Homework 2

Yixing Chen

1. General Information

1.1 Purpose

The purpose of this assignment for me is to explore and compare the methods to reduce overfitting. The main methods used here to reduce overfitting are limiting the number of hidden units, limiting the size of the weights and stopping the learning before it has time to overfit.

1.2 Dataset

Like assignment 1, the dataset I use for this assignment is 50,000 IMDB movie reviews. The dataset is downloaded from http://ai.stanford.edu/~amaas/data/sentiment/. All the reviews of this dataset are binary sentiment classification. The text reviews are labelled as 1 or 0 for positive or negative judgment, respectively.

```
import pandas as pd
import numpy as np
import tensorflow as tf
data = pd.DataFrame()
data = pd.read_csv('movie_data.csv', encoding='utf-8')

X_train = data.loc[:24999, 'review'].values
y_train = data.loc[:24999, 'sentiment'].values
X_val = data.loc[25000:, 'review'].values
y_val = data.loc[25000:, 'sentiment'].values
```

25000 reviews are used as training dataset and the other 25000 reviews are used as validation dataset.

2. Architecture and Code of Homework1

2.1 Word Embedding

Word Embedding used in this assignment is generated. All reviews need to be encoded as a unique representation. The text corpus of the reviews is vectorized after turning each text into a sequence of integers. The Python package used here is Keras. After all the words are transformed into integers, every review is converted to a vector of integers.

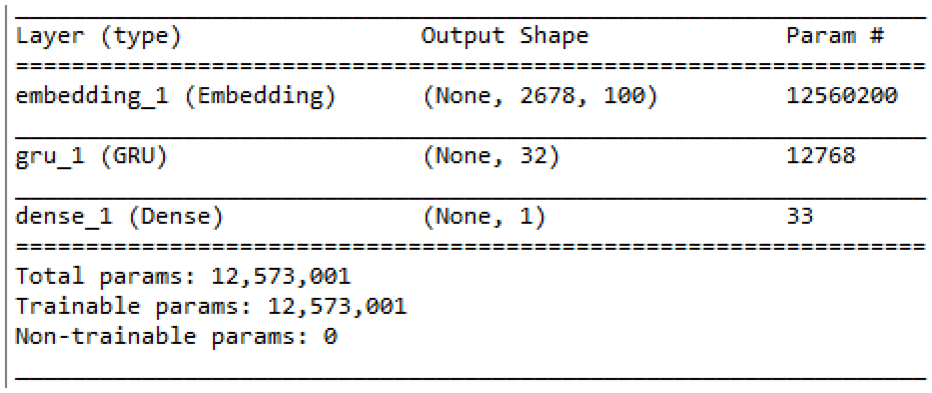
All the vectors are defined to have the same length. The length is *(max_length)*, which is the length of the text which has most words. In this assignment, this value equals 2678. For the vector whose length is less than 2678, ‘0’s are supplemented before it.

```
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.preprocessing.sequence import pad_sequences

tokenizer_object = Tokenizer()
total_reviews = X_train + X_test
tokenizer_object.fit_on_texts(total_reviews)

# pad sequences
max_length = max([len(s.split()) for s in total_reviews])
#max_length=500

# define vocabulary size
vocab_size = len(tokenizer_object.word_index) + 1

X_train_tokens =   tokenizer_object.texts_to_sequences(X_train)
X_test_tokens = tokenizer_object.texts_to_sequences(X_test)


X_train_pad = pad_sequences(X_train_tokens, maxlen=max_length, padding='pre')
X_test_pad = pad_sequences(X_test_tokens, maxlen=max_length, padding='pre')
```

The vocabulary size *(vocab_size)* of network input is the number of all vocabulary in Keras plus 1. The dimension of embedding layer *(embedding_dim)* is set to 100.

2.2 Building Model

The parameters of the model in homework1 is shown as the followings:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 2678, 100)         12560200
_________________________________________________________________
gru_1 (GRU)                  (None, 32)                12768
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 12,573,001
Trainable params: 12,573,001
Non-trainable params: 0
_________________________________________________________________
```

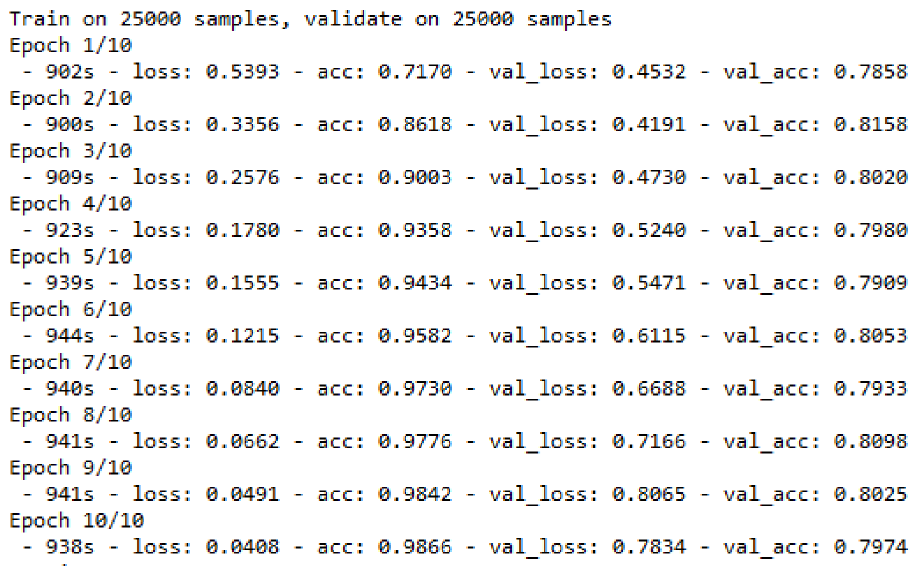
```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 100

