

Names: Cheng Zhao, Xicheng Tang

NetID: CZ1933, XT607

Project: Hierarchical Attentional Hybrid Neural Networks for Document Classification

## Hierarchical Attentional Hybrid Neural Networks for Document Classification

Document classification is a challenging task with important applications. Deep learning approaches to the problem have gained much attention. Despite the progress, the proposed models do not incorporate the knowledge of the document structure in the architecture efficiently and not take into account the contexting dependent importance of words and sentences. In this paper, we propose a new approach based on convolutional neural networks, gated recurrent units and attention mechanisms for document classification tasks. The datasets IMDB Movie Reviews and Yelp were used in experiments. The proposed method improves the results of current attention-based approaches

Reference: Abreu, J., Fred, L., Macêdo, D., & Zanchettin, C. (2019). Hierarchical Attentional Hybrid Neural Networks for Document Classification. arXiv preprint arXiv:1901.06610.

Hierarchical Attention Network is designed to capture two basic insights about document structure.

- First, construct a document representation by building representations of sentences and then aggregating them into a document representation since we know how a document structure is constructed.
- Second, two levels of attention mechanism were introduced to the model, which helped us to sort out the important words in a sentence and important sentences in a document.

## Two core advantages

**First:** Use the original hierarchical structure of the document (a sentence is composed of words, and a document is composed of sentences), first use word vectors to represent the sentence, and then use sentence vectors to construct the information representation of the document.

**Second:** In the document, the contribution of sentences to the importance of the document is different, and the contribution of words in the sentence to the importance of the sentence is also different. The importance of words and sentences depends on the context. The same word has different importance in different contexts. In order to describe this situation, the Attention mechanism is introduced to describe this importance.

Attention mechanism can bring two advantages: one can improve classification performance, and the second can increase the importance of identifying words or sentences that affect the final classification decision.

## ▼ Hierarchical Attention Network

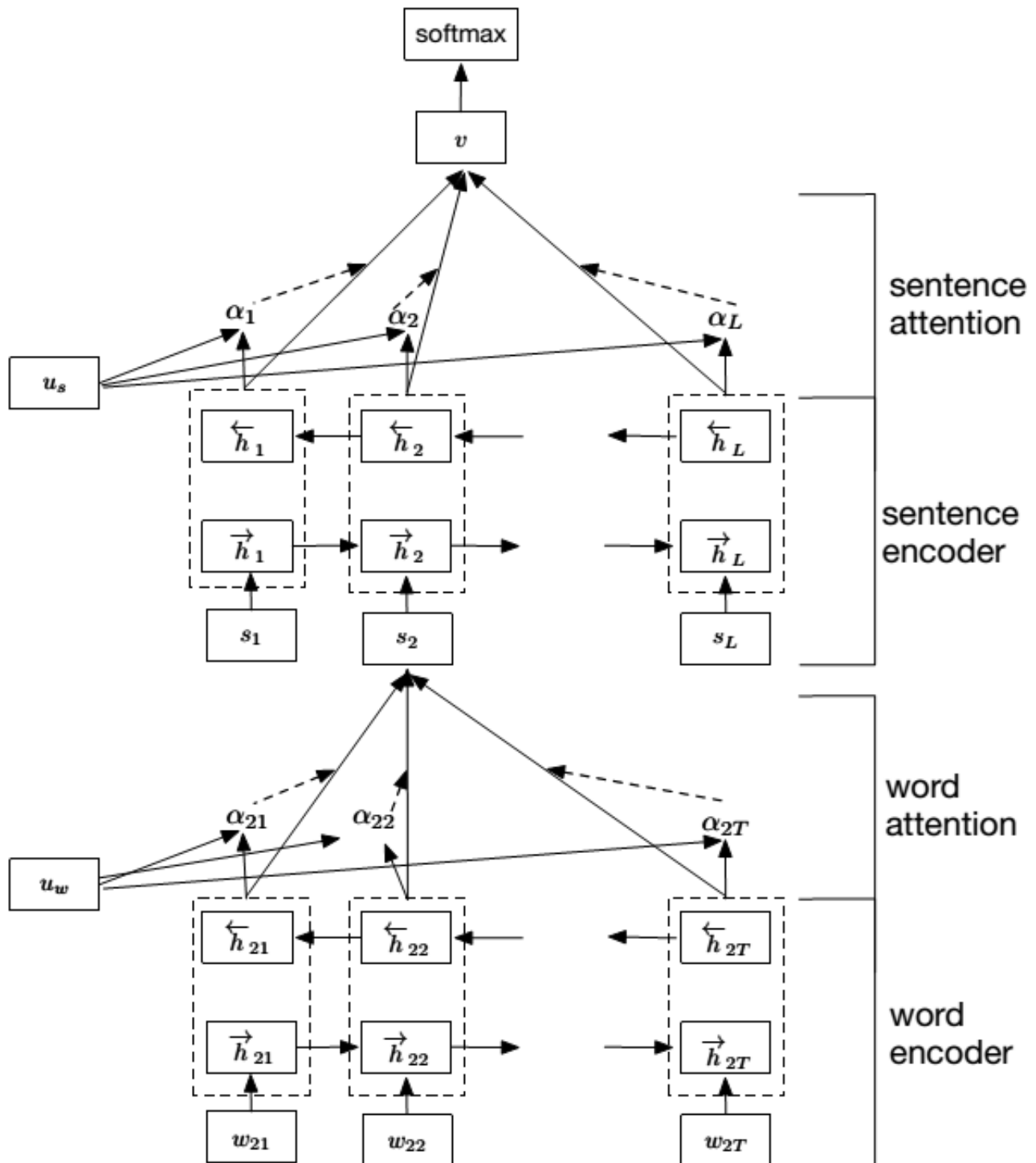
---

The Hierarchical Attention Network(HAN) that is designed to capture two basic insights about document structure.

First, we could construct a document representation by first building representations of sentences and then aggregating those into a document representation because documents have a hierarchical structure (words form sentence and sentences form a document).

