In [1]:
```
import numpy as np
import pandas as pd
import os
from tqdm import tqdm
tqdm.pandas()
```

In [2]:
```
TEXT_COL = 'comment_text'
EMB_PATH = '../input/fasttext-crawl-300d-2m/crawl-300d-2M.vec'
train = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/train.csv', index_col='id')
test = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/test.csv', index_col='id')
```

```
/opt/conda/lib/python3.6/site-packages/numpy/lib/arraysetops.py:569: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```

In [3]:
```
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')

def load_embeddings(embed_dir=EMB_PATH):
    embedding_index = dict(get_coefs(*o.strip().split(" ")) for o in tqdm(open(embed_dir)))
    return embedding_index

def build_embedding_matrix(word_index, embeddings_index, max_features, lower = True, verbose = True):
    embedding_matrix = np.zeros((max_features, 300))
    for word, i in tqdm(word_index.items(),disable = not verbose):
        if lower:
            word = word.lower()
        if i >= max_features: continue
        try:
            embedding_vector = embeddings_index[word]
        except:
            embedding_vector = embeddings_index["unknown"]
        if embedding_vector is not None:
            # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

def build_matrix(word_index, embeddings_index):
    embedding_matrix = np.zeros((len(word_index) + 1,300))
    for word, i in word_index.items():
        try:
            embedding_matrix[i] = embeddings_index[word]
        except:
            embedding_matrix[i] = embeddings_index["unknown"]
    return embedding_matrix
```

In [4]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import gc

maxlen = 220
max_features = 100000
embed_size = 300
tokenizer = Tokenizer(num_words=max_features, lower=True) #filters = ''
#tokenizer = text.Tokenizer(num_words=max_features)
print('fitting tokenizer')
tokenizer.fit_on_texts(list(train[TEXT_COL]) + list(test[TEXT_COL]))
word_index = tokenizer.word_index
X_train = tokenizer.texts_to_sequences(list(train[TEXT_COL]))
y_train = train['target'].values
X_test = tokenizer.texts_to_sequences(list(test[TEXT_COL]))

X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)


del tokenizer
gc.collect()
```

```
Using TensorFlow backend.
```

```
fitting tokenizer
```

Out[4]:
```
0
```

In [5]:
```
embeddings_index = load_embeddings()
```

```
2000001it [02:12, 15062.90it/s]
```

In [6]:
```
embedding_matrix = build_matrix(word_index, embeddings_index)
```

In [7]:
```
del embeddings_index
gc.collect()
```

Out[7]:
```
0
```

In [8]:
```
from keras import backend as K
from keras.engine.topology import Layer
from keras import initializers, regularizers, constraints, optimizers, layers

class Attention(Layer):
    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')

        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)

        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)

        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight((input_shape[-1],),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
        self.features_dim = input_shape[-1]

        if self.bias:
            self.b = self.add_weight((input_shape[1],),
                                     initializer='zero',
                                     name='{}_b'.format(self.name),
                                     regularizer=self.b_regularizer,
                                     constraint=self.b_constraint)
        else:
            self.b = None

        self.built = True

    def compute_mask(self, input, input_mask=None):
        return None

    def call(self, x, mask=None):
        features_dim = self.features_dim
        step_dim = self.step_dim

        eij = K.reshape(K.dot(K.reshape(x, (-1, features_dim)),
                        K.reshape(self.W, (features_dim, 1))), (-1, step_dim))

        if self.bias:
            eij += self.b

        eij = K.tanh(eij)

        a = K.exp(eij)

        if mask is not None:
            a *= K.cast(mask, K.floatx())

        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())

        a = K.expand_dims(a)
        weighted_input = x * a
        return K.sum(weighted_input, axis=1)

    def compute_output_shape(self, input_shape):
        return input_shape[0],  self.features_dim
```

In [9]:
```
import keras.layers as L
from keras.models import Model
from keras.optimizers import Adam

def build_model(verbose = False, compile = True):
    sequence_input = L.Input(shape=(maxlen,), dtype='int32')
    embedding_layer = L.Embedding(len(word_index) + 1,
                                  300,
                                  weights=[embedding_matrix],
                                  input_length=maxlen,
                                  trainable=False)
    x = embedding_layer(sequence_input)
    x = L.SpatialDropout1D(0.2)(x)
    x = L.Bidirectional(L.CuDNNLSTM(64, return_sequences=True))(x)

    att = Attention(maxlen)(x)
    avg_pool1 = L.GlobalAveragePooling1D()(x)
    max_pool1 = L.GlobalMaxPooling1D()(x)

    x = L.concatenate([att,avg_pool1, max_pool1])

    preds = L.Dense(1, activation='sigmoid')(x)


    model = Model(sequence_input, preds)
    if verbose:
        model.summary()
    if compile:
        model.compile(loss='binary_crossentropy',optimizer=Adam(0.005),metrics=['acc'])
    return model
```