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(GRU(units=32,    dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

2.3 Training Model

The neural network model is trained with training dataset and the 25,000 text dataset is used to validate the accuracy of the model, which is not ever used in training process. The batch size is set to 128. The training epoch is set to 10. The learning rate is the default value, 0.001.

```
# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model.summary())

print('Train...')

model.fit(X_train_pad, y_train, batch_size=128, epochs=10,
validation_data=(X_test_pad, y_ val), verbose=2)
```

The training process is shown as the following:



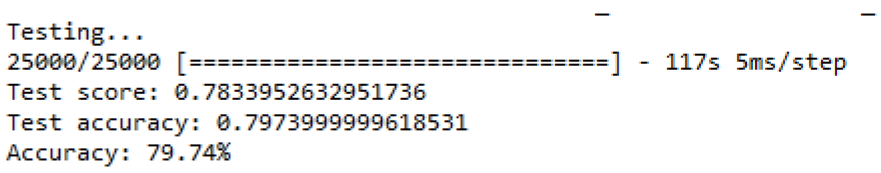
```
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 902s - loss: 0.5393 - acc: 0.7170 - val_loss: 0.4532 - val_acc: 0.7858
Epoch 2/10
 - 900s - loss: 0.3356 - acc: 0.8618 - val_loss: 0.4191 - val_acc: 0.8158
Epoch 3/10
 - 909s - loss: 0.2576 - acc: 0.9003 - val_loss: 0.4730 - val_acc: 0.8020
Epoch 4/10
 - 923s - loss: 0.1780 - acc: 0.9358 - val_loss: 0.5240 - val_acc: 0.7980
Epoch 5/10
 - 939s - loss: 0.1555 - acc: 0.9434 - val_loss: 0.5471 - val_acc: 0.7909
Epoch 6/10
 - 944s - loss: 0.1215 - acc: 0.9582 - val_loss: 0.6115 - val_acc: 0.8053
Epoch 7/10
 - 940s - loss: 0.0840 - acc: 0.9730 - val_loss: 0.6688 - val_acc: 0.7933
Epoch 8/10
 - 941s - loss: 0.0662 - acc: 0.9776 - val_loss: 0.7166 - val_acc: 0.8098
Epoch 9/10
 - 941s - loss: 0.0491 - acc: 0.9842 - val_loss: 0.8065 - val_acc: 0.8025
Epoch 10/10
 - 938s - loss: 0.0408 - acc: 0.9866 - val_loss: 0.7834 - val_acc: 0.7974
```

2.4 Validating Model

To validate the model, 25,000 test reviews are utilized. The batch size is 128. The metrics to be evaluated by the model during training and testing is accuracy. The accuracy is 79.74% after 10 epochs.

```
print('Testing...')
score, accuracy = model.evaluate(X_test_pad, y_ val, batch_size=128)

print('Test score:', score)
print('Test accuracy:', accuracy)

print("Accuracy: {0:.2%}".format(accuracy))
```

The result is shown as the following:



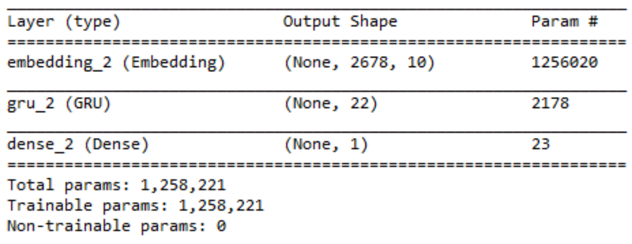
```
Testing...
25000/25000 [==============================] - 117s 5ms/step
Test score: 0.7833952632951736
Test accuracy: 0.7973999999618531
Accuracy: 79.74%
```

3. Limit the number of hidden units

The first method to reduce overfitting is limiting the number of hidden units. The experiment in homework1 sets the GRU units to 32. To validate the effect of hidden units, GRU units are set to 22 and 12, and the experiments are executed again. Thus, in the third line of the following code, GRU units are modified to 22 and 12, respectively.

```
model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(GRU(units=22,    dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

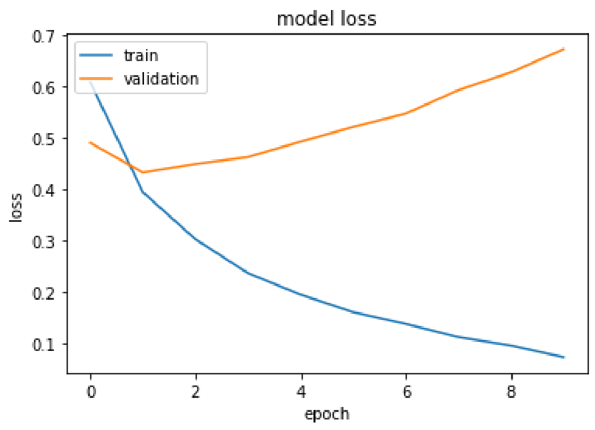
The parameters of 22 units are shown as the followings:



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 2678, 10)          1256020
_________________________________________________________________
gru_2 (GRU)                  (None, 22)                2178
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 23
=================================================================
Total params: 1,258,221
Trainable params: 1,258,221
Non-trainable params: 0
```

