Data Mining Hw2 Part 3 Report

Student ID 108011573
Name Yi-Yun Hsieh

Breif Introduction

Data

- **tweets_DM.json**
  Raw data from Twitter
- **emotion.csv**
  Lists the emotion labels per tweet_id
- **data_identification.csv**
  A file that identifies membership of training or testing set per tweet_id.
  Note that you won't be provided with the labels for the testing set, but you will have to predict for these when you make your submission.
- **sampleSubmission.csv**
  A submission format you should follow for submitting to the competition

I use two methods
(1)BOW_version.ipynb
(2)Bert_version.ipynb

Experiment result
Method (1)
Train - acc : 0.55
Validate - acc : 0.51

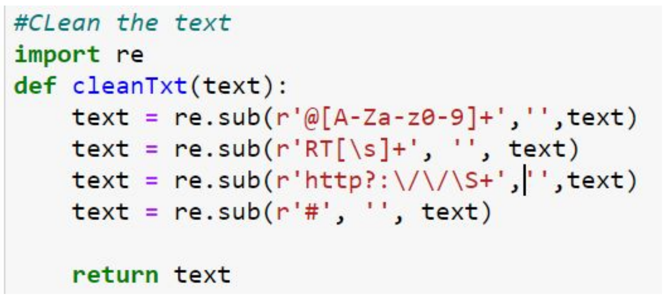
Method (2)
Train - acc : 0.84
Validate - acc : 0.55

Because method (1) got poor accuracy, so I tried method (2) which had improved a lot in accuracy of training set and validation set, but the result of the testing set still frustrated me.
I think that I encountered an overfitting problem, and can deal with it by applying data augmentation and batch normalization.

Preprocessing step

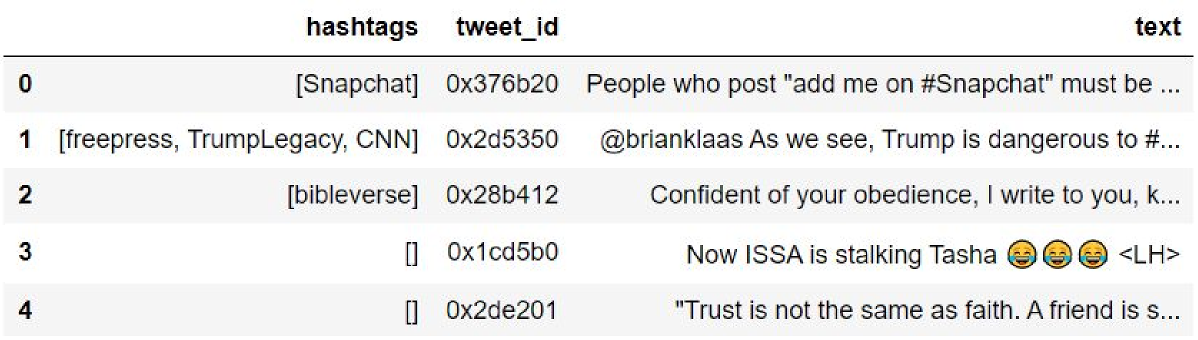
I first need to do the data cleaning the details of steps are below :
(1)Load the json and csv file - with `pd.read_json and pd.read_csv`
(2)Combine the Dataset - with `pd.merge`
(3)Text cleaning - with `cleanTxt`
(4)Create the dataset - with `to_pickle`
(5)Split the training dataset into train and validation set - with `train_test_split`

Text cleaning
I define the function named "cleanTxt" to do the data cleaning
But the thing is that I didn't do it in the beginning, after the poor acc
I search lots of articles to find out if I should eliminate some unnecessary but mistook words.



