In [1]:
```
import os
import re
import csv
import sys
import timeit
import codecs
import numpy as np
import pandas as pd
import keras.layers as KL
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from string import punctuation
from gensim.models import KeyedVectors
from keras import backend as KB
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, C
onv1D, GlobalAveragePooling1D
from keras.layers.merge import concatenate
from keras.models import Model
from keras.layers.core import Reshape, Permute, Lambda
from keras.layers.normalization import BatchNormalization
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
C:\Users\tianh\Desktop\environments\mlenv\lib\site-packages\h5py\__init__.py:
36: FutureWarning: Conversion of the second argument of issubdtype from `floa
t` to `np.floating` is deprecated. In future, it will be treated as `np.float
64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
C:\Users\tianh\Desktop\environments\mlenv\lib\site-packages\gensim\utils.py:1
167: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
BASE_DIR = 'data/'
EMBEDDING_FILE = BASE_DIR + 'GoogleNews-vectors-negative300.bin'
TRAIN_DATA_FILE = BASE_DIR + 'train.csv'
TEST_DATA_FILE = BASE_DIR + 'test.csv'
MAX_SEQUENCE_LENGTH = 30
MAX_NB_WORDS = 100000
EMBEDDING_DIM = 300
VALIDATION_SPLIT = 0.3
TIME_STEPS = 200
# if True, the attention vector is shared across the input_dimensions where th
e attention is applied.
SINGLE_ATTENTION_VECTOR = False
APPLY_ATTENTION_AFTER_LSTM = True
```

In [3]:
```
num_lstm = np.random.randint(175, 275)
num_dense = np.random.randint(100, 150)
rate_drop_lstm = 0.15 + np.random.rand() * 0.25
rate_drop_dense = 0.15 + np.random.rand() * 0.25
act = 'relu'
re_weight = True # whether to re-weight classes to fit the 17.5% share in test
 set
STAMP = 'lstm_baseline_cnn_%d_%d_%.2f_%.2f'%(num_lstm, num_dense, rate_drop_ls
tm, rate_drop_dense)
```

In [4]:
```
print('Indexing word vectors')
word2vec = KeyedVectors.load_word2vec_format(EMBEDDING_FILE, binary=True)
print('Found %s word vectors of word2vec' % len(word2vec.vocab))
```

```
Indexing word vectors
Found 3000000 word vectors of word2vec
```

In [5]:
```
print('Processing text dataset')

# The function "text_to_wordlist" is from
# https://www.kaggle.com/currie32/quora-question-pairs/the-importance-of-clean
ing-text
def text_to_wordlist(text, remove_stopwords=False, stem_words=False):
    # Clean the text, with the option to remove stopwords and to stem words.

    # Convert words to lower case and split them
    text = text.lower().split()

    # Optionally, remove stop words
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        text = [w for w in text if not w in stops]

    text = " ".join(text)

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"'", " ", text)
```

```
    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
    text = re.sub(r":", " : ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r" u s ", " american ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e - mail", "email", text)
    text = re.sub(r"j k", "jk", text)
    text = re.sub(r"\s{2,}", " ", text)

    # Optionally, shorten words to their stems
    if stem_words:
        text = text.split()
        stemmer = SnowballStemmer('english')
        stemmed_words = [stemmer.stem(word) for word in text]
        text = " ".join(stemmed_words)

    # Return a list of words
    return(text)

texts_1 = []
texts_2 = []
labels = []
with codecs.open(TRAIN_DATA_FILE, encoding='utf-8') as f:
    reader = csv.reader(f, delimiter=',')
    header = next(reader)
    for values in reader:
        texts_1.append(text_to_wordlist(values[3]))
        texts_2.append(text_to_wordlist(values[4]))
        labels.append(int(values[5]))
print('Found %s texts in train.csv' % len(texts_1))

tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts_1 + texts_2)

sequences_1 = tokenizer.texts_to_sequences(texts_1)
sequences_2 = tokenizer.texts_to_sequences(texts_2)

word_index = tokenizer.word_index
print('Found %s unique tokens' % len(word_index))

data_1 = pad_sequences(sequences_1, maxlen=MAX_SEQUENCE_LENGTH)
data_2 = pad_sequences(sequences_2, maxlen=MAX_SEQUENCE_LENGTH)
labels = np.array(labels)
print('Shape of data tensor:', data_1.shape)
print('Shape of label tensor:', labels.shape)
```

```
Processing text dataset
Found 404290 texts in train.csv
Found 85518 unique tokens
Shape of data tensor: (404290, 30)
Shape of label tensor: (404290,)
```

