Improved LSTM baseline

This kernel is a somewhat improved version of Keras - Bidirectional LSTM baseline along with some additional documentation of the steps. (NB: this notebook has been re-run on the new test set.)

In [1]:
```
import sys, os, re, csv, codecs, numpy as np, pandas as pd

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model
from keras import initializers, regularizers, constraints, optimizers, layers
```

```
Using TensorFlow backend.
```

We include the GloVe word vectors in our input files. To include these in your kernel, simple click 'input files' at the top of the notebook, and search 'glove' in the 'datasets' section.

In [2]:
```
path = '../input/'
comp = 'jigsaw-toxic-comment-classification-challenge/'
EMBEDDING_FILE=f'{path}glove6b50d/glove.6B.50d.txt'
TRAIN_DATA_FILE=f'{path}{comp}train.csv'
TEST_DATA_FILE=f'{path}{comp}test.csv'
```

Set some basic config parameters:

In [3]:
```
embed_size = 50 # how big is each word vector
max_features = 20000 # how many unique words to use (i.e num rows in embedding vector)
maxlen = 100 # max number of words in a comment to use
```

Read in our data and replace missing values:

In [4]:
```
train = pd.read_csv(TRAIN_DATA_FILE)
test = pd.read_csv(TEST_DATA_FILE)

list_sentences_train = train["comment_text"].fillna("_na_").values
list_classes = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
y = train[list_classes].values
list_sentences_test = test["comment_text"].fillna("_na_").values
```

Standard keras preprocessing, to turn each comment into a list of word indexes of equal length (with truncation or padding as needed).

In [5]:
```
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(list_sentences_train))
list_tokenized_train = tokenizer.texts_to_sequences(list_sentences_train)
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

Read the glove word vectors (space delimited strings) into a dictionary from word->vector.

In [6]:
```
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.strip().split()) for o in open(EMBEDDING_FILE))
```

Use these vectors to create our embedding matrix, with random initialization for words that aren't in GloVe. We'll use the same mean and stdev of embeddings the GloVe has when generating the random init.

In [7]:
```
all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
emb_mean,emb_std
```

Out[7]:
```
(0.020940498, 0.6441043)
```

In [8]:
```
word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

Simple bidirectional LSTM with two fully connected layers. We add some dropout to the LSTM since even 2 epochs is enough to overfit.

In [9]:
```
inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
x = Bidirectional(LSTM(50, return_sequences=True, dropout=0.1, recurrent_dropout=0.1))(x)
x = GlobalMaxPool1D()(x)
x = Dense(50, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(6, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Now we're ready to fit out model! Use `validation_split` when not submitting.

In [10]:
```
model.fit(X_t, y, batch_size=32, epochs=2, validation_split=0.1);
```

```
Train on 143613 samples, validate on 15958 samples
Epoch 1/2
143613/143613 [==============================] - 1475s 10ms/step - loss: 0.0602 - acc: 0.9792 - val_l
oss: 0.0502 - val_acc: 0.9821
Epoch 2/2
 47232/143613 [========>.....................] - ETA: 16:15 - loss: 0.0448 - acc: 0.9832
```

And finally, get predictions for the test set and prepare a submission CSV:

In [11]:
```
y_test = model.predict([X_te], batch_size=1024, verbose=1)
sample_submission = pd.read_csv(f'{path}{comp}sample_submission.csv')
sample_submission[list_classes] = y_test
sample_submission.to_csv('submission.csv', index=False)
```

```
153164/153164 [==============================] - 113s 740us/step
```

In [12]:
```

```