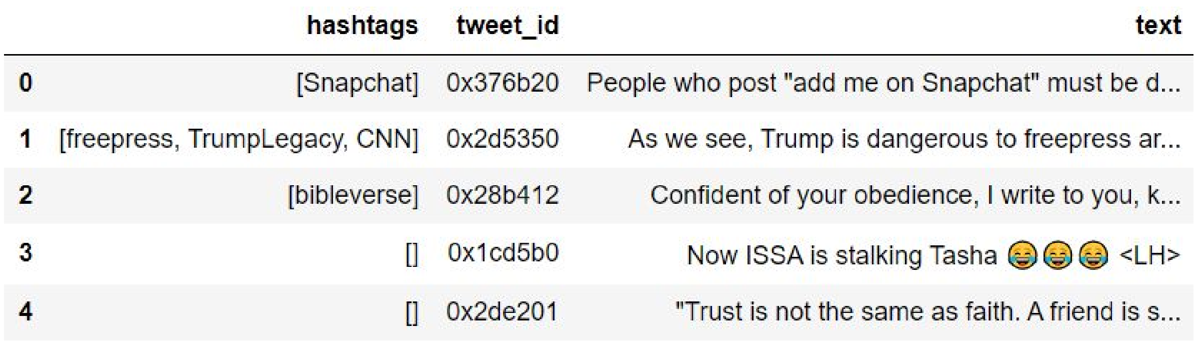
```python
#CLean the text
import re
def cleanTxt(text):
    text = re.sub(r'@[A-Za-z0-9]+','',text)
    text = re.sub(r'RT[\s]+', '', text)
    text = re.sub(r'http?:\/\/\S+','',text)
    text = re.sub(r'#', '', text)

    return text
```

<Before>



	hashtags	tweet_id	text
0	[Snapchat]	0x376b20	People who post "add me on #Snapchat" must be ...
1	[freepress, TrumpLegacy, CNN]	0x2d5350	@brianklaas As we see, Trump is dangerous to #...
2	[bibleverse]	0x28b412	Confident of your obedience, I write to you, k...
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>
4	[]	0x2de201	"Trust is not the same as faith. A friend is s...

<After>



	hashtags	tweet_id	text
0	[Snapchat]	0x376b20	People who post "add me on Snapchat" must be d...
1	[freepress, TrumpLegacy, CNN]	0x2d5350	As we see, Trump is dangerous to freepress ar...
2	[bibleverse]	0x28b412	Confident of your obedience, I write to you, k...
3	[]	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>
4	[]	0x2de201	"Trust is not the same as faith. A friend is s...

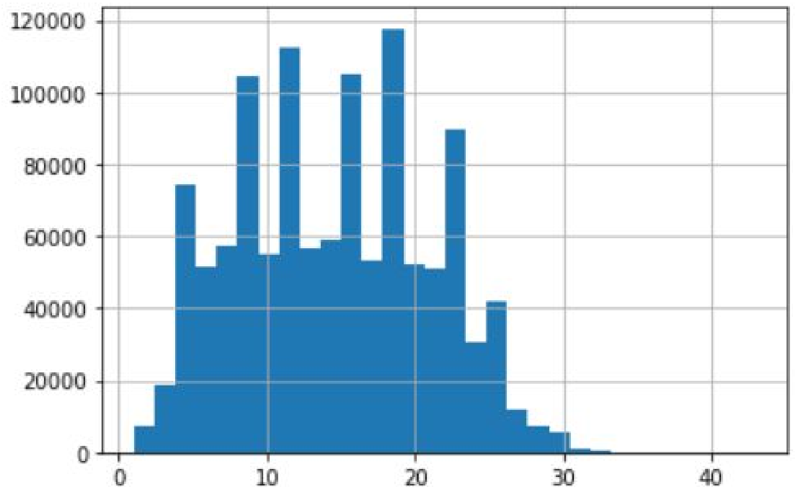
Feature engineering steps

Method (1)
I use CountVectorizer as feature engineering and set the max_features = 1000
It really took lots of time to create the BOW_1000 (about 7 mins)

```
import nltk
from sklearn.feature_extraction.text import CountVectorizer
BOW_1000 = CountVectorizer(max_features=1000, tokenizer=nltk.word_tokenize)

BOW_1000.fit(model_X_train_df['text'])
print("ok")
BOW_features_1000 = BOW_1000.transform(model_X_train_df['text'])
BOW_features_1000.shape
```

Method(2)
First thing is get the length of all text in the training set

```
# get length of all the messages in the train set
seq_len = [len(i.split()) for i in bert_X_train_df.text]

pd.Series(seq_len).hist(bins = 30)
```