Second, it is observed that different words are differentially informative to sentences which is the same as sentences to documents. Moreover, the importance of words and sentences are highly context dependent, so the research brings up two levels of attention mechanisms - one at the word level and one at the sentence level - to let the model pay more or less attention to individual words and sentences when constructing the representation of the document.

Reference: Abreu, J., Fred, L., Macêdo, D., & Zanchettin, C. (2019). Hierarchical Attentional Hybrid Neural Networks for Document Classification. arXiv preprint arXiv:1901.06610. Figure 2



**Figure 2: Hierarchical Attention Network.**

A word sequence encoder and a word-level attention layer;

- The model uses bidirectional GRU to produce annotations of words by summarizing information from both directions of hidden states.
- The model incorporates the contextual information in the annotation. The attention levels let the model pay more or less attention to individual words when construction the representation of the sentence.

A sentence encoder and a sentence-level attention layer.

- Similarly, the model uses bidirectional GRU to produce annotations of sentences by summarizing information from both directions of hidden states.
- The model incorporates the contextual information in the annotation. The attention levels let the model pay more or less attention to individual sentences when construction the representation of the document.

## Fixing some bugs on original codes

We achieve test on Yelp dataset with different parameters. It took us some time to debug to be able run the code because the code posted on github isn't working normally. We have to fix it first. Moreover, we fixed select function, for example "word\_embedding\_type = from\_scratch or pre-trained" can actually work now.

### ▼ To use Tensorflow 1 in Colab

Change the version to Tensorflow 1 so could run the codes in Colab.

```
1 %tensorflow_version 1.x

TensorFlow 1.x selected.
```

### ▼ Prompts for kaggle username and API key

Displays a field in prompt and waits for the user to input your kaggle username. By press ENTER, other field prompts for input Kaggle API key.

```
1 import os
2 from getpass import getpass
3
4 user = getpass('Kaggle Username: ')
5 key = getpass('Kaggle API key: ')
6
7 if '.kaggle' not in os.listdir('/root'):
8     !mkdir ~/.kaggle
9
10 !touch /root/.kaggle/kaggle.json
11 !chmod 666 /root/.kaggle/kaggle.json
12
13 with open('/root/.kaggle/kaggle.json', 'w') as f:
14     f.write('{"username":"%s","key":"%s"}' % (user, key))
15 !chmod 600 /root/.kaggle/kaggle.json
```

Kaggle Username: .....

Kaggle API key: .....

## Download pre-trained word embedding "wiki-news-300d-1M-subword.vec"

```
1 !kaggle datasets download -d luisfredgs/wiki-news-300d-1m-subword
2 !unzip -o wiki-news-300d-1m-subword.zip
```

## Download dataset

We choose a dataset from Yelp 2015 reviews, evaluating the reviewer's comments (text classification) and their attitudes (either positive or negative) to achieve an effective Natural Languages Processing via our method.

```

1 !kaggle datasets download -d luisfredgs/hahnn-for-document-classification
2 !unzip -o hahnn-for-document-classification.zip
3 # fixing in 400k reviews
4 !head -n400000 yelp_reviews.json > yelp_reviews_sampling.json

```

```

Downloading hahnn-for-document-classification.zip to /content
 98% 353M/361M [00:08<00:00, 51.8MB/s]
100% 361M/361M [00:08<00:00, 44.0MB/s]
Archive:  hahnn-for-document-classification.zip
  inflating: imdb_reviews.csv
  inflating: yelp_reviews.json

```

**dataset:** yelp ▼

**word\_embedding\_type:** from\_scratch ▼

**word\_vector\_model:** fasttext ▼

**rnn\_type:** GRU ▼

## ▼ Imports

Imports gensim and paramiko so we can use their libraries to process our dataset.

```

1 !pip -q install gensim
2 !python -m spacy download en_core_web_md
3 !pip -q install paramiko
4
5 import datetime, pickle, os, codecs, re, string
6 import json
7 import random
8 import numpy as np
9 import keras
10 from keras.models import *
11 from keras.layers import *
12 from keras.optimizers import *
13 from keras.callbacks import *
14 from keras import regularizers
15 from keras.preprocessing.text import Tokenizer
16 from keras.preprocessing.sequence import pad_sequences
17 from keras import backend as K
18 from keras.utils import CustomObjectScope

```

```
20 from keras.engine.topology import Layer
21 #
22 from keras.engine import InputSpec
23
24 from keras import initializers
25
26 import pandas as pd
27 from tqdm import tqdm
28
29 import string
30 from spacy.lang.en import English
31 import gensim, nltk, logging
32
33 from nltk.corpus import stopwords
34 from nltk import tokenize
35 from nltk.stem import SnowballStemmer
36
37 nltk.download('punkt')
38 nltk.download('stopwords')
39
40 from sklearn.manifold import TSNE
41 import matplotlib.pyplot as plt
42 import en_core_web_sm
43
44 from IPython.display import HTML, display
45
46 import tensorflow as tf
47
48 from numpy.random import seed
49 from tensorflow import set_random_seed
50 os.environ['PYTHONHASHSEED'] = str(1024)
51 set_random_seed(1024)
52 seed(1024)
53 np.random.seed(1024)
54 random.seed(1024)
```

Collecting en\_core\_web\_md==2.2.5

Downloading [https://github.com/explosion/spacy-models/releases/download/en\\_core\\_web\\_md-2.2.5/en\\_core\\_web\\_md-2.2.5.tar.gz](https://github.com/explosion/spacy-models/releases/download/en_core_web_md-2.2.5/en_core_web_md-2.2.5.tar.gz)

```
|████████████████████████████████████████| 96.4MB 1.1MB/s
Requirement already satisfied: spacy>=2.2.2 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/python
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/pyth
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/py
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/pytho
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python3
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python3
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/py
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dis
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: importlib-metadata>=0.20; python_version < "
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-p
Building wheels for collected packages: en-core-web-md
```

Building wheel for en-core-web-md (setup.py) ... done

Created wheel for en-core-web-md: filename=en\_core\_web\_md-2.2.5-cp36-none

Stored in directory: /tmp/pip-ephem-wheel-cache-t8ik43lb/wheels/df/94/ad/

Successfully built en-core-web-md

Installing collected packages: en-core-web-md

Successfully installed en-core-web-md-2.2.5

✓ Download and installation successful

You can now load the model via `spacy.load('en_core_web_md')`

```
|████████████████████████████████████████| 215kB 8.1MB/s
|████████████████████████████████████████| 71kB 7.3MB/s
|████████████████████████████████████████| 2.6MB 15.0MB/s
|████████████████████████████████████████| 962kB 54.1MB/s
```

Using TensorFlow backend.

