In [28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import re
from nltk.corpus import stopwords
import pickle
from tqdm import tqdm
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
import tensorflow as tf
import random
import math
from textblob import TextBlob
from collections import Counter
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk, string
from collections import Counter
from nltk.stem import WordNetLemmatizer
import Levenshtein
import fasttext
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, c
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.utils import plot_model
from tensorflow.keras import backend as K
from tensorflow.keras.regularizers import l2, l1, l1_l2
%matplotlib inline
```

In [2]:
```
import tensorflow as tf
tf.config.experimental.allow_growth = True
```

In [3]:
```
import logging
logging.getLogger('tensorflow').setLevel(logging.ERROR)  # suppress warnings
```

In [4]:
```
# Setting seed
global_seed = 42
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [5]:
```
# loading the dataframes in which features have already been extracted
train_df = pd.read_csv("train_df.csv")
val_df = pd.read_csv("val_df.csv")
test_df = pd.read_csv("test_df.csv")
```

In [6]:
```
# getting lists of sentence1 and sentence 2 for train and validation datsets
train_sentence1 = train_df['sentence1_preprocessed'].tolist()
train_sentence2 = train_df['sentence2_preprocessed'].tolist()
val_sentence1 = val_df['sentence1_preprocessed'].tolist()
val_sentence2 = val_df['sentence2_preprocessed'].tolist()
```

MLP on avg fasttext embeddings and extracted features

In [7]:
```
x_train_s1_avg_fasttext = np.load('x_train_s1_avg_fasttext.npy', mmap_mode='
x_train_s2_avg_fasttext = np.load('x_train_s2_avg_fasttext.npy', mmap_mode='
x_val_s1_avg_fasttext = np.load('x_val_s1_avg_fasttext.npy', mmap_mode='r')
x_val_s2_avg_fasttext = np.load('x_val_s2_avg_fasttext.npy', mmap_mode='r')
x_test_s1_avg_fasttext = np.load('x_test_s1_avg_fasttext.npy', mmap_mode='r'
x_test_s2_avg_fasttext = np.load('x_test_s2_avg_fasttext.npy', mmap_mode='r'
```

In [8]:
```
to_keep = ['log_bleu_score', 'sum_polarity', 'diff_subjectivity',
       'cosine_similarity', 'tfidf_cosine_sim', 'diff_num_words', 's1_CC',
       's1_IN','s1_PRP', 's1_PRP$', 's1_RB', 's1_RBR', 's1_RBS', 's1_UH', 's
       's2_PRP$', 's2_RB', 's2_RBR', 's2_RBS', 's2_UH', 'ratio_noun', 'ratio
       'diff_adjective', 'levenshtein_dist', 'tfidf_w2v_cosine',
       'tfidf_cosine_sim_ngram']
```

In [9]:
```
X_train_1 = train_df[to_keep]
y_train = train_df['gold_label']
X_val_1 = val_df[to_keep]
y_val = val_df['gold_label']
```

In [10]:
```
y_train.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , va
y_val.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , valu
```

In [11]:
```
y_train_enc = to_categorical(y_train)
y_val_enc = to_categorical(y_val)
```

In [12]:

```
input1 = Input(shape=(29,)) #for extracted features
input2 = Input(shape=(300,)) #for sentence1 embeddings
input3 = Input(shape=(300,)) #for sentence2 embeddings
merged = concatenate([input1, input2, input3]) #merging all inputs
norm1 = BatchNormalization()(merged)
norm1 = tf.expand_dims(norm1, axis=-1)
conv1 = Conv1D(filters = 128, kernel_size = 3, strides=1, padding="valid")(n
conv2 = Conv1D(filters = 64, kernel_size = 3, strides=1, padding="valid")(co
conv3 = Conv1D(filters = 32, kernel_size = 3, strides=1, padding="valid")(co
flat = Flatten()(conv3)
dense1 = Dense(units = 1024, activation = 'relu')(flat)
drop1 = Dropout(0.3)(dense1)
dense2 = Dense(units = 512, activation = 'relu')(drop1)
drop2 = Dropout(0.3)(dense2)
dense3 = Dense(units = 256, activation = 'relu')(drop2)
drop3 = Dropout(0.3)(dense3)
norm2 = BatchNormalization()(drop3)
dense4 = Dense(units = 128, activation = 'relu')(norm2)
drop4 = Dropout(0.3)(dense4)
dense5 = Dense(units = 64, activation = 'relu')(dense4)
dense6 = Dense(units = 32, activation = 'relu')(dense5)
dense7 = Dense(units = 16, activation = 'relu')(dense6)

output = Dense(units = 3, activation = 'softmax')(dense7)
model1 = Model(inputs = [input1, input2, input3], outputs = output)

```

In [13]:
```
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
# learning rate schedule
def step_decay(epoch):

    import math
    initial_lrate = 0.001
    drop = 0.8
    epochs_drop = 8.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop)
    return lrate

lrate = LearningRateScheduler(step_decay)

es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 10)
fname = 'model1.h5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate,es]

model1.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
model1.fit([X_train_1, x_train_s1_avg_fasttext, x_train_s2_avg_fasttext], y_
           validation_data = ([X_val_1, x_val_s1_avg_fasttext, x_val_s2_avg_
```

```
Epoch 1/50
8584/8584 [==============================] - 138s 15ms/step - loss: 0.8259 -
accuracy: 0.6294 - val_loss: 0.7223 - val_accuracy: 0.6886
Epoch 2/50
8584/8584 [==============================] - 133s 15ms/step - loss: 0.7248 -
accuracy: 0.6882 - val_loss: 0.6653 - val_accuracy: 0.7200
Epoch 3/50
8584/8584 [==============================] - 134s 16ms/step - loss: 0.6826 -
accuracy: 0.7109 - val_loss: 0.6237 - val_accuracy: 0.7386
Epoch 4/50
8584/8584 [==============================] - 137s 16ms/step - loss: 0.6552 -
accuracy: 0.7242 - val_loss: 0.6206 - val_accuracy: 0.7393
Epoch 5/50
8584/8584 [==============================] - 136s 16ms/step - loss: 0.6347 -
accuracy: 0.7351 - val_loss: 0.6062 - val_accuracy: 0.7470
Epoch 6/50
8584/8584 [==============================] - 137s 16ms/step - loss: 0.6181 -
accuracy: 0.7433 - val_loss: 0.5879 - val_accuracy: 0.7530
Epoch 7/50
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Sequence modeling

We will be using following general architecture for modelling:

In [15]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "https://i.imgur.com/6q4df1V.png")
```

Out[15]:

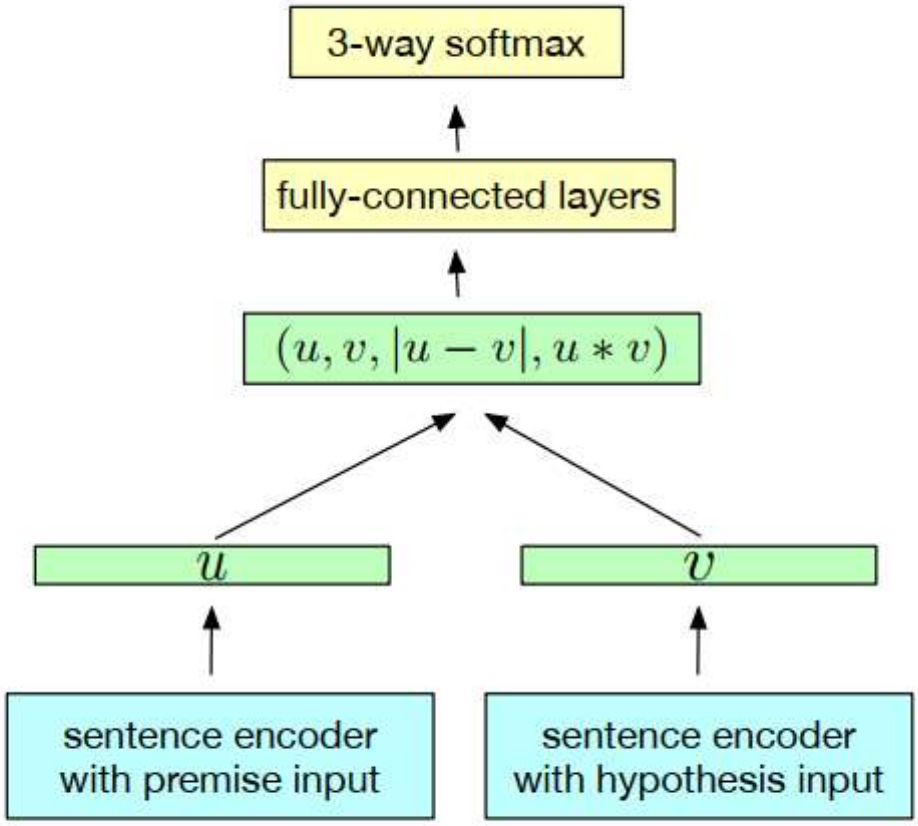


Figure 1: **Generic NLI training scheme.**

train sentence1 tokenize > token to ids > padding

In [12]:
```
num_words = None
oov_token = '<UNK>'
pad_type = 'post'
trunc_type = 'post'
# Tokenize our training data
tokenizer1 = Tokenizer(num_words=num_words, oov_token=oov_token)
tokenizer1.fit_on_texts(train_sentence1)

# Get our training data word index
word_index1 = tokenizer1.word_index

# Encode training data sentences into sequences
train_sequences_1 = tokenizer1.texts_to_sequences(train_sentence1)

# Get max training sequence length
maxlen1 = max([len(x) for x in train_sequences_1])

# Pad the training sequences
train_padded_1 = pad_sequences(train_sequences_1, padding=pad_type, truncati
```

validation sentence1 > tokenize > token to ids > pad

In [13]:
```
# Encode training data sentences into sequences
val_sequences_1 = tokenizer1.texts_to_sequences(val_sentence1)

# Pad the training sequences
val_padded_1 = pad_sequences(val_sequences_1, padding=pad_type, truncating=t
```

train sentence2 tokenize > token to ids > padding

In [14]:
```
num_words = None
oov_token = '<UNK>'
pad_type = 'post'
trunc_type = 'post'
# Tokenize our training data
tokenizer2 = Tokenizer(num_words=num_words, oov_token=oov_token)
tokenizer2.fit_on_texts(train_sentence2)

# Get our training data word index
word_index2 = tokenizer2.word_index

# Encode training data sentences into sequences
train_sequences_2 = tokenizer2.texts_to_sequences(train_sentence2)

# Get max training sequence length
maxlen2 = max([len(x) for x in train_sequences_2])

# Pad the training sequences
train_padded_2 = pad_sequences(train_sequences_2, padding=pad_type, truncati
```

validation sentence1 > tokenize > token to ids > pad

In [15]:
```
# Encode training data sentences into sequences
val_sequences_2 = tokenizer2.texts_to_sequences(val_sentence2)

# Pad the training sequences
val_padded_2 = pad_sequences(val_sequences_2, padding=pad_type, truncating=t
```