In [10]:
```
from sklearn.model_selection import KFold

splits = list(KFold(n_splits=5).split(X_train,y_train))


from keras.callbacks import EarlyStopping, ModelCheckpoint
import keras.backend as K
import numpy as np
BATCH_SIZE = 2048
NUM_EPOCHS = 100

oof_preds = np.zeros((X_train.shape[0]))
test_preds = np.zeros((X_test.shape[0]))
for fold in [0,1,2,3,4]:
    K.clear_session()
    tr_ind, val_ind = splits[fold]
    ckpt = ModelCheckpoint(f'gru_{fold}.hdf5', save_best_only = True)
    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
    model = build_model()
    model.fit(X_train[tr_ind],
          y_train[tr_ind]>0.5,
          batch_size=BATCH_SIZE,
          epochs=NUM_EPOCHS,
          validation_data=(X_train[val_ind], y_train[val_ind]>0.5),
          callbacks = [es,ckpt])

    oof_preds[val_ind] += model.predict(X_train[val_ind])[:,0]
    test_preds += model.predict(X_test)[:,0]
test_preds /= 5
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_grad.py:102: div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Train on 1443899 samples, validate on 360975 samples
Epoch 1/100
1443899/1443899 [==============================] - 138s 95us/step - loss: 0.1169 - acc: 0.9584 - val_loss: 0.0923 - val_acc: 0.9646
Epoch 2/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0935 - acc: 0.9644 - val_loss: 0.0909 - val_acc: 0.9652
Epoch 3/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0893 - acc: 0.9658 - val_loss: 0.0881 - val_acc: 0.9662
Epoch 4/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0867 - acc: 0.9664 - val_loss: 0.0927 - val_acc: 0.9633
Epoch 5/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0847 - acc: 0.9673 - val_loss: 0.0925 - val_acc: 0.9585
Epoch 6/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0831 - acc: 0.9677 - val_loss: 0.0942 - val_acc: 0.9628
Epoch 00006: early stopping
Train on 1443899 samples, validate on 360975 samples
Epoch 1/100
1443899/1443899 [==============================] - 138s 95us/step - loss: 0.1193 - acc: 0.9576 - val_loss: 0.0995 - val_acc: 0.9613
Epoch 2/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0950 - acc: 0.9639 - val_loss: 0.0930 - val_acc: 0.9634
Epoch 3/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0906 - acc: 0.9653 - val_loss: 0.0932 - val_acc: 0.9650
Epoch 4/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0881 - acc: 0.9663 - val_loss: 0.0876 - val_acc: 0.9661
Epoch 5/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0856 - acc: 0.9670 - val_loss: 0.0922 - val_acc: 0.9631
Epoch 6/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0841 - acc: 0.9676 - val_loss: 0.0880 - val_acc: 0.9654
Epoch 7/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0824 - acc: 0.9683 - val_loss: 0.0880 - val_acc: 0.9662
Epoch 00007: early stopping
Train on 1443899 samples, validate on 360975 samples
Epoch 1/100
1443899/1443899 [==============================] - 138s 95us/step - loss: 0.1179 - acc: 0.9573 - val_loss: 0.0868 - val_acc: 0.9669
Epoch 2/100
1443899/1443899 [==============================] - 137s 95us/step - loss: 0.0937 - acc: 0.9642 - val_loss: 0.0845 - val_acc: 0.9672
Epoch 3/100
 106496/1443899 [=>............................] - ETA: 1:58 - loss: 0.0902 - acc: 0.9649
```

In [11]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train>0.5,oof_preds)
```

Out[11]:
```
0.9701654911792239
```

In [12]:
```
submission = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/sample_submission.csv', index_col='id')
submission['prediction'] = test_preds
submission.reset_index(drop=False, inplace=True)
submission.head()
```

Out[12]:

	id	prediction
0	7000000	0.001369
1	7000001	0.000080
2	7000002	0.005752
3	7000003	0.001052
4	7000004	0.992313

In [13]:
```
submission.to_csv('submission.csv', index=False)
```