need to load the word embeddings from pre-trained word embedding file. In this study, we use glove.6B.100d.txt file, which has 17493 words with 100 dimension each.

The existed code allows us to use "CNN" or "LSTM" for parameters['char_mode']. This is the mode for the character-level encoder. The existed code only has LSTM mode for word-level encoder.

Under the existed setting, I run the code with “CNN” and “LSTM” for parameters[‘char_mode’]. The results of F1 scores and training losses are presented in Figure 1. Limited by the GPU resource, all the training in this study is set to 30 epochs. Comparing the result and losses to the standard 50 epochs training, the results are identical.

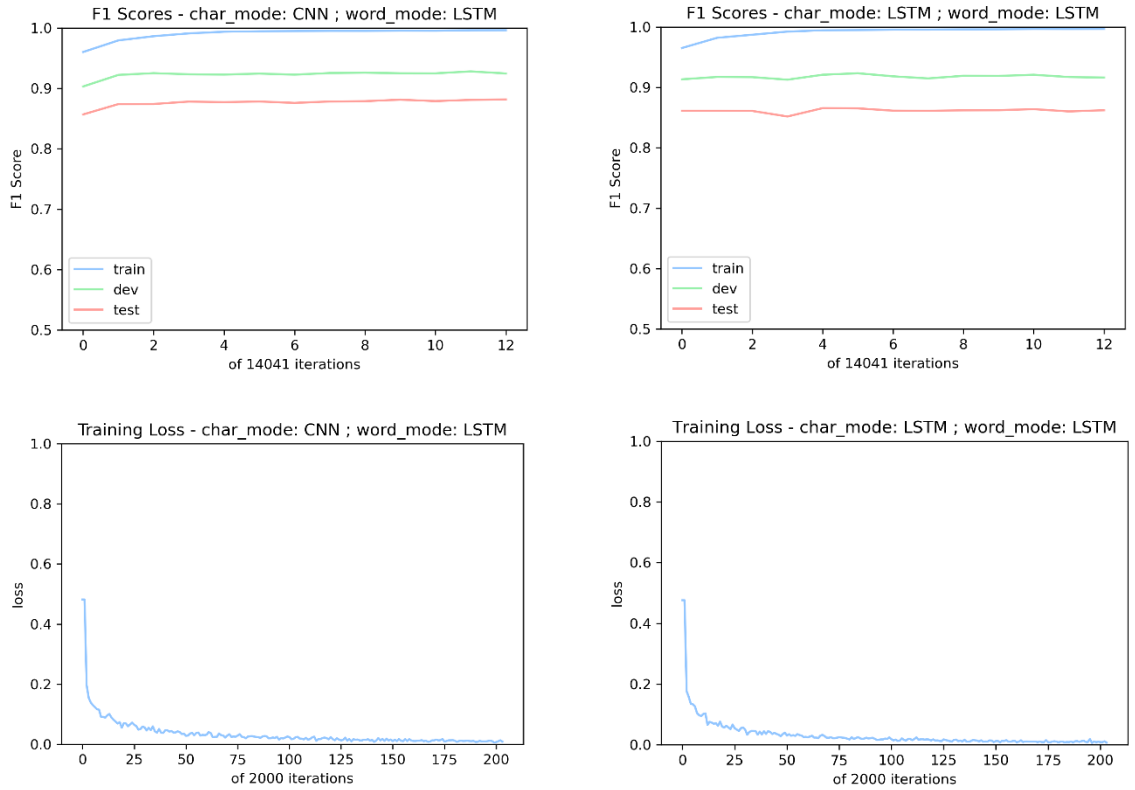

Figure 1. char_mode CNN word_mode LSTM vs. char_mode LSTM word_mode LSTM

The best F1 scores on test set:

Model Parameters	**Best F1 Score on Test Set**
char CNN; word LSTM	0.8813
char LSTM; word LSTM	0.8657

(iii) Go through the model part. Please read the default implementations carefully.

For the CNN model of the character embedding, the existed model padded each word based on the maximum word length of the sentence. The LSTM word embedding is created by implementing a nn.LSTM layer in the model BiLSTM_CRF class, the input channel is adapt to both LSTM and CNN char mode’s output.

The get_lstm_features implemented the date flow directions of model layers. The embedding feeding into the word LSTM layer incorporated both character embeddings and word embedding. The character embedding could come from either LSTM or CNN depends on its char mode selection.