In [16]:
```
vocab_size_1 = len(word_index1) +1
vocab_size_2 = len(word_index2)+1
```

In [17]:
```
print(vocab_size_1)
print(vocab_size_2)
```

```
17997
30076
```

GloVe embeddings

Sentence1 tokens glove embeddings

In [18]:
```
#getting glove embeddings for tokens
embeddings_index_1 = dict()
f = open('glove.6B.300d.txt', encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index_1[word] = coefs
f.close()

embedding_matrix_glove_1 = np.zeros((vocab_size_1, 300))
for word, i in word_index1.items():
    embedding_vector = embeddings_index_1.get(word)
    if embedding_vector is not None:
        embedding_matrix_glove_1[i] = embedding_vector
```

Sentence2 tokens glove embeddings

In [19]:
```
embedding_matrix_glove_2 = np.zeros((vocab_size_2, 300))
for word, i in word_index2.items():
    embedding_vector = embeddings_index_1.get(word)
    if embedding_vector is not None:
        embedding_matrix_glove_2[i] = embedding_vector
```

In [20]:
```
print(embedding_matrix_glove_1.shape)
print(embedding_matrix_glove_2.shape)
```

```
(17997, 300)
(30076, 300)
```

Fasttext embeddings

In [25]:
```
#loading oretrained fasttext model
#https://fasttext.cc/docs/en/crawl-vectors.html
import fasttext.util
fasttext.util.download_model('en', if_exists='ignore')  # English
ft = fasttext.load_model('cc.en.300.bin')
```

Sentence1 tokens fasttext embeddings

In [26]:
```
embedding_matrix_fasttext_1 = np.zeros((vocab_size_1, 300))
for word, i in word_index1.items():
    embedding_vector = ft.get_word_vector(word)
    if embedding_vector is not None:
        embedding_matrix_fasttext_1[i] = embedding_vector
```

Sentence2 tokens fasttext embeddings

In [27]:
```
embedding_matrix_fasttext_2 = np.zeros((vocab_size_2, 300))
for word, i in word_index2.items():
    embedding_vector = ft.get_word_vector(word)
    if embedding_vector is not None:
        embedding_matrix_fasttext_2[i] = embedding_vector
```

Encoding target variable

In [28]:
```
y_train = train_df['gold_label']
y_val = val_df['gold_label']
```

In [ ]:
```

```

In [ ]:
```
#converting target categories to numbers: neutral:0, contradiction:1, entail
y_train.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , va
y_val.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , valu
```

In [31]:
```
#one hot encoding target
y_train_enc = to_categorical(y_train)
y_val_enc = to_categorical(y_val)
```

LSTM with glove embeddings

we will be taking max over each time step of LSTM

In [47]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_1], trainable = False)(in
lstm_s1 = LSTM(128, return_sequences = True)(emb1)
#lstm1 = LSTM(128, return_sequences = True)(lstm_s1)
rs1 = tf.keras.layers.Reshape(target_shape=(maxlen1,128,1))(lstm_s1)
avg1 = tf.keras.layers.TimeDistributed(tf.keras.layers.GlobalMaxPool1D())(rs
rs2 = tf.keras.layers.Reshape(target_shape=(maxlen1,))(avg1)
s1_dense = Dense(units = 300, activation = 'relu' )(rs2)


#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_2], trainable = False)(in
lstm_s2 = LSTM(128, return_sequences = True)(emb2)
#lstm2 = LSTM(128, return_sequences = True)(lstm_s2)
rs3 = tf.keras.layers.Reshape(target_shape=(maxlen2,128,1))(lstm_s2)
avg2 = tf.keras.layers.TimeDistributed(tf.keras.layers.GlobalAveragePooling1
rs4 = tf.keras.layers.Reshape(target_shape=(maxlen2,))(avg2)
s2_dense = Dense(units = 300, activation = 'relu' )(rs4)


#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
prod = tf.math.multiply(s1_dense, s2_dense)
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

#concatenating all merged representations
concatenated =  concatenate([merged_representations, prod, abs_diff])

#MLP
dense1 = Dense(units = 100, activation = 'relu')(concatenated)
drop1 = Dropout(0.3)(dense1)
dense2 = Dense(units = 50, activation = 'relu')(drop1)
drop2 = Dropout(0.3)(dense2)
dense3 = Dense(units = 25, activation = 'relu')(drop2)
drop3 = Dropout(0.3)(dense3)
output = Dense(units = 3, activation = 'softmax')(drop3)
model1 = Model(inputs = [input1, input2], outputs = output)


```

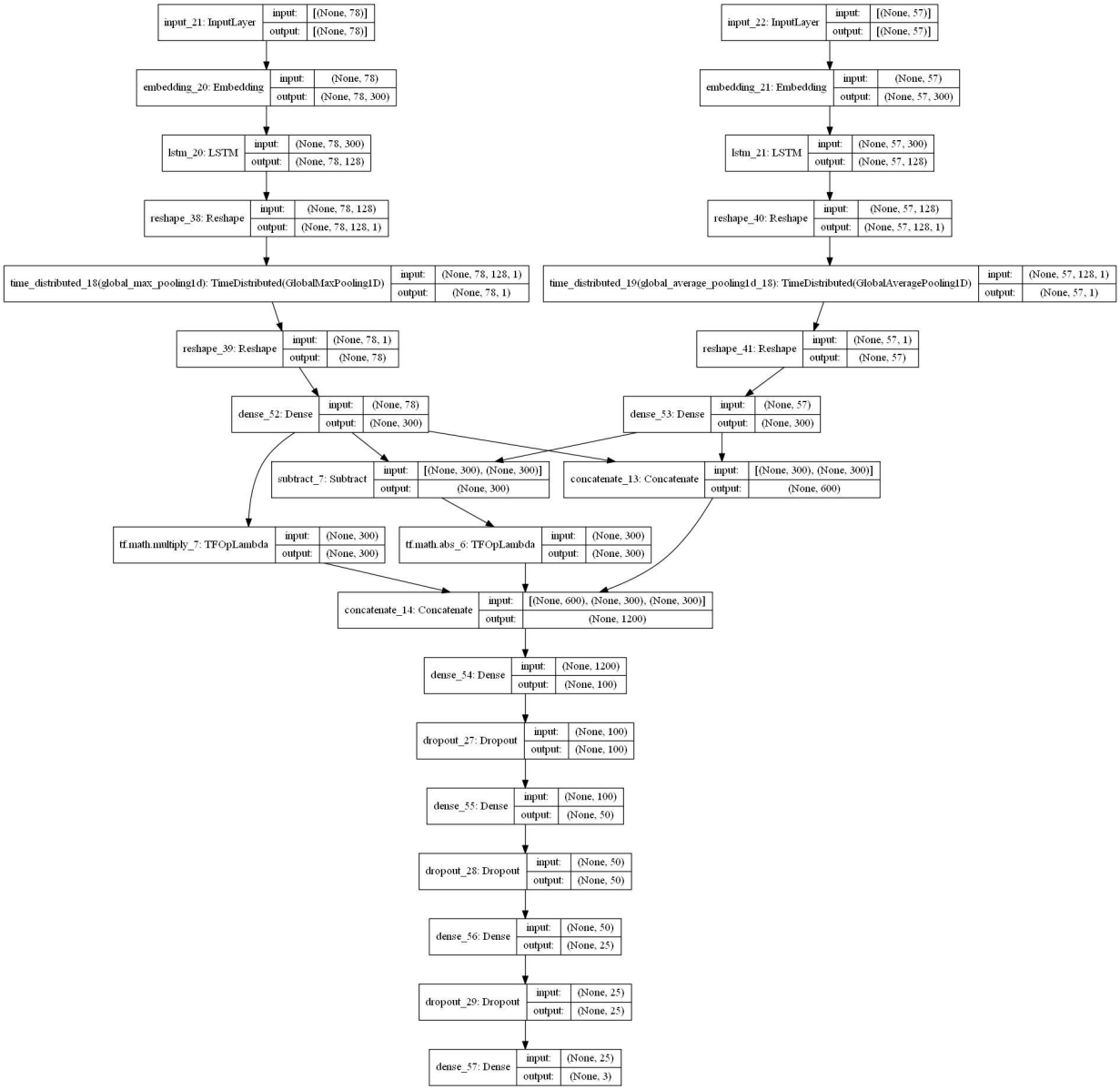
In [48]:
```
1 model1.summary()
```