[nltk\_data] Downloading package punkt to /root/nltk\_data...

[nltk\_data] Unzipping tokenizers/punkt.zip.

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

## ► Text preprocessing

This step is preprocessing texts that need to be learned in order to have easier learning process. For example, we are going to get rid of all unnecessary symbols.

[ ] ↪ 1 cell hidden



## ▼ Use pre-trained word embeddings

Alternative use the pre-trained words from "wiki-news-300d-1M-subword.vec" (1 million word vectors trained on Wikipedia 2017 for word embeddings so we can compare them to the word embeddings on our datasets.

```

1 def load_subword_embedding_300d(word_index):
2     print('load_subword_embedding...')
3     embeddings_index = {}
4     f = codecs.open("wiki-news-300d-1M-subword.vec", encoding='utf-8')
5     for line in tqdm(f):
6         values = line.rstrip().rsplit(' ')
7         word = values[0]
8         coefs = np.asarray(values[1:], dtype='float32')
9         embeddings_index[word] = coefs
10    f.close()
11    print('found %s word vectors' % len(embeddings_index))
12
13    #embedding matrix
14    print('preparing embedding matrix...')
15    words_not_found = []
16
17    embedding_matrix = np.zeros((len(word_index) + 1, 300))
18
19    for word, i in word_index.items():
20        embedding_vector = embeddings_index.get(word)
21        if (embedding_vector is not None) and len(embedding_vector) > 0:
22
23            embedding_matrix[i] = embedding_vector
24        else:
25            words_not_found.append(word)
26
27    return embedding_matrix

```

## ► Plot word embedding chart

This is a demonstration of the location of each word in embedding matrix, i.e., this is what the embedding matrix look like.

[ ] ↪ 1 cell hidden

## ▸ Normalize texts

This is the step that translate to a canonical form that machine can understand it better.

```
[ ] ↪ 1 cell hidden
```

## ▸ Training word embeddings

This step is training the weight of each word in each sentence from a trained dataset.

```
[ ] ↪ 1 cell hidden
```

## ▸ Load datasets

1) Creating function for loading Yelp Dataset with 500k reviews

2) Creating function for loading IMDb with 50k reviews

```
[ ] ↪ 1 cell hidden
```

## ▸ Attention Layer

Check ([Bahdanau et al., 2015](#)).

Since each word will have different weight in different sentence due to the context, we use attention layer to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector.

```
[ ] ↪ 1 cell hidden
```

## ▸ Remove and create a new "saved\_models" directory

```
[ ] ↪ 1 cell hidden
```

## ▸ Model architecture

Layer 1: **Word Encoder**

First of all, a machine can only understand its own languages, so we have to translate English words to something that computers can read. That is, we embedded words to vectors through an embedding matrix. Here, we trained our embedding matrix instead of using the pretrained one, and the dimension of the word vector is 200. After that, we use a bidirectional GRU (Gated Recurrent Unit) to get annotations of words by summarizing information from both directions of words.

$$\begin{aligned}x_{it} &= W_e w_{it}, t \in [1, T] \\ \vec{h}_{it} &= \overrightarrow{GRU}(x_{it}), t \in [1, T] \\ \overleftarrow{h}_{it} &= \overleftarrow{GRU}(x_{it}), t \in [T, 1]\end{aligned}$$

where  $w_{it}$  is the  $t$ th word in a sentence, and  $x_{it}$  is the word vector that we get from embedding matrix  $W_e$ .  $\vec{h}_{it}$  and  $\overleftarrow{h}_{it}$  are forward and backward hidden states, respectively.

### Layer 2: **Word Attention**

Since words can have different meaning and importances depend on the context, we form a sentence vector by collecting those information of words through attention mechanism.

"Thus, we first feed the word annotation  $h_{it}$  through a one-layer MLP to get  $u_{it}$  as a hidden representation of  $h_{it}$ , then we measure the importance of the word as the similarity of  $u_{it}$  with a word level context vector  $u_w$  and get a normalized importance weight  $\alpha_{it}$  through a softmax function. After that, we compute the sentence vector  $s_i$  as a weighted sum of the word annotations based on the weights."(Hierarchical, by Yang)

$$\begin{aligned}u_{it} &= \tanh(W_w h_{it} + b_w), \\ \alpha_{it} &= \frac{\exp(u_{it}^T u_w)}{\sum \exp(u_{it}^T u_w)}, \\ s_i &= \sum \alpha_{it} h_{it},\end{aligned}$$

where  $h_{it} = [\vec{h}_{it}, \overleftarrow{h}_{it}]$ .

### Layer 3: **Sentence encoder**

First of all, we used sentence encoder layer to get a document vector, which is very similar to the word encoder. A bidirectional GRU is used to encode the sentences, and then concatenate vectors of both directions to get an annotation of sentence.

$$\begin{aligned}\vec{h}_i &= \overrightarrow{GRU}(s_i), i \in [1, L], \\ \overleftarrow{h}_i &= \overleftarrow{GRU}(s_i), i \in [L, 1],\end{aligned}$$

where  $h_i = [\vec{h}_i, \overleftarrow{h}_i]$ .

### Layer 4: **Sentence Attention**

Secondly, we use attention mechanism and sentence level context vector to measure the importance of the sentences (the sentence level context vector can be randomly initialized and jointly learned during the training process).

$$\begin{aligned} u_i &= \tanh(W_w h_i + b_w), \\ \alpha_i &= \frac{\exp(u_i^T u_s)}{\sum \exp(u_i^T u_s)}, \\ s_i &= \sum \alpha_i h_i, \end{aligned}$$

### Layer 5: Document Classification

Finally, we will input the document vector, which gained from sentence attention by summarizing all the information of sentences in a document and input its value to Softmax - layer to achieve the results.

$$p = \text{softmax}(W_c + b_c),$$

and we use the negative log likelihood of the correct labels as training loss:

$$L = - \sum \log(p_{dj})$$

We also define word and sentence encoder for different rnn\_type, thus define parameters for train() function of our model. Finally, structure plot functions.

