Report for HW4

By Linsen Li

Task 1: Document Classification (50 points) Use the same datasets as Assignment 1. Classify text paragraphs into three categories: Fyodor Dostoyevsky, Arthur Conan Doyle, and Jane Austen by building your own classifiers. The data provided is from Project Gutenberg.

1.1 Size of the training set and validation set

We have a dataset of 2758 examples:

	paragraph	category
0	part i book i the history of a family chapter ...	0.0
1	alexey fyodorovitch karamazov was the third so...	0.0
2	he was married twice and had three sons the el...	0.0
3	immediately after the elopement adela da ivano...	0.0
4	one would think that you d got a promotion fyo...	0.0
...	...	...
927	in spite of his being allowed once more to liv...	2.0
928	i will not say that i am disappointed my dear ...	2.0
929	the whole of lucy s behaviour in the affair an...	2.0
930	creating the works from public domain print ed...	2.0
931	1 c the project gutenberg literary archive fou...	2.0

2758 rows × 2 columns

- The total number of examples for category 0 is: 952
- The total number of examples for category 1 is: 874
- The total number of examples for category 2 is: 932

We split our data into:

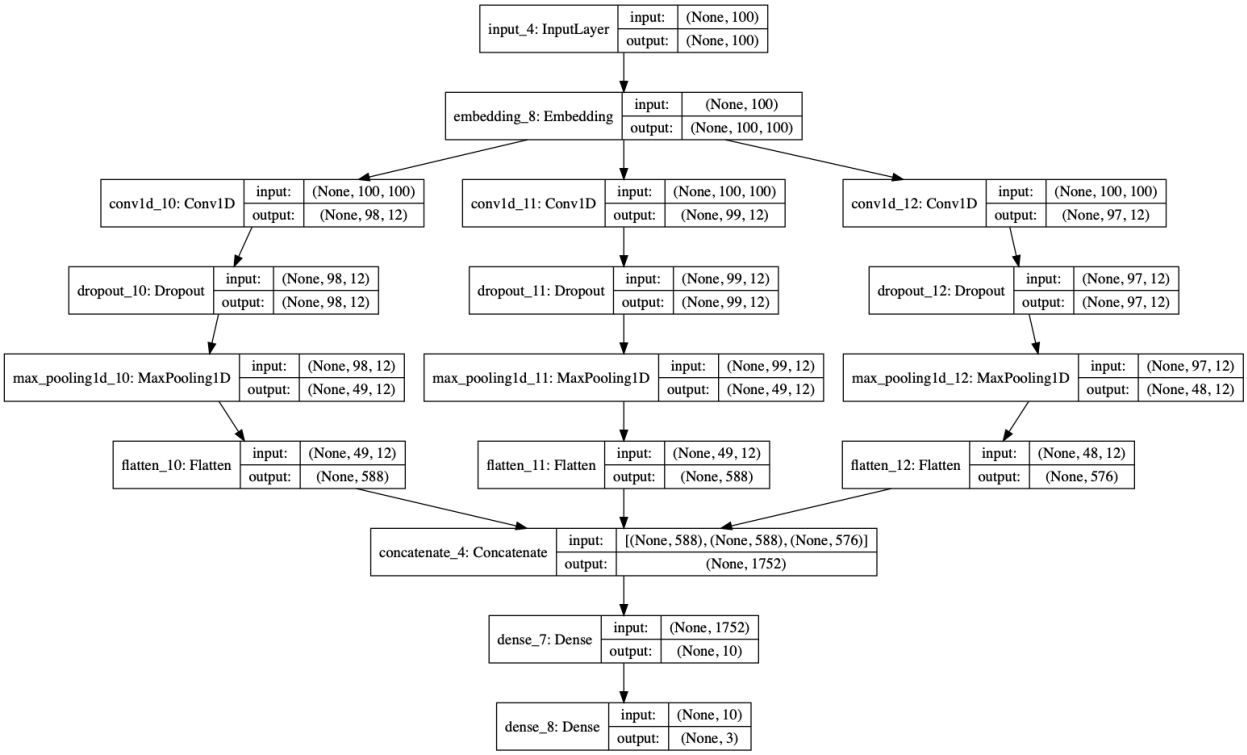
```
Shape of x_train: (1964, 100)
Shape of y_train: (1964, 3)
Shape of x_validation: (104, 100)
Shape of y_validation: (104, 3)
Shape of x_test: (690, 100)
Shape of y_test: (690, 3)
```

- Size of the training set: 1964-by-100
- Size of the validation set: 104-by-100

1.2 Parameters for the model

We Develop a Multichannel CNN Model to implement text classification. For task 1 we use three channels. Each channel includes a convolutional layer (each layer includes convolution, activation, dropout, maxpooling, flatten).

