Names: Cheng Zhao, Xicheng Tang

NetID: CZ1933, XT607

Prject: 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 inportant words in a sentence and inportant sentences in a document.

Saved successfully!

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

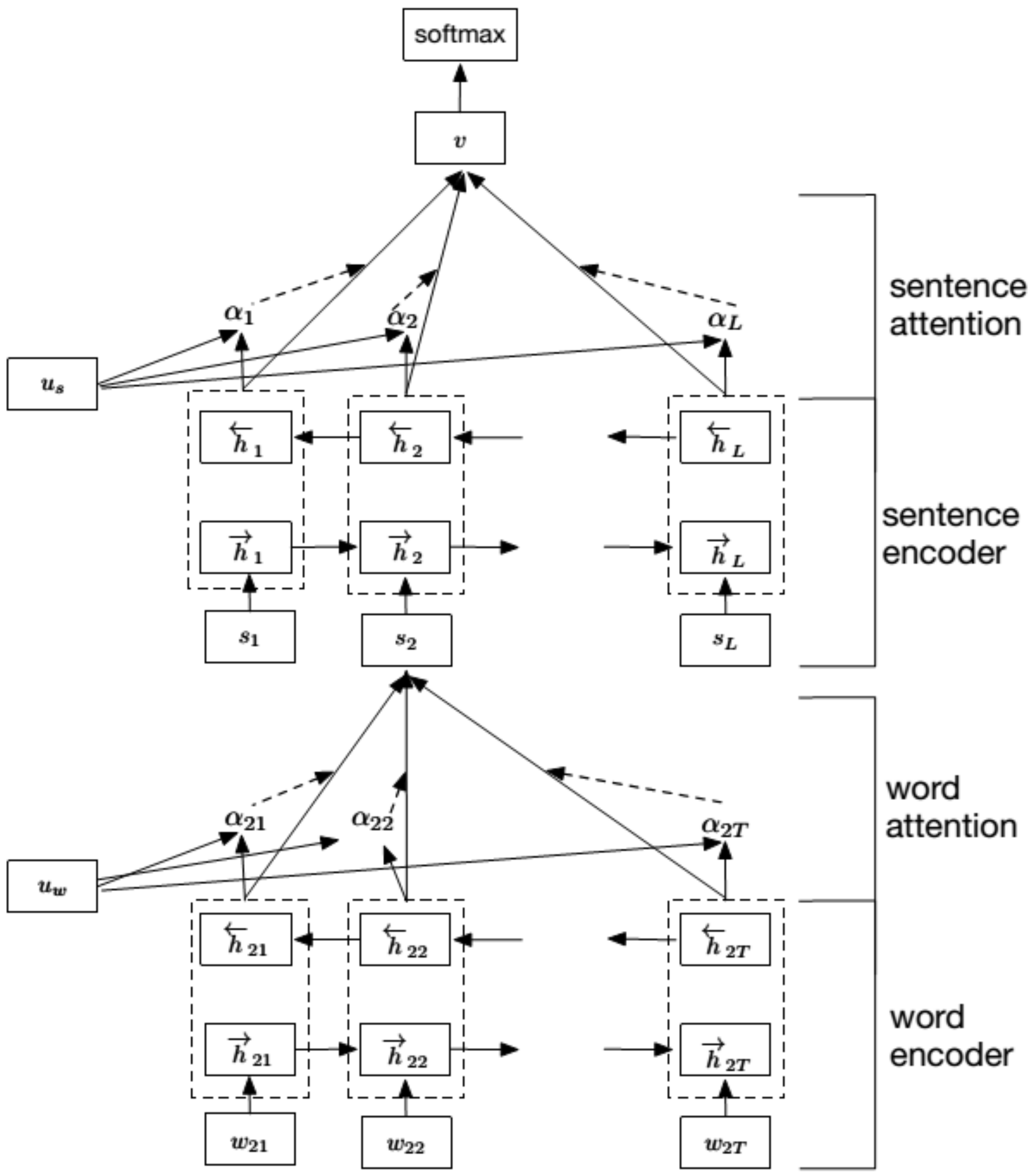


Figure 2: Hierarchical Attention Network.

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.

```
%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.

```
import os
from getpass import getpass

user = getpass('Kaggle Username: ')
key = getpass('Kaggle API key: ')

if '.kaggle' not in os.listdir('/root'):
    !mkdir ~/.kaggle

!touch /root/.kaggle/kaggle.json
!chmod 666 /root/.kaggle/kaggle.json

with open('/root/.kaggle/kaggle.json', 'w') as f:
    f.write('{"username":"%s","key":"%s"}' % (user, key))
!chmod 600 /root/.kaggle/kaggle.json
```

```
Kaggle Username: ··········
Kaggle API key: ··········
```

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

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