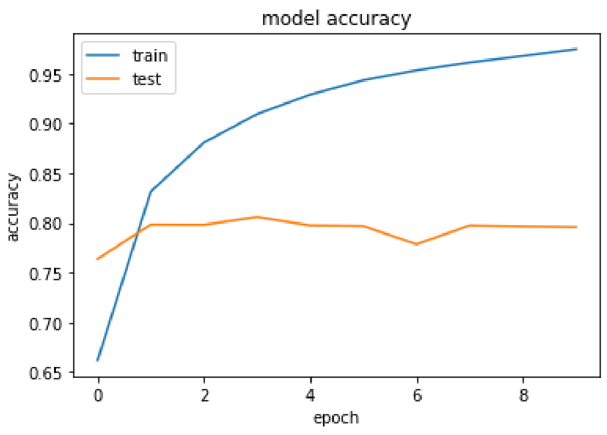
The training process is shown as the followings:

```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 342s - loss: 0.6088 - acc: 0.6616 - val_loss: 0.4910 - val_acc: 0.7637
Epoch 2/10
 - 334s - loss: 0.3951 - acc: 0.8315 - val_loss: 0.4331 - val_acc: 0.7983
Epoch 3/10
 - 336s - loss: 0.3036 - acc: 0.8809 - val_loss: 0.4497 - val_acc: 0.7981
Epoch 4/10
 - 344s - loss: 0.2376 - acc: 0.9096 - val_loss: 0.4634 - val_acc: 0.8059
Epoch 5/10
 - 333s - loss: 0.1958 - acc: 0.9291 - val_loss: 0.4932 - val_acc: 0.7975
Epoch 6/10
 - 338s - loss: 0.1617 - acc: 0.9436 - val_loss: 0.5218 - val_acc: 0.7969
Epoch 7/10
 - 346s - loss: 0.1389 - acc: 0.9536 - val_loss: 0.5478 - val_acc: 0.7784
Epoch 8/10
 - 352s - loss: 0.1132 - acc: 0.9615 - val_loss: 0.5935 - val_acc: 0.7974
Epoch 9/10
 - 351s - loss: 0.0968 - acc: 0.9680 - val_loss: 0.6280 - val_acc: 0.7964
Epoch 10/10
 - 355s - loss: 0.0743 - acc: 0.9748 - val_loss: 0.6730 - val_acc: 0.7958
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

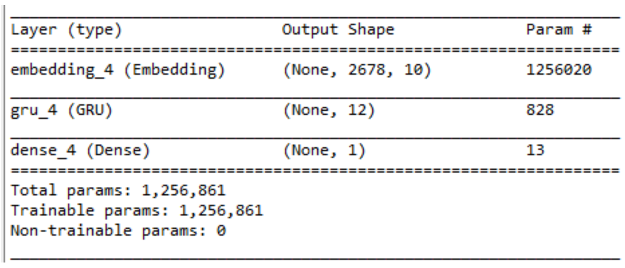


The validation result is shown as the followings:

```
Testing...
25000/25000 [==============================] - 55s 2ms/step
Test score: 0.6729865133714676
Test accuracy: 0.7958399999618531
Accuracy: 79.58%
```

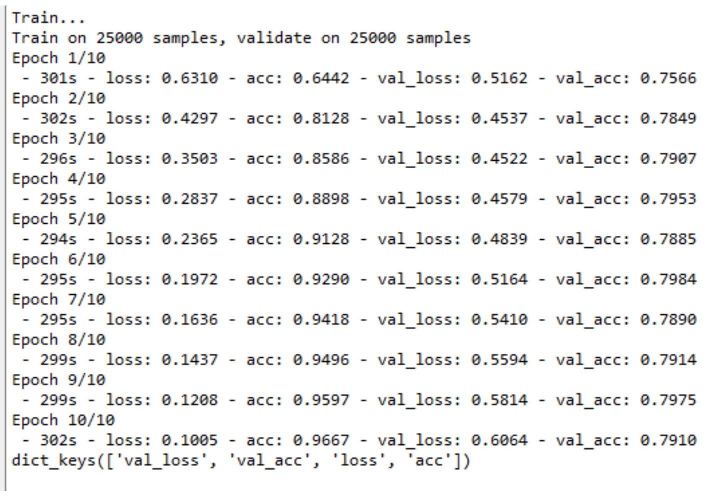


The parameters of 12 units are shown as the followings:

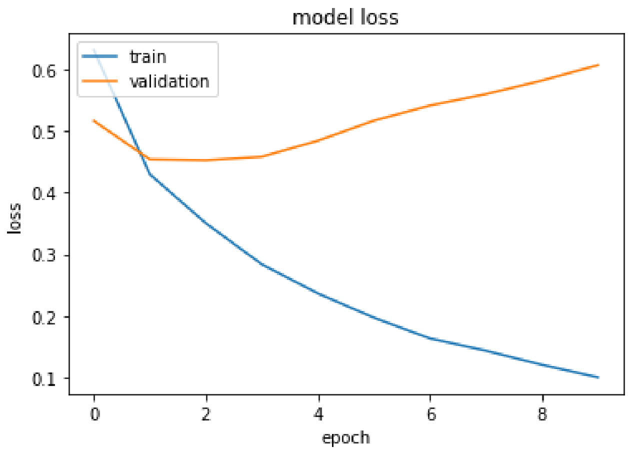