Based on the result above, take the average of length and let the tensorflow do the padding :
max_seq_len = 15

Use the “trans” function in ktrain

```
class_n = ['anger', 'anticipation', 'disgust', 'fear', 'sadness', 'surprise', 'trust','joy']
```

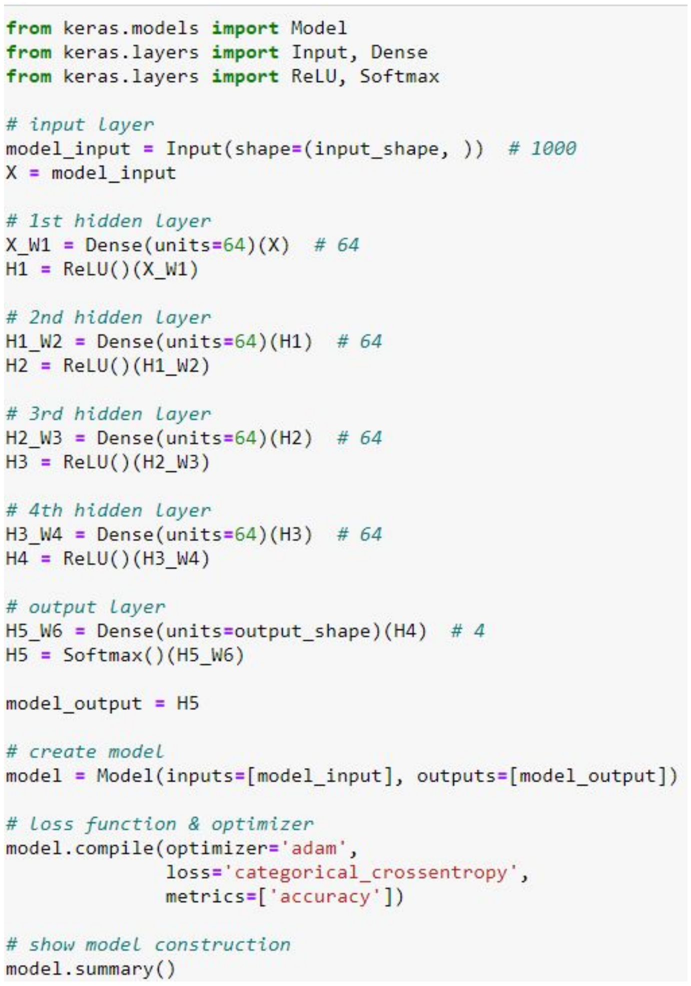
```
trans = text.Transformer(Model_name = 'distilbert-base-uncased', maxlen=15, class_names=class_n)
```

```
train_data = trans.preprocess_train(X_train, y_train)
test_data = trans.preprocess_test(X_test, y_test)
```

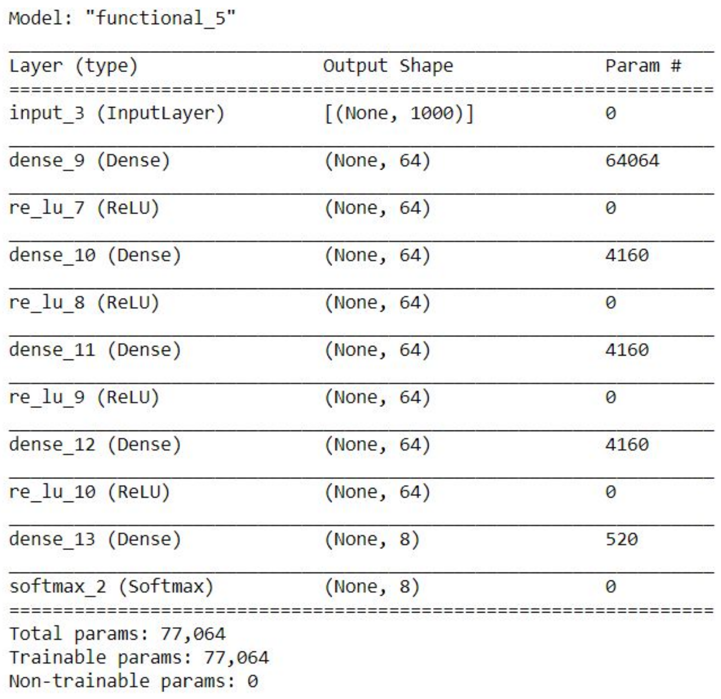
(contains 'distilbert-base-uncased' And it also visualize the loading time

