Baseline Modeling

In [1]:

```
import os
import json
import gc
import pickle

import numpy as np
import pandas as pd
from tqdm import tqdm_notebook as tqdm
from tensorflow.keras.models import Model
from tensorflow.keras.layers import *
from tensorflow.keras.preprocessing import text, sequence
import fasttext
```

Build Dataset

In [2]:

```
# https://stackoverflow.com/questions/20885797/iteratively-parse-json-file
def build_train(train_path, n_rows=200000, sampling_rate=15):
    with open(train_path) as f:
        processed_rows = []

        for i in tqdm(range(n_rows)):
            line = f.readline()
            if not line:
                break

            line = json.loads(line)

            text = line['document_text'].split(' ')
            question = line['question_text']
            annotations = line['annotations'][0]

            for i, candidate in enumerate(line['long_answer_candidates']):
                label = i == annotations['long_answer']['candidate_index']

                start = candidate['start_token']
                end = candidate['end_token']

                if label or (i % sampling_rate == 0):
                    processed_rows.append({
                        'text': " ".join(text[start:end]),
                        'is_long_answer': label,
                        'question': question,
                        'annotation_id': annotations['annotation_id']
                    })

        train = pd.DataFrame(processed_rows)

        return train
```

In [3]:

```
def build_test(test_path):
    with open(test_path) as f:
        processed_rows = []

        for line in tqdm(f):
            line = json.loads(line)

            text = line['document_text'].split(' ')
            question = line['question_text']
            example_id = line['example_id']

            for candidate in line['long_answer_candidates']:
                start = candidate['start_token']
                end = candidate['end_token']

                processed_rows.append({
                    'text': " ".join(text[start:end]),
                    'question': question,
                    'example_id': example_id,
                    'sequence': f'{start}:{end}'

                })

        test = pd.DataFrame(processed_rows)

    return test
```

In [4]:

```
directory = '/kaggle/input/tensorflow2-question-answering/'
train_path = directory + 'simplified-nq-train.jsonl'
test_path = directory + 'simplified-nq-test.jsonl'

train = build_train(train_path)
test = build_test(test_path)
train_target = train.is_long_answer.astype(int).values
```

In [5]:

```
train.sample(5)
```

Out[5]:

	text	is_long_answer	question	annotation_id
1807747	<Li> Welkom Hoërskool , Welkom </Li>	False	list of agricultural high schools in south africa	17414750892970110456
1541210	<Li> Jump up ^ `` Diaghilev London Walk '' . V...	False	at its first performance the rite of spring pr...	2290474378537056466
210418	<Table> British Columbia Liberal Party leaders...	True	who won the liberal leadership in british colu...	4027934080044497844
107785	<Tr> <Th> Slovakia (Singles Digitál Top 100)...	False	justin bieber i'll show you mp3 song download	10991333931554020085
1590139	<P> The 21 World Cup tournaments have been won...	True	which african country has ever won the world cup	13524104377121675701

In [6]:

```
test.head()
```

Out[6]:

	text	question	example_id	sequence
0	<Table> <Tr> <Th_colspan="2"> High Commission ...	who is the south african high commissioner in ...	-1220107454853145579	18:136
1	<Tr> <Th_colspan="2"> High Commission of South...	who is the south african high commissioner in ...	-1220107454853145579	19:30
2	<Tr> <Th> Location </Th> <Td> Trafalgar Square...	who is the south african high commissioner in ...	-1220107454853145579	34:45
3	<Tr> <Th> Address </Th> <Td> Trafalgar Square ...	who is the south african high commissioner in ...	-1220107454853145579	45:59

	text	question	example_id	sequence
4	<Tr> <Th> Coordinates </Th> <Td> 51 ° 30′ 30 ...	who is the south african high commissioner in ...	1220107454853145579	69.126

In [7]:

```
train_target
```

Out[7]:

```
array([0, 0, 0, ..., 1, 0, 0])
```

Preprocessing

In [8]:

```
def find_avg_length(df):
    length = []
    df.apply(lambda x: length.append(len(x.split())))
    return np.mean(length)

def count_less_than_x(df, thr):
    temp = df.apply(lambda x: True if len(x.split())>thr else False)
    count = temp[temp == False]
    return (len(count)*100)/len(df)
```

In [9]:

```
print("The average length of all the document_text is {}".format(find_avg_length(train['t
ext'])))
```

```
The average length of all the document_text is 86.9223448525609
```