The following is the structure of our CNN model:

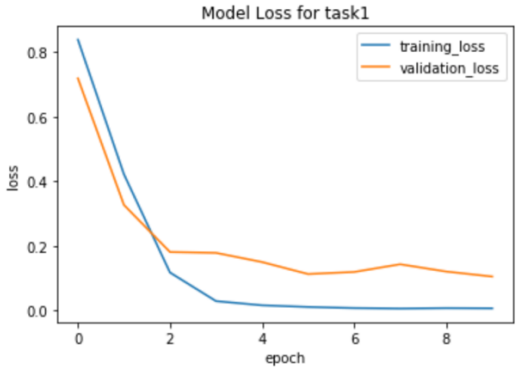


We have filter1, filter2, filter3:

- Filter1: number of filters is 12, filter size is 3
- Filter2: number of filters is 12, filter size is 2
- Filter3: number of filters is 12, filter size is 4

We use “categorical_crossentropy ” as our loss function. We use ‘adam’ as our optimizer with learning rate as 0.001.

1.3 Plot for training and validation loss



We train the model for only 10 epochs because we find that the loss converges very soon. There is no need to train for more epochs. Fortunately, we didn't have the overfitting problem.

1.4 Report recall and precision for task 1

```
In [14]: loss_and_acc = model.evaluate([X_test], label_test)
         print('loss = ' + str(loss_and_acc[0]))
         print('accuracy = ' + str(loss_and_acc[1]))

690/690 [==============================] - 0s 153us/step
loss = 0.11386912482912126
accuracy = 0.9550724625587463
```

```
In [16]: # Report the recall and precision for each category on the test sets
         from sklearn.metrics import classification_report
         y_pred_NN = model.predict([X_test], batch_size=16, verbose=1)
         y_pred_bool = np.argmax(y_pred_NN, axis=1)
         print(classification_report(y_test, y_pred_bool))

690/690 [==============================] - 0s 284us/step
              precision    recall  f1-score   support

         0.0       0.94      0.96      0.95       231
         1.0       1.00      0.92      0.96       219
         2.0       0.94      0.98      0.96       240

    accuracy                           0.96       690
   macro avg       0.96      0.95      0.96       690
weighted avg       0.96      0.96      0.96       690
```

The recall and precision for our model on test set is above. We find that our model preforms robust on test set.

Task 2: Sentiment Analysis (50 points) This is a multi-domain sentiment dataset with positive or negative sentiment annotations. We only use the book reviews for this assignment. There are 1000 positive book reviews and 1000 negative book reviews.

2.1 Size of the training set and validation set

We have a dataset of 2000 examples: 1000 positive reviews and 1000 negative reviews:

	reviews	category
0	Bridget Jones, modern day woman, brillant and ...	1.0
1	I am ordering copies for all 23 middle school ...	1.0
2	As a casual piano player and a Broadway fanati...	1.0
3	This is one of the best biographies I have eve...	1.0
4	I read this book many, many years ago on a ver...	1.0
...	...	...
995	This book provides excellent information about...	0.0
996	I really didn't enjoy this book. I am half It...	0.0
997	This book is a place to start at best. The ma...	0.0
998	I was barely able to finish this book. Armstro...	0.0
999	"The Burning" was a big letdown after reading ...	0.0

2000 rows × 2 columns

We split our data into:

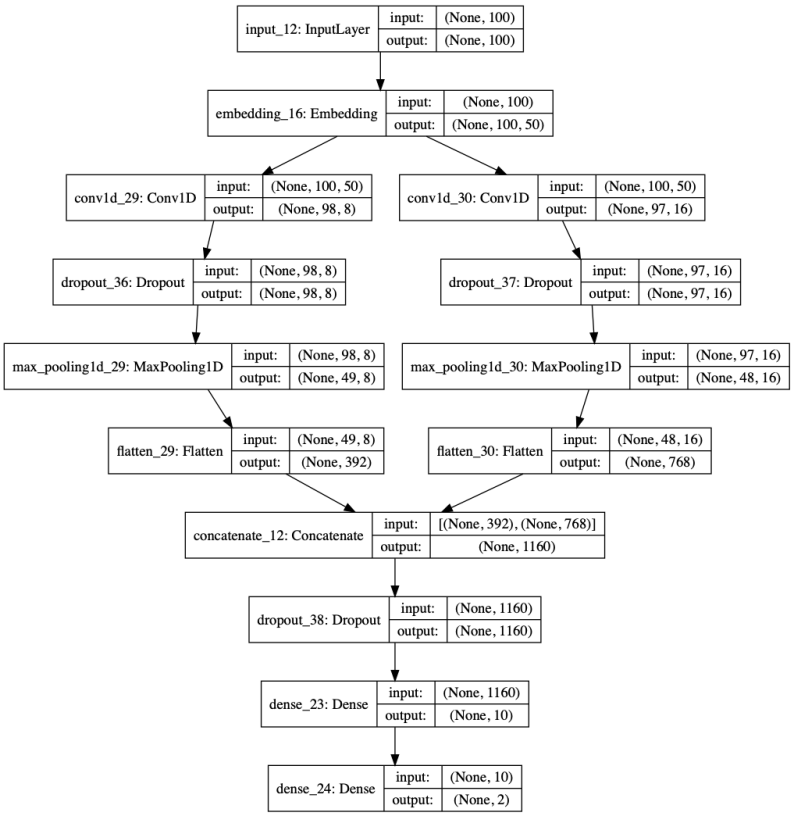
```
Shape of x_train: (1620, 100)
Shape of y_train: (1620,)
Shape of x_validation: (180, 100)
Shape of y_validation: (180,)
Shape of x_test: (200, 100)
Shape of y_test: (200,)
```