Model Implementation

Method (1)
Similar to lab2, I add more hidden layer to the nn model (more parameters)

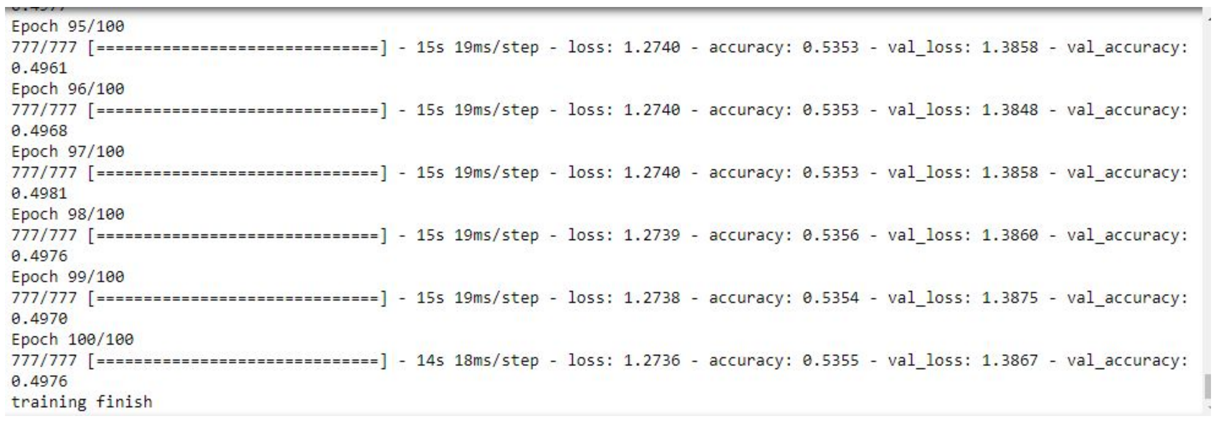


```python
from keras.models import Model
from keras.layers import Input, Dense
from keras.layers import ReLU, Softmax

# input layer
model_input = Input(shape=(input_shape, ))  # 1000
X = model_input

# 1st hidden layer
X_W1 = Dense(units=64)(X)  # 64
H1 = ReLU()(X_W1)

# 2nd hidden layer
H1_W2 = Dense(units=64)(H1)  # 64
H2 = ReLU()(H1_W2)

# 3rd hidden layer
H2_W3 = Dense(units=64)(H2)  # 64
H3 = ReLU()(H2_W3)

# 4th hidden layer
H3_W4 = Dense(units=64)(H3)  # 64
H4 = ReLU()(H3_W4)

# output layer
H5_W6 = Dense(units=output_shape)(H4)  # 4
H5 = Softmax()(H5_W6)

model_output = H5

# create model
model = Model(inputs=[model_input], outputs=[model_output])

# loss function & optimizer
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# show model construction
model.summary()
```



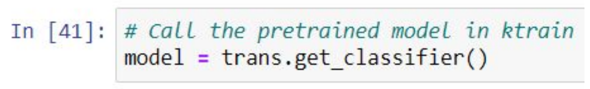
```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 1000)]            0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                64064
_________________________________________________________________
re_lu_7 (ReLU)               (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                4160
_________________________________________________________________
re_lu_8 (ReLU)               (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                4160
_________________________________________________________________
re_lu_9 (ReLU)               (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                4160
_________________________________________________________________
re_lu_10 (ReLU)              (None, 64)                0
_________________________________________________________________
dense_13 (Dense)             (None, 8)                 520
_________________________________________________________________
softmax_2 (Softmax)          (None, 8)                 0
=================================================================
Total params: 77,064
Trainable params: 77,064
Non-trainable params: 0
```