Reference: Zichao Yang<sup>1</sup>, Diyi Yang<sup>1</sup>, Chris Dyer<sup>1</sup>, Xiaodong He<sup>2</sup>, Alex Smola<sup>1</sup>, Eduard Hovy  
Carnegie Mellon University, <sup>2</sup>Microsoft Research, Redmond. (2016) Hierarchical Attentional Hybrid Neural Networks for Document Classification.

Abreu, J., Fred, L., Macêdo, D., & Zanchettin, C. (2019). Hierarchical Attentional Hybrid Neural Networks for Document Classification. arXiv preprint arXiv:1901.06610.

[ ] ↪ 1 cell hidden

## ▼ Load dataset

This might take several minutes, depending your dataset

```

1 YELP_DATA_PATH = 'yelp_reviews_sampling.json'
2 IMDB_DATA_PATH = 'imdb_reviews.csv'
3 SAVED_MODEL_DIR = 'saved_models'
4 SAVED_MODEL_FILENAME = 'model.h5'
5
6 if dataset is 'yelp':
7     (X, Y) = load_data_yelp(path=YELP_DATA_PATH, size=400000)
8 else:
9     (X, Y) = load_data_imdb(path=IMDB_DATA_PATH, size=49000)
10

```

## ▼ Plots Word Embedding chart

Visualize the word embedding in the graph

```

1 limit = 200
2 vector_dim = 200
3
4 # Fasttext
5 filename = './fasttext_model.txt'
6 model = gensim.models.FastText.load(filename)
7 words = []
8 embedding = np.array([])
9 i = 0
10 for word in model.wv.vocab:
11     if i == limit: break
12
13     words.append(word)
14     embedding = np.append(embedding, model[word])
15     i += 1
16
17 embedding = embedding.reshape(limit, vector_dim)
18 tsne = TSNE(n_components=2)
19 low_dim_embedding = tsne.fit_transform(embedding)
20
21 plot_with_labels(low_dim_embedding, words)

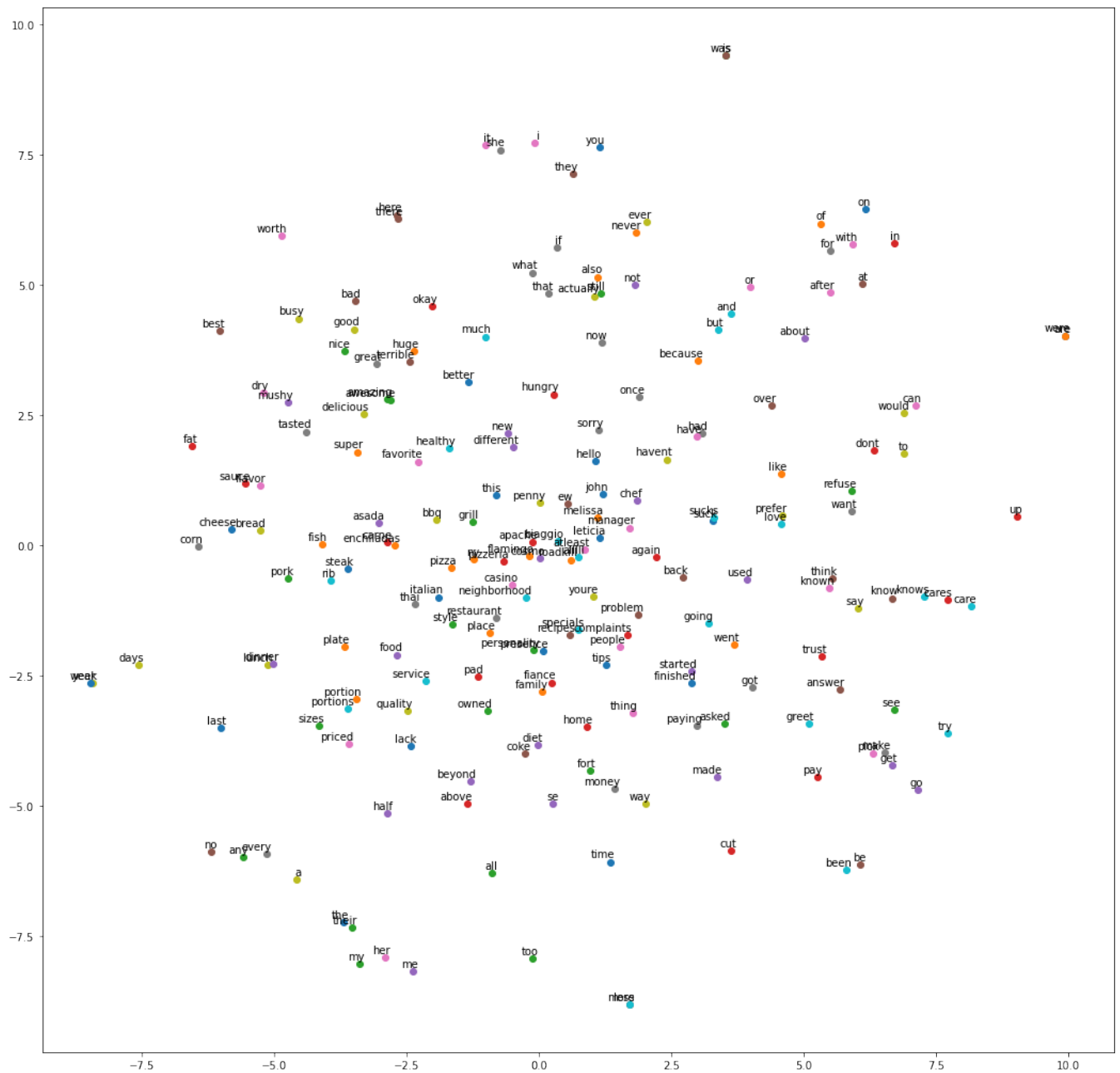
```

```

2020-10-16 05:58:43,222 : INFO : loading FastText object from ./fasttext_mo
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:252: Us
'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
2020-10-16 05:58:44,604 : INFO : loading wv recursively from ./fasttext_mod
2020-10-16 05:58:44,605 : INFO : loading vectors_ngrams from ./fasttext_mod
2020-10-16 05:58:44,664 : INFO : setting ignored attribute vectors_norm to
2020-10-16 05:58:44,666 : INFO : setting ignored attribute vectors_vocab_no
2020-10-16 05:58:44,667 : INFO : setting ignored attribute vectors_ngrams_n
2020-10-16 05:58:44,668 : INFO : setting ignored attribute buckets_word to