In [10]:

```
for i in [100, 200, 250, 300, 350, 400]:
    print("Percentage of documents with length less than {}: {}".format(i, count_less_th
an_x(train['text'], i)))
```

```
Percentage of documents with length less than 100: 77.94376116706712
Percentage of documents with length less than 200: 91.82614912177553
Percentage of documents with length less than 250: 94.70552242362689
Percentage of documents with length less than 300: 96.26487945030713
Percentage of documents with length less than 350: 97.24196205304415
Percentage of documents with length less than 400: 97.83677585114253
```

In [11]:

```
print("The average length of all the Question_text is {}".format(find_avg_length(train['q
uestion'])))
```

```
The average length of all the Question_text is 9.19653882054959
```

In [12]:

```
for i in [10, 15, 20, 25]:
    print("Percentage of Question with length less than {}: {}".format(i, count_less_tha
n_x(train['question'], i)))
```

```
Percentage of Question with length less than 10: 83.60692756196063
Percentage of Question with length less than 15: 98.81807293233004
Percentage of Question with length less than 20: 99.99412054240858
Percentage of Question with length less than 25: 100.0
```

In [13]:

```
def texts_to_sequences(train, test, tokenizer):
    train_text = tokenizer.texts_to_sequences(train.text.values)
    train_questions = tokenizer.texts_to_sequences(train.question.values)
    test_text = tokenizer.texts_to_sequences(test.text.values)
    test_questions = tokenizer.texts_to_sequences(test.question.values)
    return train_text, train_questions, test_text, test_questions
def pad_sequence(train_, test_, padding_var = 20):

    train_var = sequence.pad_sequences(train_, maxlen=padding_var)
    test_var = sequence.pad_sequences(test_, maxlen=padding_var)
    return train_var, test_var
```

In [14]:

```
def Build_dataset(train, test):
    print("="*100)
    print("Tokenizing the sequences...")
    tokenizer = text.Tokenizer(lower=False, num_words=80000)

    t = train['text'].append(train['question'], ignore_index=True).to_frame(name = 'text
')
    tokenizer.fit_on_texts(t['text'])
    print("Done")
    print("="*100)
    print()
    print("Starting Text to sequences process...")
    train_text, train_questions, test_text, test_questions = texts_to_sequences(train, t
est, tokenizer)
    print("Done")
    print("="*100)
    print()

    print("Padding Each Sequences...")
    train_text, test_text = pad_sequence(train_text, test_text, 300)
    train_questions, test_questions = pad_sequence(train_questions, test_questions, 20)
    print("Done")
    return train_text, train_questions, test_text, test_questions, tokenizer
```

In [15]:

```
train_text, train_questions, test_text, test_questions, tokenizer = Build_dataset(train,
test)
```

```
==========================================================================================
===========
Tokenizing the sequences...
Done
==========================================================================================
===========

Starting Text to sequences process...
Done
==========================================================================================
===========

Padding Each Sequences...
Done
```

In [16]:

```
# saving
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [17]:

```
train_questions[0]
```

Out[17]:

```
array([    0,     0,     0,     0,     0,     0,     0,     0,    43,
          13,     3,    78,   648,   255,     6, 22867,     8,   586,
        6071,  4866], dtype=int32)
```

In [18]:

```
del train
```

Modelling

In [19]:

```
def build_embedding_matrix(tokenizer, path):
    embedding_matrix = np.zeros((tokenizer.num_words + 1, 300))
    ft_model = fasttext.load_model(path)

    for word, i in tokenizer.word_index.items():
        if i >= tokenizer.num_words - 1:
            break
        embedding_matrix[i] = ft_model.get_word_vector(word)

    return embedding_matrix
```

In [20]:

```
def build_model(embedding_matrix):
    embedding = Embedding(
        *embedding_matrix.shape,
        weights=[embedding_matrix],
        trainable=False,
        mask_zero=True
    )

    q_in = Input(shape=(None,))
    q = embedding(q_in)
    q = SpatialDropout1D(0.2)(q)
    q = Bidirectional(LSTM(100, return_sequences=True))(q)
    q = GlobalMaxPooling1D()(q)


    t_in = Input(shape=(None,))
    t = embedding(t_in)
    t = SpatialDropout1D(0.2)(t)
    t = Bidirectional(LSTM(150, return_sequences=True))(t)
    t = GlobalMaxPooling1D()(t)

    x = concatenate([q, t])
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)

    out = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[t_in, q_in], outputs=out)
    model.compile(loss='binary_crossentropy', optimizer='adam')

    return model
```

In [21]:

```
path = '/kaggle/input/fasttext-crawl-300d-2m-with-subword/crawl-300d-2m-subword/crawl-300
d-2M-subword.bin'
embedding_matrix = build_embedding_matrix(tokenizer, path)
```

In [22]:

```
model = build_model(embedding_matrix)
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
__________
Layer (type)                    Output Shape          Param #     Connected to
```

