In [1]:
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input
 directory

import os
print(os.listdir("../input"))
print(os.listdir("../input/glove-global-vectors-for-word-representation"))
print(os.listdir("../input/jigsaw-unintended-bias-in-toxicity-classification"))

# Any results you write to the current directory are saved as output.
```

```
['glove-global-vectors-for-word-representation', 'jigsaw-unintended-bias-in-toxicity-classification']
['glove.6B.50d.txt', 'glove.6B.200d.txt', 'glove.6B.100d.txt']
['train.csv', 'sample_submission.csv', 'test.csv']
```

In [2]:
```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import datetime
import os
import pandas as pd
import numpy as np
import pkg_resources
import seaborn as sns
import time
import scipy.stats as stats

from sklearn import metrics
from sklearn import model_selection

from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding
from keras.layers import Input
from keras.layers import Conv1D
from keras.layers import MaxPooling1D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import Dense
from keras.optimizers import RMSprop
from keras.models import Model
from keras.models import load_model
```

```
Using TensorFlow backend.
```

Load and pre-process the data set

In [3]:
```python
train = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/train.csv')
print('loaded %d records' % len(train))

# Make sure all comment_text values are strings
train['comment_text'] = train['comment_text'].astype(str)

# List all identities
identity_columns = [
    'male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',
    'muslim', 'black', 'white', 'psychiatric_or_mental_illness']

# Convert taget and identity columns to booleans
def convert_to_bool(df, col_name):
    df[col_name] = np.where(df[col_name] >= 0.5, True, False)

def convert_dataframe_to_bool(df):
    bool_df = df.copy()
    for col in ['target'] + identity_columns:
        convert_to_bool(bool_df, col)
    return bool_df

train = convert_dataframe_to_bool(train)
```

```
loaded 1804874 records
```

Split the data into 80% train and 20% validate sets

In [4]:
```python
train_df, validate_df = model_selection.train_test_split(train, test_size=0.2)
print('%d train comments, %d validate comments' % (len(train_df), len(validate_df)))
```

```
1443899 train comments, 360975 validate comments
```

Create a text tokenizer

In [5]:
```python
MAX_NUM_WORDS = 10000
TOXICITY_COLUMN = 'target'
TEXT_COLUMN = 'comment_text'

# Create a text tokenizer.
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(train_df[TEXT_COLUMN])

# All comments must be truncated or padded to be the same length.
MAX_SEQUENCE_LENGTH = 250
def pad_text(texts, tokenizer):
    return pad_sequences(tokenizer.texts_to_sequences(texts), maxlen=MAX_SEQUENCE_LENGTH)
```

Define and train a Convolutional Neural Net for classifying toxic comments

In [6]:
```python
EMBEDDINGS_PATH = '../input/glove-global-vectors-for-word-representation/glove.6B.100d.txt'
EMBEDDINGS_DIMENSION = 100
DROPOUT_RATE = 0.3
LEARNING_RATE = 0.00005
NUM_EPOCHS = 10
BATCH_SIZE = 128

def train_model(train_df, validate_df, tokenizer):
    # Prepare data
    train_text = pad_text(train_df[TEXT_COLUMN], tokenizer)
    train_labels = to_categorical(train_df[TOXICITY_COLUMN])
    validate_text = pad_text(validate_df[TEXT_COLUMN], tokenizer)
    validate_labels = to_categorical(validate_df[TOXICITY_COLUMN])

    # Load embeddings
    print('loading embeddings')
    embeddings_index = {}
    with open(EMBEDDINGS_PATH) as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs

    embedding_matrix = np.zeros((len(tokenizer.word_index) + 1,
                                 EMBEDDINGS_DIMENSION))
    num_words_in_embedding = 0
    for word, i in tokenizer.word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            num_words_in_embedding += 1
            # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector

    # Create model layers.
    def get_convolutional_neural_net_layers():
        """Returns (input_layer, output_layer)"""
        sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
        embedding_layer = Embedding(len(tokenizer.word_index) + 1,
                                    EMBEDDINGS_DIMENSION,
                                    weights=[embedding_matrix],
                                    input_length=MAX_SEQUENCE_LENGTH,
                                    trainable=False)
        x = embedding_layer(sequence_input)
        x = Conv1D(128, 2, activation='relu', padding='same')(x)
        x = MaxPooling1D(5, padding='same')(x)
        x = Conv1D(128, 3, activation='relu', padding='same')(x)
        x = MaxPooling1D(5, padding='same')(x)
        x = Conv1D(128, 4, activation='relu', padding='same')(x)
        x = MaxPooling1D(40, padding='same')(x)
        x = Flatten()(x)
        x = Dropout(DROPOUT_RATE)(x)
        x = Dense(128, activation='relu')(x)
        preds = Dense(2, activation='softmax')(x)
        return sequence_input, preds

    # Compile model.
    print('compiling model')
    input_layer, output_layer = get_convolutional_neural_net_layers()
    model = Model(input_layer, output_layer)
    model.compile(loss='categorical_crossentropy',
                  optimizer=RMSprop(lr=LEARNING_RATE),
                  metrics=['acc'])

    # Train model.
    print('training model')
    model.fit(train_text,
              train_labels,
              batch_size=BATCH_SIZE,
              epochs=NUM_EPOCHS,
              validation_data=(validate_text, validate_labels),
              verbose=2)

    return model

model = train_model(train_df, validate_df, tokenizer)
```

```
loading embeddings
compiling model
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_def_lib
rary.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed i
n 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:34
45: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be remo
ved in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
training model
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:306
6: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Use tf.cast instead.
Train on 1443899 samples, validate on 360975 samples
Epoch 1/10
 - 92s - loss: 0.1954 - acc: 0.9327 - val_loss: 0.1673 - val_acc: 0.9407
Epoch 2/10
 - 89s - loss: 0.1644 - acc: 0.9417 - val_loss: 0.1586 - val_acc: 0.9436
Epoch 3/10
 - 90s - loss: 0.1576 - acc: 0.9439 - val_loss: 0.1566 - val_acc: 0.9447
Epoch 4/10
 - 90s - loss: 0.1540 - acc: 0.9450 - val_loss: 0.1536 - val_acc: 0.9456
Epoch 5/10
 - 89s - loss: 0.1512 - acc: 0.9459 - val_loss: 0.1513 - val_acc: 0.9456
Epoch 6/10
 - 89s - loss: 0.1491 - acc: 0.9464 - val_loss: 0.1500 - val_acc: 0.9459
Epoch 7/10
 - 88s - loss: 0.1474 - acc: 0.9468 - val_loss: 0.1506 - val_acc: 0.9457
Epoch 8/10
 - 88s - loss: 0.1459 - acc: 0.9472 - val_loss: 0.1490 - val_acc: 0.9458
Epoch 9/10
 - 88s - loss: 0.1450 - acc: 0.9473 - val_loss: 0.1472 - val_acc: 0.9468
Epoch 10/10
 - 88s - loss: 0.1440 - acc: 0.9478 - val_loss: 0.1473 - val_acc: 0.9460
```

Generate model predictions on the validation set

In [7]:
```python
MODEL_NAME = 'my_model'
validate_df[MODEL_NAME] = model.predict(pad_text(validate_df[TEXT_COLUMN], tokenizer))[:, 1]
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#index
ing-view-versus-copy
```

In [8]:
```python
validate_df.head()
```

Out[8]:

	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat	asian	atheist	bisexual	black	buddh
759210	5049616	False	Springer,\n\nIt is RICH you somehow look at po...	0.0	0.0	0.0	0.000000	0.0	NaN	NaN	NaN	False	N
1340596	5753854	False	ummmm 90% in my hood	0.0	0.0	0.0	0.000000	0.0	NaN	NaN	NaN	False	N
700140	4979226	False	well said sir.	0.0	0.0	0.0	0.000000	0.0	NaN	NaN	NaN	False	N
186419	469117	False	Growing up in the foster system in Alaska for ...	0.0	0.0	0.0	0.166667	0.0	NaN	NaN	NaN	False	N
1715924	6225642	False	How many friday nights in a year is that?	0.0	0.0	0.0	0.000000	0.0	NaN	NaN	NaN	False	N

Define bias metrics, then evaluate our new model for bias using the validation set predictions

In [9]:
```python
SUBGROUP_AUC = 'subgroup_auc'
BPSN_AUC = 'bpsn_auc'  # stands for background positive, subgroup negative
BNSP_AUC = 'bnsp_auc'  # stands for background negative, subgroup positive