```
Model: "model_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_21 (InputLayer)        [(None, 78)]              0
_________________________________________________________________
input_22 (InputLayer)        [(None, 57)]              0
_________________________________________________________________
embedding_20 (Embedding)     (None, 78, 300)           5399100   input_21[0][0]
_________________________________________________________________
embedding_21 (Embedding)     (None, 57, 300)           9022800   input_22[0][0]
_________________________________________________________________
lstm_20 (LSTM)               (None, 78, 128)           219648    embedding_20
[0][0]
_________________________________________________________________
lstm_21 (LSTM)               (None, 57, 128)           219648    embedding_21
[0][0]
_________________________________________________________________
reshape_38 (Reshape)         (None, 78, 128, 1)        0         lstm_20[0][0]
_________________________________________________________________
reshape_40 (Reshape)         (None, 57, 128, 1)        0         lstm_21[0][0]
_________________________________________________________________
time_distributed_18 (TimeDistri (None, 78, 1)          0         reshape_38[0]
[0]
_________________________________________________________________
time_distributed_19 (TimeDistri (None, 57, 1)          0         reshape_40[0]
[0]
_________________________________________________________________
reshape_39 (Reshape)         (None, 78)                0         time_distribut
ed_18[0][0]
_________________________________________________________________
reshape_41 (Reshape)         (None, 57)                0         time_distribut
ed_19[0][0]
_________________________________________________________________
dense_52 (Dense)             (None, 300)               23700     reshape_39[0]
[0]
_________________________________________________________________
dense_53 (Dense)             (None, 300)               17400     reshape_41[0]
[0]
_________________________________________________________________
```

```
__________________
subtract_7 (Subtract)           (None, 300)          0           dense_52[0][0]
                                                                 dense_53[0][0]

__________________
concatenate_13 (Concatenate)    (None, 600)          0           dense_52[0][0]
                                                                 dense_53[0][0]

__________________
tf.math.multiply_7 (TFOpLambda) (None, 300)          0           dense_52[0][0]
                                                                 dense_53[0][0]

__________________
tf.math.abs_6 (TFOpLambda)      (None, 300)          0           subtract_7[0]
[0]

__________________
concatenate_14 (Concatenate)    (None, 1200)         0           concatenate_13
[0][0]
                                                                 tf.math.multip
ly_7[0][0]
                                                                 tf.math.abs_6
[0][0]

__________________
dense_54 (Dense)                (None, 100)          120100      concatenate_14
[0][0]

__________________
dropout_27 (Dropout)            (None, 100)          0           dense_54[0][0]

__________________
dense_55 (Dense)                (None, 50)           5050        dropout_27[0]
[0]

__________________
dropout_28 (Dropout)            (None, 50)           0           dense_55[0][0]

__________________
dense_56 (Dense)                (None, 25)           1275        dropout_28[0]
[0]

__________________
dropout_29 (Dropout)            (None, 25)           0           dense_56[0][0]

__________________
dense_57 (Dense)                (None, 3)            78          dropout_29[0]
[0]
================================================================================
==================
Total params: 15,028,799
Trainable params: 606,899
Non-trainable params: 14,421,900

__________________
```

In [49]:
```
1 plot_model(model1, to_file='model1.png', show_shapes=True)
```

Out[49]:



In [50]:

```
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
# learning rate schedule
def step_decay(epoch):

    import math
    initial_lrate = 0.001
    drop = 0.9
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop)
    return lrate

lrate = LearningRateScheduler(step_decay)

es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 5)
fname = 'best_model1.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate, es, checkpoint]
```

In [51]:
```
model1.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
model1.fit([train_padded_1, train_padded_2], y_train_enc, epochs = 50, batch
          validation_data = ([val_padded_1, val_padded_2], y_val_enc), call
```

```
Epoch 1/50
537/537 [==============================] - 39s 65ms/step - loss: 0.9134 - accur
acy: 0.5709 - val_loss: 0.8058 - val_accuracy: 0.6463
Epoch 2/50
537/537 [==============================] - 35s 65ms/step - loss: 0.8029 - accur
acy: 0.6473 - val_loss: 0.7561 - val_accuracy: 0.6712
Epoch 3/50
537/537 [==============================] - 35s 66ms/step - loss: 0.7634 - accur
acy: 0.6719 - val_loss: 0.7162 - val_accuracy: 0.6897
Epoch 4/50
537/537 [==============================] - 36s 66ms/step - loss: 0.7351 - accur
acy: 0.6884 - val_loss: 0.7070 - val_accuracy: 0.6974
Epoch 5/50
537/537 [==============================] - 36s 67ms/step - loss: 0.7105 - accur
acy: 0.7022 - val_loss: 0.6901 - val_accuracy: 0.7050
Epoch 6/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6918 - accur
acy: 0.7124 - val_loss: 0.6741 - val_accuracy: 0.7141
Epoch 7/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6735 - accur
acy: 0.7219 - val_loss: 0.6777 - val_accuracy: 0.7163
Epoch 8/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6583 - accur
acy: 0.7300 - val_loss: 0.6552 - val_accuracy: 0.7258
Epoch 9/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6453 - accur
acy: 0.7365 - val_loss: 0.6544 - val_accuracy: 0.7295
Epoch 10/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6304 - accur
acy: 0.7435 - val_loss: 0.6642 - val_accuracy: 0.7297
Epoch 11/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6202 - accur
acy: 0.7486 - val_loss: 0.6491 - val_accuracy: 0.7321
Epoch 12/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6105 - accur
acy: 0.7535 - val_loss: 0.6457 - val_accuracy: 0.7327
Epoch 13/50
537/537 [==============================] - 36s 67ms/step - loss: 0.6012 - accur
acy: 0.7573 - val_loss: 0.6528 - val_accuracy: 0.7295
Epoch 14/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5917 - accur
acy: 0.7619 - val_loss: 0.6603 - val_accuracy: 0.7330
Epoch 15/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5830 - accur
acy: 0.7667 - val_loss: 0.6537 - val_accuracy: 0.7326
Epoch 16/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5754 - accur
acy: 0.7699 - val_loss: 0.6627 - val_accuracy: 0.7337
Epoch 17/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5672 - accur
acy: 0.7737 - val_loss: 0.6585 - val_accuracy: 0.7339
Epoch 18/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5610 - accur
```

```
acy: 0.7770 - val_loss: 0.6708 - val_accuracy: 0.7333
Epoch 19/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5519 - accur
acy: 0.7809 - val_loss: 0.6789 - val_accuracy: 0.7317
Epoch 20/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5401 - accur
acy: 0.7870 - val_loss: 0.6895 - val_accuracy: 0.7291
Epoch 21/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5349 - accur
acy: 0.7891 - val_loss: 0.6818 - val_accuracy: 0.7301
Epoch 22/50
537/537 [==============================] - 36s 67ms/step - loss: 0.5267 - accur
acy: 0.7932 - val_loss: 0.6844 - val_accuracy: 0.7318
```

Out[51]: <keras.callbacks.History at 0x1def70a6cf8>

LSTM model with fasttext embeddings

we will be taking max over each time step of LSTM

In [73]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_fasttext_1], trainable = False)
lstm_s1 = LSTM(128, return_sequences = True)(emb1)
print(lstm_s1.shape)
#lstm1 = LSTM(128, return_sequences = True)(lstm_s1)
rs1 = tf.keras.layers.Reshape(target_shape=(maxlen1,128,1))(lstm_s1)
avg1 = tf.keras.layers.TimeDistributed(tf.keras.layers.GlobalMaxPool1D())(rs
rs2 = tf.keras.layers.Reshape(target_shape=(maxlen1,))(avg1)
s1_dense = Dense(units = 100, activation = 'relu' )(rs2)


#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_fasttext_2], trainable = False)
lstm_s2 = LSTM(128, return_sequences = True)(emb2)
#lstm2 = LSTM(128, return_sequences = True)(lstm_s2)
rs3 = tf.keras.layers.Reshape(target_shape=(maxlen2,128,1))(lstm_s2)
avg2 = tf.keras.layers.TimeDistributed(tf.keras.layers.GlobalAveragePooling1
rs4 = tf.keras.layers.Reshape(target_shape=(maxlen2,))(avg2)
s2_dense = Dense(units = 100, activation = 'relu' )(rs4)


#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
prod = tf.math.multiply(s1_dense, s2_dense)
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

#concatenating all merged representations
concatenated =  concatenate([merged_representations, prod, abs_diff])

#MLP
dense1 = Dense(units = 100, activation = 'relu')(concatenated)
drop1 = Dropout(0.3)(dense1)
dense2 = Dense(units = 25, activation = 'relu')(drop1)
output = Dense(units = 3, activation = 'softmax')(dense2)
model2 = Model(inputs = [input1, input2], outputs = output)