Training time : around 5 hrs

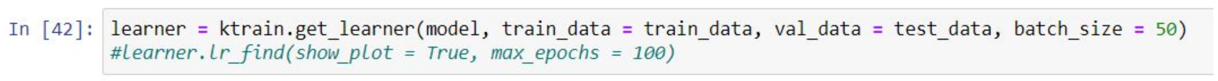


```
Epoch 95/100
777/777 [==============================] - 15s 19ms/step - loss: 1.2740 - accuracy: 0.5353 - val_loss: 1.3858 - val_accuracy:
0.4961
Epoch 96/100
777/777 [==============================] - 15s 19ms/step - loss: 1.2740 - accuracy: 0.5353 - val_loss: 1.3848 - val_accuracy:
0.4968
Epoch 97/100
777/777 [==============================] - 15s 19ms/step - loss: 1.2740 - accuracy: 0.5353 - val_loss: 1.3858 - val_accuracy:
0.4981
Epoch 98/100
777/777 [==============================] - 15s 19ms/step - loss: 1.2739 - accuracy: 0.5356 - val_loss: 1.3860 - val_accuracy:
0.4976
Epoch 99/100
777/777 [==============================] - 15s 19ms/step - loss: 1.2738 - accuracy: 0.5354 - val_loss: 1.3875 - val_accuracy:
0.4970
Epoch 100/100
777/777 [==============================] - 14s 18ms/step - loss: 1.2736 - accuracy: 0.5355 - val_loss: 1.3867 - val_accuracy:
0.4976
training finish
```

Method (2)
In ktrain, it contain learner and trans functions to call the model and the training steps



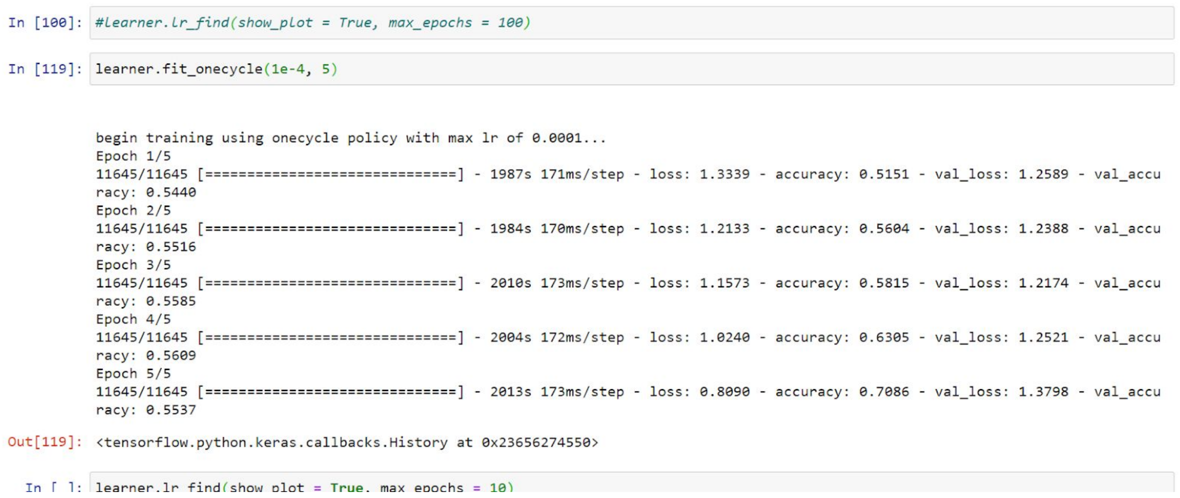
```
In [41]: # Call the pretrained model in ktrain
         model = trans.get_classifier()
```