The model class BiLSTM_CRF has other hyperparameters, such as char_out_dimension and char_embedding_dim, which can be turned if needed. Since we are using glove.6B.100d.txt pre-trained word embedding file, our word embedding dim is fixed to 100.

(iv) Replace the LSTM-based word-level encoder with a CNN layer. The CNN layer should have the same output dimensions (out_channels) as the LSTM

In order to replace the LSTM-based word-level encoder with a CNN layer. I created another hyperparameter parameters['word_mode']. Just like parameters['char_mode'], it can be selected from either “CNN” or “LSTM”. In cased of word mode selection is “LSTM”, the model would be running the existed code for LSTM word-level encoder.

To add the CNN level word encoder, I implemented a nn.Conv2d layer in the model class BiLSTM_CRF, its output channels is set to 2*parameters[‘word_lstm_dim’]. In function get_lstm_features, I take the concatenated word and character level representation and feed it into the newly added nn.Conv2d CNN layer. I also created a max_pool2d layer after the CNN to pool the results and match the dimension to the last layer hidden2tag, which outputs the final results.

(v) Report the test set results when you use only one such CNN layer in your network. Report results when you use an LSTM-based character-level encoder. In each case, report the number of parameters in your model.

Since we have implemented the one-layer CNN level word encoder. We are now able to run two more experiments, the results are showing in figure 2.

1. Char CNN; Word CNN
2. Char LSTM; Word CNN

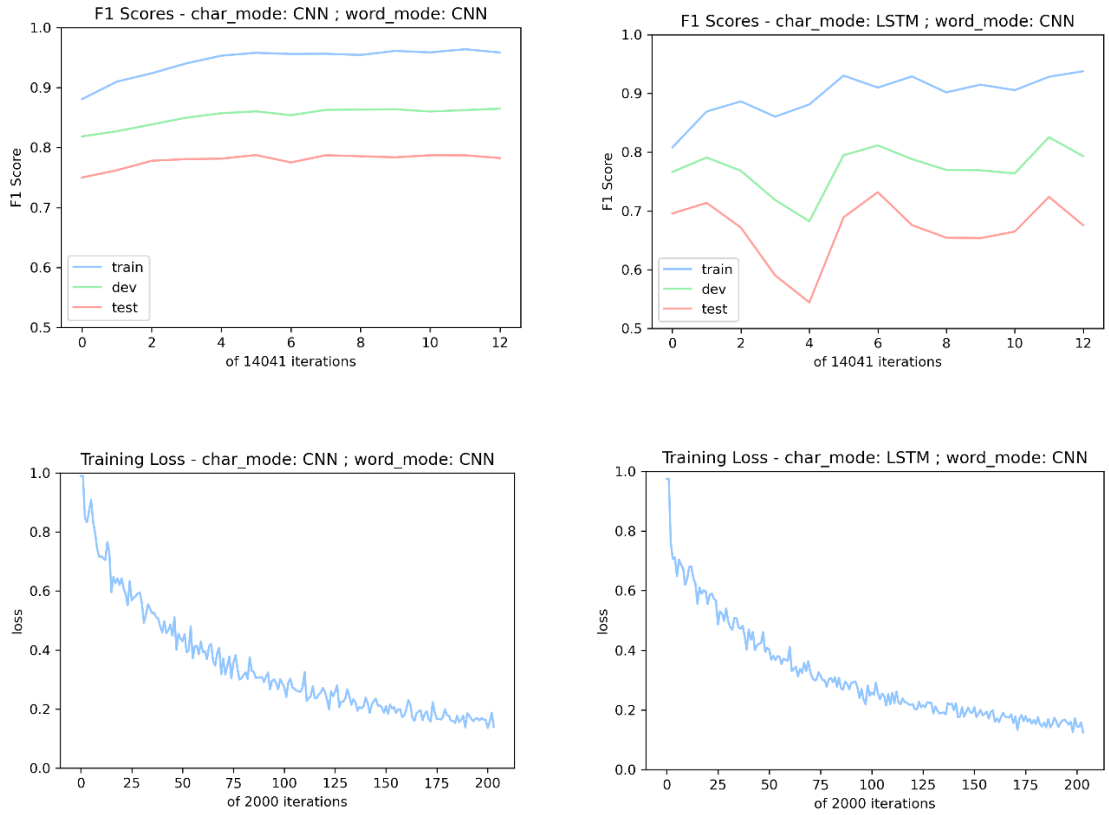

Figure 2. char_mode CNN word_mode CNN vs. char_mode LSTM word_mode CNN

The best F1 scores on test set:

Model Parameters	**Best F1 Score on Test Set**
char CNN; word CNN	0.7494
char LSTM; word CNN	0.7319

Parameters used in the model:

	char CNN; word CNN	**char LSTM; word CNN**
Char Encoder	char_out_dimension = 20 char_embedding_dim = 30	char_lstm_dim = 10
Word Encoder	Layer of nn.conv2d: word_embedding_dim =100 kernel_size = (3, word_embedding_dim) = (3, 100) Padding = (1, 0)	

From the results we can see, using a single layer of CNN replacing the LSTM word encoder caused the performance drop on both CNN and LSTM character encoder. One of the reasons could be a single layer of CNN does not have enough representation power. By increasing the depth of the CNN encoder, we would increase the representation capacity of the CNN layers, which could potentially improve the accuracy.

(vi) Increase the number of CNN layers in your network and see the impact on your results.

For the model to switch from different types of CNN word encoder, I incorporated a new parameter, parameters['word_cnn_mode']. It could be switched from 'single' – one layer CNN, 'double' – two layers of CNN and 'dilated' – three layers of dilated CNN(for Qvii).

For the word encoder to perform the best, the out channels for the first layer and the input of the second layer are set at the middle point of the final layer's out channel. I use a larger kernel size for the first layer, and a standard 3 by 3 kernel size for the second layer. This structure is likely to learn feature representation in different levels, so it might help in our case.

After switched to two layers of CNN, the performance did improve slightly from the one-layer model (figure 3).

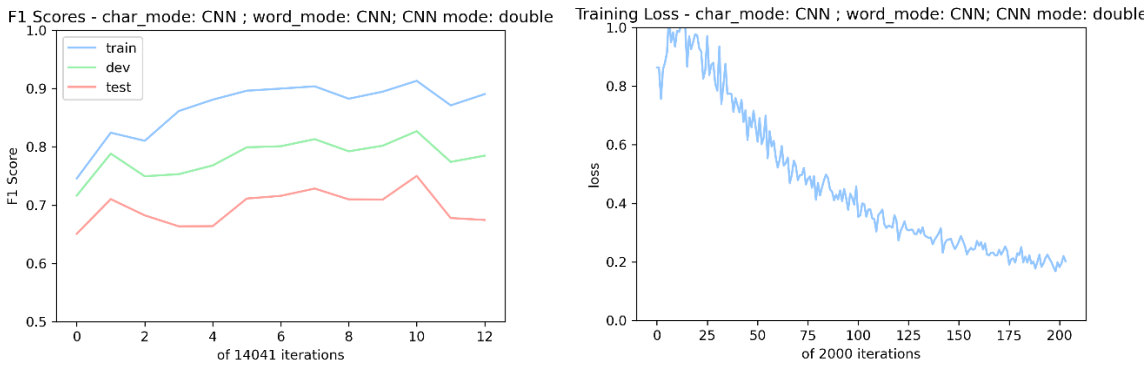

Figure 3. char_mode CNN word_mode CNN – double layer CNN

The best F1 scores on test set:

Model Parameters	**Best F1 Score on Test Set**
char CNN; word CNN – one layer (from Qv)	0.7494
char CNN; word CNN – two layers	0.7502

(vii) Replace the simple convolution layers in steps(iv-v) by dilated convolution layers. Report the NER results with the newly implemented dilated layers.

I added another mode for word cnn mode, "dialated". I implemented 3 layers of CNNs in the main model and in the get_lstm_features.

	in channels	out channels	kernel size	dilation
First layer	1	hidden dim	(3, word embedding dim) = (3, 100)	1
Second layer	hidden dim	2/3 hidden dim	(3, 3)	2
Third layer	2/3 hidden dim	2 hidden dim	(3, 3)	3

Dilated convolution delivers a wider field of view while keep the computational cost same, while we can not afford larger kernels this is a useful technique to expand the area of the input coverage without pooling.