```
!kaggle datasets download -d luisfredgs/hahnn-for-document-classification
!unzip -o hahnn-for-document-classification.zip
# fixing in 400k reviews
!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.

```
!pip -q install gensim
!python -m spacy download en_core_web_md
!pip -q install paramiko

import datetime, pickle, os, codecs, re, string
import json
import random
import numpy as np
import keras
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import *
from keras import regularizers
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras import backend as K
from keras.utils import CustomObjectScope
from keras.engine.topology import Layer

#
from keras.engine import InputSpec

from keras import initializers

import pandas as pd
from tqdm import tqdm

import string
from spacy.lang.en import English
import gensim, nltk, logging

from nltk.corpus import stopwords
from nltk import tokenize
from nltk.stem import SnowballStemmer

nltk.download('punkt')
nltk.download('stopwords')

from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import en_core_web_sm

from IPython.display import HTML, display

import tensorflow as tf

from numpy.random import seed
from tensorflow import set_random_seed
os.environ['PYTHONHASHSEED'] = str(1024)
set_random_seed(1024)
seed(1024)
```

```
np.random.seed(1024)
random.seed(1024)
```

```
Collecting en_core_web_md==2.2.5
  Downloading https://github.com/explosion/spacy-models/releases/download/en_c
     |████████████████████████████████| 96.4MB 1.1MB/s
Requirement already satisfied: spacy>=2.2.2 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/pyt
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.6
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/python3.6
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.6
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/pytho
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/python3.
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python3.6/
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python3.6/
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/pytho
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/d
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/
Requirement already satisfied: importlib-metadata>=0.20; python_version < "3.8
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-pack
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-an
  Stored in directory: /tmp/pip-ephem-wheel-cache-t8ik43lb/wheels/df/94/ad/f5c
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.

```python
def load_subword_embedding_300d(word_index):
    print('load_subword_embedding...')
    embeddings_index = {}
    f = codecs.open("wiki-news-300d-1M-subword.vec", encoding='utf-8')
    for line in tqdm(f):
        values = line.rstrip().rsplit(' ')
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    f.close()
    print('found %s word vectors' % len(embeddings_index))

    #embedding matrix
    print('preparing embedding matrix...')
    words_not_found = []

    embedding_matrix = np.zeros((len(word_index) + 1, 300))

    for word, i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if (embedding_vector is not None) and len(embedding_vector) > 0:

            embedding_matrix[i] = embedding_vector
        else:
            words_not_found.append(word)

    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 though 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.

$$x_{it} = W_e w_{it}, t \in [1, T]$$
$$\overrightarrow{h_{it}} = \overrightarrow{GRU}(x_{it}), t \in [1, T]$$
$$\overleftarrow{h_{it}} = \overleftarrow{GRU}(x_{it}), t \in [T, 1]$$

where w_{it} is the tth word in a sentence, and x_{it} is the word vector that we get from embedding matrix W_{it}. $\overrightarrow{h_{it}}$ 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 α_{it} through a softmax function. After that, we compute the sentence vector s_i as a wighted sum of the word annotations based on the weights."(Hierarchical, by Yang)

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

where $h_{it} = [\overrightarrow{h_{it}}, \overleftarrow{h_{it}}]$.

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 concatente vectors of both directions to get an annotation of sentence.

$$\overrightarrow{h_i} = \overrightarrow{GRU}(s_i), i \in [1, L],$$
$$\overleftarrow{h_i} = \overleftarrow{GRU}(s_i), i \in [L, 1],$$

where $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$.

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).

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

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 Sofmax - layer to achieve the results.

$$p = softmax(W_c + b_c),$$

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. Finially, structure plot functions.

Reference: Zichao Yang1, Diyi Yang1, Chris Dyer1, Xiaodong He2, Alex Smola1, Eduard Hovy Carnegie Mellon University, 2Microsoft 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

```
YELP_DATA_PATH = 'yelp_reviews_sampling.json'
IMDB_DATA_PATH = 'imdb_reviews.csv'
SAVED_MODEL_DIR = 'saved_models'
SAVED_MODEL_FILENAME = 'model.h5'

if dataset is 'yelp':
    (X, Y) = load_data_yelp(path=YELP_DATA_PATH, size=400000)
else:
    (X, Y) = load_data_imdb(path=IMDB_DATA_PATH, size=49000)