```

```
(None, 78, 128)
```

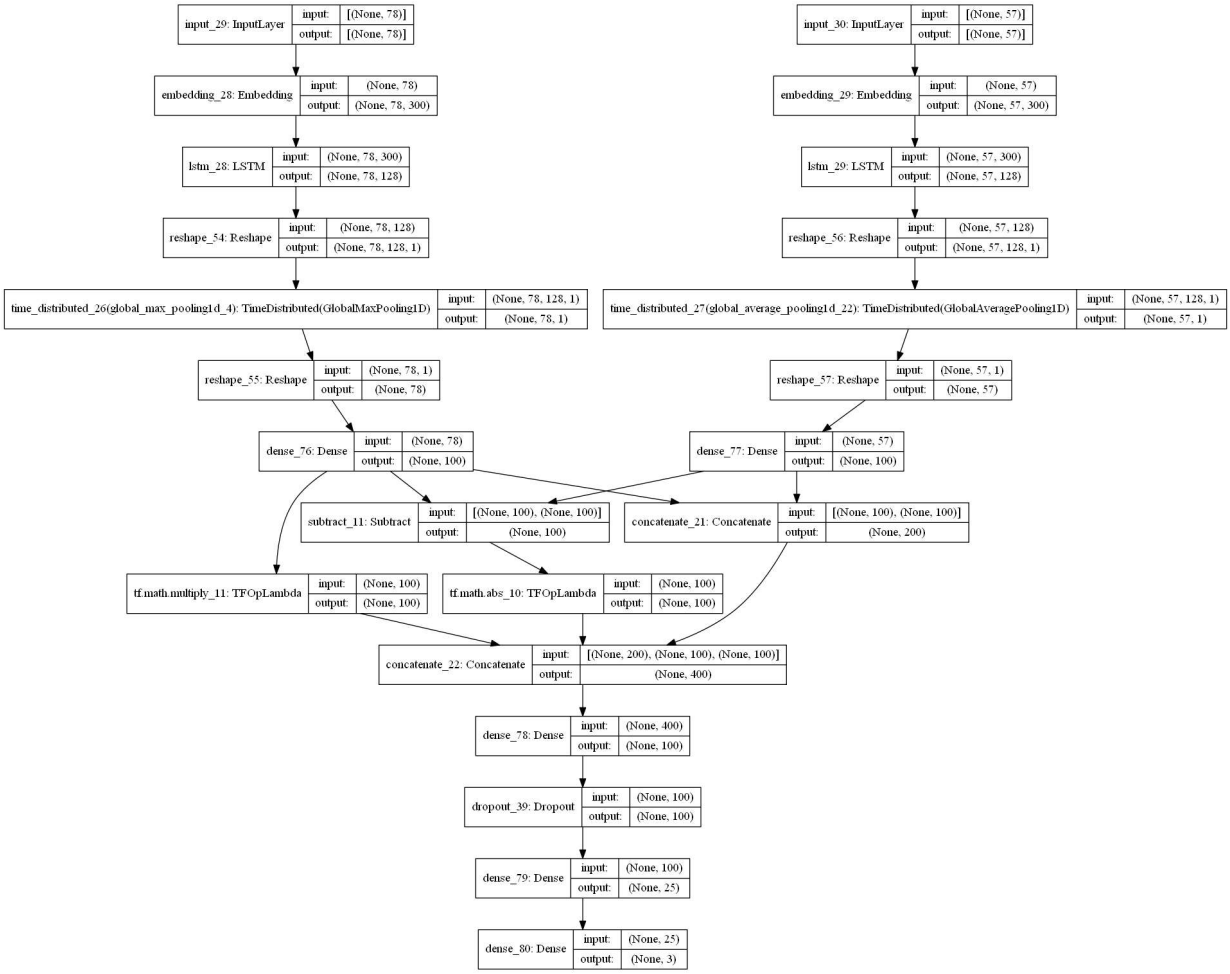
In [68]:
```
1 model2.summary()
```

```
Model: "model_13"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_29 (InputLayer)           [(None, 78)]         0
__________________________________________________________________________________________________
input_30 (InputLayer)           [(None, 57)]         0
__________________________________________________________________________________________________
embedding_28 (Embedding)        (None, 78, 300)      5399100     input_29[0][0]
__________________________________________________________________________________________________
embedding_29 (Embedding)        (None, 57, 300)      9022800     input_30[0][0]
__________________________________________________________________________________________________
lstm_28 (LSTM)                  (None, 78, 128)      219648      embedding_28
[0][0]
__________________________________________________________________________________________________
lstm_29 (LSTM)                  (None, 57, 128)      219648      embedding_29
[0][0]
__________________________________________________________________________________________________
reshape_54 (Reshape)            (None, 78, 128, 1)   0           lstm_28[0][0]
__________________________________________________________________________________________________
reshape_56 (Reshape)            (None, 57, 128, 1)   0           lstm_29[0][0]
__________________________________________________________________________________________________
time_distributed_26 (TimeDistri (None, 78, 1)        0           reshape_54[0]
[0]
__________________________________________________________________________________________________
time_distributed_27 (TimeDistri (None, 57, 1)        0           reshape_56[0]
[0]
__________________________________________________________________________________________________
reshape_55 (Reshape)            (None, 78)           0           time_distribut
ed_26[0][0]
__________________________________________________________________________________________________
reshape_57 (Reshape)            (None, 57)           0           time_distribut
ed_27[0][0]
__________________________________________________________________________________________________
dense_76 (Dense)                (None, 100)          7900        reshape_55[0]
[0]
__________________________________________________________________________________________________
dense_77 (Dense)                (None, 100)          5800        reshape_57[0]
[0]
__________________________________________________________________________________________________
```

```
__________________
subtract_11 (Subtract)          (None, 100)          0           dense_76[0][0]
                                                                 dense_77[0][0]

__________________
concatenate_21 (Concatenate)    (None, 200)          0           dense_76[0][0]
                                                                 dense_77[0][0]

__________________
tf.math.multiply_11 (TFOpLambda (None, 100)          0           dense_76[0][0]
                                                                 dense_77[0][0]

__________________
tf.math.abs_10 (TFOpLambda)     (None, 100)          0           subtract_11[0]
[0]

__________________
concatenate_22 (Concatenate)    (None, 400)          0           concatenate_21
[0][0]
                                                                 tf.math.multip
ly_11[0][0]
                                                                 tf.math.abs_10
[0][0]

__________________
dense_78 (Dense)                (None, 100)          40100       concatenate_22
[0][0]

__________________
dropout_39 (Dropout)            (None, 100)          0           dense_78[0][0]

__________________
dense_79 (Dense)                (None, 25)           2525        dropout_39[0]
[0]

__________________
dense_80 (Dense)                (None, 3)            78          dense_79[0][0]
==================================================================================
==================
Total params: 14,917,599
Trainable params: 495,699
Non-trainable params: 14,421,900

__________________
```

In [69]:
```
plot_model(model2, to_file='model2.png', show_shapes=True)
```

Out[69]:



In [70]:
```
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
# learning rate schedule
def step_decay(epoch):

    import math
    initial_lrate = 0.001
    drop = 0.9
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop)
    return lrate

lrate = LearningRateScheduler(step_decay)

es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 5)
fname = 'best_model2.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate, checkpoint]
```

In [71]:
```
model2.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
model2.fit([train_padded_1, train_padded_2], y_train_enc, epochs = 50, batch
          validation_data = ([val_padded_1, val_padded_2], y_val_enc), call
```

```
Epoch 1/50
537/537 [==============================] - 39s 65ms/step - loss: 0.9331 - acc
uracy: 0.5515 - val_loss: 0.8487 - val_accuracy: 0.6084
Epoch 2/50
537/537 [==============================] - 35s 65ms/step - loss: 0.8365 - acc
uracy: 0.6173 - val_loss: 0.8064 - val_accuracy: 0.6341
Epoch 3/50
537/537 [==============================] - 35s 66ms/step - loss: 0.8049 - acc
uracy: 0.6353 - val_loss: 0.7946 - val_accuracy: 0.6428
Epoch 4/50
537/537 [==============================] - 35s 66ms/step - loss: 0.7847 - acc
uracy: 0.6486 - val_loss: 0.7751 - val_accuracy: 0.6569
Epoch 5/50
537/537 [==============================] - 35s 66ms/step - loss: 0.7614 - acc
uracy: 0.6643 - val_loss: 0.7468 - val_accuracy: 0.6719
Epoch 6/50
537/537 [==============================] - 36s 66ms/step - loss: 0.7399 - acc
uracy: 0.6770 - val_loss: 0.7252 - val_accuracy: 0.6874
Epoch 7/50
```

Bidirectional LSTM with fasttext embedding

In [27]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_fasttext_1], trainable = False)
lstm_s1 = Bidirectional(LSTM(200, return_sequences = True) , merge_mode = 'c
lstm1 = Bidirectional(LSTM(100, return_sequences = True))(lstm_s1)
rs1 = tf.keras.layers.Reshape(target_shape=(maxlen1,200,1))(lstm1)
avg1 = tf.keras.layers.TimeDistributed(tf.keras.layers.GlobalMaxPool1D())(rs
rs2 = tf.keras.layers.Reshape(target_shape=(maxlen1,))(avg1)
s1_dense = Dense(units = 100, activation = 'relu' )(rs2)


#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_fasttext_2], trainable = False)
lstm_s2 = Bidirectional(LSTM(200, return_sequences = True))(emb2)
lstm2 = Bidirectional(LSTM(100, return_sequences = True))(lstm_s2)
rs3 = tf.keras.layers.Reshape(target_shape=(maxlen2,200,1))(lstm2)
avg2 = tf.keras.layers.TimeDistributed(tf.keras.layers.GlobalAveragePooling1
rs4 = tf.keras.layers.Reshape(target_shape=(maxlen2,))(avg2)
s2_dense = Dense(units = 100, activation = 'relu' )(rs4)


#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
prod = tf.math.multiply(s1_dense, s2_dense)
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

#concatenating all merged representations
concatenated =  concatenate([merged_representations, prod, abs_diff])