```

```
2020-10-16 05:58:44,669 : INFO : loading vocabulary recursively from ./fast
2020-10-16 05:58:44,670 : INFO : loading trainables recursively from ./fast
2020-10-16 05:58:44,671 : INFO : loading vectors_ngrams_lockf from ./fastte
2020-10-16 05:58:44,727 : INFO : loaded ./fasttext_model.txt
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: Deprecatio
```



## ▼ Taining model

We first pick yelp database for testing and choose "word\_embedding\_type = from\_scratch, word\_vector\_mode = fasttext, rnn\_type = GRU"

```
1 K.clear_session()
2 model = HAHNetwork()
3 model.train(X, Y, batch_size=64, epochs=8, embeddings_path=True, saved_model_
```

```
2020-10-16 10:27:20,302 : INFO : loading FastText object from ./fasttext_mo
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:252: Us
'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
2020-10-16 10:27:21,439 : INFO : loading wv recursively from ./fasttext_mod
2020-10-16 10:27:21,441 : INFO : loading vectors_ngrams from ./fasttext_mod
2020-10-16 10:27:21,521 : INFO : setting ignored attribute vectors_norm to
2020-10-16 10:27:21,525 : INFO : setting ignored attribute vectors_vocab_no
2020-10-16 10:27:21,526 : INFO : setting ignored attribute vectors_ngrams_n
2020-10-16 10:27:21,526 : INFO : setting ignored attribute buckets_word to
2020-10-16 10:27:21,527 : INFO : loading vocabulary recursively from ./fast
2020-10-16 10:27:21,528 : INFO : loading trainables recursively from ./fast
2020-10-16 10:27:21,529 : INFO : loading vectors_ngrams_lockf from ./fastte
2020-10-16 10:27:22,045 : INFO : loaded ./fasttext_model.txt
tracking <tf.Variable 'word_attention/Variable:0' shape=(100, 1) dtype=floa
tracking <tf.Variable 'word_attention/Variable_1:0' shape=(50,) dtype=float
tracking <tf.Variable 'word_attention/Variable_2:0' shape=(50, 1) dtype=flo
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected
input_1 (InputLayer)	(None, 50)	0	
word_embeddings (Embedding)	(None, 50, 200)	33156200	input_1[0]
dropout_1 (Dropout)	(None, 50, 200)	0	word_embed
conv1d_1 (Conv1D)	(None, 50, 64)	38464	dropout_1[

conv1d_2 (Conv1D)	(None, 50, 64)	51264	dropout_1[
conv1d_3 (Conv1D)	(None, 50, 64)	64064	dropout_1[
max_pooling1d_1 (MaxPooling1D)	(None, 16, 64)	0	conv1d_1[0
max_pooling1d_2 (MaxPooling1D)	(None, 12, 64)	0	conv1d_2[0
max_pooling1d_3 (MaxPooling1D)	(None, 10, 64)	0	conv1d_3[0
concatenate_1 (Concatenate)	(None, 38, 64)	0	max_poolin max_poolin max_poolin
dropout_2 (Dropout)	(None, 38, 64)	0	concatenat
bidirectional_1 (Bidirectional)	(None, 38, 100)	34800	dropout_2[
dense_transform_word (Dense)	(None, 38, 100)	10100	bidirectio
word_attention (Attention)	(None, 100)	200	dense_tran
=====			
Total params: 33,355,092			
Trainable params: 33,355,092			
Non-trainable params: 0			

```

tracking <tf.Variable 'sentence_attention/Variable:0' shape=(100, 1) dtype=
tracking <tf.Variable 'sentence_attention/Variable_1:0' shape=(50,) dtype=f
tracking <tf.Variable 'sentence_attention/Variable_2:0' shape=(50, 1) dtype
Model: "model_2"

```

Layer (type)	Output Shape	Param #
=====		
input_2 (InputLayer)	(None, 15, 50)	0
time_distributed_1 (TimeDist	(None, 15, 100)	33355092
dropout_3 (Dropout)	(None, 15, 100)	0
bidirectional_2 (Bidirection	(None, 15, 100)	45600
dense_transform_sentence (De	(None, 15, 100)	10100
sentence_attention (Attentio	(None, 100)	200
dense_1 (Dense)	(None, 5)	505
=====		
Total params: 33,411,497		
Trainable params: 33,411,497		
Non-trainable params: 0		