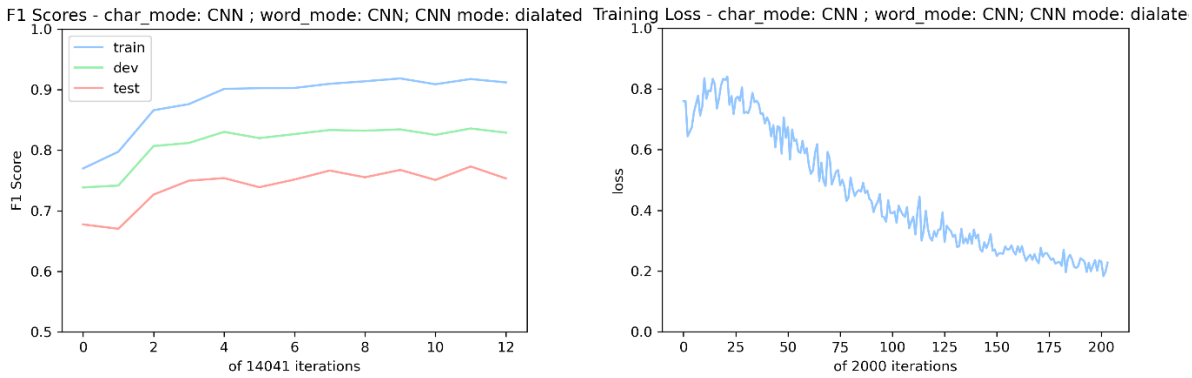

Figure 4. char_mode CNN word_mode CNN – dilated CNN layers

The best F1 scores on test set:

Model Parameters	Best F1 Score on Test Set	Improvement from base
char CNN; word CNN – one layer (base line)	0.7494	N/A
char CNN; word CNN – two layers	0.7502	0.101%
char CNN; word CNN – three layers (dilated)	0.7732	3.169%

Adding the 3 dilated CNN layers did help improve performance 3.169% from the baseline model (the one-layer CNN). Pooling has the same effects of keeping the computation cost the same but would reduce the resolution. For our NER NLP task, adding dilated convolution layers is a effective way of boosting performance.

(viii) Replace the CRF output layer with a softmax output and report the results.

The advantage of CRF over a regular Softmax is, CRF considered the neighboring words. As in our task the whole sequence of input is important information source, Softmax could leave important information behind.

Figure 5. char_mode CNN word_mode CNN – dilated CNN layers - CRF turns off

Model Parameters	**Best F1 Score on Test Set**	**Improvement**
char CNN; word CNN – three layers (dilated) CRF output layer	0.7731885348050561	N/A
char CNN; word CNN – three layers (dilated) Softmax output layer	0.6559383119604392	-15.165%

The results match with our expectation. From the training losses we can see the loss is much lower, but F1 score is not improving. This meaning the model is not learning as it should. NLP task is different from other deep learning applications such as object detection. Incorporate the information in sequence with CRF and Viterbi algorithm help with our NER task.

(ix) Please summarize your observations.

I summarized the best F1 scores and the training time for all the models in the study in Figure 6 and Figure 7. From the previous comparison we come to the below conclusions:

1. LSTM is computationally expensive compare with CNN. This is due to RNNs are very memory intensive in backpropagation, since we want our RNN to remember more back in time. LSTM also provides good results, there is not CNN word encoder outperform a LSTM word encoder.
2. Adding layers to CNN can improve its representation capacity enhance improve the performance. However, in our NER NLP task, simply adding layers does not boost the

performance as much as dilated CNN layers. Dilated CNN layers can expand the range of input while keep the computation cost relatively low. It is helpful to us while we cannot use a larger kernel size.

3. In learning tasks that sequence contains important information, CRF is a better choice than the standard Softmax. Softmax assumes that the likelihood of the tag is conditionally independent given the RNN states, thus is not suitable for our NER task.

