PRML Assignment 4

1 Experiment Setting

- **Core Packages**: Pytorch, FastNLP, numpy and _pickle. Their versions are shown in Table 1.
- **Python Version**: 3.6

Package	FastNLP	torch	numpy	gensim	glove-python	_pickle
Version	0.4.1	0.2.2	1.16.2	3.7.3	0.1.0	-

Table 1: Core Packages

2 Dataset

2.1 Dataset Selection

The 20 newsgroups text dataset is the main dataset of this assignment. I have splited the original training data into new training data and new validation data by ratio 0.2. **During my experiments, I did not only run my models on a part of the dataset as what we did in assignment 2, but also explore how well my models would perform on the whole 20 classes dataset.**

2.2 Data Preprocessing

I use DataSet and Vocabulary, which are very helpful tools of fastNLP, to help me preprocess and pack the data. And then **_pickle** tool is used for storing my datasets and vocabulary objects as .pkl files. Here I will give main details about my preprocessing.

1. Clean the Sentence Data. Just as what we did in Assignment2, I firstly do cleaning to each sentence. For example, punctuations, extra whitespaces, some special characters like '\' will be removed.

Listing 1: Main Code for Cleaning Sentences

```
input_data = (input_data).lower()
input_data = re.sub("\d","", input_data)
input_data = re.sub(r"[{}]+".format(string.punctuation)," ",input_data)
input_data = re.sub(r"[{}]+".format(string.whitespace)," ",input_data)
input_data = input_data.replace('\\', " ")
input_data = re.split(' ', input_data)
input_data.remove('')
```

2. Pack as DataSet Object. I wrap the word lists as the form of Instance. After doing that, it will be easy to append my data into a certain fastNLP.DataSet.

Listing 2: Append Data into DataSet

```
dataset = DataSet()
for i,input_data in enumerate(data):
    input_data = clean_data(input_data)
    dataset.append(Instance(target = int(target[i]), words = input_data))
```

3. Build Word Vocabulary and Map Word to Index. This work is quite easy if we ultilize fastNLP.Vocabulary. Codes are shown as bellow.

Listing 3: Build Vocabulary and Do Mapping

```
vocab = Vocabulary(min_freq=10).from_dataset(dataset, field_name='words')
vocab.index_dataset(dataset, field_name='words',new_field_name='input_data')
vocab.index_dataset(test_data, field_name='words',new_field_name='input_data')
```

4.Split dataset into train dataset and Validation dataset. My final step is to split the dataset into train data and validation data properly. Split ratio I choose is 0.8, which means 80% of the original training data will become my formal training data, and the other 20% data will become validation data.

3 Model

1. CNN. CNN is widely used not only in CV demain, but also in many NLP tasks. I construct a simple CNN model for this text classification task. As we can see in Figure 1, the path **Word Embedding** >> **Convolutional Layers** >> **FC** is my CNN architecture this time. Details of the parameters setting are as bellow.

Parameters	Embedding Dim	Kernel Sizes	Kernel Nums	Dropout	Batch Size	Learning Rate
Setting	256	3,4,5	100,100,100	0.5	32	1e-3

Table 2: CNN Model Parameters Setting

2. LSTM & B-LSTM. As we can see in Figure 1, the path of **Word Embedding** >> **LSTM Layers** >> **FC** mainly summarized my LSTM architecture. On the basic of LSTM, I reverse the input data and run through another LSTM Cells. After then, two output features will be connected as a whole feature and run through a fc layer.

Parameters	Embedding Dim	Layer num	Hidden Size	lstm Dropout	Batch Size	Learning Rate
Setting	256	2	256	0.5	32	1e-3

Table 3: LSTM & B-LSTM Model Parameters Setting

3. Bert. I tried bert model just for fun. Since there exists implementation in fastNLP, it is easy to use it for the classification task. But there are many parameters that need to be modified.

For example, the default fitted input length of the pre-trained Bert is 512, but in our dataset, there exists many long passages. Some passages length are even more than 10000, and because of the padding operation and passage length, the computation while training will occupy too many resources that we might not affordable. So I finally cut all the passages' length under 2500 and set maxPositionEmbeddings = 2500. The parameters of my bert model are set as the below table shows.