In [6]:
```
print('Preparing embedding matrix')

nb_words = min(MAX_NB_WORDS, len(word_index))+1

embedding_matrix = np.zeros((nb_words, EMBEDDING_DIM))
for word, i in word_index.items():
    if word in word2vec.vocab:
        embedding_matrix[i] = word2vec.word_vec(word)
print('Null word embeddings: %d' % np.sum(np.sum(embedding_matrix, axis=1) ==
0))
```

```
Preparing embedding matrix
Null word embeddings: 37391
```

In [7]:
```
perm = np.random.permutation(len(data_1))
idx_train = perm[:int(len(data_1)*(1-VALIDATION_SPLIT))]
idx_val = perm[int(len(data_1)*(1-VALIDATION_SPLIT)):]

data_1_train = np.vstack((data_1[idx_train], data_2[idx_train]))
data_2_train = np.vstack((data_2[idx_train], data_1[idx_train]))
labels_train = np.concatenate((labels[idx_train], labels[idx_train]))

data_1_val = np.vstack((data_1[idx_val], data_2[idx_val]))
data_2_val = np.vstack((data_2[idx_val], data_1[idx_val]))
labels_val = np.concatenate((labels[idx_val], labels[idx_val]))

weight_val = np.ones(len(labels_val))
if re_weight:
    weight_val *= 0.472001959
    weight_val[labels_val==0] = 1.309028344
```

In [15]:
```
train_orig =  pd.read_csv(TRAIN_DATA_FILE, header=0)
df1 = train_orig[['question1']].copy()
df2 = train_orig[['question2']].copy()
df2.rename(columns = {'question2':'question1'},inplace=True)

train_questions = df1.append(df2)
train_questions.drop_duplicates(subset = ['question1'],inplace=True)

train_questions.reset_index(inplace=True,drop=True)
questions_dict = pd.Series(train_questions.index.values,index=train_questions.
question1.values).to_dict()
train_cp = train_orig.copy()
train_cp.drop(['qid1','qid2'],axis=1,inplace=True)
train_cp['q1_hash'] = train_cp['question1'].map(questions_dict)
train_cp['q2_hash'] = train_cp['question2'].map(questions_dict)
q1_vc = train_cp.q1_hash.value_counts().to_dict()
q2_vc = train_cp.q2_hash.value_counts().to_dict()
def try_apply_dict(x,dict_to_apply):
    try:
        return dict_to_apply[x]
    except KeyError:
        return 0
#map to frequency space
train_cp['q1_freq'] = train_cp['q1_hash'].map(lambda x: try_apply_dict(x,q1_vc
) + try_apply_dict(x,q2_vc))
train_cp['q2_freq'] = train_cp['q2_hash'].map(lambda x: try_apply_dict(x,q1_vc
) + try_apply_dict(x,q2_vc))
train_comb = train_cp[train_cp['is_duplicate'] >= 0][['id','q1_hash','q2_hash'
,'q1_freq','q2_freq','is_duplicate']]
corr_mat = train_comb.corr()
# corr_mat.head()
print(train_comb.shape)
```

```
(404290, 6)
```