```
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_4 (Embedding)      (None, 2678, 10)          1256020   
_________________________________________________________________
gru_4 (GRU)                  (None, 12)                828       
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 13        
=================================================================
Total params: 1,256,861
Trainable params: 1,256,861
Non-trainable params: 0
_________________________________________________________________
```

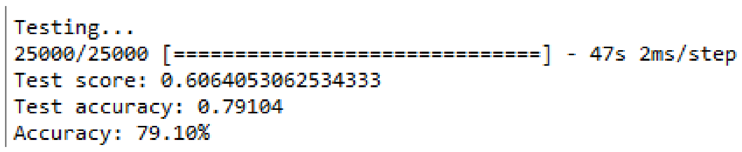
The training process is shown as the followings:



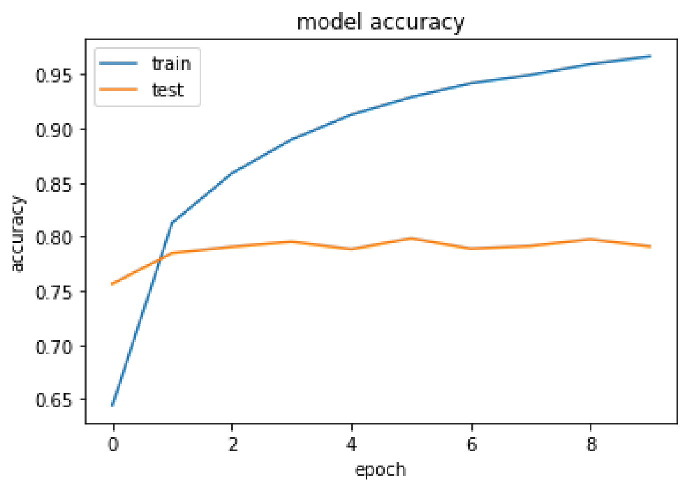
```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 301s - loss: 0.6310 - acc: 0.6442 - val_loss: 0.5162 - val_acc: 0.7566
Epoch 2/10
 - 302s - loss: 0.4297 - acc: 0.8128 - val_loss: 0.4537 - val_acc: 0.7849
Epoch 3/10
 - 296s - loss: 0.3503 - acc: 0.8586 - val_loss: 0.4522 - val_acc: 0.7907
Epoch 4/10
 - 295s - loss: 0.2837 - acc: 0.8898 - val_loss: 0.4579 - val_acc: 0.7953
Epoch 5/10
 - 294s - loss: 0.2365 - acc: 0.9128 - val_loss: 0.4839 - val_acc: 0.7885
Epoch 6/10
 - 295s - loss: 0.1972 - acc: 0.9290 - val_loss: 0.5164 - val_acc: 0.7984
Epoch 7/10
 - 295s - loss: 0.1636 - acc: 0.9418 - val_loss: 0.5410 - val_acc: 0.7890
Epoch 8/10
 - 299s - loss: 0.1437 - acc: 0.9496 - val_loss: 0.5594 - val_acc: 0.7914
Epoch 9/10
 - 299s - loss: 0.1208 - acc: 0.9597 - val_loss: 0.5814 - val_acc: 0.7975
Epoch 10/10
 - 302s - loss: 0.1005 - acc: 0.9667 - val_loss: 0.6064 - val_acc: 0.7910
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



The validation result is shown as the followings:



```
Testing...
25000/25000 [==============================] - 47s 2ms/step
Test score: 0.6064053062534333
Test accuracy: 0.79104
Accuracy: 79.10%
```



With the results above, a form is produced to compare the model performance with different GRU units.

epoch	Units=12		Units=22		Units=32	
	Validation loss	Validation accuracy	Validation loss	Validation accuracy	Validation loss	Validation accuracy
1	0.5162	0.7566	0.4910	0.7637	0.4911	0.7711
2	0.4537	0.7849	0.4331	0.7983	0.4034	0.8190
3	0.4522	0.7904	0.4497	0.7981	0.4238	0.8116
4	0.4579	0.7953	0.4634	0.8059	0.4539	0.8121
5	0.4839	0.7885	0.4932	0.7975	0.4763	0.8169
6	0.5164	0.7984	0.5218	0.7964	0.5420	0.8112
7	0.5410	0.7890	0.5478	0.7784	0.5673	0.8023
8	0.5594	0.7914	0.5935	0.7974	0.5909	0.8092
9	0.5814	0.7975	0.6280	0.7964	0.6449	0.8058
10	0.6064	0.7910	0.6730	0.7958	0.6765	0.7998

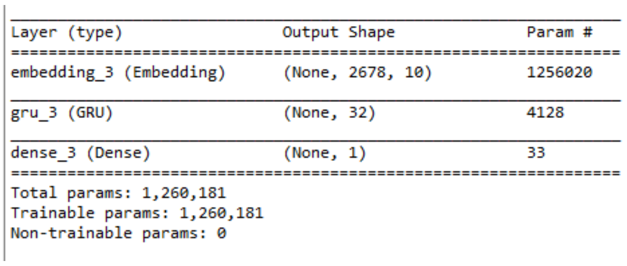
Compared to the model validation accuracy 79.74% in homework1, the accuracy for 22 units and 12 units are 79.58% and 79.10%, respectively. And the training process of 32 units shows its validation loss is smaller that 22 units and 12 units. Thus, 32 units is optimal for the best performance.