Parameters	Vocab Size	Hid Size	Num Hid	Num Atten	Itermediate	Drop	M-Pos-Embed	lr
Setting	256	512	2	2	512	0.1	2500	1e-4

Table 4: BERT Model Parameters Setting

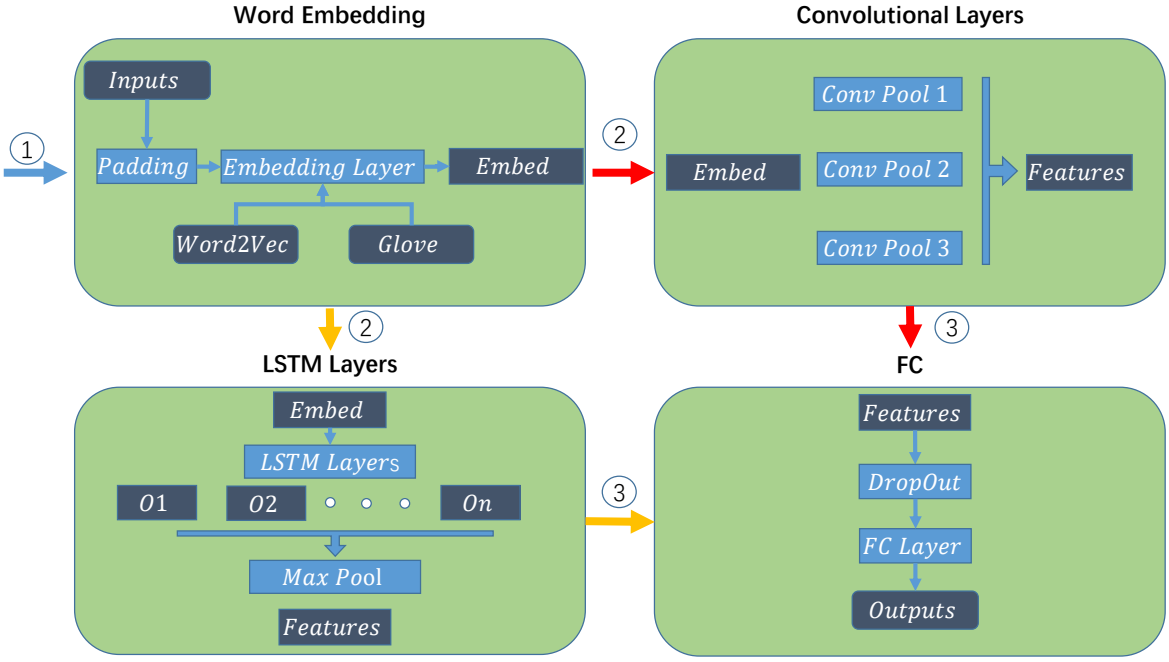

Figure 1: This figure shows the main architecture of my CNN Model and LSTM Model.

4 Pooling Way of LSTM Output

There are multiple ways to ultilize the LSTM Cells outputs. **Mean Pooling, Max Pooling, Min Pooling or even just use the last step outputs**. I have done some experiments on different pooling ways and the comparison results are recorded in Table 5. No matter using LSTM or B-LSTM, max pooling and min pooling always correspond to the best results. So in the following experiments, I choose to use max pooling method to deal with the outputs of the LSTM cells.

Model	Embedding Way	Pooling Way	Dataset	Accuracy
LSTM	Word2Vec	Max Pool	Large	**0.824349**
LSTM		Min Pool		**0.823686**
LSTM		Mean Pool		0.795273
LSTM		Output[-1]		low
B-LSTM		Max Pool		**0.843999**
B-LSTM		Min Pool		**0.840813**
B-LSTM		Mean Pool		0.818109
B-LSTM		Latest Output		low

Table 5: Comparison of different kinds of pooling ways.

5 Word2Vec & Glove

Word Embedding is an significant part of our models. The embedding quality will inference the final model performances a lot, which is also proved by the following experiments. In this assignment, I have seperately tried **Word2Vec** and **Glove** Algorithms to construct the word embeddings. These embedding vectors will be put into nn.embedding layer as its initial value. Implementation details are illustrated below.

I use Gensim and glove-python packages to train word2vec embeddings and glove embeddings. Since they may not perfectly fit the nerual network layers after the embedding layer, I use wrod2vec embedding matrix or glove matrix as the initial value of torch.nn.embedding, but still let the embedding matrix values update when training the nerual network.

