CSCI 544 Deep Learning 2 Report
Ziyin Zhang
ziyinz@usc.edu

1 Model Architecture

The model I finally used, in Keras' equivalence (not completely equivalent to TensorFlow's), was:

```
model = Sequential()
model.add(InputLayer(input_shape = (MAX_LENGTH, )))
model.add(Embedding(n_voc, 128, mask_zero = True))
model.add(Bidirectional(LSTM(96, return_sequences = True)))
model.add(TimeDistributed(Dense(n_tag)))
model.add(Activation('softmax'))
```

A summary of the model is as following:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 114, 128)          3031680
_________________________________________________________________
bidirectional_1 (Bidirection (None, 114, 192)          172800
_________________________________________________________________
time_distributed_1 (TimeDist (None, 114, 37)           7141
_________________________________________________________________
activation_1 (Activation)    (None, 114, 37)           0
=================================================================
Total params: 3,211,621
Trainable params: 3,211,621
Non-trainable params: 0
_________________________________________________________________
```

When implementing in TensorFlow, you can sum up all trainable variables and make sure the number of trainable params is as same as Keras' equivalent model shows.

2 Hyperparameters

There were in total 4 hyperparameters to determine: the batch size; the learning rate of the Adam optimizer; the input dropout rate (dropout rate in Keras) of the LSTM cell and the state dropout rate (recurrent dropout rate in Keras) of the LSTM cell.

Unlike Keras, which doesn't have built-in support to change hyperparameters during training unless using callbacks or customized layers, we can easily change the hyperparameters by feeding the desired numbers as long as we use placeholders to build them.

3 Training Process

For each language, I trained the model for 5 epochs. After 5 epochs, `train_epoch()` will return `False` and thus stop training.

The following is a summary of the choice of hyperparameters for each epoch:

	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
batch size	32	32	64	64	64

learning rate	$10^{-1.9}$	10^{-3}	$10^{-2.1}$	10^{-3}	$10^{-2.5}$
input dropout	0	0	0	0	0
state dropout	0	0	0.3	0.3×0.4	0.3×0.4^2

4 Refinement: Tried and Worked

4.1 Using bidirectional network

Do it. You'd be paid.

4.2 Using LSTM cell

If your program runs fast enough, i.e. your program does not worry about exceeding time limit, do it. You'd be paid.

4.3 Fine tune using smaller or cyclical learning rate

The learning rate I used was based on Cyclical Learning Rates for Training Neural Networks. However, I manually set the learning rate. The learning rate was oscillating between 10^{-3} and 10^{-2}, with a roughly exponential decay of the maximum learning rate.

4.4 Input dropout

It did work, but not so obvious. My experiments showed the LSTM-based model was sensitive to it. It may be attributed to that the prediction made by the model come more from the historical memory, and as LSTM can retrain longer term of memory, a masked input may prevent enough information fed to the model.

4.5 State dropout

My experiments showed that if we used dropout (input or recurrent) as the first (or second) epoch of training, the convergence speed would be largely slowed down. Besides, keeping a constant dropout rate at every epoch was actually not a good idea. The model performed better than it did without dropout after several epochs, but then its loss went up and accuracy went down. It could be viewed as somehow overfitting. Because the memory network was starved for input. So, to trade off convergence speed and performance, I first trained the model without dropout, then used an exponentially decaying dropout rate to complete remaining training.

4.6 Adjusting the hidden size of the LSTM cell.

Experiment showed that 90~108 were good choices for the hidden size. 128 is more prone to overfitting.

4.7 Early Stopping

Maybe the dataset is "easy", my model converged fast and reached the highest testing accuracy after only several epochs. Experience showed that 5 epochs was enough. I didn't have time to implement early stopping so I just let `train_epoch()` down after 5 epochs.

5 Tried but Didn't Worked

5.1 L1/L2 regularization on dense layer

Experimented, but seemed to not have obvious effect. Not very sure for the reasons…

5.2 Adjusting the embedding size

128 seemed to be a magic number. Other numbers did not perform so well…