4. Limit the size of weights

The second method to reduce overfitting is limiting the size of weights. The experiment in homework1 sets the embedding dimension of model first layer to 100. To reduce the size of weights, embedding dimension needs to be reduced. Thus, embedding dimensions are set to 10 and 1 and the experiments are executed again. In the fourth line of the following code, embedding dimensions are modified to 10 and 1, respectively.

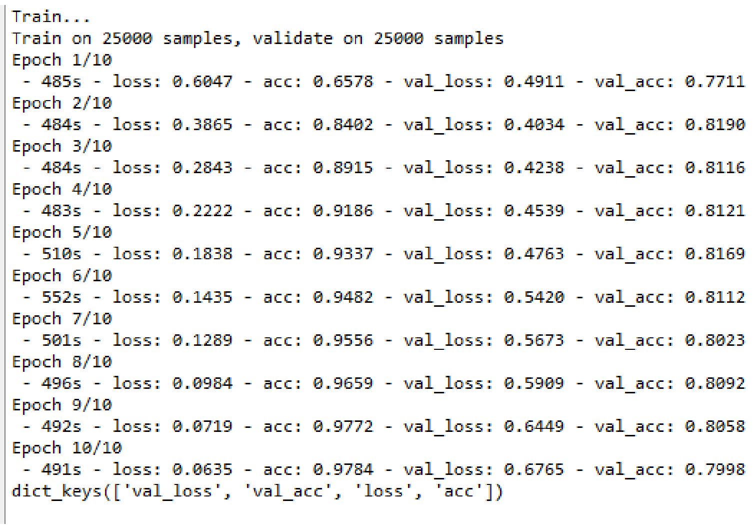
```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 10

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(GRU(units=32,    dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

The parameters when embedding dimension is equal to 10 are shown as the followings:

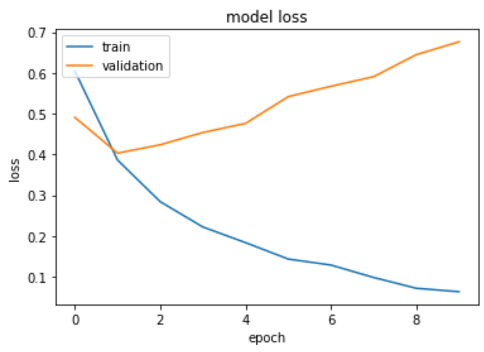


```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 2678, 10)          1256020
_________________________________________________________________
gru_3 (GRU)                  (None, 32)                4128
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,260,181
Trainable params: 1,260,181
Non-trainable params: 0
```

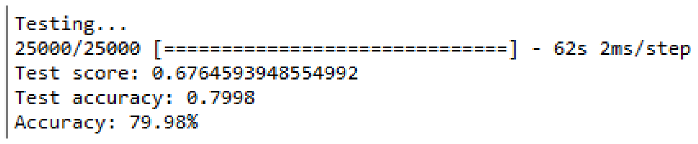
The training process is shown as the followings:



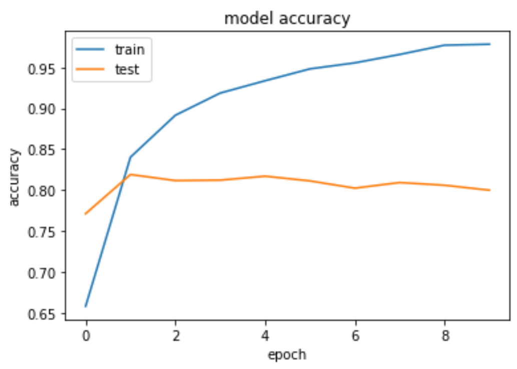
```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 485s - loss: 0.6047 - acc: 0.6578 - val_loss: 0.4911 - val_acc: 0.7711
Epoch 2/10
 - 484s - loss: 0.3865 - acc: 0.8402 - val_loss: 0.4034 - val_acc: 0.8190
Epoch 3/10
 - 484s - loss: 0.2843 - acc: 0.8915 - val_loss: 0.4238 - val_acc: 0.8116
Epoch 4/10
 - 483s - loss: 0.2222 - acc: 0.9186 - val_loss: 0.4539 - val_acc: 0.8121
Epoch 5/10
 - 510s - loss: 0.1838 - acc: 0.9337 - val_loss: 0.4763 - val_acc: 0.8169
Epoch 6/10
 - 552s - loss: 0.1435 - acc: 0.9482 - val_loss: 0.5420 - val_acc: 0.8112
Epoch 7/10
 - 501s - loss: 0.1289 - acc: 0.9556 - val_loss: 0.5673 - val_acc: 0.8023
Epoch 8/10
 - 496s - loss: 0.0984 - acc: 0.9659 - val_loss: 0.5909 - val_acc: 0.8092
Epoch 9/10
 - 492s - loss: 0.0719 - acc: 0.9772 - val_loss: 0.6449 - val_acc: 0.8058
Epoch 10/10
 - 491s - loss: 0.0635 - acc: 0.9784 - val_loss: 0.6765 - val_acc: 0.7998
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