```

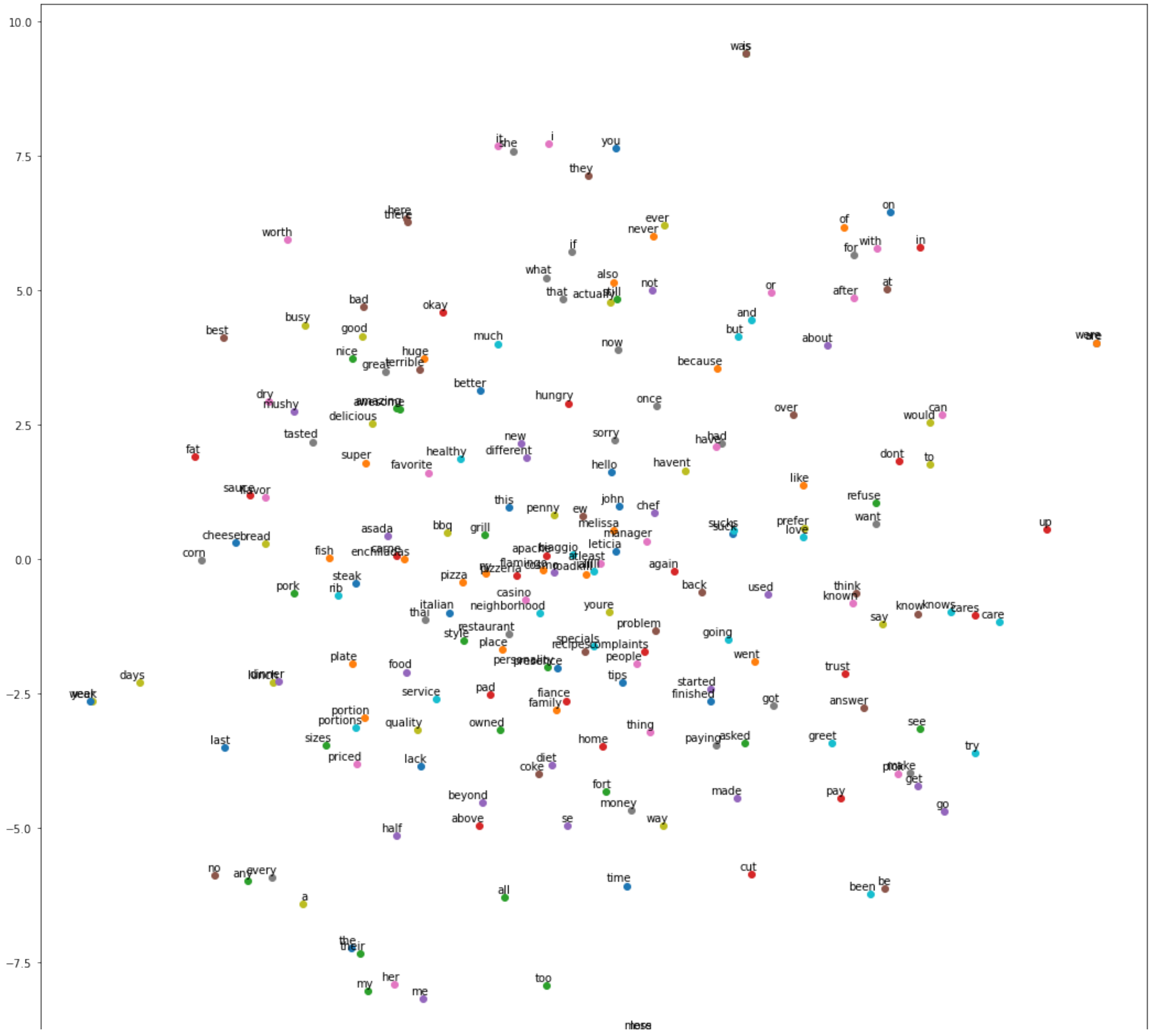
Plots Word Embedding chart

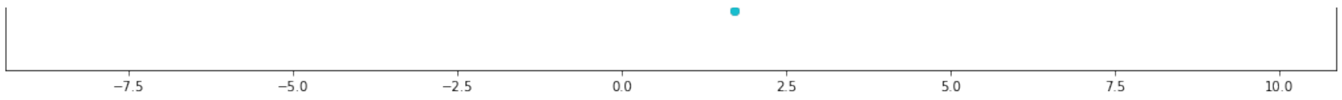
Visualize the word embedding in the graph

```
limit = 200
vector_dim = 200

# Fasttext
filename = './fasttext_model.txt'
model =  gensim.models.FastText.load(filename)
words = []
embedding = np.array([])
i = 0
for word in model.wv.vocab:
    if i == limit: break

    words.append(word)
    embedding = np.append(embedding, model[word])
    i += 1

embedding = embedding.reshape(limit, vector_dim)
tsne = TSNE(n_components=2)
low_dim_embedding = tsne.fit_transform(embedding)

```

```
21 plot_with_labels(low_dim_embedding, words)
```

```
2020-10-16 05:58:43,222 : INFO : loading FastText object from ./fasttext_model
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:252: UserW
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
2020-10-16 05:58:44,604 : INFO : loading wv recursively from ./fasttext_model.
2020-10-16 05:58:44,605 : INFO : loading vectors_ngrams from ./fasttext_model.
2020-10-16 05:58:44,664 : INFO : setting ignored attribute vectors_norm to Nor
2020-10-16 05:58:44,666 : INFO : setting ignored attribute vectors_vocab_norm
2020-10-16 05:58:44,667 : INFO : setting ignored attribute vectors_ngrams_norm
2020-10-16 05:58:44,668 : INFO : setting ignored attribute buckets_word to Nor
2020-10-16 05:58:44,669 : INFO : loading vocabulary recursively from ./fasttex
2020-10-16 05:58:44,670 : INFO : loading trainables recursively from ./fasttex
2020-10-16 05:58:44,671 : INFO : loading vectors_ngrams_lockf from ./fasttext_
2020-10-16 05:58:44,727 : INFO : loaded ./fasttext_model.txt
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: DeprecationWa
```