#MLP
dense1 = Dense(units = 100, activation = 'relu')(concatenated)
drop1 = Dropout(0.3)(dense1)
dense2 = Dense(units = 50, activation = 'relu')(drop1)
drop2 = Dropout(0.2)(dense2)
dense3 = Dense(units = 10, activation = 'relu')(drop2)
output = Dense(units = 3, activation = 'softmax')(dense3)
model3 = Model(inputs = [input1, input2], outputs = output)


```

In [28]:
```
1 model3.summary()
```

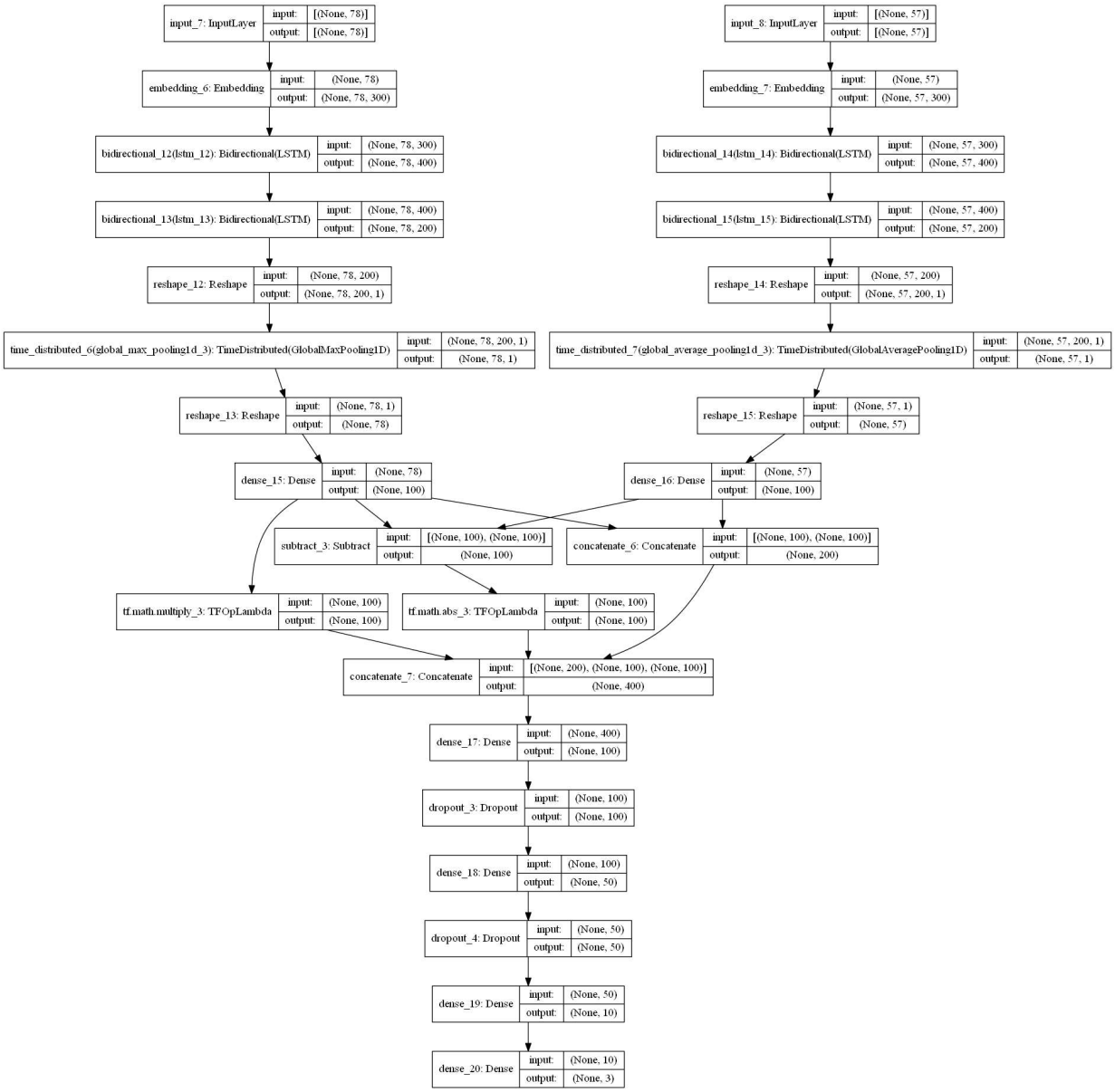
```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_7 (InputLayer)         [(None, 78)]              0         
_________________________________________________________________
input_8 (InputLayer)         [(None, 57)]              0         
_________________________________________________________________
embedding_6 (Embedding)      (None, 78, 300)           5399100   
_________________________________________________________________
embedding_7 (Embedding)      (None, 57, 300)           9022800   
_________________________________________________________________
bidirectional_12 (Bidirectional (None, 78, 400)        801600    
_________________________________________________________________
bidirectional_14 (Bidirectional (None, 57, 400)        801600    
_________________________________________________________________
bidirectional_13 (Bidirectional (None, 78, 200)        400800    
_________________________________________________________________
bidirectional_15 (Bidirectional (None, 57, 200)        400800    
_________________________________________________________________
reshape_12 (Reshape)         (None, 78, 200, 1)        0         
_________________________________________________________________
reshape_14 (Reshape)         (None, 57, 200, 1)        0         
_________________________________________________________________
time_distributed_6 (TimeDistrib (None, 78, 1)          0         
_________________________________________________________________
time_distributed_7 (TimeDistrib (None, 57, 1)          0         
_________________________________________________________________
reshape_13 (Reshape)         (None, 78)                0         
_________________________________________________________________
reshape_15 (Reshape)         (None, 57)                0         
```

```
ed_7[0][0]
__________________________________________________________________________________________________
dense_15 (Dense)                (None, 100)          7900        reshape_13[0]
[0]
__________________________________________________________________________________________________
dense_16 (Dense)                (None, 100)          5800        reshape_15[0]
[0]
__________________________________________________________________________________________________
subtract_3 (Subtract)           (None, 100)          0           dense_15[0][0]
                                                                 dense_16[0][0]
__________________________________________________________________________________________________
concatenate_6 (Concatenate)     (None, 200)          0           dense_15[0][0]
                                                                 dense_16[0][0]
__________________________________________________________________________________________________
tf.math.multiply_3 (TFOpLambda) (None, 100)          0           dense_15[0][0]
                                                                 dense_16[0][0]
__________________________________________________________________________________________________
tf.math.abs_3 (TFOpLambda)      (None, 100)          0           subtract_3[0]
[0]
__________________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 400)          0           concatenate_6
[0][0]
                                                                 tf.math.multip
ly_3[0][0]
                                                                 tf.math.abs_3
[0][0]
__________________________________________________________________________________________________
dense_17 (Dense)                (None, 100)          40100       concatenate_7
[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 100)          0           dense_17[0][0]
__________________________________________________________________________________________________
dense_18 (Dense)                (None, 50)           5050        dropout_3[0]
[0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 50)           0           dense_18[0][0]
__________________________________________________________________________________________________
dense_19 (Dense)                (None, 10)           510         dropout_4[0]
[0]
__________________________________________________________________________________________________
dense_20 (Dense)                (None, 3)            33          dense_19[0][0]
==================================================================================
==================
Total params: 16,886,093
```

```
Trainable params: 2,464,193
Non-trainable params: 14,421,900
```

In [29]:
```
1 plot_model(model3, to_file='model3.png', show_shapes=True)
```

Out[29]:



In [30]:

```
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
# learning rate schedule
def step_decay(epoch):

    import math
    initial_lrate = 0.001
    drop = 0.9
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop)
    return lrate

lrate = LearningRateScheduler(step_decay)

es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 6)
fname = 'best_model3.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [es, lrate, checkpoint]
```

In [31]:
```
model3.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
model3.fit([train_padded_1, train_padded_2], y_train_enc, epochs = 50, batch
           validation_data = ([val_padded_1, val_padded_2], y_val_enc), call
```

```
Epoch 1/50
4292/4292 [==============================] - 327s 73ms/step - loss: 0.8976 - ac
curacy: 0.5820 - val_loss: 0.8098 - val_accuracy: 0.6443
Epoch 2/50
4292/4292 [==============================] - 321s 75ms/step - loss: 0.8047 - ac
curacy: 0.6427 - val_loss: 0.7841 - val_accuracy: 0.6557
Epoch 3/50
4292/4292 [==============================] - 323s 75ms/step - loss: 0.7770 - ac
curacy: 0.6584 - val_loss: 0.7618 - val_accuracy: 0.6654
Epoch 4/50
4292/4292 [==============================] - 320s 74ms/step - loss: 0.7573 - ac
curacy: 0.6696 - val_loss: 0.7518 - val_accuracy: 0.6745
Epoch 5/50
4292/4292 [==============================] - 323s 75ms/step - loss: 0.7414 - ac
curacy: 0.6775 - val_loss: 0.7477 - val_accuracy: 0.6746
Epoch 6/50
4292/4292 [==============================] - 324s 75ms/step - loss: 0.7270 - ac
curacy: 0.6858 - val_loss: 0.7446 - val_accuracy: 0.6781
Epoch 7/50
4292/4292 [==============================] - 324s 75ms/step - loss: 0.7127 - ac
curacy: 0.6929 - val_loss: 0.7415 - val_accuracy: 0.6776
Epoch 8/50
4292/4292 [==============================] - 324s 75ms/step - loss: 0.6988 - ac
curacy: 0.6999 - val_loss: 0.7452 - val_accuracy: 0.6812
Epoch 9/50
4292/4292 [==============================] - 324s 76ms/step - loss: 0.6834 - ac
curacy: 0.7079 - val_loss: 0.7467 - val_accuracy: 0.6822
Epoch 10/50
4292/4292 [==============================] - 324s 76ms/step - loss: 0.6649 - ac
curacy: 0.7170 - val_loss: 0.7536 - val_accuracy: 0.6783
Epoch 11/50
4292/4292 [==============================] - 340s 79ms/step - loss: 0.6477 - ac
curacy: 0.7254 - val_loss: 0.7567 - val_accuracy: 0.6794
Epoch 12/50
4292/4292 [==============================] - 326s 76ms/step - loss: 0.6300 - ac
curacy: 0.7341 - val_loss: 0.7668 - val_accuracy: 0.6749
Epoch 13/50
4292/4292 [==============================] - 326s 76ms/step - loss: 0.6121 - ac
curacy: 0.7429 - val_loss: 0.7776 - val_accuracy: 0.6730
Epoch 14/50
4292/4292 [==============================] - 326s 76ms/step - loss: 0.5939 - ac
curacy: 0.7515 - val_loss: 0.8028 - val_accuracy: 0.6672
Epoch 15/50
4292/4292 [==============================] - 325s 76ms/step - loss: 0.5749 - ac
curacy: 0.7604 - val_loss: 0.8195 - val_accuracy: 0.6703
```

Out[31]: <keras.callbacks.History at 0x214a0428978>

Bi-LSTM with attention with glove embeddings

In [32]:
```
import logging
logging.getLogger('tensorflow').setLevel(logging.ERROR)  # suppress warnings
```

Following self attention has been applied:

$$\overline{h_i} = v_i \cdot \tanh(W \cdot h_i + b) + v_b$$
$$\alpha_i = softmax(\overline{h_i})$$
$$u = \sum \alpha_i h_i$$

where, h_i represent lstm hidden states, W and v represent affine transformations to calculate weights for each hidden states

In [24]:
```
class Attention_Custom(tf.keras.layers.Layer):