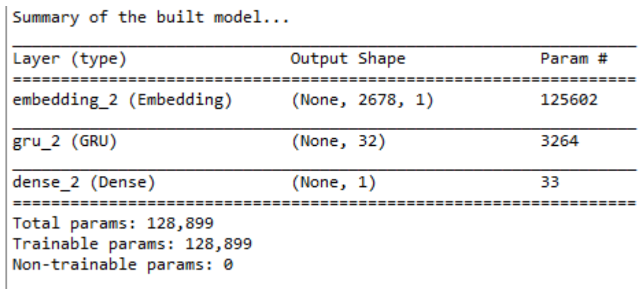
The validation result is shown as the followings:



```
Testing...
25000/25000 [==============================] - 62s 2ms/step
Test score: 0.6764593948554992
Test accuracy: 0.7998
Accuracy: 79.98%
```

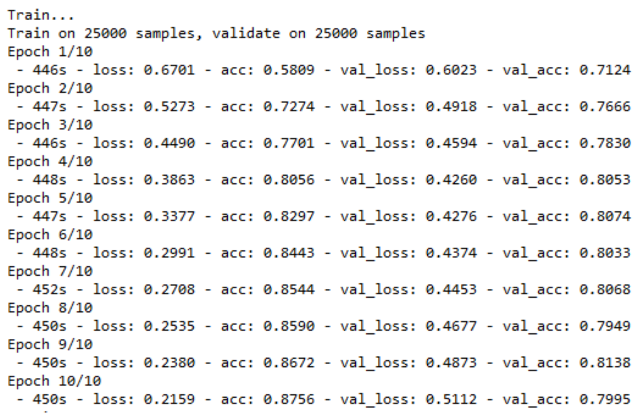


The parameters when embedding dimension is equal to 1 are shown as the followings:



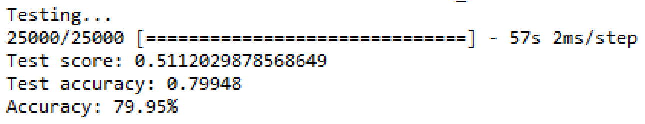
```
Summary of the built model...
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 2678, 1)           125602
_________________________________________________________________
gru_2 (GRU)                  (None, 32)                3264
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 128,899
Trainable params: 128,899
Non-trainable params: 0
```

The training process is shown as the followings:



```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 446s - loss: 0.6701 - acc: 0.5809 - val_loss: 0.6023 - val_acc: 0.7124
Epoch 2/10
 - 447s - loss: 0.5273 - acc: 0.7274 - val_loss: 0.4918 - val_acc: 0.7666
Epoch 3/10
 - 446s - loss: 0.4490 - acc: 0.7701 - val_loss: 0.4594 - val_acc: 0.7830
Epoch 4/10
 - 448s - loss: 0.3863 - acc: 0.8056 - val_loss: 0.4260 - val_acc: 0.8053
Epoch 5/10
 - 447s - loss: 0.3377 - acc: 0.8297 - val_loss: 0.4276 - val_acc: 0.8074
Epoch 6/10
 - 448s - loss: 0.2991 - acc: 0.8443 - val_loss: 0.4374 - val_acc: 0.8033
Epoch 7/10
 - 452s - loss: 0.2708 - acc: 0.8544 - val_loss: 0.4453 - val_acc: 0.8068
Epoch 8/10
 - 450s - loss: 0.2535 - acc: 0.8590 - val_loss: 0.4677 - val_acc: 0.7949
Epoch 9/10
 - 450s - loss: 0.2380 - acc: 0.8672 - val_loss: 0.4873 - val_acc: 0.8138
Epoch 10/10
 - 450s - loss: 0.2159 - acc: 0.8756 - val_loss: 0.5112 - val_acc: 0.7995
```

The validation result is shown as the followings:



```
Testing...
25000/25000 [==============================] - 57s 2ms/step
Test score: 0.5112029878568649
Test accuracy: 0.79948
Accuracy: 79.95%
```

With the results above, a form is produced to compare the model performance with different embedding dimensions.

epoch	Embedding dimension=100		Embedding dimension=10		Embedding dimension=1	
	Validation loss	Validation accuracy	Validation loss	Validation accuracy	Validation loss	Validation accuracy
1	0.4532	0.7858	0.4911	0.7711	0.6021	0.7124
2	0.4191	0.8158	0.4034	0.8190	0.4918	0.7666
3	0.4730	0.8020	0.4238	0.8116	0.4594	0.7830
4	0.5240	0.7980	0.4539	0.8121	0.4260	0.8053
5	0.5471	0.7909	0.4763	0.8169	0.4276	0.8074
6	0.6115	0.8053	0.5420	0.8112	0.4374	0.8033
7	0.6688	0.7933	0.5673	0.8023	0.4453	0.8068
8	0.7166	0.8098	0.5909	0.8092	0.4677	0.7949
9	0.8065	0.8025	0.6449	0.8058	0.4873	0.8138
10	0.7834	0.7974	0.6765	0.7998	0.5112	0.7995