Taining model

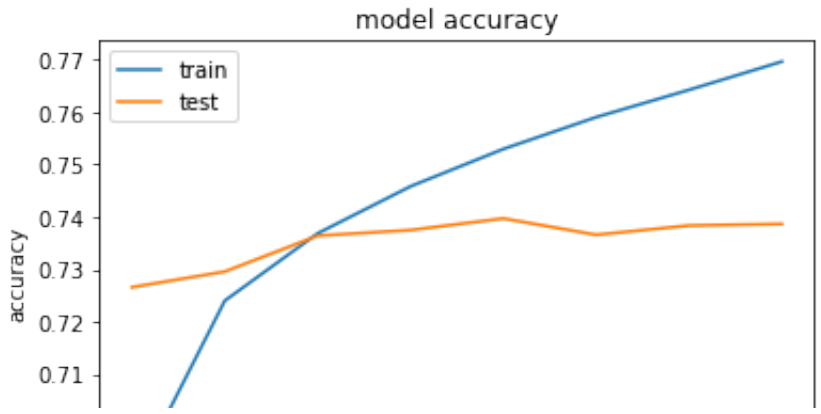
We first pick yelp database for testing and choose "word_embedding_type = from_scratch, word_vector_mode = fasttext, rnn_type = GRU"

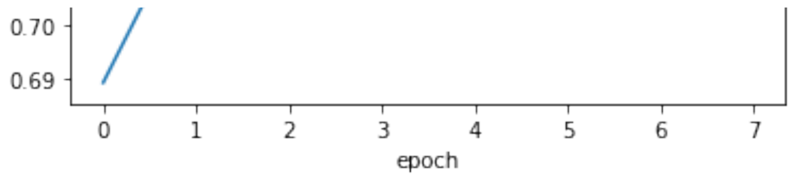
```
K.clear_session()
model = HAHNetwork()
model.train(X, Y, batch_size=64, epochs=8, embeddings_path=True, saved_model_dir
```

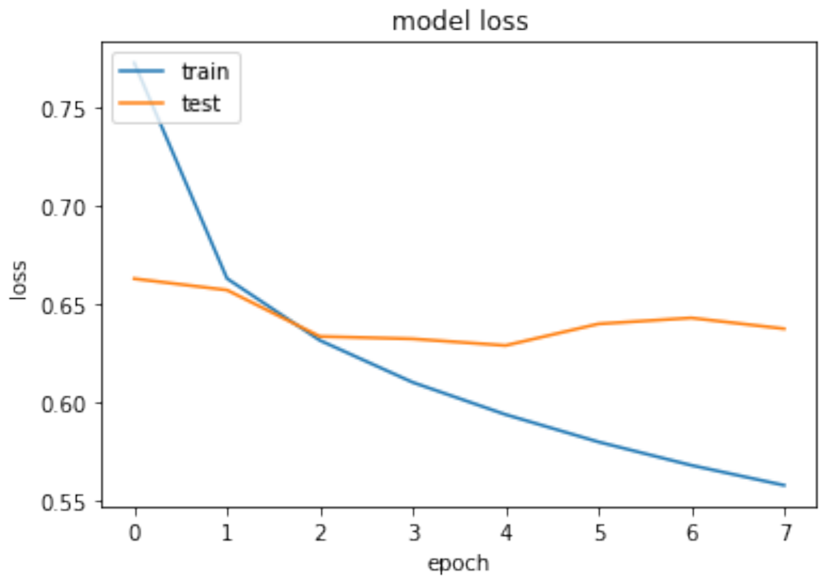
```
2020-10-16 10:27:20,302 : INFO : loading FastText object from ./fasttext_model
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:252: UserW
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
2020-10-16 10:27:21,439 : INFO : loading wv recursively from ./fasttext_model.
2020-10-16 10:27:21,441 : INFO : loading vectors_ngrams from ./fasttext_model.
2020-10-16 10:27:21,521 : INFO : setting ignored attribute vectors_norm to Non
2020-10-16 10:27:21,525 : INFO : setting ignored attribute vectors_vocab_norm
2020-10-16 10:27:21,526 : INFO : setting ignored attribute vectors_ngrams_norm
2020-10-16 10:27:21,526 : INFO : setting ignored attribute buckets_word to Non
2020-10-16 10:27:21,527 : INFO : loading vocabulary recursively from ./fasttex
```

```
2020-10-16 10:27:21,528 : INFO : loading trainables recursively from ./fasttex
2020-10-16 10:27:21,529 : INFO : loading vectors_ngrams_lockf from ./fasttext_
2020-10-16 10:27:22,045 : INFO : loaded ./fasttext_model.txt
tracking <tf.Variable 'word_attention/Variable:0' shape=(100, 1) dtype=float32
tracking <tf.Variable 'word_attention/Variable_1:0' shape=(50,) dtype=float32>
tracking <tf.Variable 'word_attention/Variable_2:0' shape=(50, 1) dtype=float3
Model: "model_1"
________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
input_1 (InputLayer)            (None, 50)           0
________________________________________________________________________________
word_embeddings (Embedding)     (None, 50, 200)      33156200    input_1[0][0]
________________________________________________________________________________
dropout_1 (Dropout)             (None, 50, 200)      0           word_embeddir
________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 50, 64)       38464       dropout_1[0][
________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 50, 64)       51264       dropout_1[0][
________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 50, 64)       64064       dropout_1[0][
________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 16, 64)       0           conv1d_1[0][0
________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 12, 64)       0           conv1d_2[0][0
________________________________________________________________________________
max_pooling1d_3 (MaxPooling1D)  (None, 10, 64)       0           conv1d_3[0][0
________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 38, 64)       0           max_pooling1c
                                                                 max_pooling1c
                                                                 max_pooling1c
________________________________________________________________________________
dropout_2 (Dropout)             (None, 38, 64)       0           concatenate_1
________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 38, 100)      34800       dropout_2[0][
________________________________________________________________________________
dense_transform_word (Dense)    (None, 38, 100)      10100       bidirectional
________________________________________________________________________________
word_attention (Attention)      (None, 100)          200         dense_transfc
================================================================================
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=flc
tracking <tf.Variable 'sentence_attention/Variable_1:0' shape=(50,) dtype=floa
tracking <tf.Variable 'sentence_attention/Variable_2:0' shape=(50, 1) dtype=fl
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: 0.7730
Epoch 2/8
    64/360000 [..............................] - ETA: 10:56 - loss: 0.7916 - a
  'skipping.' % (self.monitor), RuntimeWarning)
360000/360000 [==============================] - 625s 2ms/step - loss: 0.6633
Epoch 3/8
360000/360000 [==============================] - 627s 2ms/step - loss: 0.6319
Epoch 4/8
360000/360000 [==============================] - 630s 2ms/step - loss: 0.6105
Epoch 5/8
360000/360000 [==============================] - 631s 2ms/step - loss: 0.5941
Epoch 6/8
360000/360000 [==============================] - 628s 2ms/step - loss: 0.5802
Epoch 7/8
360000/360000 [==============================] - 625s 2ms/step - loss: 0.5683
Epoch 8/8
360000/360000 [==============================] - 624s 2ms/step - loss: 0.5582
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy', 'lr'])
```