def compute_auc(y_true, y_pred):
    try:
        return metrics.roc_auc_score(y_true, y_pred)
    except ValueError:
        return np.nan

def compute_subgroup_auc(df, subgroup, label, model_name):
    subgroup_examples = df[df[subgroup]]
    return compute_auc(subgroup_examples[label], subgroup_examples[model_name])

def compute_bpsn_auc(df, subgroup, label, model_name):
    """Computes the AUC of the within-subgroup negative examples and the background positive example
s."""
    subgroup_negative_examples = df[df[subgroup] & ~df[label]]
    non_subgroup_positive_examples = df[~df[subgroup] & df[label]]
    examples = subgroup_negative_examples.append(non_subgroup_positive_examples)
    return compute_auc(examples[label], examples[model_name])

def compute_bnsp_auc(df, subgroup, label, model_name):
    """Computes the AUC of the within-subgroup positive examples and the background negative example
s."""
    subgroup_positive_examples = df[df[subgroup] & df[label]]
    non_subgroup_negative_examples = df[~df[subgroup] & ~df[label]]
    examples = subgroup_positive_examples.append(non_subgroup_negative_examples)
    return compute_auc(examples[label], examples[model_name])

def compute_bias_metrics_for_model(dataset,
                                   subgroups,
                                   model,
                                   label_col,
                                   include_asegs=False):
    """Computes per-subgroup metrics for all subgroups and one model."""
    records = []
    for subgroup in subgroups:
        record = {
            'subgroup': subgroup,
            'subgroup_size': len(dataset[dataset[subgroup]])
        }
        record[SUBGROUP_AUC] = compute_subgroup_auc(dataset, subgroup, label_col, model)
        record[BPSN_AUC] = compute_bpsn_auc(dataset, subgroup, label_col, model)
        record[BNSP_AUC] = compute_bnsp_auc(dataset, subgroup, label_col, model)
        records.append(record)
    return pd.DataFrame(records).sort_values('subgroup_auc', ascending=True)

bias_metrics_df = compute_bias_metrics_for_model(validate_df, identity_columns, MODEL_NAME, TOXICITY_CO
LUMN)
bias_metrics_df
```

Out[9]:

	bnsp_auc	bpsn_auc	subgroup	subgroup_auc	subgroup_size
6	0.960970	0.747399	black	0.800943	3025
7	0.961173	0.757097	white	0.806505	5047
2	0.956972	0.771142	homosexual_gay_or_lesbian	0.806888	2210
5	0.953894	0.802245	muslim	0.829797	4202
1	0.935772	0.868028	female	0.864950	10804
8	0.956977	0.834996	psychiatric_or_mental_illness	0.868650	954
4	0.940793	0.853562	jewish	0.871105	1570
0	0.949669	0.854961	male	0.879031	8898
3	0.929157	0.906609	christian	0.902127	8004

Calculate the final score

In [10]:
```python
def calculate_overall_auc(df, model_name):
    true_labels = df[TOXICITY_COLUMN]
    predicted_labels = df[model_name]
    return metrics.roc_auc_score(true_labels, predicted_labels)

def power_mean(series, p):
    total = sum(np.power(series, p))
    return np.power(total / len(series), 1 / p)

def get_final_metric(bias_df, overall_auc, POWER=-5, OVERALL_MODEL_WEIGHT=0.25):
    bias_score = np.average([
        power_mean(bias_df[SUBGROUP_AUC], POWER),
        power_mean(bias_df[BPSN_AUC], POWER),
        power_mean(bias_df[BNSP_AUC], POWER)
    ])
    return (OVERALL_MODEL_WEIGHT * overall_auc) + ((1 - OVERALL_MODEL_WEIGHT) * bias_score)

get_final_metric(bias_metrics_df, calculate_overall_auc(validate_df, MODEL_NAME))
```

Out[10]:
```
0.8835451822570127
```

Prediction on Test data

In [11]:
```python
test = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/test.csv')
submission = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/sample_submission.
csv', index_col='id')
```

In [12]:
```python
submission['prediction'] = model.predict(pad_text(test[TEXT_COLUMN], tokenizer))[:, 1]
submission.to_csv('submission.csv')
```