    def __init__(self,att_units, inp_shape):
        super().__init__()
        self.att_units = att_units
        self.inp_shape = inp_shape
        self.num_dim = self.inp_shape[-1] ##dimensions per time step
        self.ts = self.inp_shape[-2]  ##number of timesteps
        self.w = self.add_weight(shape = (self.num_dim,self.att_units), init
        self.b = self.add_weight(shape = (self.ts,self.att_units), initializ
        self.v = self.add_weight(shape = (self.att_units, self.num_dim), ini
        self.v_b = self.add_weight(shape = (self.ts, self.num_dim))

    def call(self,x):
        score = K.dot((K.tanh(K.dot(x,self.w)+self.b)),self.v)+self.v_b
        weights = K.softmax(score, axis=1)
        weighted_states = x*weights
        output = K.sum(weighted_states, axis=1)

        return weights, output

```

In [40]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_1], trainable = False)(in
#lstm_s1 = Bidirectional(LSTM(256, return_sequences = True) , merge_mode = '
lstm1 = Bidirectional(LSTM(128, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb1)
att_1 = Attention_Custom(64, lstm1.shape)
att_weights_1, context_1 = att_1.call(lstm1)
s1_dense = Dense(units = 300, activation = 'relu' )(context_1)


#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_2], trainable = False)(in
#lstm_s2 = Bidirectional(LSTM(20, return_sequences = True))(emb2)
lstm2 = Bidirectional(LSTM(128, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb2)
att_2 = Attention_Custom(64, lstm2.shape)
att_weights_2, context_2 = att_2.call(lstm2)
s2_dense = Dense(units = 300, activation = 'relu' )(context_2)

#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
#point-wise product of sentence representations
prod = tf.math.multiply(s1_dense, s2_dense)
#point-wise absolute difference of sentence representations
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

#concatenating
concatenated =  concatenate([merged_representations, prod, abs_diff])
# concatenated = tf.expand_dims(concatenated, axis=-1)
# conv1 = Conv1D(filters = 64, kernel_size = 3, strides=1, padding='valid')(
# conv2 = Conv1D(filters = 32, kernel_size = 3, strides=1, padding='valid')(
# conv3 = Conv1D(filters = 16, kernel_size = 3, strides=1, padding='valid')(

# flat = Flatten()(conv3)
#MLP and threeway classifier
dense1 = Dense(units = 512, activation = 'relu')(concatenated)
#drop1 = Dropout(0.3)(dense1)
# dense2 = Dense(units = 50, activation = 'relu')(drop1)
output = Dense(units = 3, activation = 'softmax')(dense1)
model7 = Model(inputs = [input1, input2], outputs = output)


```

In [41]:
```
1 model7.summary()
```

```
Model: "model_6"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_13 (InputLayer)           [(None, 78)]         0
__________________________________________________________________________________________________
input_14 (InputLayer)           [(None, 57)]         0
__________________________________________________________________________________________________
embedding_12 (Embedding)        (None, 78, 300)      5399100     input_13[0]
[0]
__________________________________________________________________________________________________
embedding_13 (Embedding)        (None, 57, 300)      9022800     input_14[0]
[0]
__________________________________________________________________________________________________
```

In [42]:

```
import tensorflow_addons as tfa
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
# learning rate schedule
def step_decay(epoch):

    import math
    initial_lrate = 0.1
    drop = 0.9
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop)
    return lrate

#lrate = LearningRateScheduler(step_decay)
lrate = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=10, m
es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 10)
fname = 'best_model7.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate,es]
#optimizer=tfa.optimizers.SGDW(weight_decay=0.99,momentum=0.9,learning_rate=
#optimizer=tfa.optimizers.AdamW(weight_decay=0.99,learning_rate=0.001)
optimizer=tf.optimizers.Adam(learning_rate=0.001)


#sgd = SGD(learning_rate = 0.1, )
model7.compile(optimizer = optimizer, loss = 'categorical_crossentropy', met
model7.fit([train_padded_1, train_padded_2], y_train_enc, epochs = 100, batc
          validation_data = ([val_padded_1, val_padded_2], y_val_enc), call
```

```
Epoch 1/100
8584/8584 [==============================] - 337s 39ms/step - loss: 0.7877 - ac
curacy: 0.6600 - val_loss: 0.6641 - val_accuracy: 0.7384
Epoch 2/100
8584/8584 [==============================] - 339s 39ms/step - loss: 0.6718 - ac
curacy: 0.7328 - val_loss: 0.6226 - val_accuracy: 0.7617
Epoch 3/100
8584/8584 [==============================] - 334s 39ms/step - loss: 0.6368 - ac
curacy: 0.7543 - val_loss: 0.5997 - val_accuracy: 0.7712
Epoch 4/100
8584/8584 [==============================] - 341s 40ms/step - loss: 0.6154 - ac
curacy: 0.7662 - val_loss: 0.5864 - val_accuracy: 0.7770
Epoch 5/100
8584/8584 [==============================] - 341s 40ms/step - loss: 0.5994 - ac
curacy: 0.7760 - val_loss: 0.5791 - val_accuracy: 0.7908
Epoch 6/100
8584/8584 [==============================] - 356s 41ms/step - loss: 0.5876 - ac
curacy: 0.7830 - val_loss: 0.5771 - val_accuracy: 0.7918
Epoch 7/100
8584/8584 [==============================] - 341s 40ms/step - loss: 0.5776 - ac
curacy: 0.7884 - val_loss: 0.5724 - val_accuracy: 0.7929
Epoch 8/100
8584/8584 [==============================] - 334s 39ms/step - loss: 0.5691 - ac
curacy: 0.7937 - val_loss: 0.5616 - val_accuracy: 0.7991
Epoch 9/100
8584/8584 [==============================] - 333s 39ms/step - loss: 0.5621 - ac
curacy: 0.7977 - val_loss: 0.5693 - val_accuracy: 0.7982
Epoch 10/100
```

```
8584/8584 [==============================] - 334s 39ms/step - loss: 0.5554 - ac
curacy: 0.8014 - val_loss: 0.5792 - val_accuracy: 0.7997
Epoch 11/100
8584/8584 [==============================] - 332s 39ms/step - loss: 0.5498 - ac
curacy: 0.8049 - val_loss: 0.5648 - val_accuracy: 0.8038
Epoch 12/100
8584/8584 [==============================] - 340s 40ms/step - loss: 0.5440 - ac
curacy: 0.8079 - val_loss: 0.5774 - val_accuracy: 0.7994
Epoch 13/100
8584/8584 [==============================] - 339s 39ms/step - loss: 0.5396 - ac
curacy: 0.8104 - val_loss: 0.5761 - val_accuracy: 0.8025
Epoch 14/100
8584/8584 [==============================] - 334s 39ms/step - loss: 0.5349 - ac
curacy: 0.8127 - val_loss: 0.5764 - val_accuracy: 0.8041
Epoch 15/100
8584/8584 [==============================] - 331s 39ms/step - loss: 0.5312 - ac
curacy: 0.8146 - val_loss: 0.5680 - val_accuracy: 0.8068
Epoch 16/100
8584/8584 [==============================] - 332s 39ms/step - loss: 0.5263 - ac
curacy: 0.8172 - val_loss: 0.5717 - val_accuracy: 0.8104
Epoch 17/100
8584/8584 [==============================] - 335s 39ms/step - loss: 0.5231 - ac
curacy: 0.8194 - val_loss: 0.5851 - val_accuracy: 0.8022
Epoch 18/100
8584/8584 [==============================] - 336s 39ms/step - loss: 0.5194 - ac
curacy: 0.8213 - val_loss: 0.5743 - val_accuracy: 0.8074
Epoch 19/100
8584/8584 [==============================] - 328s 38ms/step - loss: 0.5156 - ac
curacy: 0.8231 - val_loss: 0.5862 - val_accuracy: 0.8032
Epoch 20/100
8584/8584 [==============================] - 332s 39ms/step - loss: 0.5117 - ac
curacy: 0.8248 - val_loss: 0.5768 - val_accuracy: 0.8093
Epoch 21/100
8584/8584 [==============================] - 323s 38ms/step - loss: 0.5088 - ac
curacy: 0.8266 - val_loss: 0.5843 - val_accuracy: 0.8068
Epoch 22/100
8584/8584 [==============================] - 322s 37ms/step - loss: 0.5067 - ac
curacy: 0.8275 - val_loss: 0.5950 - val_accuracy: 0.8028
Epoch 23/100
8584/8584 [==============================] - 321s 37ms/step - loss: 0.5036 - ac
curacy: 0.8299 - val_loss: 0.5988 - val_accuracy: 0.8024
Epoch 24/100
8584/8584 [==============================] - 321s 37ms/step - loss: 0.5011 - ac
curacy: 0.8307 - val_loss: 0.5991 - val_accuracy: 0.8079
Epoch 25/100
8584/8584 [==============================] - 322s 37ms/step - loss: 0.4988 - ac
curacy: 0.8317 - val_loss: 0.5854 - val_accuracy: 0.8067
Epoch 26/100
8584/8584 [==============================] - 321s 37ms/step - loss: 0.4957 - ac
curacy: 0.8338 - val_loss: 0.6001 - val_accuracy: 0.8017
```