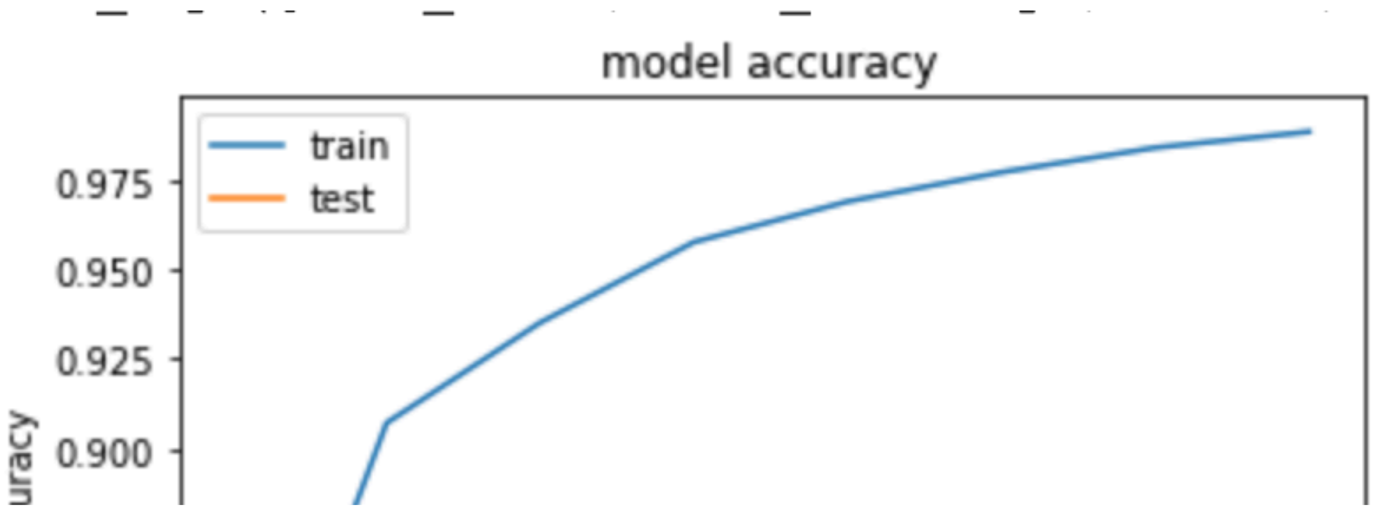


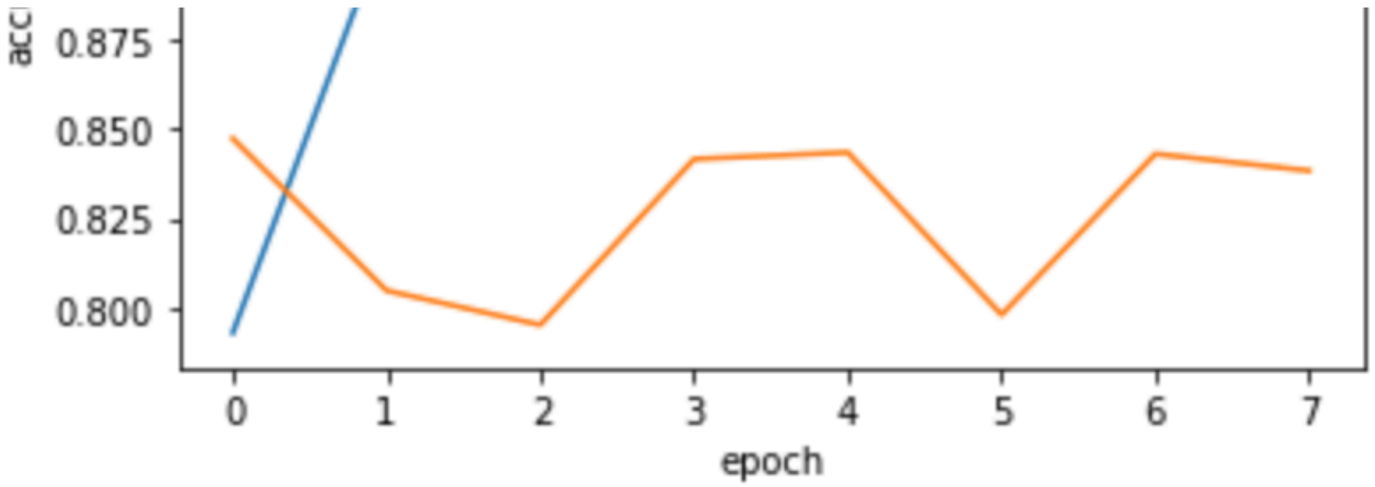




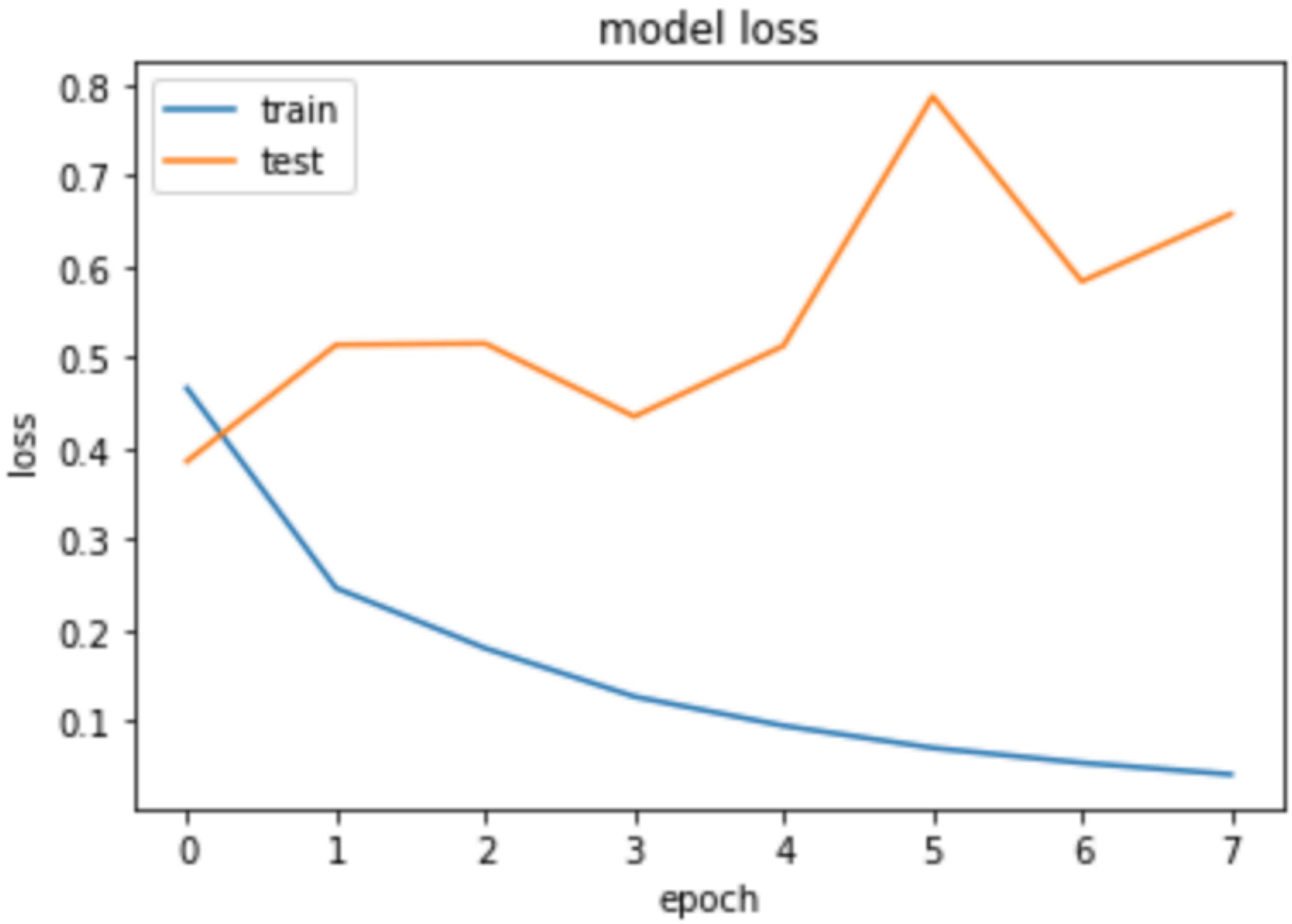