```
==========================================================================================
=========
input_1 (InputLayer)            [(None, None)]       0

__________________________________________________________________________________________
_________
input_2 (InputLayer)            [(None, None)]       0

__________________________________________________________________________________________
_________
embedding (Embedding)           (None, None, 300)    24000300     input_1[0][0]

                                                                  input_2[0][0]

__________________________________________________________________________________________
_________
spatial_dropout1d (SpatialDropo (None, None, 300)    0            embedding[0][0]

__________________________________________________________________________________________
_________
spatial_dropout1d_1 (SpatialDro (None, None, 300)    0            embedding[1][0]

__________________________________________________________________________________________
_________
bidirectional (Bidirectional)   (None, None, 200)    320800       spatial_dropout1d[0][0]

__________________________________________________________________________________________
_________
bidirectional_1 (Bidirectional) (None, None, 300)    541200       spatial_dropout1d_1[0][0
]
__________________________________________________________________________________________
_________
global_max_pooling1d (GlobalMax (None, 200)          0            bidirectional[0][0]

__________________________________________________________________________________________
_________
global_max_pooling1d_1 (GlobalM (None, 300)          0            bidirectional_1[0][0]

__________________________________________________________________________________________
_________
concatenate (Concatenate)       (None, 500)          0            global_max_pooling1d[0]
[0]
                                                                  global_max_pooling1d_1
[0][0]
__________________________________________________________________________________________
_________
dense (Dense)                   (None, 512)          256512       concatenate[0][0]

__________________________________________________________________________________________
_________
dropout (Dropout)               (None, 512)          0            dense[0][0]

__________________________________________________________________________________________
_________
dense_1 (Dense)                 (None, 256)          131328       dropout[0][0]

__________________________________________________________________________________________
_________
dropout_1 (Dropout)             (None, 256)          0            dense_1[0][0]

__________________________________________________________________________________________
_________
dense_2 (Dense)                 (None, 1)            257          dropout_1[0][0]

==========================================================================================
=========
Total params: 25,250,397
Trainable params: 1,250,097
Non-trainable params: 24,000,300
__________________________________________________________________________________________
_________
```

In [27]:

```
from tensorflow.keras.callbacks import ModelCheckpoint
filepath="training_1/weights-improvement-{epoch:02d}.ckpt"
checkpoint_dir = os.path.dirname(filepath)
# Create a callback that saves the model's weights
cp_callback = ModelCheckpoint(filepath=filepath,save_weights_only=True, verbose=1, save_
best_only=True)
```

In [29]:

```
history = model.fit(
    [train_text, train_questions],
    train_target,
    epochs=2,
    validation_split=0.2,
    batch_size=256# ,callbacks=[cp_callback]

)
```

```
Train on 1537556 samples, validate on 384390 samples
Epoch 1/2
1537556/1537556 [==============================] - 6529s 4ms/sample - loss: 0.1325 - val_
loss: 0.1185
Epoch 2/2
1537556/1537556 [==============================] - 6524s 4ms/sample - loss: 0.1175 - val_
loss: 0.1090
```

Save Model

In [30]:

```
model.save('model.h5')
```

Inference Step

In [59]:

```
test_target = model.predict([test_text, test_questions], batch_size=256)
```

In [97]:

```
test['target'] = test_target

result = (
    test.query('target > 0.05').groupby('example_id').max().reset_index().loc[:, ['examp
le_id', 'sequence']]
)

result = pd.concat([
    result.assign(example_id=lambda example_id: example_id + '_long'),
    result.assign(example_id=lambda example_id: example_id + '_short')
])

result.head()
```

Out[97]:

	example_id	sequence
0	-1011141123527297803_long	931:1088
1	-1028916936938579349_long	781:923
2	-1055197305756217938_long	741:998
3	-1074129516932871805_long	89:103
4	-1114334749483663139_long	968:1083

In [104]:

```
result.shape
```

Out[104]:

```
(686, 2)
```

Submit

In [102]:

```
submission = pd.read_csv("../input/tensorflow2-question-answering/sample_submission.csv")
final_submission = (
    submission.drop(columns='PredictionString').merge(result, on=['example_id'], how='le
ft')
)
final_submission = final_submission.rename(columns={'sequence': 'PredictionString'})
final_submission.to_csv("submission.csv", index=False)
```

In [103]:

```
final_submission.head()
```

Out[103]:

	example_id	PredictionString
0	-1011141123527297803_long	931:1088
1	-1011141123527297803_short	931:1088
2	-1028916936938579349_long	781:923
3	-1028916936938579349_short	781:923
4	-1055197305756217938_long	741:998

In [ ]:

```

```