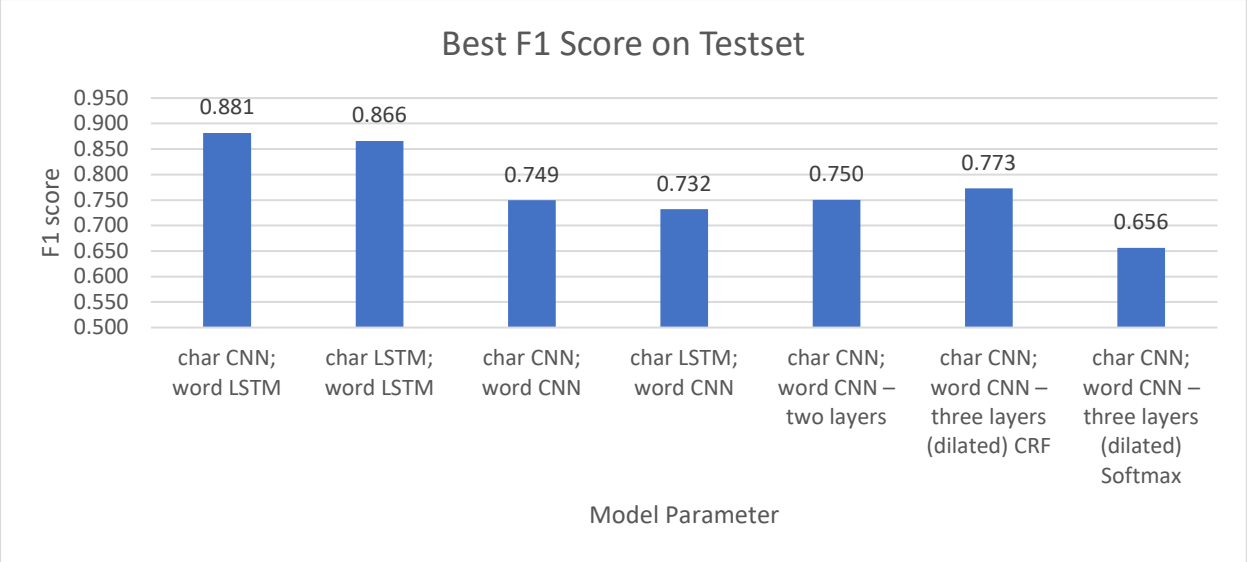

Figure 6. the Best F1 scores on test – all models

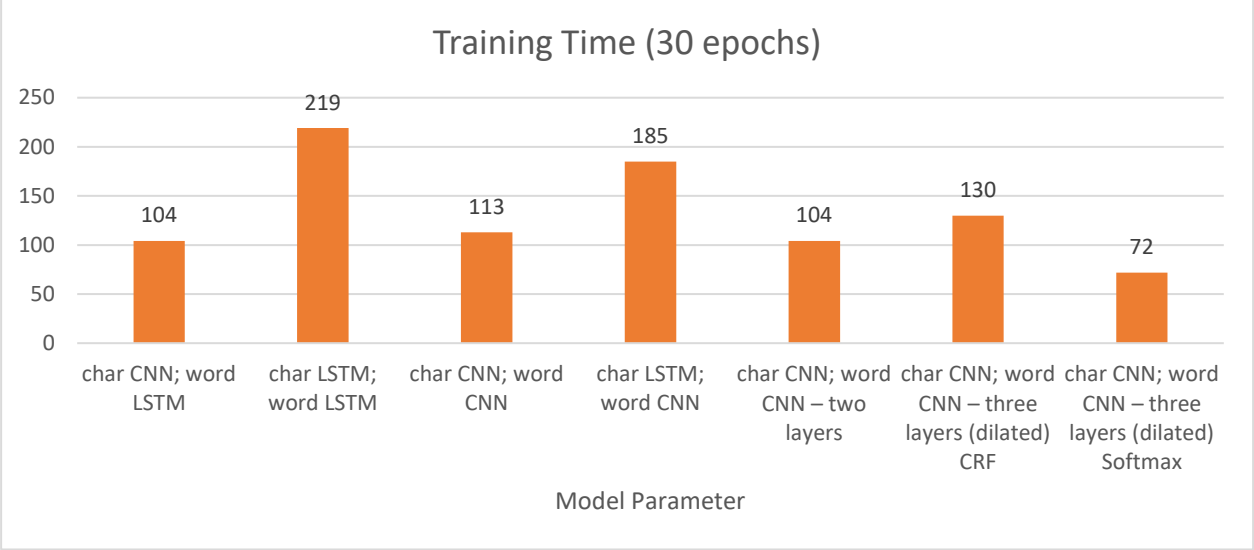

Figure 7. Training Time (30 epochs) – all models

QUESTION TWO

To help me in future planning with the assignment exercises, please mention how much time you spent on each of the questions.

Question #	Time Spent
i	4 hour
ii	3 hours
iii	10 hours
iv	40 hours
v	2 hours
vi	4 hours
vii	3 hours
viii	2 hours
ix	5 hours