Out[42]: <keras.callbacks.History at 0x1f54c56d438>

Bi-LSTM with Attention on fasttext embeddings

In [34]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_fasttext_1], trainable = False)
#lstm_s1 = Bidirectional(LSTM(256, return_sequences = True) , merge_mode = '
lstm1 = Bidirectional(LSTM(128, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb1)
att_1 = Attention_Custom(64, lstm1.shape)
att_weights_1, context_1 = att_1.call(lstm1)
s1_dense = Dense(units = 300, activation = 'relu' )(context_1)


#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_fasttext_2], trainable = False)
#lstm_s2 = Bidirectional(LSTM(20, return_sequences = True))(emb2)
lstm2 = Bidirectional(LSTM(128, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb2)
att_2 = Attention_Custom(64, lstm2.shape)
att_weights_2, context_2 = att_2.call(lstm2)
s2_dense = Dense(units = 300, activation = 'relu' )(context_2)

#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
#point-wise product of sentence representations
prod = tf.math.multiply(s1_dense, s2_dense)
#point-wise absolute difference of sentence representations
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

#concatenating
concatenated =  concatenate([merged_representations, prod, abs_diff])
# concatenated = tf.expand_dims(concatenated, axis=-1)
# conv1 = Conv1D(filters = 64, kernel_size = 3, strides=1, padding='valid')(
# conv2 = Conv1D(filters = 32, kernel_size = 3, strides=1, padding='valid')(
# conv3 = Conv1D(filters = 16, kernel_size = 3, strides=1, padding='valid')(

# flat = Flatten()(conv3)
#MLP and threeway classifier
dense1 = Dense(units = 512, activation = 'relu')(concatenated)
#drop1 = Dropout(0.3)(dense1)
# dense2 = Dense(units = 50, activation = 'relu')(drop1)
output = Dense(units = 3, activation = 'softmax')(dense1)
model8 = Model(inputs = [input1, input2], outputs = output)


```

In [35]:
```
1 model8.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)            [(None, 78)]         0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 57)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 78, 300)      5399100     input_4[0]
[0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 57, 300)      9022800     input_5[0]
[0]
__________________________________________________________________________________________________
```

In [36]:
```
import tensorflow_addons as tfa
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
# learning rate schedule
def step_decay(epoch):

    import math
    initial_lrate = 0.1
    drop = 0.9
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop)
    return lrate

#lrate = LearningRateScheduler(step_decay)
lrate = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=10, m
es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 10)
fname = 'best_model8.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate,es]
#optimizer=tfa.optimizers.SGDW(weight_decay=0.99,momentum=0.9,learning_rate=
#optimizer=tfa.optimizers.AdamW(weight_decay=0.99,learning_rate=0.001)
optimizer=tf.optimizers.Adam(learning_rate=0.001)


#sgd = SGD(learning_rate = 0.1, )
model8.compile(optimizer = optimizer, loss = 'categorical_crossentropy', met
model8.fit([train_padded_1, train_padded_2], y_train_enc, epochs = 100, batc
          validation_data = ([val_padded_1, val_padded_2], y_val_enc), call
```

```
Epoch 1/100
8584/8584 [==============================] - 335s 38ms/step - loss: 0.8649 -
accuracy: 0.6084 - val_loss: 0.7642 - val_accuracy: 0.6739
Epoch 2/100
8584/8584 [==============================] - 337s 39ms/step - loss: 0.7751 -
accuracy: 0.6686 - val_loss: 0.7407 - val_accuracy: 0.6846
Epoch 3/100
8584/8584 [==============================] - 337s 39ms/step - loss: 0.7234 -
accuracy: 0.7017 - val_loss: 0.6674 - val_accuracy: 0.7303
Epoch 4/100
8584/8584 [==============================] - 338s 39ms/step - loss: 0.6876 -
accuracy: 0.7219 - val_loss: 0.6438 - val_accuracy: 0.7430
Epoch 5/100
8584/8584 [==============================] - 341s 40ms/step - loss: 0.6680 -
accuracy: 0.7338 - val_loss: 0.6266 - val_accuracy: 0.7544
Epoch 6/100
8584/8584 [==============================] - 337s 39ms/step - loss: 0.6515 -
accuracy: 0.7426 - val_loss: 0.6153 - val_accuracy: 0.7569
Epoch 7/100
```

In [ ]:
```
1
```

In [ ]:
```
1
```

tuning Bi-lstm, attention with glove

In [37]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_1], trainable = False)(in
#lstm_s1 = Bidirectional(LSTM(256, return_sequences = True) , merge_mode = '
lstm1 = Bidirectional(LSTM(256, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb1)
att_1 = Attention_Custom(512, lstm1.shape)
att_weights_1, context_1 = att_1.call(lstm1)
s1_dense = Dense(units = 300, activation = 'relu' )(context_1)


#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_2], trainable = False)(in
#lstm_s2 = Bidirectional(LSTM(20, return_sequences = True))(emb2)
lstm2 = Bidirectional(LSTM(256, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb2)
att_2 = Attention_Custom(512, lstm2.shape)
att_weights_2, context_2 = att_2.call(lstm2)
s2_dense = Dense(units = 300, activation = 'relu' )(context_2)

#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
#point-wise product of sentence representations
prod = tf.math.multiply(s1_dense, s2_dense)
#point-wise absolute difference of sentence representations
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

#concatenating
concatenated =  concatenate([merged_representations, prod, abs_diff])
# concatenated = tf.expand_dims(concatenated, axis=-1)
# conv1 = Conv1D(filters = 64, kernel_size = 3, strides=1, padding='valid')(
# conv2 = Conv1D(filters = 32, kernel_size = 3, strides=1, padding='valid')(
# conv3 = Conv1D(filters = 16, kernel_size = 3, strides=1, padding='valid')(

# flat = Flatten()(conv3)
#MLP and threeway classifier
dense1 = Dense(units = 512, activation = 'relu')(concatenated)
dense2 = Dense(units = 128, activation = 'relu')(dense1)
#drop1 = Dropout(0.3)(dense1)
# dense2 = Dense(units = 50, activation = 'relu')(drop1)
output = Dense(units = 3, activation = 'softmax')(dense2)
model9 = Model(inputs = [input1, input2], outputs = output)


```

In [39]:
```
1 model9.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_6 (InputLayer)            [(None, 78)]         0
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 57)]         0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 78, 300)      5399100     input_6[0]
[0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 57, 300)      9022800     input_7[0]
[0]
__________________________________________________________________________________________________
```

In [40]:

```
import tensorflow_addons as tfa
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
lrate = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=10, m
es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 5)
fname = 'best_model9.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate,es]
#optimizer=tfa.optimizers.SGDW(weight_decay=0.99,momentum=0.9,learning_rate=
#optimizer=tfa.optimizers.AdamW(weight_decay=0.99,learning_rate=0.001)
optimizer=tf.optimizers.Adam(learning_rate=0.001)


#sgd = SGD(learning_rate = 0.1, )
model9.compile(optimizer = optimizer, loss = 'categorical_crossentropy', met
model9.fit([train_padded_1, train_padded_2], y_train_enc, epochs = 100, batc
          validation_data = ([val_padded_1, val_padded_2], y_val_enc), call
```

```
Epoch 1/100
8584/8584 [==============================] - 378s 43ms/step - loss: 0.7961 - ac
curacy: 0.6583 - val_loss: 0.6676 - val_accuracy: 0.7347
Epoch 2/100
8584/8584 [==============================] - 379s 44ms/step - loss: 0.6795 - ac
curacy: 0.7322 - val_loss: 0.6228 - val_accuracy: 0.7649
Epoch 3/100
8584/8584 [==============================] - 378s 44ms/step - loss: 0.6436 - ac
curacy: 0.7542 - val_loss: 0.6054 - val_accuracy: 0.7734
Epoch 4/100
8584/8584 [==============================] - 381s 44ms/step - loss: 0.6209 - ac
curacy: 0.7679 - val_loss: 0.5962 - val_accuracy: 0.7821
Epoch 5/100
8584/8584 [==============================] - 379s 44ms/step - loss: 0.6051 - ac
curacy: 0.7781 - val_loss: 0.5895 - val_accuracy: 0.7888
Epoch 6/100
8584/8584 [==============================] - 381s 44ms/step - loss: 0.5924 - ac
curacy: 0.7869 - val_loss: 0.5746 - val_accuracy: 0.7937
Epoch 7/100
8584/8584 [==============================] - 388s 45ms/step - loss: 0.5822 - ac
curacy: 0.7930 - val_loss: 0.5774 - val_accuracy: 0.7972
Epoch 8/100
8584/8584 [==============================] - 387s 45ms/step - loss: 0.5724 - ac
curacy: 0.7991 - val_loss: 0.5743 - val_accuracy: 0.7989
Epoch 9/100
8584/8584 [==============================] - 383s 45ms/step - loss: 0.5651 - ac
curacy: 0.8040 - val_loss: 0.5880 - val_accuracy: 0.7973
Epoch 10/100
8584/8584 [==============================] - 383s 45ms/step - loss: 0.5583 - ac
curacy: 0.8083 - val_loss: 0.5851 - val_accuracy: 0.7995
Epoch 11/100
8584/8584 [==============================] - 387s 45ms/step - loss: 0.5527 - ac
curacy: 0.8126 - val_loss: 0.5864 - val_accuracy: 0.8010
Epoch 12/100
8584/8584 [==============================] - 387s 45ms/step - loss: 0.5465 - ac
curacy: 0.8157 - val_loss: 0.5822 - val_accuracy: 0.8029
Epoch 13/100
8584/8584 [==============================] - 390s 45ms/step - loss: 0.5419 - ac
curacy: 0.8187 - val_loss: 0.5809 - val_accuracy: 0.8058
```

```
Epoch 14/100
8584/8584 [==============================] - 388s 45ms/step - loss: 0.5365 - ac
curacy: 0.8221 - val_loss: 0.5994 - val_accuracy: 0.8005
Epoch 15/100
8584/8584 [==============================] - 389s 45ms/step - loss: 0.5328 - ac
curacy: 0.8250 - val_loss: 0.6054 - val_accuracy: 0.7972
Epoch 16/100
8584/8584 [==============================] - 389s 45ms/step - loss: 0.5286 - ac
curacy: 0.8275 - val_loss: 0.6007 - val_accuracy: 0.8097
Epoch 17/100
8584/8584 [==============================] - 388s 45ms/step - loss: 0.5257 - ac
curacy: 0.8293 - val_loss: 0.6020 - val_accuracy: 0.8063
Epoch 18/100
8584/8584 [==============================] - 375s 44ms/step - loss: 0.5208 - ac
curacy: 0.8318 - val_loss: 0.6119 - val_accuracy: 0.8055
Epoch 19/100
8584/8584 [==============================] - 381s 44ms/step - loss: 0.5174 - ac
curacy: 0.8346 - val_loss: 0.6115 - val_accuracy: 0.8087
Epoch 20/100
8584/8584 [==============================] - 381s 44ms/step - loss: 0.5147 - ac
curacy: 0.8360 - val_loss: 0.6146 - val_accuracy: 0.8100
Epoch 21/100
8584/8584 [==============================] - 381s 44ms/step - loss: 0.5119 - ac
curacy: 0.8385 - val_loss: 0.6076 - val_accuracy: 0.8078
Epoch 22/100
8584/8584 [==============================] - 378s 44ms/step - loss: 0.5088 - ac
curacy: 0.8397 - val_loss: 0.6135 - val_accuracy: 0.8062
Epoch 23/100
8584/8584 [==============================] - 587s 68ms/step - loss: 0.5061 - ac
curacy: 0.8418 - val_loss: 0.6267 - val_accuracy: 0.8039
Epoch 24/100
8584/8584 [==============================] - 384s 45ms/step - loss: 0.5033 - ac
curacy: 0.8431 - val_loss: 0.6533 - val_accuracy: 0.8062
Epoch 25/100
8584/8584 [==============================] - 385s 45ms/step - loss: 0.5006 - ac
curacy: 0.8447 - val_loss: 0.6251 - val_accuracy: 0.8045
```