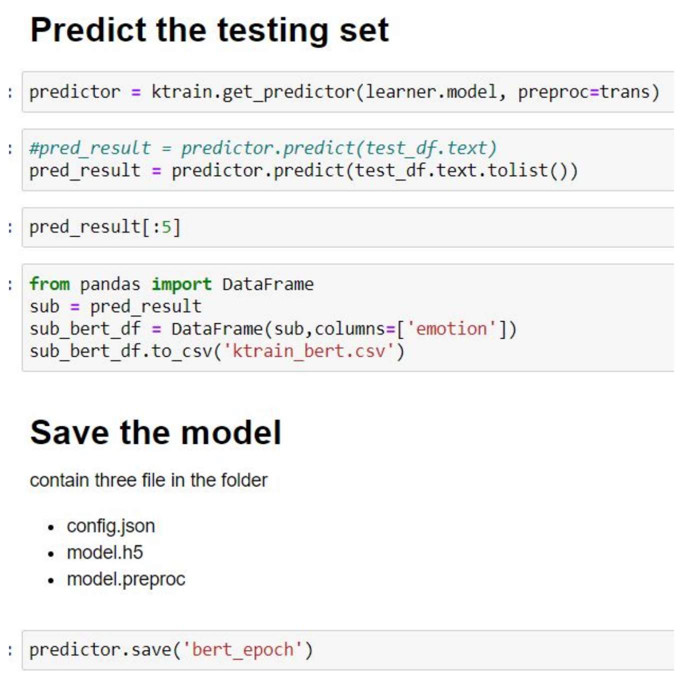
```
In [42]: learner = ktrain.get_learner(model, train_data = train_data, val_data = test_data, batch_size = 50)
         #learner.lr_find(show_plot = True, max_epochs = 100)
```

The batch_size here I can only set 50 or 100 due to the limitation of my GPU
If it is larger than 100, the system will give me error message
The Attention takes time to train, each epoch needs around thirty minutes to train and validate

Due to the time limitation, I set the number of epochs 10
It should be larger if I want better accuracy



```
In [100]: #learner.lr_find(show_plot = True, max_epochs = 100)

In [119]: learner.fit_onecycle(1e-4, 5)

          begin training using onecycle policy with max lr of 0.0001...
          Epoch 1/5
          11645/11645 [==============================] - 1987s 171ms/step - loss: 1.3339 - accuracy: 0.5151 - val_loss: 1.2589 - val_accu
          racy: 0.5440
          Epoch 2/5
          11645/11645 [==============================] - 1984s 170ms/step - loss: 1.2133 - accuracy: 0.5604 - val_loss: 1.2388 - val_accu
          racy: 0.5516
          Epoch 3/5
          11645/11645 [==============================] - 2010s 173ms/step - loss: 1.1573 - accuracy: 0.5815 - val_loss: 1.2174 - val_accu
          racy: 0.5585
          Epoch 4/5
          11645/11645 [==============================] - 2004s 172ms/step - loss: 1.0240 - accuracy: 0.6305 - val_loss: 1.2521 - val_accu
          racy: 0.5609
          Epoch 5/5
          11645/11645 [==============================] - 2013s 173ms/step - loss: 0.8090 - accuracy: 0.7086 - val_loss: 1.3798 - val_accu
          racy: 0.5537

Out[119]: <tensorflow.python.keras.callbacks.History at 0x23656274550>

  In [ ]: learner.lr_find(show_plot = True, max_epochs = 10)
```



Predict the testing set

```
: predictor = ktrain.get_predictor(learner.model, preproc=trans)

: #pred_result = predictor.predict(test_df.text)
  pred_result = predictor.predict(test_df.text.tolist())

: pred_result[:5]

: from pandas import DataFrame
  sub = pred_result
  sub_bert_df = DataFrame(sub,columns=['emotion'])
  sub_bert_df.to_csv('ktrain_bert.csv')
```

Save the model

contain three file in the folder

- config.json
- model.h5
- model.preproc

```
: predictor.save('bert_epoch')
```

Problems

(1)The interesting thing is that when I start with feature engineering by using BOW countvectorizer, the countvectorizer can not read the data frame until I reload it with pickle file and change to dataframe, so the next time I will learn from the experience to load data with picke format!

(2)Fine-tune the model to see the difference result
epoch from 100 to 2000
batch size from 1000 to 10000
But both my training and validation acc is around 0.55/0.51
I think the method (1) worked bad (Due to its simple nn model and BOW), so I tries method(2)

Summary

I am quite new to the NLP and Deep learning models.
The preprocessing task takes lots of time to transform the raw data into our custom dataset.
I learn a lot from this hw2 and kaggle competition.Thanks!

I find out that Bert learns well for the training set, but the validation acc is bad, I think overfitting is the problem that I faced.

During the Kaggle competition, method (1) got 0.2 but method (2) got 0.24.
I think I need to work hard on Bert again!