TREC 2021 Deep Learning Track (Information Retrieval)
Task1
Documents Ranking

Auther: Jingxuan Liu
Id: 173098550

In [2]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing import text, sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Conv1D, GlobalMaxPooling1D, MaxPooling1D
from sklearn.model_selection import train_test_split
import json
pd.options.mode.chained_assignment = None
```

In [2]:
```
def get_document(document_id):
    (string1, string2, bundlenum, position) = document_id.split('_')
    assert string1 == 'msmarco' and string2 == 'doc'

    with open(f'./msmarco_v2_doc/msmarco_doc_{bundlenum}', 'rt', encoding='utf8') as in_fh:
        in_fh.seek(int(position))
        json_string = in_fh.readline()
        document = json.loads(json_string)
        assert document['docid'] == document_id
        return document

document = get_document('msmarco_doc_31_726131')
print(document.keys())
```

```
dict_keys(['url', 'title', 'headings', 'body', 'docid'])
```

Prepare Training Data

In [3]:
```
train_qrels_df = pd.read_csv('./train/docv2_train_qrels.tsv', names = ['0','file','1'], header = None,delimiter = "\t")
train_queries_df = pd.read_csv('./train/docv2_train_queries.tsv', delimiter = "\t" , header=None, names = ['id','topic'
train_top100_df = pd.read_csv('./train/docv2_train_top100.txt', delimiter = " ", names = ['id','used','file','rank','sc
```

Dictionary for Document and Title

In [4]:
```
docs = train_qrels_df['file'].values
documents_index = dict()
for doc in docs:
    document = get_document(doc)
    if document['title'] != "":
        documents_index[doc] = document['title']
print(f'Found {len(documents_index)} doc number.')
dict(list(documents_index.items())[0:5])
```

Out[4]:
```
Found 267649 doc number.
{'msmarco_doc_10_1691063043': 'French and the francophonie in Canada',
 'msmarco_doc_05_72507775': 'Westminster, California (CA 92683) profile: population, maps, real estate, averages, homes,
statistics, relocation, travel, jobs, hospitals, schools, crime, moving, houses, news, sex offenders',
 'msmarco_doc_19_673141443': 'Westminster, California - Wikipedia',
 'msmarco_doc_19_673231526': 'Westminster, Massachusetts - Wikipedia',
 'msmarco_doc_19_673209131': 'Westminster, Maryland - Wikipedia'}
```

Dictionary for Id and Topic

In [5]:
```
topics_index = train_queries_df.set_index('id').to_dict()['topic']
print(f'Found {len(topics_index)} doc number.')
dict(list(topics_index.items())[0:10])
```

Out[5]:
```
Found 322196 doc number.
{121352: 'define extreme',
 510633: 'tattoo fixers how much does it cost',
 674172: 'what is a bank transit number',
 570009: 'what are the four major groups of elements',
 54528: 'blood clots in urine after menopause',
 738368: 'what is delta in2ition',
 507001: 'symptoms of an enlarged heart in dogs',
 466926: 'number of times congress voted to repeal aca',
 224811: 'how does a firefly light up',
 918533: 'what was introduced to the human diet in what year'}
```

In [6]:
```
train_top100_df['id'] = train_top100_df.id.map(topics_index)
train_top100_df['file'] = train_top100_df.file.map(documents_index)
train_top100_df = train_top100_df.dropna()
```