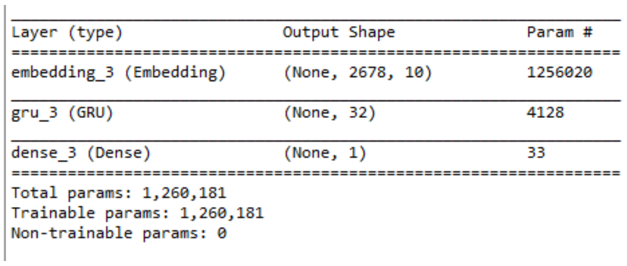
From the form above, the epochs of each embedding dimension where validation loss is the least, is 2, 2, 5, respectively. And when the embedding dimension is equal to 1, the model has the least validation loss. Since the number of IMDB reviews in the training dataset is heavily less than numbers of parameters of the model, overfitting cannot be eliminated but only reduced. Thus, in this assignment, when the embedding dimension is 1, the model has the least number of parameters, and the model is less likely affected by overfitting.

5. Stop the learning before it has time to overfit

The third method to reduce overfitting is stopping the learning before it has time to overfit. The functions, Earlystopping and ModelCheckpoint, are used to stop the model fitting when the validation loss begins to increase after former decreasing. In homework1, Earlystopping and ModelCheckpoint are not used, so no matter how the validation loss goes. The experiment fit the model in 10 epochs. However, in this assignment, the experiment does not have to fit it in 10 epochs. The corresponding code is shown as the followings:

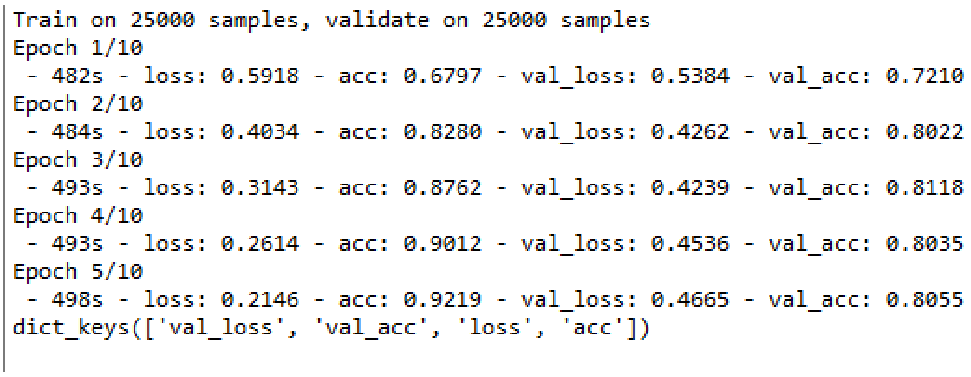
```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 10

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(GRU(units=32,    dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

callbacks = [EarlyStopping(monitor='val_loss', patience=2),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_loss',
save_best_only=True)]

# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model.summary())

print('Train...')

history = model.fit(X_train_pad, y_train, batch_size=128, epochs=10,
callbacks=callbacks, validation_data=(X_test_pad, y_test), verbose=2)
```

The GRU units are set to 32 and embedding dimension is set to 10.

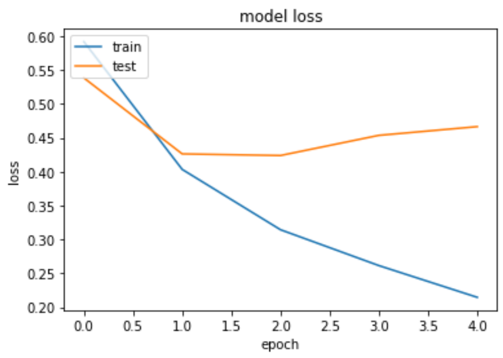


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 2678, 10)          1256020
_________________________________________________________________
gru_3 (GRU)                  (None, 32)                4128
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,260,181
Trainable params: 1,260,181
Non-trainable params: 0
```

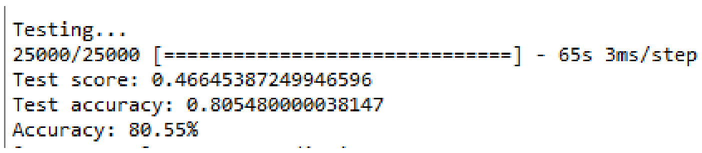
The training process is shown as the followings:



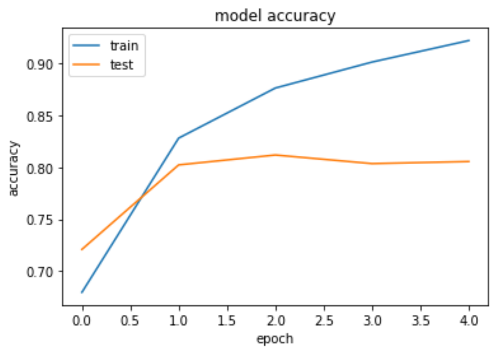
```
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 482s - loss: 0.5918 - acc: 0.6797 - val_loss: 0.5384 - val_acc: 0.7210
Epoch 2/10
 - 484s - loss: 0.4034 - acc: 0.8280 - val_loss: 0.4262 - val_acc: 0.8022
Epoch 3/10
 - 493s - loss: 0.3143 - acc: 0.8762 - val_loss: 0.4239 - val_acc: 0.8118
Epoch 4/10
 - 493s - loss: 0.2614 - acc: 0.9012 - val_loss: 0.4536 - val_acc: 0.8035
Epoch 5/10
 - 498s - loss: 0.2146 - acc: 0.9219 - val_loss: 0.4665 - val_acc: 0.8055
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



