ADL HW1 Report

- Q1:Data processing (Use sample code.)
  - a. How do you tokenize the data:
    - intent_cls: I use “encode_batch” function in the sample code (utils.py -> class Vocab). To begin with, the Vocab is built by “preprocess_intent.py” given by TAs, they use the most common word in train and eval data based on vocab_size. This “encode_batch” function aims to tokenize a batch of sentences and pad the sentences to same length (the length of the longest sentence). Besides, since the “encode_batch” function tokenize sentences by characters, I create a new function called “encode_batch1” which tokenize sentences by words. I concatenate the characters’ token and the words’ token before embedding.
    - slot_tag: I use “encode_batch1” which created by myself and this function tokenize sentences by words. Besides, since I try some different models like CNN-BiGRU, I create another function called “encode_char”, this function helps me to tokenize character based on words in a sentence and pad them to the same length (I set the length to 15).
  - b. The pre-trained embedding you used.
    - The pre-trained embedding is also given by TAs, both intent_cls and slot_tag use “glove.840B.300d”.

- Q2:Describe your intent classification model.
  - a. model:
    1. gru_out, h = GRU(word and character embedding of their token)
    2. out = concatenate(gru_out[:,0],gru_out[:,-1]),1). Since I set bidirectional to True, I concatenate the first GRU out and the last GRU out to get more information.
    3. final_out = Linear(out). This is final output so that return it.
  - b. performance of model: The public score is 0.90888.
  - c. loss function: CrossEntropyLoss.

- d. The optimization algorithm: The algorithm I used is “Adam”. The batch size is 32. The initial learning rate is 0.001 and I adjust learning rate every 10 epochs (the learning rate would * 0.1 for every 10 epochs).

- Q3:Describe your slot tagging model.
  - a. model:
    1. char_representation, h = GRU(char_embedding). I feed char embedding into GRU.
    2. new_input = Dropout(Concatenate(char_representation, word_embedding)). Concatenate the char_representation and the word embedding and then dropout.
    3. gru_out, h = GRU(new_input). Feed the new_input into GRU.
    4. out = Linear(gru_out). Feed gru_out into linear layer.
    5. final_out = log_softmax(gru_out). This is final output so that return it.
  - b. performance of model: The public score is 0.788.
  - c. loss function: The loss function is created by myself. I use the concept of CrossEntropyLoss but sum up all the loss of labels except padding words. Finally, return average of loss.
  - d. The optimization algorithm: The algorithm I used is “Adam”. The batch size is 32. The initial learning rate is 0.0005 and I adjust learning rate every 10 epochs (the learning rate would * 0.1 for every 10 epochs).

- Q4:Sequence Tagging Evaluation:
  - classification_report:

	precision	recall	f1-score	support
date	0.80	0.79	0.80	206
first_name	0.96	0.99	0.98	102
last_name	0.93	0.83	0.88	78
people	0.75	0.74	0.75	238
time	0.89	0.87	0.88	218
micro avg	0.84	0.83	0.83	842
macro avg	0.87	0.85	0.86	842
weighted avg	0.84	0.83	0.83	842

- Differences between the evaluation method in seqval token accuracy, and joint accuracy (I use true positive as TP, true negative as TN, false positive as FP, and false negative as FN below):
  - accuracy_score: (TP+TN)/(number of data). When the labels of data are unbalance, accuracy score would not be a good measurement. (Ex: 100 data with 10 positive and 90 negative, if the model always output negative and it can get high accuracy.)
  - precision_score: (TP)/(TP+FP). This measurement focus on number of false positive, which means that it cares about false alarm.
  - recall_score: (TP)/(TP+FN). This measurement focus on number of false negative, which means that it cares about missing rate.
  - f1_score: 2 ∗ Precision ∗ Recall / (Precision + Recall). This measurement integrates Precision and Recall.
  - token accuracy: (number of correct labels) / (label number of all the data). This measurement focuses more on label of each word.
  - joint accuracy: (number of correction) / (number of data). Note that all the labels should be correct in a sentence. This measurement focuses more on whole sentence.

- Q5:Compare with different configurations
  - Improvement and Bonus tricks:
    - About learning rate: I define a function called adjust_learning_rate to adjust learning rate every 10 epoch.
    - For slot tagging, I try 3 different models:
      - Bi-GRU with word embedding:
        1. gru_out, h = GRU(word embedding)
        2. final_out = log_softmax(Linear(gru_out))
      - CNN + Bi-GRU (word and char embed):
        1. char_encode = CNN(char_token). I use 4 different kernel sizes in CNN. So, there are 4 convolution layers and I

concatenate these output, feed them into a Linear layer, and return it.

2. word_char_embed = concatenate(char_encode, word_embedding).
3. gru_out, h = GRU(word_char_embed).
4. final_out = log_softmax(Linear(gru_out)).

- RNN for character representation + Bi-GRU:
  1. char_representation, h = GRU(char_embedding). I feed char embedding into GRU.
  2. new_input = Dropout(Concatenate(char_representation, word_embedding)). Concatenate the char_representation and the word embedding and then dropout.
  3. gru_out, h = GRU(new_input). Feed the new_input into GRU.
  4. out = Linear(gru_out). Feed gru_out into linear layer.
  5. final_out = log_softmax(gru_out). This is final output so that return it.
- The table contains some experiment. I finally choose “RNN for character representation + Bi-GRU” model since it performs good result.

	hidden size	dropout	Others	Highest validation accuracy
GRU (word embed)	600	0.3		0.782
GRU (word embed)	1000	0.3		0.808
GRU (word embed)	1000	0.1		0.814
CNN + GRU (word and char embed)	600	0.3	active func in linear layer: Relu	0.781
CNN + GRU (word and char embed)	600	0.3	active func in linear layer: tanh	0.783
CNN + GRU (word and char embed)	1000	0.3	active func in linear layer: Relu	0.801
RNN for character representation + Bi-GRU	1000	0.3		0.797
RNN for character representation + Bi-GRU	1000	0.3	without dropout embedding.	0.798
RNN for character representation + Bi-GRU	1000	0.3	change the num_layer of char representation from 2 to 3	0.825