Extend work on IMDB database

Apply our model on IMDB datase, 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".

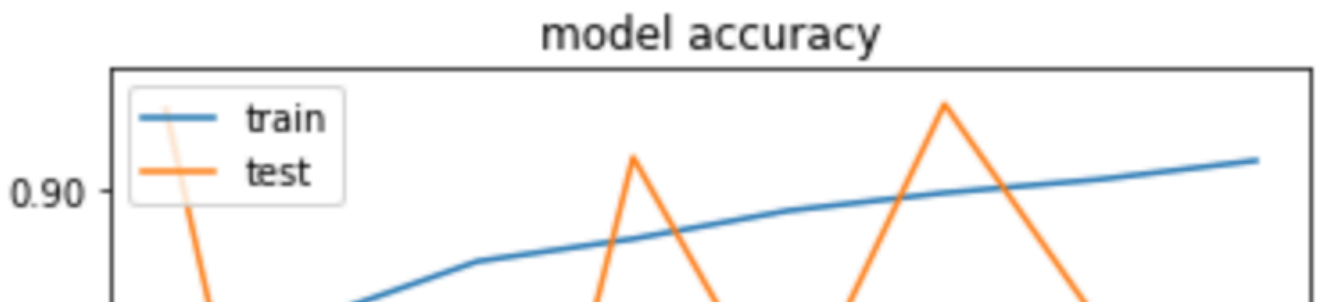




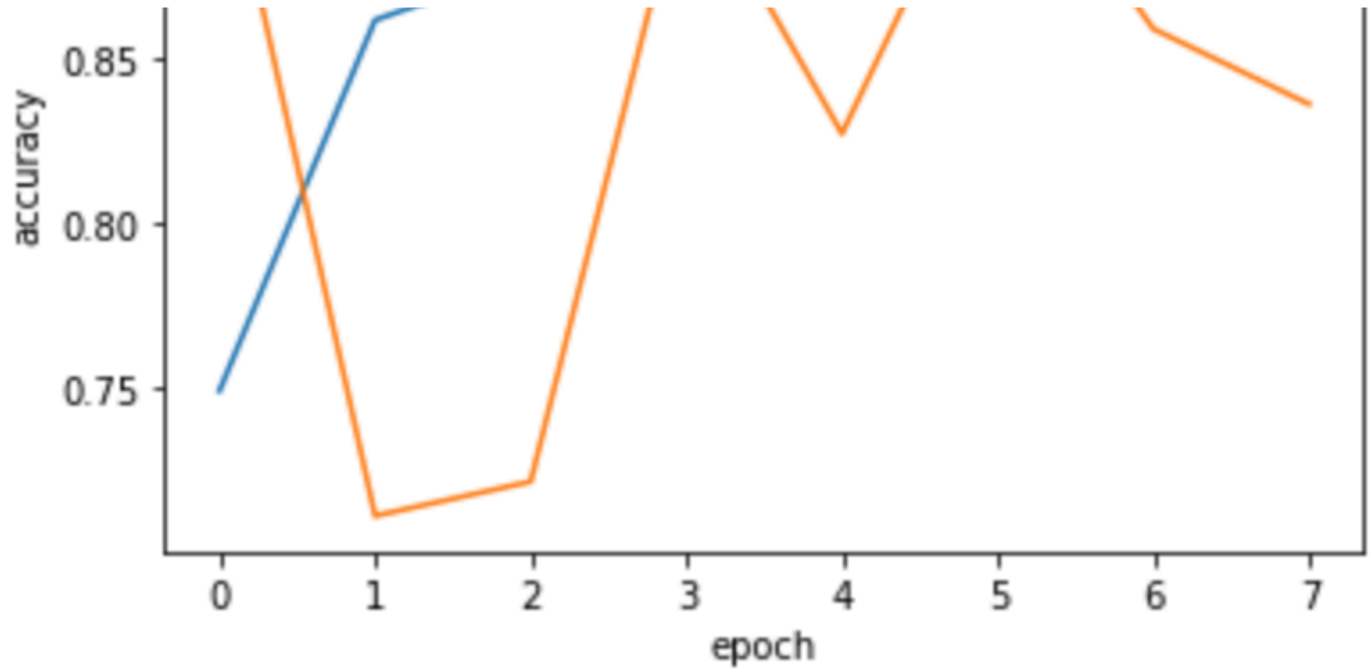
acc
0.875
0.850
0.825
0.800
0
1
2
3
4
5
6
7
epoch

