In [6]:
```
import pandas as pd
import numpy as np
import pickle
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activat
ion, Conv1D, GRU, Bidirectional
from keras.layers import GlobalAveragePooling1D, GlobalMaxPooling1D, con
catenate, SpatialDropout1D
from keras.models import Model, load_model
from keras.callbacks import Callback, ModelCheckpoint, EarlyStopping
from keras.optimizers import Adam
from sklearn.metrics import roc_auc_score, f1_score, precision_recall_fs
core_support
import logging
import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

In [7]:
```
# Train test split ratio of 0.3
X_train = pd.read_pickle("../../Preprocessing/Data/X_train.pkl")
X_test = pd.read_pickle("../../Preprocessing/Data/X_test.pkl")
y_train = pd.read_pickle("../../Preprocessing/Data/y_train.pkl")
y_test = pd.read_pickle("../../Preprocessing/Data/y_test.pkl")

# Url - https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-30
0d-2M.vec.zip
# pre-trained FastText vector file
# word million word vectors trained on Common Crawl (600B tokens)
embedding_path = "Data/crawl-300d-2M.vec"

# no. of dimensions for each vector
embed_size = 300
# maximum number of unique words to be considered
max_features = 100000
# maximum length of a comment
max_len = 220
```

In [8]:
```
# vectorize a text corpus
# turns text into sequence of space-separated sequence of words
# sequences are split into list of tokens
# they will be indexed or vectorized

# create tokenizer
token = Tokenizer(num_words=max_features, lower=True)
# fit the tokenizer on training data
# create internal vocabulary index based on word frequency
token.fit_on_texts(X_train)

# convert comment texts to their numeric counterparts
# transform each text in texts to sequence of integers
# i.e replaces each word in text with corresponding integer value from
# word_index dictionary
X_train = token.texts_to_sequences(X_train)
X_test = token.texts_to_sequences(X_test)
```

In [9]:
```
# padding to make comments uniform in length
# output will be padded sequence of numbers
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)
```

In [10]:
```
# load FastText word embeddings
def get_coefficients(word,*arr):
  return word, np.asarray(arr, dtype='float32')

embeddings_index = dict(get_coefficients(*v.strip().split(" ")) for v in
open(embedding_path))

# create embedding matrix that contains words in our corpus and their
# corresponding values from FastText embeddings
word_index = token.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.zeros((nb_words, embed_size))
for word, index in word_index.items():
    if index >= max_features:
      continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
      embedding_matrix[index] = embedding_vector
```

In [0]:
```
class RocAucEvaluation(Callback):
    def __init__(self, validation_data=(), interval=1):
        super(Callback, self).__init__()

        self.interval = interval
        self.X_val, self.y_val = validation_data

    def on_epoch_end(self, epoch, logs={}):
        if epoch % self.interval == 0:
            y_pred = self.model.predict(self.X_val, verbose=0)
            score = roc_auc_score(self.y_val, y_pred)
            # results = self.model.evaluate(self.X_val, self.y_val, verb
ose=1)
            # precision, recall, fscore, _ = precision_recall_fscore_sup
port(y_pred, self.y_val)
            # print("\n ROC-AUC - epoch: {:d} - score: {:.6f} - score:
 {:.6f} - score: {:.6f} - score: {:.6f}".format(epoch+1, score, precisio
n, recall, fscore))
            # print("\n Results: test_loss: {:d} - test_accuracy: {:.6
f}".format(results[0], results[1]))
            print("\n ROC-AUC - epoch: {:d} - score: {:.6f}".format(epoc
h+1, score))
```

In [0]:
```
# calculating ROC_AUC score after every epoch
ra_val = RocAucEvaluation(validation_data=(X_test, y_test), interval=1)
# stop training when a monitored quantity (specified by monitor attribut
e) has stopped improving
# patience = number of epochs that produced the monitored quantity with
 no improvement after which training will be stopped
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=3)
```

In [0]:
```
def build_model(lr = 0.0, lr_d = 0.0, units = 0, dr = 0.0):
  # instantiates a Keras tensor
  inp = Input(shape=(max_len,))
  # first layer
  # matrix is used to initialize weights in the Embedding layer of the m
odel
  # trainable=False to prevent the weights from being updated during tra
ining
  x = Embedding(nb_words, embed_size, weights = [embedding_matrix], trai
nable = False)(inp)
  # drops entire 1D feature maps (channels) instead of individual elemen
ts
  # randomly setting a fraction rate (dr) of input units to 0 at each up
date, to prevent overfitting
  x1 = SpatialDropout1D(dr)(x)