```
train_top100_df['x_train'] = train_top100_df['id']+" "+train_top100_df['file']
train_top100_df.head(5)
```

Out[6]:

	id	used	file	rank	score	username	x_train
26	Another name for the primary visual cortex is	Q0	Visual cortex - Wikipedia	27	11.718999	Anserini	Another name for the primary visual cortex is...
31	Another name for the primary visual cortex is	Q0	What Are the Functions of the Occipital Lobe? ...	32	11.654000	Anserini	Another name for the primary visual cortex is...
33	Another name for the primary visual cortex is	Q0	Occipital lobe - Wikipedia	34	11.637600	Anserini	Another name for the primary visual cortex is...
54	Another name for the primary visual cortex is	Q0	Motor cortex - Wikipedia	55	11.456200	Anserini	Another name for the primary visual cortex is...
56	Another name for the primary visual cortex is	Q0	Motor cortex - Wikipedia	57	11.455900	Anserini	Another name for the primary visual cortex is...

Tokenize and Pad Data for Training Data

In [7]:

```
max_feature = 20000
max_text_length = 20

x = train_top100_df['x_train'].values
y = train_top100_df['score'].values

x_tokenizer = text.Tokenizer(max_feature)
x_tokenizer.fit_on_texts(list(x))
x_tokenized = x_tokenizer.texts_to_sequences(x)
x_train_val = sequence.pad_sequences(x_tokenized, maxlen = max_text_length)
x_train_val
```

Out[7]:

```
array([[    0,     0,     0, ...,  2082,  2894,     9],
       [    0,     0,  1037, ...,  4241,    40,    99],
       [    0,     0,     0, ...,  9828,  4241,     9],
       ...,
       [    0,     0,     0, ..., 11625,  1999,     9],
       [   38,     3,  7035, ...,   762, 13080, 18735],
       [    0,     1,    38, ...,    76,   269,  1243]], dtype=int32)
```

Prepare Embedding Matrix using Pre-trained GloVe Embeddings Data

In [8]:

```
#!wget http://nlp.stanford.edu/data/glove.6B.zip
#!unzip -q glove.6B.zip

embedding_dim = 100
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print(f'Found {len(embeddings_index)} word vectors.')
```

```
Found 400000 word vectors.
```

In [9]:

```
embedding_matrix = np.zeros((max_feature, embedding_dim))
for word, index in x_tokenizer.word_index.items():
    if index > max_feature - 1:
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
embedding_matrix
```

Out[9]:

```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.15180001,  0.38409001,  0.89340001, ..., -0.27123001,
         0.22157   ,  0.92111999],
       [-0.54263997,  0.41475999,  1.03219998, ..., -1.29690003,
         0.76217002,  0.46349001],
       ...,
       [ 0.85588002,  0.44924   ,  0.40204   , ..., -0.14808001,
        -0.054877  , -0.19088   ],
       [-0.14505   ,  0.11744   ,  0.018758  , ...,  0.2252    ,
        -0.14031   , -0.078794  ],
       [ 0.61238998,  0.37639001,  0.16402   , ..., -0.33844   ,
        -0.29087001,  0.057413  ]])
```

Build the Model

Add Embedding Layer

In [10]:
```
model = Sequential()
model.add(Embedding(max_feature,
                    embedding_dim,
                    embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix),
                    trainable=False))
model.add(Dropout(0.2))
```

Build Rest Model

In [11]:
```
filters = 250
kernel_size = 3
hidden_dims = 250

model.add(Conv1D(filters,
                 kernel_size,
                 padding= 'valid'))
model.add(MaxPooling1D())
model.add(Conv1D(filters,
                 5,
                 padding = 'valid',
                 activation = 'relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(hidden_dims, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation = 'relu'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         2000000
_________________________________________________________________
dropout (Dropout)            (None, None, 100)         0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 250)         75250
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, None, 250)         0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, None, 250)         312750
_________________________________________________________________
global_max_pooling1d (Global (None, 250)               0
_________________________________________________________________
dense (Dense)                (None, 250)               62750
_________________________________________________________________
dropout_1 (Dropout)          (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 251
=================================================================
Total params: 2,451,001
Trainable params: 451,001
Non-trainable params: 2,000,000
_________________________________________________________________
```

In [12]:
```
model.compile(loss='mean_squared_error', optimizer='adam')
```

Trian the Model

In [13]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train_val, y,
                                                  test_size = 0.15, random_state = 1)
```