In [8]:
```
def model_conv1D_(emb_matrix):

    # The embedding layer containing the word vectors
    emb_layer = Embedding(
        input_dim=emb_matrix.shape[0],
        output_dim=emb_matrix.shape[1],
        weights=[emb_matrix],
        input_length=30,
        trainable=False
    )

    # 1D convolutions that can iterate over the word vectors
    conv1 = Conv1D(filters=128, kernel_size=1, padding='same', activation=
'relu')
    conv2 = Conv1D(filters=128, kernel_size=2, padding='same', activation=
'relu')
    conv3 = Conv1D(filters=128, kernel_size=3, padding='same', activation=
'relu')
    conv4 = Conv1D(filters=128, kernel_size=4, padding='same', activation=
'relu')
    conv5 = Conv1D(filters=32, kernel_size=5, padding='same', activation='r
```

```
elu')
    conv6 = Conv1D(filters=32, kernel_size=6, padding='same', activation='r
elu')

    # Define inputs
    seq1 = Input(shape=(30,))
    seq2 = Input(shape=(30,))

    # Run inputs through embedding
    emb1 = emb_layer(seq1)
    emb2 = emb_layer(seq2)

    # Run through CONV + GAP layers
    conv1a = conv1(emb1)
    glob1a = GlobalAveragePooling1D()(conv1a)
    conv1b = conv1(emb2)
    glob1b = GlobalAveragePooling1D()(conv1b)

    conv2a = conv2(emb1)
    glob2a = GlobalAveragePooling1D()(conv2a)
    conv2b = conv2(emb2)
    glob2b = GlobalAveragePooling1D()(conv2b)

    conv3a = conv3(emb1)
    glob3a = GlobalAveragePooling1D()(conv3a)
    conv3b = conv3(emb2)
    glob3b = GlobalAveragePooling1D()(conv3b)

    conv4a = conv4(emb1)
    glob4a = GlobalAveragePooling1D()(conv4a)
    conv4b = conv4(emb2)
    glob4b = GlobalAveragePooling1D()(conv4b)

    conv5a = conv5(emb1)
    glob5a = GlobalAveragePooling1D()(conv5a)
    conv5b = conv5(emb2)
    glob5b = GlobalAveragePooling1D()(conv5b)

    conv6a = conv6(emb1)
    glob6a = GlobalAveragePooling1D()(conv6a)
    conv6b = conv6(emb2)
    glob6b = GlobalAveragePooling1D()(conv6b)

    mergea = concatenate([glob1a, glob2a, glob3a, glob4a, glob5a, glob6a])
    mergeb = concatenate([glob1b, glob2b, glob3b, glob4b, glob5b, glob6b])

    # We take the explicit absolute difference between the two sentences
    # Furthermore we take the multiply different entries to get a different
 measure of equalness
    diff = Lambda(lambda x: KB.abs(x[0] - x[1]), output_shape=(4 * 128 + 2*
32,))([mergea, mergeb])
    mul = Lambda(lambda x: x[0] * x[1], output_shape=(4 * 128 + 2*32,))([me
rgea, mergeb])

#     # Add the magic features
#     magic_input = Input(shape=(5,))
#     magic_dense = BatchNormalization()(magic_input)
```

```
#     magic_dense = Dense(64, activation='relu')(magic_dense)

#     # Add the distance features (these are now TFIDF (character and wor
d), Fuzzy matching,
#     # nb char 1 and 2, word mover distance and skew/kurtosis of the sente
nce vector)
#     distance_input = Input(shape=(20,))
#     distance_dense = BatchNormalization()(distance_input)
#     distance_dense = Dense(128, activation='relu')(distance_dense)

    # Merge the Magic and distance features with the difference layer
    merge = concatenate([diff, mul])

    # The MLP that determines the outcome
    x = Dropout(0.2)(merge)
    x = BatchNormalization()(x)
    x = Dense(300, activation='relu')(x)

    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)
    pred = Dense(1, activation='sigmoid')(x)

    # model = Model(inputs=[seq1, seq2, magic_input, distance_input], outpu
ts=pred)
    model = Model(inputs=[seq1, seq2], outputs=pred)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['a
cc'])

    return model
```

In [9]:
```
model = model_conv1D_(embedding_matrix)
```

In [10]:
```
if re_weight:
    class_weight = {0: 1.309028344, 1: 0.472001959}
else:
    class_weight = None
```