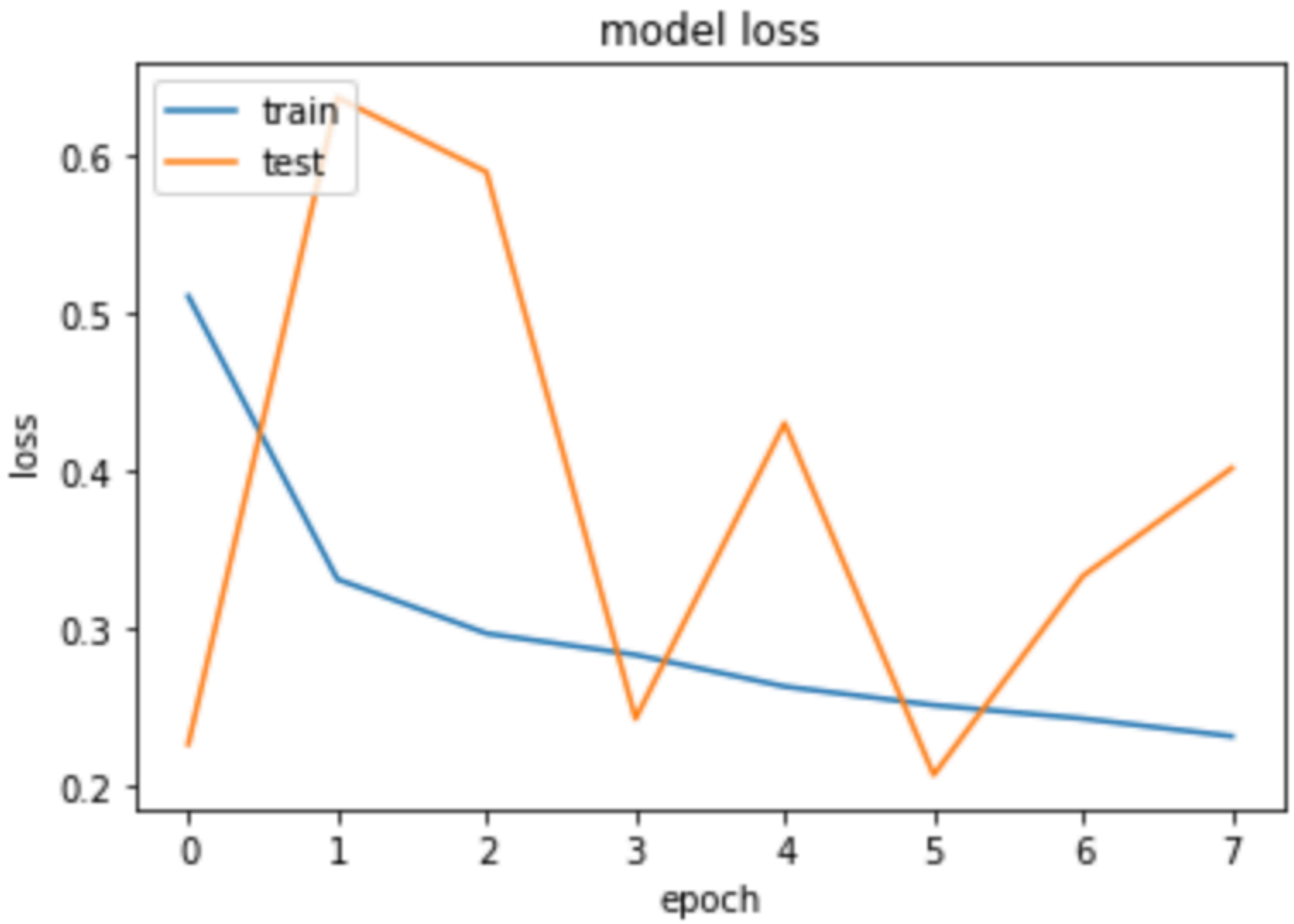


model loss
train
test
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
loss
0
1
2
3
4
5
6
7
epoch



model accuracy
train
test
0.90


accuracy
0.85
0.80
0.75
0
1
2
3
4
5
6
7
epoch


model loss
train
test
loss
0.6
0.5
0.4
0.3
0.2
0
1
2
3
4
5
6
7
epoch

Conclusion

We can easily conclude that the accuracy on IMDB dataset is obviously better than Yelp datase 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.