PA3 Report

Kevin Gao

a) Structure of my code

My code is in the jupyter notebook submitted. And my notebook can be divided into four parts: Data loading, Data iterator, Training and Evaluation, and Experiments. To run my code, just run through all the cells

Here is the Structure:

```
PA3
|-------- Data loading
|         |-------- read_in_gold_data(filename: str): Read in the labeled gold data into a list
|                   from a file
|         |-------- read_in_plain_data(filename:str): Read in plain text data for sequence
|                   labeling, assuming a one-sentence-per-line style
|         |-------- Construct a vocab using the data read in
|         |-------- get_glove_vocab(embedding_module): Generate the vocab using GloVe
|
|-------- Data iterator
|         |-------- batchify(iterable, n): Generate n batches of data
|         |-------- class CustomBucketSampler: Assign sentences with similar length to the
|         |         same bucket
|         |         |--------  __init__(self, dataset, buckets, batch_size):
|         |         |-------- __len__(self): return the lenth of the dataset
|         |         |-------- __iter__(self): Assign data to appropriate buckes, and assign the
|         |                   with length higher the the max boundary to the last bucket
|         |         |-------- collate_fn(batch): control how to batchify data using from-corpus
|                              embedding
|         |         |-------- collate_fnGlove(batch): control how to batchify data using GloVe
|
|-------- Training and Evaluation
|         |-------- train(model, dl_dataset, num_epoches, pad_idx): train the given model
|         |         using corresponding dataset for given epochs, yield the loss and accuracy
|         |         for each epoch
|         |-------- evaluate(model, dl_dataset, tag_pad_idx): Evaluate the trained model
|         |         using given validation set
|
|-------- Experiments
|         |-------- Experiment 1
|         |         |-------- class BiLSTM(nn.Module):
|         |         |         |-------- __init__(self, input_dim, output_dim, device, src_pad_idx)
|         |         |         |-------- init_hidden(self, batchSize): initialize the hidden layers
|         |         |         |-------- forward(self, x): forward propagation
|         |         |-------- class BiLSTM_greedy(nn.Module):
|         |         |         |-------- __init__(self, intput_dim, output_dim, device, src_pad_idx)
|         |         |         |--------loss_compute(self, src, tgt, criterion): compute the nll loss
|         |         |         |--------forward(self, preds, y, tag_pad_idx): calculate the forward
|         |         |         |         score
|         |         |-------- class BiLSTM_Glove(nn.Module):
|         |         |         |-------- __init__(self, input_dim, output_dim, device, src_pad_idx)
|         |         |         |-------- init_hidden(self, batchSize): initialize the hidden layers
|         |         |         |-------- forward(self, x): forward propagation
|         |         |-------- class BiLSTM_Glove_greedy(nn.Module):
|         |         |         |-------- __init__(self, intput_dim, output_dim, device, src_pad_idx)
|         |         |         |--------loss_compute(self, src, tgt, criterion): compute the nll loss
|         |         |         |--------forward(self, preds, y, tag_pad_idx): calculate the forward
|         |         |         |         score
|         |-------- Experiment 2
|         |         |-------- class CharEmbeddings(nn.Module)
|         |         |         |--------__init__(self, char_input, char_embed_dim, char_pad_idx)
|         |         |         |--------forward(self, x, raw, char_lenght, char_max_length)
|         |         |-------- class CharCNN(nn.Module)
|         |         |         |--------__init__()
|         |         |         |--------forward()
|         |         |-------- class BiLSTM_CNN(nn.Module)
|         |         |         |--------__init__()
|         |         |         |--------init_hidden()
|         |         |         |--------forward()
|         |         |-------- class BiLSTM_CNN_greedy(nn.Module)
|         |         |         |--------__init__()
|         |         |         |--------loss_compute()
|         |         |         |--------forward()
|         |-------- Experiment 3
|         |         |--------class CRF(nn.Module)
|         |         |         |--------__init__()
|         |         |         |--------reset_parameters()
|         |         |         |--------forward()
|         |         |         |--------score(): calculate emission score and transcore
|         |         |         |--------viterbi()
|         |         |-------- class BiLSTM_CRF(nn.Module)
|         |         |         |--------__init__()
|         |         |         |--------neg_log_likelihood()
|         |         |         |--------forward()
```