- Size of the training set: 1620-by-100
- Size of the validation set: 180-by-100

2.2 Parameters for the model

We also use multichannel CNN in task 2. However, in order to avoid the overfitting problem, we only use two channels. Each channel includes a convolutional layer (each layer includes convolution, activation, dropout, maxpooling, flatten). We also use a very high dropout rate as 0.8 before the dense layer. The main reason is that we only have 1620 sample in our training data, while the number of the parameters is over 1 million.

The following is the structure of our CNN model:

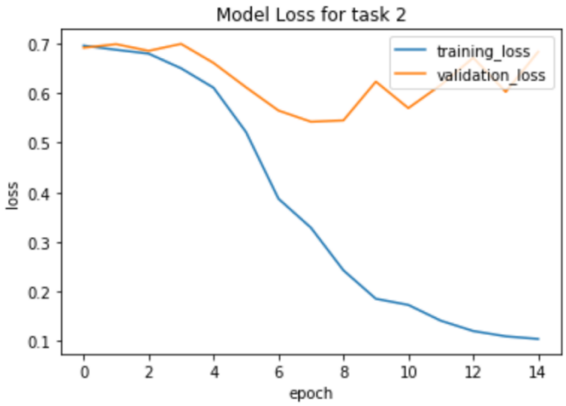


We have filter1, filter2:

- Filter1: number of filters is 8, filter size is 3
- Filter2: number of filters is 16, filter size is 4

We use “binary_crossentropy” as our loss function. We use ‘adam’ as our optimizer with learning rate as 0.001.

2.3 Plot for training and validation loss



We train the model for 15 epochs. We find that there is still overfitting problem. But our CNN model is as simple as it can be: we only have two CNN layers. As a result, we think the problem is caused by the embedding layer.

2.4 Report accuracy score for task 2 on test data.

```
200/200 [==============================] - 0s 98us/step
loss = 0.555784466266632
accuracy = 0.7450000047683716
```

```
|: # Report the recall and precision for each category on the test
from sklearn.metrics import classification_report
y_pred_NN = model.predict([x_test], batch_size=16, verbose=1)
y_pred_bool = np.argmax(y_pred_NN, axis=1)
print(classification_report(y_test, y_pred_bool))
```

```
200/200 [==============================] - 1s 3ms/step
              precision    recall  f1-score   support

         0.0       0.73      0.73      0.73        94
         1.0       0.76      0.75      0.76       106

    accuracy                           0.74       200
   macro avg       0.74      0.74      0.74       200
weighted avg       0.75      0.74      0.75       200
```

We test our model on the 200 samples test set. The test accuracy is 0.745. The result is not very acceptable although it is the best we can get.

2.5 Some improvement for model: Using Pre-Trained GloVe Embedding

As mentioned before, in order to get rid of the overfitting problem in the embedding layer, we use the pre-trained glove embedding. Now we set all parameters in embedding layer untrainable. As a result, we only have to train the parameters in convoluted layers and dense layers.

Firstly, we need to build our own embedding layer. We need the “glove.6B.100d.txt” file. This file contains 400000-word vectors whose dimension is 100:

```
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

```
# create a weight matrix for words in training docs
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
embedding_matrix.shape
```

```
(22348, 100)
```

Then, we build the similar CNN model as before, the only different is we use this pre-trained embedding layer. And the result is below:

```
# evaluate the model
loaded_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
loss, accuracy = loaded_model.evaluate(x_test, y_test_c, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

```
Accuracy: 81.999999
```

```
# Report the recall and precision for each category on the test sets
from sklearn.metrics import classification_report
y_pred_NN = loaded_model.predict([x_test], batch_size=16, verbose=1)
y_pred_bool = np.argmax(y_pred_NN, axis=1)
print(classification_report(y_test, y_pred_bool))
```

```
200/200 [==============================] - 2s 8ms/step
              precision    recall  f1-score   support

         0.0       0.80      0.83      0.81        94
         1.0       0.84      0.81      0.83       106

    accuracy                           0.82       200
   macro avg       0.82      0.82      0.82       200
weighted avg       0.82      0.82      0.82       200
```

We find that the test accuracy has been improved to 0.819.