```

Train on 360000 samples, validate on 40000 samples
Epoch 1/8

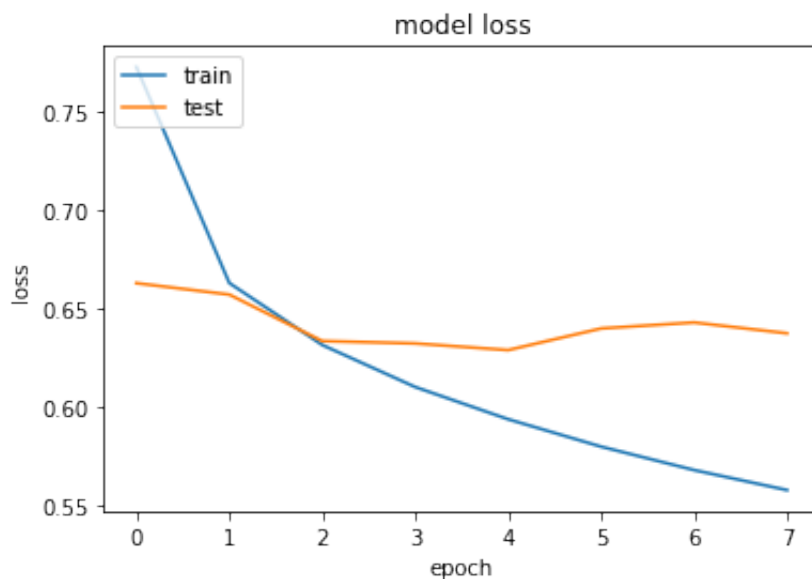
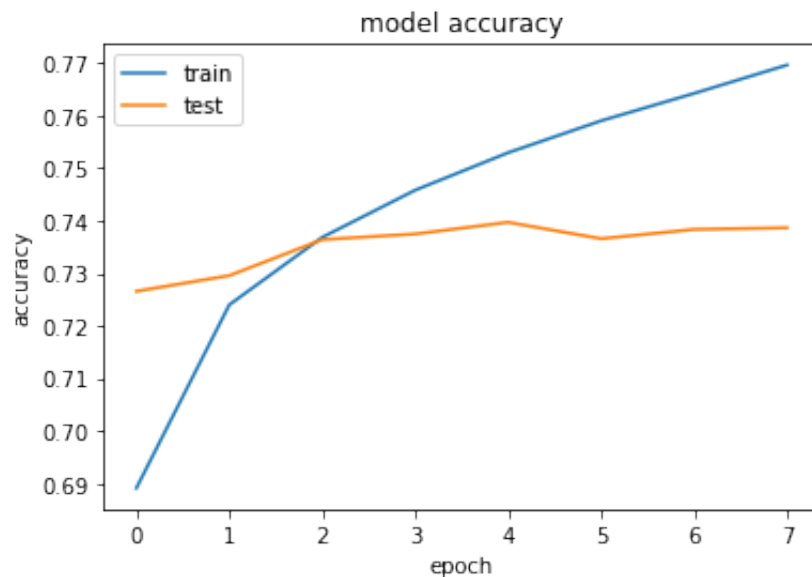
```



```

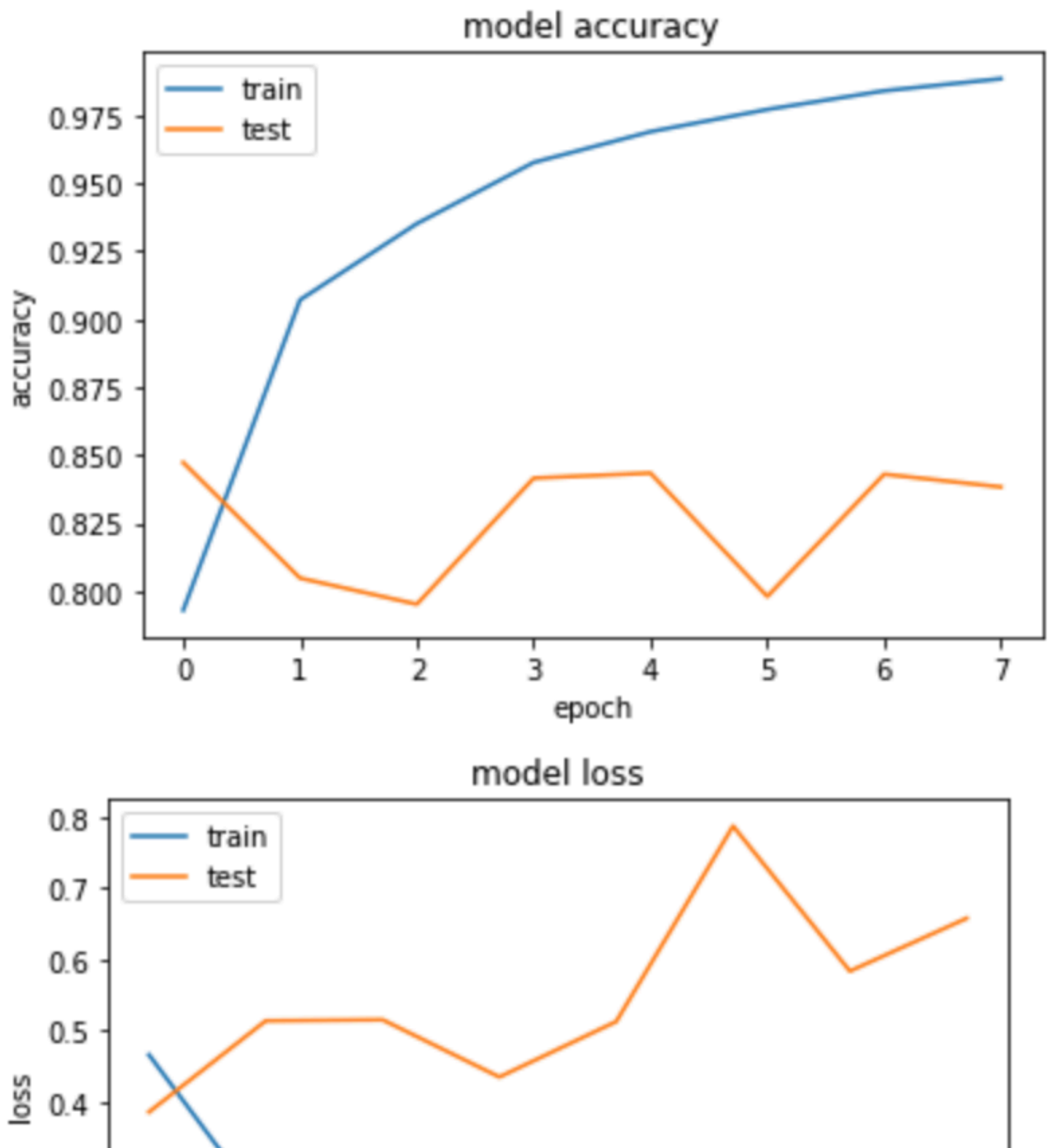
360000/360000 [=====] - 627s 2ms/step - loss: 0.77
Epoch 2/8
  64/360000 [.....] - ETA: 10:56 - loss: 0.7916
'skipping.' % (self.monitor), RuntimeWarning)
360000/360000 [=====] - 625s 2ms/step - loss: 0.66
Epoch 3/8
360000/360000 [=====] - 627s 2ms/step - loss: 0.63
Epoch 4/8
360000/360000 [=====] - 630s 2ms/step - loss: 0.61
Epoch 5/8
360000/360000 [=====] - 631s 2ms/step - loss: 0.59
Epoch 6/8
360000/360000 [=====] - 628s 2ms/step - loss: 0.58
Epoch 7/8
360000/360000 [=====] - 625s 2ms/step - loss: 0.56
Epoch 8/8
360000/360000 [=====] - 624s 2ms/step - loss: 0.55
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy', 'lr'])