Out[40]: <keras.callbacks.History at 0x216243f4b38>

In [ ]:
```
1
```

Bi-LSTM with Attention on fasttext embeddings and extracted features

In [49]:

```
emb_dim = 300

#getting sentence 1 representations
input1 = Input(shape=(maxlen1,)) #for sentence1 embeddings
emb1 = Embedding(input_dim = vocab_size_1, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_1], trainable = False)(in
#lstm_s1 = Bidirectional(LSTM(256, return_sequences = True) , merge_mode = '
lstm1 = Bidirectional(LSTM(256, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb1)
att_1 = Attention_Custom(512, lstm1.shape)
att_weights_1, context_1 = att_1.call(lstm1)
print(context_1.shape)
s1_dense = Dense(units = 300, activation = 'relu' )(context_1)
print(s1_dense.shape)

#getting sentence2 representations
input2 = Input(shape=(maxlen2,)) #for sentence2 embeddings
emb2 = Embedding(input_dim = vocab_size_2, output_dim = emb_dim, input_lengt
                 weights = [embedding_matrix_glove_2], trainable = False)(in
#lstm_s2 = Bidirectional(LSTM(20, return_sequences = True))(emb2)
lstm2 = Bidirectional(LSTM(256, return_sequences = True, dropout=0.3, kernel
                      merge_mode = 'concat')(emb2)
att_2 = Attention_Custom(512, lstm2.shape)
att_weights_2, context_2 = att_2.call(lstm2)
s2_dense = Dense(units = 300, activation = 'relu' )(context_2)
print(s2_dense.shape)
#merging sentence1 and sentence2 representations
merged_representations = concatenate([s1_dense, s2_dense])
#point-wise product of sentence representations
prod = tf.math.multiply(s1_dense, s2_dense)
#point-wise absolute difference of sentence representations
subtracted = tf.keras.layers.Subtract()([s1_dense, s2_dense])
abs_diff = tf.math.abs(subtracted)

input3 = Input(shape=(29,)) #for extracted features
dense_1 = Dense(units = 16, activation = 'relu')(input3)

#concatenating
concatenated =  concatenate([merged_representations, prod, abs_diff, dense_1
# concatenated = tf.expand_dims(concatenated, axis=-1)
# conv1 = Conv1D(filters = 64, kernel_size = 3, strides=1, padding='valid')(
# conv2 = Conv1D(filters = 32, kernel_size = 3, strides=1, padding='valid')(
# conv3 = Conv1D(filters = 16, kernel_size = 3, strides=1, padding='valid')(

# flat = Flatten()(conv3)
#MLP and threeway classifier
dense1 = Dense(units = 512, activation = 'relu')(concatenated)
dense2 = Dense(units = 128, activation = 'relu')(dense1)
#drop1 = Dropout(0.3)(dense1)
# dense2 = Dense(units = 50, activation = 'relu')(drop1)
output = Dense(units = 3, activation = 'softmax')(dense2)
model10 = Model(inputs = [input1, input2, input3], outputs = output)


```

```
(None, 512)
(None, 300)
```

```
(None, 300)
```

In [26]:
```
1 model10.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 78)]         0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 57)]         0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 78, 300)      5399100     input_2[0]
[0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 57, 300)      9022800     input_3[0]
[0]
__________________________________________________________________________________________________
```

In [27]:

```
import tensorflow_addons as tfa
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping,
import tensorflow as tf
lrate = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1, patience=10, m
es = EarlyStopping(monitor='val_accuracy', mode='max', patience = 5)
fname = 'best_model10.hdf5'
checkpoint = ModelCheckpoint(fname, monitor="val_accuracy", mode="max", save
callbacks = [lrate,es]
#optimizer=tfa.optimizers.SGDW(weight_decay=0.99,momentum=0.9,learning_rate=
#optimizer=tfa.optimizers.AdamW(weight_decay=0.99,learning_rate=0.001)
optimizer=tf.optimizers.Adam(learning_rate=0.001)


#sgd = SGD(learning_rate = 0.1, )
model10.compile(optimizer = optimizer, loss = 'categorical_crossentropy', me
model10.fit([train_padded_1, train_padded_2, X_train_1], y_train_enc, epochs
          validation_data = ([val_padded_1, val_padded_2, X_val_1], y_val_e
```

```
Epoch 1/100
8584/8584 [==============================] - 386s 44ms/step - loss: 0.8149 - ac
curacy: 0.6518 - val_loss: 0.6799 - val_accuracy: 0.7253
Epoch 2/100
8584/8584 [==============================] - 393s 46ms/step - loss: 0.7116 - ac
curacy: 0.7116 - val_loss: 0.6541 - val_accuracy: 0.7416
Epoch 3/100
8584/8584 [==============================] - 393s 46ms/step - loss: 0.6529 - ac
curacy: 0.7451 - val_loss: 0.6061 - val_accuracy: 0.7687
Epoch 4/100
8584/8584 [==============================] - 393s 46ms/step - loss: 0.6226 - ac
curacy: 0.7624 - val_loss: 0.5893 - val_accuracy: 0.7775
Epoch 5/100
8584/8584 [==============================] - 394s 46ms/step - loss: 0.6039 - ac
curacy: 0.7732 - val_loss: 0.5804 - val_accuracy: 0.7899
Epoch 6/100
8584/8584 [==============================] - 394s 46ms/step - loss: 0.5888 - ac
curacy: 0.7826 - val_loss: 0.5782 - val_accuracy: 0.7843
Epoch 7/100
8584/8584 [==============================] - 396s 46ms/step - loss: 0.5801 - ac
curacy: 0.7881 - val_loss: 0.5885 - val_accuracy: 0.7877
Epoch 8/100
8584/8584 [==============================] - 391s 46ms/step - loss: 0.5766 - ac
curacy: 0.7918 - val_loss: 0.5671 - val_accuracy: 0.7991
Epoch 9/100
8584/8584 [==============================] - 392s 46ms/step - loss: 0.5669 - ac
curacy: 0.7968 - val_loss: 0.5760 - val_accuracy: 0.7914
Epoch 10/100
8584/8584 [==============================] - 391s 46ms/step - loss: 0.5607 - ac
curacy: 0.8010 - val_loss: 0.5740 - val_accuracy: 0.7993
Epoch 11/100
8584/8584 [==============================] - 391s 46ms/step - loss: 0.5560 - ac
curacy: 0.8045 - val_loss: 0.5765 - val_accuracy: 0.7980
Epoch 12/100
8584/8584 [==============================] - 386s 45ms/step - loss: 0.5485 - ac
curacy: 0.8083 - val_loss: 0.5911 - val_accuracy: 0.7930
Epoch 13/100
8584/8584 [==============================] - 384s 45ms/step - loss: 0.5427 - ac
curacy: 0.8112 - val_loss: 0.5747 - val_accuracy: 0.7977
```

```
Epoch 14/100
8584/8584 [==============================] - 384s 45ms/step - loss: 0.5374 - ac
curacy: 0.8147 - val_loss: 0.5971 - val_accuracy: 0.7932
Epoch 15/100
8584/8584 [==============================] - 384s 45ms/step - loss: 0.5313 - ac
curacy: 0.8179 - val_loss: 0.5945 - val_accuracy: 0.7986
```

Out[27]: <keras.callbacks.History at 0x191810855f8>

the best performance by a model until now is val acc = 0.81, model: bi-lstm with self attention and glove embeddings

In [ ]:
```
1
```