In [11]:
```
print(STAMP)

early_stopping =EarlyStopping(monitor='val_loss', patience=5)
bst_model_path = STAMP + '.h5'
model_checkpoint = ModelCheckpoint(bst_model_path, save_best_only=True, save_w
eights_only=True)

hist = model.fit([data_1_train, data_2_train], labels_train, \
        validation_data=([data_1_val, data_2_val], labels_val, weight_val), \
        epochs=50, batch_size=2048, shuffle=True, \
        class_weight=class_weight, callbacks=[model_checkpoint])

model.load_weights(bst_model_path)
bst_val_score = min(hist.history['val_loss'])
```

```
lstm_baseline_cnn_261_139_0.16_0.29
Train on 566006 samples, validate on 242574 samples
Epoch 1/50
566006/566006 [==============================] - 65s 114us/step - loss: 0.3
748 - acc: 0.7434 - val_loss: 0.2880 - val_acc: 0.7641
Epoch 2/50
566006/566006 [==============================] - 61s 109us/step - loss: 0.2
519 - acc: 0.8186 - val_loss: 0.2754 - val_acc: 0.7868
Epoch 3/50
566006/566006 [==============================] - 61s 109us/step - loss: 0.2
068 - acc: 0.8575 - val_loss: 0.2778 - val_acc: 0.8336
Epoch 4/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.1
741 - acc: 0.8839 - val_loss: 0.2732 - val_acc: 0.8223
Epoch 5/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.1
496 - acc: 0.9023 - val_loss: 0.2884 - val_acc: 0.8438
Epoch 6/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.1
314 - acc: 0.9158 - val_loss: 0.2810 - val_acc: 0.8334
Epoch 7/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.1
164 - acc: 0.9263 - val_loss: 0.3087 - val_acc: 0.8448
Epoch 8/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.1
068 - acc: 0.9332 - val_loss: 0.3102 - val_acc: 0.8392
Epoch 9/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
970 - acc: 0.9399 - val_loss: 0.3247 - val_acc: 0.8440
Epoch 10/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
893 - acc: 0.9455 - val_loss: 0.3394 - val_acc: 0.8486
Epoch 11/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
835 - acc: 0.9494 - val_loss: 0.3352 - val_acc: 0.8469
Epoch 12/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
787 - acc: 0.9525 - val_loss: 0.3311 - val_acc: 0.8471
Epoch 13/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
731 - acc: 0.9562 - val_loss: 0.3357 - val_acc: 0.8459
Epoch 14/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
702 - acc: 0.9585 - val_loss: 0.3543 - val_acc: 0.8490
Epoch 15/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
668 - acc: 0.9608 - val_loss: 0.3578 - val_acc: 0.8516
Epoch 16/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
638 - acc: 0.9622 - val_loss: 0.3695 - val_acc: 0.8551
Epoch 17/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
618 - acc: 0.9638 - val_loss: 0.3592 - val_acc: 0.8516
Epoch 18/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
594 - acc: 0.9652 - val_loss: 0.3749 - val_acc: 0.8513
Epoch 19/50
```

```
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
570 - acc: 0.9668 - val_loss: 0.3998 - val_acc: 0.8574
Epoch 20/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
551 - acc: 0.9680 - val_loss: 0.3865 - val_acc: 0.8575
Epoch 21/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
528 - acc: 0.9694 - val_loss: 0.3755 - val_acc: 0.8533
Epoch 22/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
520 - acc: 0.9700 - val_loss: 0.3740 - val_acc: 0.8523
Epoch 23/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
503 - acc: 0.9712 - val_loss: 0.3929 - val_acc: 0.8554
Epoch 24/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
492 - acc: 0.9719 - val_loss: 0.4165 - val_acc: 0.8599
Epoch 25/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
473 - acc: 0.9732 - val_loss: 0.4167 - val_acc: 0.8588
Epoch 26/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
463 - acc: 0.9737 - val_loss: 0.3932 - val_acc: 0.8574
Epoch 27/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
454 - acc: 0.9742 - val_loss: 0.4018 - val_acc: 0.8546
Epoch 28/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
447 - acc: 0.9749 - val_loss: 0.4403 - val_acc: 0.8594
Epoch 29/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
431 - acc: 0.9757 - val_loss: 0.3843 - val_acc: 0.8541
Epoch 30/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
427 - acc: 0.9761 - val_loss: 0.4344 - val_acc: 0.8595
Epoch 31/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
409 - acc: 0.9766 - val_loss: 0.4272 - val_acc: 0.8599
Epoch 32/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
414 - acc: 0.9770 - val_loss: 0.3977 - val_acc: 0.8539
Epoch 33/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
401 - acc: 0.9776 - val_loss: 0.3888 - val_acc: 0.8524
Epoch 34/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
395 - acc: 0.9781 - val_loss: 0.4109 - val_acc: 0.8594
Epoch 35/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
371 - acc: 0.9794 - val_loss: 0.4387 - val_acc: 0.8589
Epoch 36/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
378 - acc: 0.9790 - val_loss: 0.4105 - val_acc: 0.8566
Epoch 37/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
373 - acc: 0.9794 - val_loss: 0.4149 - val_acc: 0.8592
Epoch 38/50
```

```
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
366 - acc: 0.9796 - val_loss: 0.4279 - val_acc: 0.8591
Epoch 39/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
359 - acc: 0.9803 - val_loss: 0.4127 - val_acc: 0.8575
Epoch 40/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
344 - acc: 0.9809 - val_loss: 0.4482 - val_acc: 0.8615
Epoch 41/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
348 - acc: 0.9809 - val_loss: 0.4303 - val_acc: 0.8592
Epoch 42/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
342 - acc: 0.9812 - val_loss: 0.4335 - val_acc: 0.8609
Epoch 43/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
345 - acc: 0.9811 - val_loss: 0.4249 - val_acc: 0.8608
Epoch 44/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
333 - acc: 0.9817 - val_loss: 0.4159 - val_acc: 0.8600
Epoch 45/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
327 - acc: 0.9821 - val_loss: 0.4467 - val_acc: 0.8577
Epoch 46/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
328 - acc: 0.9818 - val_loss: 0.4379 - val_acc: 0.8596
Epoch 47/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
309 - acc: 0.9831 - val_loss: 0.4229 - val_acc: 0.8589
Epoch 48/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
308 - acc: 0.9831 - val_loss: 0.4246 - val_acc: 0.8597
Epoch 49/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
310 - acc: 0.9830 - val_loss: 0.4316 - val_acc: 0.8583
Epoch 50/50
566006/566006 [==============================] - 62s 109us/step - loss: 0.0
307 - acc: 0.9832 - val_loss: 0.4402 - val_acc: 0.8595
```