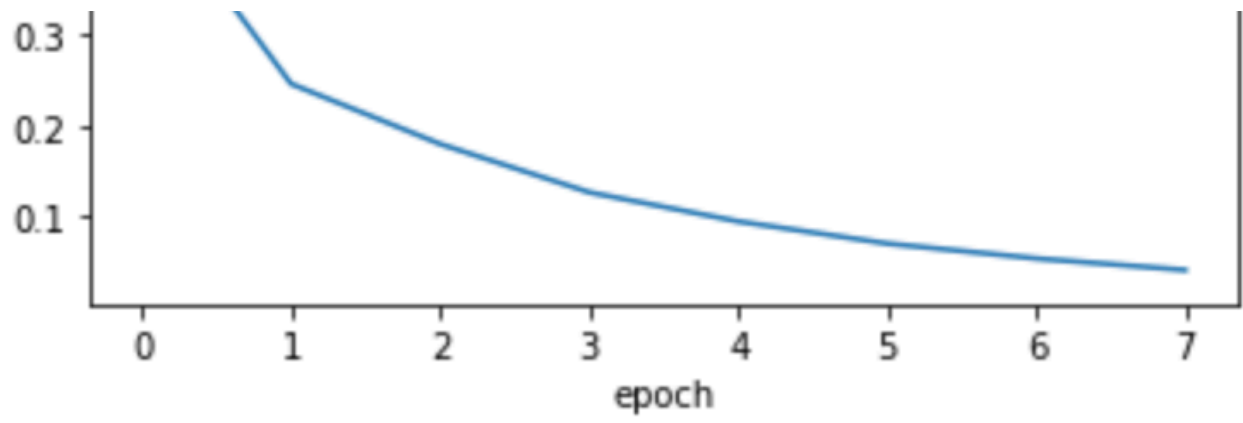
```

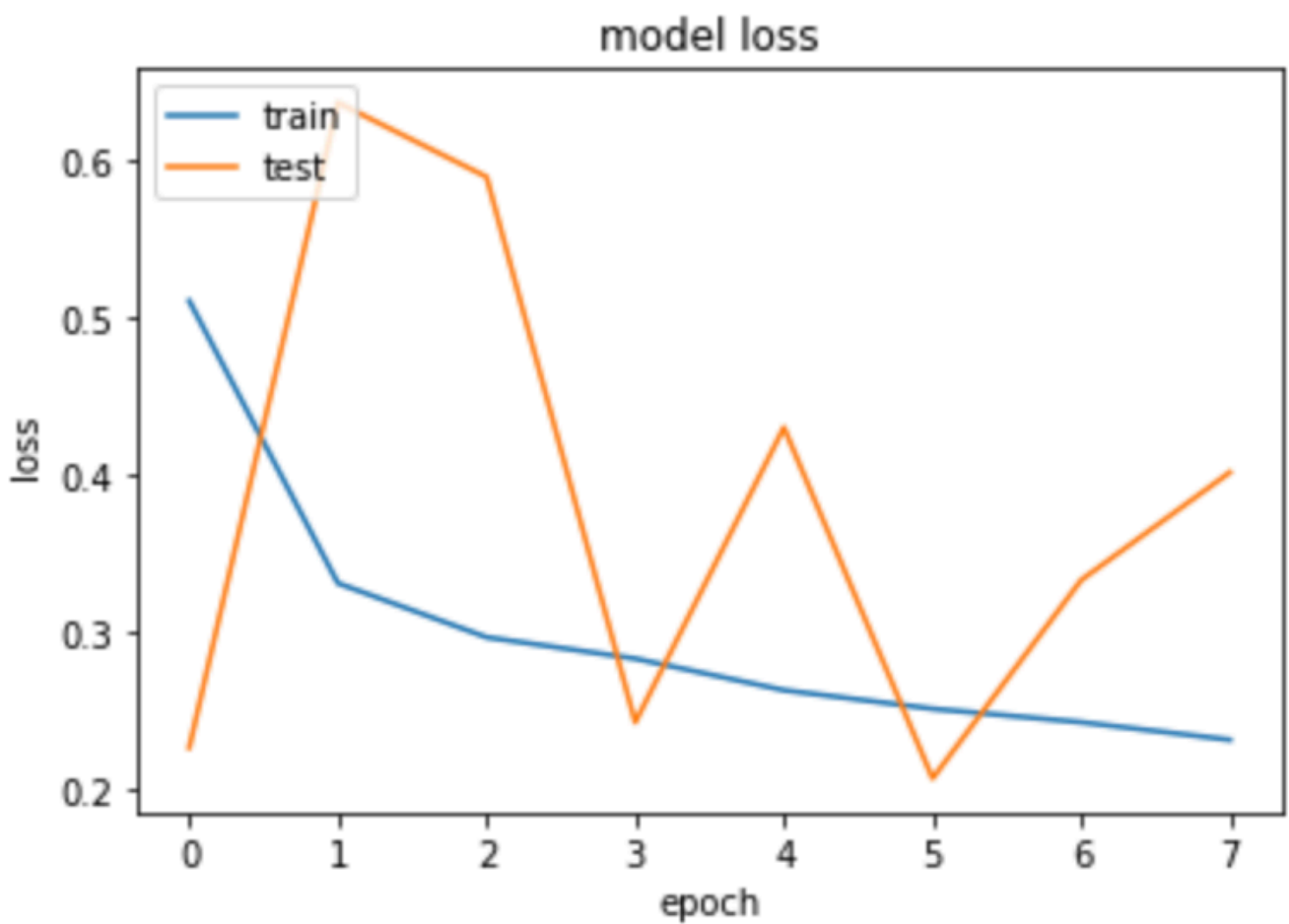
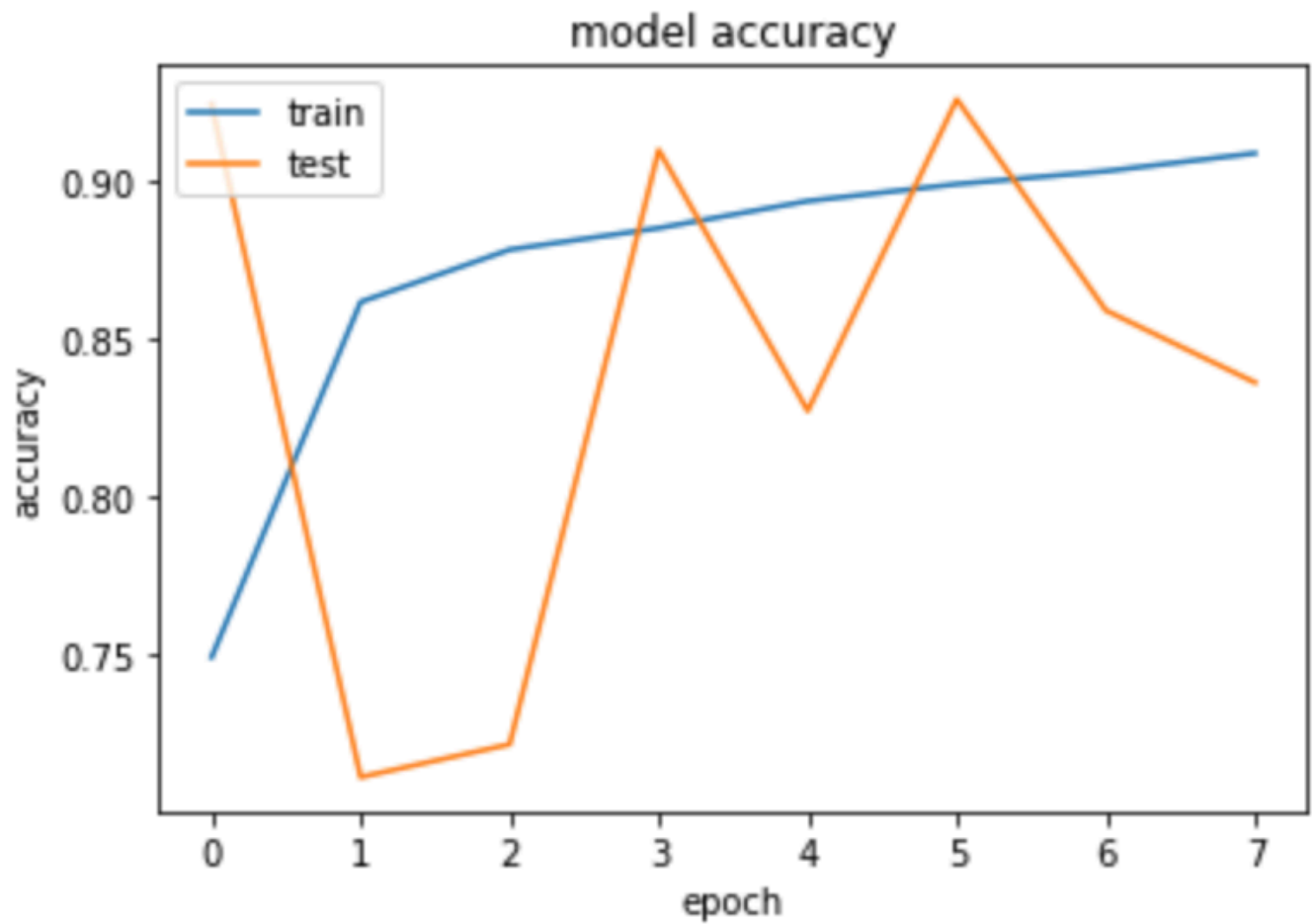


## ▼ Extend work on IMDB database

Apply our model on IMDB dataset, to see how its performance on this model. We also enable the pre-train word embedding (wiki-news-300d-1M-subword.vec) and compare its accuracy and loss graphs between IMDB with trained word embeddings and IMDB with pre-word embeddings. We believe this model could easily expand to any dataset if it satisfies the pattern "reviews / comments / articles + rates / sentiments / assessment".







# Conclusion

We can easily conclude that the accuracy on IMDB dataset is obviously better than Yelp dataset because the reviews in Yelp have more complicated sentences and large articles, it makes our model hard to predict. Thus, compared to above plots, we can tell that the accuracy between train word embedding on IMDB dataset and pre-train word embedding(wiki-news-300-1M-subword.vec) is similar, and the test loss are similar as well. Therefore, we have known that either we choose word\_embedding from\_scratch or use pre-trained, they are do not have different on accuracy.

In this notebook, we have proformed a document classification method by utilizing HAHNN architecture, which improved the accuracy score compare to other methods by approximately 3-5%. The attention mechanism of HAGNN architecture aggregates important words into sentence vectors and then aggregating important sentence vectors to a document vector. Even though, this method has its weakness as other methods have, for example it is relatively difficult to distinguish sarcastic reviews from others (actually we could get rid of these reviews by pre-precessing the datebase. For example, we could omit those admiring comments but with low rates/stars), and moreover, it does not perform well on a dataset with large context (like Yelp reviews) either. However, its attention mechanism of the model still improved the ability of classification and highlighted the importance of words and sentences that relative to final decision. All in all, this model has the best results compared to other methods, we might keep researching to further improve its performance on large/complicated sentences articles.