(b) Settings and Results

This experiment is conducted on my own laptop, and following are the plots of the results:

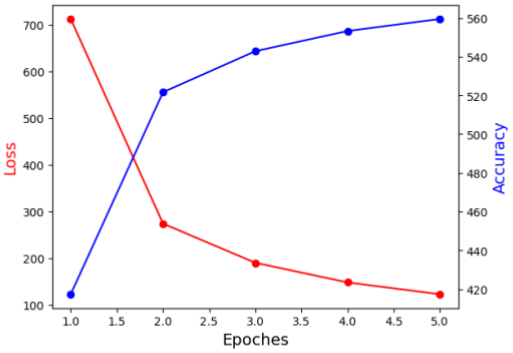

Fig 1: Performance curves of the BiLSTM with random embedding

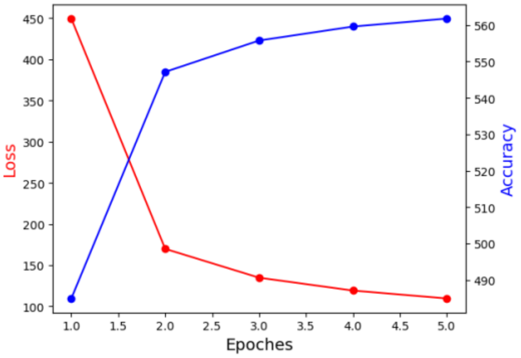

Fig 2: Performance curves of the BiLSTM with GloVe embedding

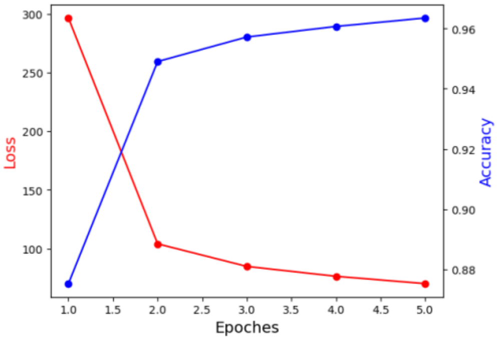

Fig 3: Performance curves of the BiLSTM with CharCNN embedding

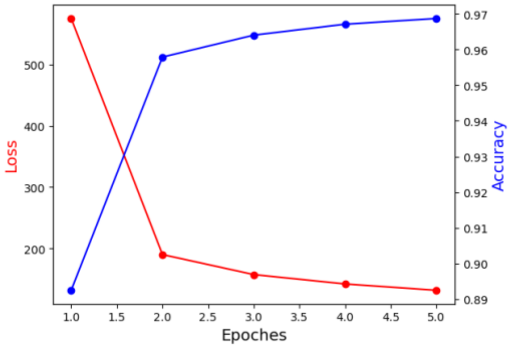

Fig 4: Performance curves of the BiLSTM with CRF

All 4 experiments have reached a relatively high high accuracy at the 2nd epoch, with CRF-BiLSTM and CharCNN-BiLSTM reaching accuracy of .96. For the 1st epoch, CRF-BiLSTM has the best accuracy of .89, which outreaches the 1st-epoch accuracy of BiLSTMs with only individual encoding, respectively .74 and .64. The performance of contextual embedding is much better than the individual word embedding from the perspective of accuracy. However, it takes much longer time to train the contextual models.

(c) Insights

The CharCNN and CRF has a better performance implies that the suffices and the words in the neighborhood can help determine the label. Also they are slower Because I used CPU for the whole notebook. CPU is much more suitable for the sequential model (experiment 1), and when GPU is available, we can use GPU to accelerate the computation in the case of parallel computing.