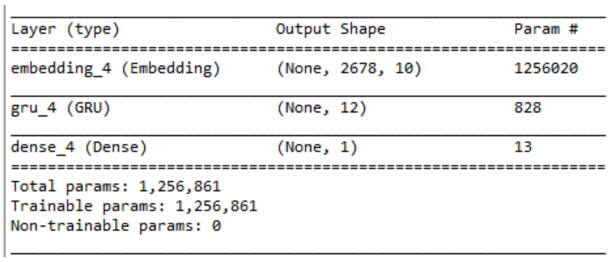
The validation result is shown as the followings:



```
Testing...
25000/25000 [==============================] - 65s 3ms/step
Test score: 0.46645387249946596
Test accuracy: 0.805480000038147
Accuracy: 80.55%
```

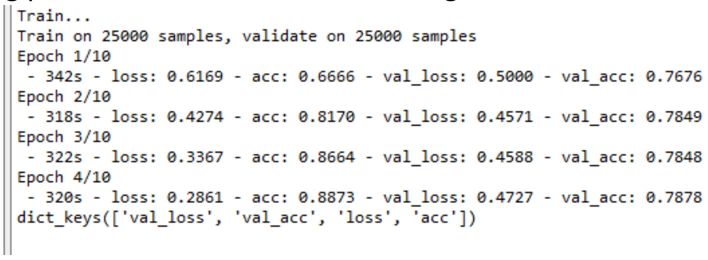


Now, early stopping is executed with another group of parameters. The GRU units are 12, and embedding dimension is equal to 1.



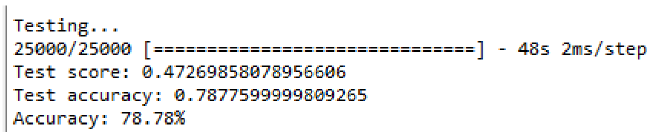
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 2678, 10)          1256020
_________________________________________________________________
gru_4 (GRU)                  (None, 12)                828
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 13
=================================================================
Total params: 1,256,861
Trainable params: 1,256,861
Non-trainable params: 0
_________________________________________________________________
```

The training process is shown as the followings:



```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 342s - loss: 0.6169 - acc: 0.6666 - val_loss: 0.5000 - val_acc: 0.7676
Epoch 2/10
 - 318s - loss: 0.4274 - acc: 0.8170 - val_loss: 0.4571 - val_acc: 0.7849
Epoch 3/10
 - 322s - loss: 0.3367 - acc: 0.8664 - val_loss: 0.4588 - val_acc: 0.7848
Epoch 4/10
 - 320s - loss: 0.2861 - acc: 0.8873 - val_loss: 0.4727 - val_acc: 0.7878
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

The validation result is shown as the followings:


```
Testing...
25000/25000 [==============================] - 48s 2ms/step
Test score: 0.47269858078956606
Test accuracy: 0.7877599999809265
Accuracy: 78.78%
```

With the results above, a form is produced to compare the model performance with different embedding dimensions.

epoch	Embedding dimension=10, GRU units=32		Embedding dimension=10, GRU units=12	
	Validation loss	Validation accuracy	Validation loss	Validation accuracy
1	0.5384	0.7210	0.5000	0.7676
2	0.4262	0.8022	0.4571	0.7849
3	0.4239	0.8118	0.4588	0.7848
4	0.4536	0.8035	0.4727	0.7878
5	0.4665	0.8055	stopped	Stopped
6	stopped	stopped	Stopped	Stopped
7	stopped	stopped	Stopped	stopped
8	stopped	Stopped	stopped	stopped
9	stopped	stopped	stopped	Stopped
10	stopped	stopped	stopped	stopped

From the form above, the model fitting stops two epochs after it reaches the least validation loss, but the parameters with least validation loss are adopted as the final model parameter. The first group takes 5 epochs to stop while the second takes 4 epochs.

6. Conclusion

In this assignment, three methods to reduce overfitting are implemented. Because the number of IMDB reviews of training dataset is less than the number of parameters, overfitting cannot be eliminated here. For the method of limiting the number of hidden units, the optimal number of GRU units is 32. If GRU units are less, the network cannot convey much information to the next layer, thus the validation loss could be greater. If GRU units are more, the parameters could increase heavily, which is also not good for validation loss. For the method of limiting the size of the weights, embedding dimensions are limited to limit the size of the weights. The experiments show that if the embedding dimension is set to 1, the model has the best performance. This could possibly be caused by the limited number of training data tuples. If the number of tuples is greater than the number of parameters, the model with larger embedding dimension could have the best performance. For the method of stopping the learning before it has time to overfit, the model fitting stops when it reaches the least validation loss. So, it seems to perform best among these three methods, because the former two methods get the parameters which has already been overfit, while the third method have the optimal parameters.