In [14]:
```
model.fit(x_train, y_train,
          batch_size = 32,
          epochs = 10,
          validation_data = (x_val, y_val))
```

```
Epoch 1/10
36916/36916 [==============================] - 185s 5ms/step - loss: 8.4429 - val_loss: 7.4058
Epoch 2/10
36916/36916 [==============================] - 190s 5ms/step - loss: 7.1149 - val_loss: 6.4973
Epoch 3/10
36916/36916 [==============================] - 219s 6ms/step - loss: 6.7526 - val_loss: 6.5247
Epoch 4/10
36916/36916 [==============================] - 234s 6ms/step - loss: 6.4529 - val_loss: 6.0008
Epoch 5/10
36916/36916 [==============================] - 228s 6ms/step - loss: 6.2993 - val_loss: 5.8188
Epoch 6/10
36916/36916 [==============================] - 233s 6ms/step - loss: 6.1738 - val_loss: 5.8334
Epoch 7/10
36916/36916 [==============================] - 235s 6ms/step - loss: 6.0732 - val_loss: 6.1077
Epoch 8/10
36916/36916 [==============================] - 231s 6ms/step - loss: 6.0327 - val_loss: 5.7452
Epoch 9/10
36916/36916 [==============================] - 221s 6ms/step - loss: 6.0175 - val_loss: 5.8306
```

```
Epoch 10/10
36916/36916 [==============================] - 227s 6ms/step - loss: 5.9336 - val_loss: 6.3716
```

Out[14]: <tensorflow.python.keras.callbacks.History at 0x1c1e0d0d0>

Evaluate Model
Part1: documents ranking

In [15]:
```
test_queries_df = pd.read_csv('./test/2021_queries.tsv', delimiter = "\t" , header=None, names = ['id','topic'])
test_queries_df.head(5)
```

Out[15]:

	id	topic
0	787021	what is produced by muscle
1	1049187	who recorded be my baby
2	1049519	who said no one can make you feel inferior
3	788054	what is ptf
4	2082	At about what age do adults normally begin to ...

In [16]:
```
new_topics_index = test_queries_df.set_index('id').to_dict()['topic']
topic_values = test_queries_df['id'].values
docs = train_qrels_df['file'].values
```

Mapping Documents to Each Topics

In [17]:
```
rows = []
for t in topic_values[:300]:
    for d in docs[:100]:
        rows.append([t, d])
```

In [18]:
```
rank_df = pd.DataFrame(rows, columns = ['id', 'file'])
test_df = rank_df.copy()
test_df['topic'] = test_df.id.map(new_topics_index)
test_df['title'] = test_df.file.map(documents_index)
test_df = test_df.dropna()
test_df['x_test'] = test_df['topic'] + " " + test_df['title']
```

In [19]:
```
x_test = test_df['x_test'].values
x_test_tokenized = x_tokenizer.texts_to_sequences(x_test)
x_testing = sequence.pad_sequences(x_test_tokenized, maxlen = max_text_length)
y_testing = model.predict(x_testing, verbose=1, batch_size=32)
```

```
919/919 [==============================] - 2s 2ms/step
```

In [20]:
```
test_df['score'] = [float(x) for x in y_testing]
test_df = test_df.drop(columns=['x_test','title','topic'])
rank_df = test_df.sort_values(by=['id','score'], ascending = False)
```

In [21]:
```
rank_df['rank'] = rank_df.groupby('id')['score'].rank(ascending=False).astype(int)
np.savetxt(r'./doc_ranking.txt', rank_df.values,fmt='%s')
rank_df
```