In Table 6 and Table 7, we can see that word2vec and glove methods really make a difference. And here I will briefly list the my steps of using Word2Vec. Steps of Glove are nearly the same. If you want more details, you can watch the code in packdata_word2vec.py and packdata_glove.py files.
s

1. word2vec training. Firstly, I build a sentence list. Then I use gensim.model.word2vec to train a word2vec mapping table.

Listing 4: word2vec training

```
sentence_list = []

for i in dataset:
  sentence_list.append(i['words'])

model = word2vec.Word2Vec(sentence_list , hs=1,min_count=10,window=64,size=256)
```

2. **Store word2vec matrix**. Using pikle package, we can store the matrix as the form of pkl file. And it is easy to load it back.

Listing 5: word2vec mapping table store

```
import _pickle as pickle

pickle.dump(embedding_weight , open("./dataset/embedding_weight.pkl", "wb"), 2)
```

3. **torch.nn.embedding initialization**. As is said above, the word2vec matrix mainly used to initialize the torch.nn.embedding weight.

Listing 6: word2vec mapping table store

```
import _pickle as pickle

if use_word2vec:
  self.embeddings.weight.data.copy_(
    torch.from_numpy(
      pickle.load(open(embedding_weight_path , 'rb'))
    ))
```

6 Experiment Results

1. On a four classes dataset. Compare Logistic Model, CNN, LSTM, B-LSTM model.
2. On a 20 classes dataset. Compare Logistic Model, CNN, LSTM, B-LSTM model.

As we can see in Table 6 and Table 7, using word2vec or glove to initial the embedding layer is actually a good way to improve model performances, which also shows the importance of weights initialization for nerual network. And B-LSTM with word2vec gains the highest accuracy while testing.

Model	Embedding Way	Embedding Size	Accuracy
CNN	torch.nn.Embedding	256	0.93984
LSTM			0.92246
B-LSTM			0.93382
CNN	Word2Vec	256	0.951872
LSTM			**0.937166**
B-LSTM			**0.956551**
CNN	Glove	256	**0.953209**
LSTM			0.921123
B-LSTM			0.945187
Logistic	Bag of Words	-	0.933155
Small Bert + fc	-	-	**0.949198**

Table 6: Experiment Results on Small Data

Model	Embedding Way	Embedding Size	Accuracy
CNN	torch.nn.Embedding	256	0.775757
LSTM		256	0.745751
B-LSTM		256	0.777881
CNN	Word2Vec	256	0.802974
LSTM		256	**0.824349**
B-LSTM		256	**0.843999**
CNN	Glove	256	**0.820499**
LSTM		256	0.774164
B-LSTM		256	0.827138
Small Bert + fc	-	-	**0.817977**

Table 7: Experiment Results on Big Data

7 Code Structure

Filename	Function
packdata.py	利用fastNLP处理打包数据，存为pkl格式，包括训练集、测试集、词表
packdata_word2vec.py	利用fastNLP和word2vec算法处理打包数据，存为pkl格式
packdata_glove.py	利用fastNLP和 glove 算法处理打包数据，存为pkl格式
utils.py	工具函数定义（包括数据集获取，small参数设置为true为获取小数据）
models.py	模型定义
config.py	训练时参数设置
train.py	模型训练与保存
test_config.py	测试时参数设置
test.py	测试模型

Table 8: Code illustration.

8 About fastNLP

一些小建议

1. 创建的Dataset对象中，保存了所有数据并都存在内存中，如果遇到大规模数据可能较难处理,内存不一定存放的下。在处理超大数据集时，可以类似仿照Tensorflow读取tfrecords格式文件的方式，通过创建缓冲队列来完成数据的读取。

2. 当验证集比较大时，验证时间常常占据很大一部分的训练时间，一次验证可能要花费非常长的时间，而在进行验证集验证时训练过程也相应停止，这会导致整体训练时间延长。我认为验证部分可以通过开多线

程或者多进程的方式监视进行。验证过程中不希望影响训练过程。不过这个情况在本次实验不明显。

3. Trainer最终保存的是验证集上表现最优的结果，而其他训练过程中的模型没有得到保存，但是，最优模型附近保存的模型参数通常是非常有用的，一般只要通过简单的参数融合技巧能够使得模型得到提升。所以我认为fastNLP的trainer过程也许可以多传一些参数来指定最终将会连续保存的模型数量。

4. 希望在保存模型的同时也保存模型训练中的各种中间训练参数，并提供在意外中断后能够方便恢复训练的功能。