  # bi-directional GRU and LSTM to keep contextual information in both d
irections
  x = Bidirectional(GRU(units, return_sequences = True))(x1)
  x = Conv1D(64, kernel_size = 2, padding = "valid", kernel_initializer
= "he_uniform")(x)

  y = Bidirectional(LSTM(units, return_sequences = True))(x1)
  y = Conv1D(64, kernel_size = 2, padding = "valid", kernel_initializer
= "he_uniform")(y)

  # to minimize overfitting
  avg_pool1 = GlobalAveragePooling1D()(x)
  max_pool1 = GlobalMaxPooling1D()(x)

  avg_pool2 = GlobalAveragePooling1D()(y)
  max_pool2 = GlobalMaxPooling1D()(y)

  # returns a single tensor by merging all the pooling layers
  x = concatenate([avg_pool1, max_pool1, avg_pool2, max_pool2])
  # Output layer which classifies a given comment into one of 6 toxic le
vels
  x = Dense(6, activation = "sigmoid")(x)

  # this model includes all layers required in computation of x given by
inp
  model = Model(inputs = inp, outputs = x)
  # configure model for training
  model.compile(loss = "binary_crossentropy", optimizer = Adam(lr = lr,
decay = lr_d), metrics = ["accuracy"])
  # trains the model for fixed number of epochs
  history = model.fit(X_train, y_train, batch_size = 128, epochs = 3, va
lidation_data = (X_test, y_test),
                      verbose = 1, callbacks = [ra_val, early_stop])
  # model = load_model(file_path)
  return model
```

In [0]:
```
# learning_rate for optimizer    = 1e-4
# learning_rate decay            = 0
# Output dimensionality for LSTM = 128
# dropout_rate                   = 0.2
model = build_model(lr = 1e-4, lr_d = 0, units = 128, dr = 0.2)
```

```
Train on 39912 samples, validate on 19659 samples
Epoch 1/3
39912/39912 [==============================] - 397s 10ms/step - loss:
0.2412 - acc: 0.9144 - val_loss: 0.1320 - val_acc: 0.9510

 ROC-AUC - epoch: 1 - score: 0.931762
Epoch 2/3
39912/39912 [==============================] - 387s 10ms/step - loss:
0.1290 - acc: 0.9505 - val_loss: 0.1192 - val_acc: 0.9536

 ROC-AUC - epoch: 2 - score: 0.942917
Epoch 3/3
39912/39912 [==============================] - 387s 10ms/step - loss:
0.1194 - acc: 0.9531 - val_loss: 0.1125 - val_acc: 0.9563

 ROC-AUC - epoch: 3 - score: 0.949765
```

In [0]:
```
pickle.dump(model.history, open('Models/FastText_NN.sav', 'wb'))
```

In [0]:
```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 220)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 220, 300)     18345300    input_1[0][0]
__________________________________________________________________________________________________
spatial_dropout1d_1 (SpatialDro (None, 220, 300)     0           embedding_1[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 220, 256)     329472      spatial_dropout1d_1[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 220, 256)     439296      spatial_dropout1d_1[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 219, 64)      32832       bidirectional_1[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 219, 64)      32832       bidirectional_2[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_1 (Glo (None, 64)           0           conv1d_1[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 64)           0           conv1d_1[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_2 (Glo (None, 64)           0           conv1d_2[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_2 (GlobalM (None, 64)           0           conv1d_2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 256)          0           global_average_pooling1d_1[0][0]
                                                                 global_max_pooling1d_1[0][0]
                                                                 global_average_pooling1d_2[0][0]
                                                                 global
```

```
_max_pooling1d_2[0][0]
__________________________________________________________________________________
_______________________________
dense_1 (Dense)                 (None, 6)            1542        concat
enate_1[0][0]
==================================================================================
===========================
Total params: 19,181,274
Trainable params: 835,974
Non-trainable params: 18,345,300
__________________________________________________________________________________
_______________________________
```

In [0]:
```
results = model.evaluate(X_test, y_test, verbose=1)

print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

```
19659/19659 [==============================] - 250s 13ms/step
Test Loss: 0.11250274048393247
Test Accuracy: 0.9563049891847725
```