Out[21]:

	id	file	score	rank
24692	1136769	msmarco_doc_58_863235543	13.081055	1
24691	1136769	msmarco_doc_19_777567412	12.619779	2
24674	1136769	msmarco_doc_08_1708693754	12.508640	3
24685	1136769	msmarco_doc_10_452791723	12.342521	4
24634	1136769	msmarco_doc_41_412361142	12.064927	5
...	...	...	...	...
438	2082	msmarco_doc_05_261279398	8.629291	94
409	2082	msmarco_doc_03_889237013	8.259294	95
480	2082	msmarco_doc_10_342737855	7.871660	96
464	2082	msmarco_doc_03_891708626	7.736861	97
407	2082	msmarco_doc_10_783928151	7.708967	98

29400 rows × 4 columns

Part2: top100_reranking

I used docv2_tran_top100.txt for reranking

In [22]:
```
test_top100_df = pd.read_csv('./train/docv2_train_top100.txt', delimiter = " ", names = ['id','used','file','rank','sco
test_top100_df.head(10)
```

Out[22]:

	id	used	file	rank	score	username
0	3	Q0	msmarco_doc_22_449579381	1	12.077800	Anserini
1	3	Q0	msmarco_doc_16_3765146983	2	11.966400	Anserini
2	3	Q0	msmarco_doc_16_3772357379	3	11.966399	Anserini
3	3	Q0	msmarco_doc_00_821814871	4	11.954800	Anserini
4	3	Q0	msmarco_doc_01_1914713906	5	11.925100	Anserini
5	3	Q0	msmarco_doc_00_599701433	6	11.904700	Anserini
6	3	Q0	msmarco_doc_04_176608381	7	11.897000	Anserini
7	3	Q0	msmarco_doc_08_185507485	8	11.857600	Anserini
8	3	Q0	msmarco_doc_05_1605175028	9	11.819000	Anserini
9	3	Q0	msmarco_doc_26_1982585323	10	11.798000	Anserini

In [23]:
```
test_top100_df['topic'] = test_top100_df['id'].map(topics_index)
test_top100_df['title'] = test_top100_df['file'].map(documents_index)
test_top100_df = test_top100_df.dropna()
test_top100_df['x_test'] = test_top100_df['topic'] + " " + test_top100_df['title']
```

In [24]:
```
x_top100_testing = test_top100_df['x_test'].values
x_top100_test_tokenized = x_tokenizer.texts_to_sequences(x_top100_testing)
x_top100_testing = sequence.pad_sequences(x_top100_test_tokenized, maxlen = max_text_length)
y_top100_testing = model.predict(x_top100_testing, verbose=1, batch_size=32)
```

```
43431/43431 [==============================] - 74s 2ms/step
```

In [25]:
```
rerank_top100_df = test_top100_df.drop(columns = ['x_test', 'title', 'topic', 'username', 'score', 'rank', 'used'])
rerank_top100_df['score'] = [float(x) for x in y_top100_testing]
rerank_top100_df = rerank_top100_df.sort_values(by=['id','score'], ascending = False)
```

In [26]:
```
rerank_top100_df['rank'] = rerank_top100_df.groupby('id').rank(ascending=False).astype(int)
np.savetxt(r'./doc_top100_reranking.txt', rerank_top100_df.values,fmt='%s')
rerank_top100_df
```

Out[26]:

	id	file	score	rank
32218735	1185869	msmarco_doc_10_423537489	10.674301	1
32218767	1185869	msmarco_doc_17_1254574904	10.617181	2
32218720	1185869	msmarco_doc_06_1612776115	10.541935	3
32218758	1185869	msmarco_doc_17_4719433890	10.375751	4
32218754	1185869	msmarco_doc_30_214786134	9.955307	5
...	...	...	...	...
84	3	msmarco_doc_18_3676159488	10.051237	3
54	3	msmarco_doc_18_2698878789	9.724071	4
56	3	msmarco_doc_18_423924267	9.724071	4
26	3	msmarco_doc_19_478270426	9.580538	6
67	3	msmarco_doc_16_2338409058	9.008437	7

1389777 rows × 4 columns

In [ ]: