Experiment - 5

Convolutional Neural networks for Text Classification (Word2Vec generated features)

- Keras Example (https://github.com/fchollet/keras/tree/master/examples)
- CNN for Sentence Classification in Keras (https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras)
- Time series example (http://machinelearningmastery.com/time-series-prediction-with-deep-learning-in-python-with-keras/)
- Good article on optimizing gradient descent (http://sebastianruder.com/optimizing-gradient-descent/index.html)

In [26]:
```
import pandas as pd
```

In [27]:
```
# Read training and testing data
train = pd.read_csv('data/train.csv') # category, text
test = pd.read_csv('data/test.csv') # category, text

# Replace NaN with ''
train = train.fillna('')
test = test.fillna('')

# Shapes
train_n = train.shape[0]
test_n = test.shape[0]
print train_n + test_n
```

```
17647
```

In [28]:
```
# Concatenate training and testind data
df = pd.concat([train, test])
print df.shape
```

```
(17647, 2)
```

In [29]:
```
# Data: X and y
# Label encoding: y
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(df['category'])
print 'Number of classes: ', len(list(le.classes_))
y_c = le.transform(df['category'])

# Label binarizer
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
lb.fit(df['category'])
y = lb.transform(df['category'])
print y.shape

X = list(df['text'].values)
```

```
Number of classes:  17
(17647, 17)
```

In [30]:
```
import numpy as np
import re
import itertools
from collections import Counter

# Function to clean text - Source: https://github.com/dennybritz/cnn-te
def clean_str(string):
    """
    Tokenization/string cleaning for all datasets except for SST.
    Original taken from https://github.com/yoonkim/CNN_sentence/blob/ma
    """
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()

# Function to pad texts
def pad_sentences(sentences, padding_word="<PAD/>"):
    """
    Pads all sentences to the same length. The length is defined by the
    Returns padded sentences.
    """
    sequence_length = max(len(x) for x in sentences)
    padded_sentences = []
    for i in range(len(sentences)):
        sentence = sentences[i]
        num_padding = sequence_length - len(sentence)
        new_sentence = sentence + [padding_word] * num_padding
        padded_sentences.append(new_sentence)
    return padded_sentences
```

In [31]:
```
# Data preparation

# Remove leading characters
X_strip = [s.strip() for s in X]

# Clean strings
X_clean = [clean_str(s) for s in X_strip]

# Create list of lists
X_list = [s.split(" ") for s in X_clean]

# Pad text
X_pad = pad_sentences(X_list)

# Build vocabulary
word_counts = Counter(itertools.chain(*X_pad))

# Mapping from index to word
vocabulary_inv = [w[0] for w in word_counts.most_common()]

# Mapping from word to index
vocabulary = {w: i for i, w in enumerate(vocabulary_inv)}

# X data
X_data = np.array([[vocabulary[word] for word in sentence] for sentence
```

In [32]:
```
# Create testing set and training set
mask = range(train_n, train_n + test_n)
X_test = X_data[mask]
y_test = y_c[mask]
print X_test.shape, y_test.shape

mask = range(train_n)
X_train = X_data[mask]
y_train = y_c[mask]
print X_train.shape, y_train.shape
```

```
(3599, 66) (3599,)
(14048, 66) (14048,)
```

Word2Vec

In [33]:
```
# Multiprocessing
from multiprocessing import cpu_count

# word2vec
from gensim.models import word2vec
```

In [51]:
```
# Model:
#       size = 100 as per http://arxiv.org/pdf/1408.5882v2.pdf
#       window = 5 max distance between the current and predicted word
#       min_count` = 1 (ignore all words with total frequency lower tha

# Initiate model
num_features = 300
downsampling = 1e-3   # Downsample setting for frequent words

# Create sentence matrix
X_train_sent = [[vocabulary_inv[w] for w in s] for s in X_train]

embedding_model = word2vec.Word2Vec(X_train_sent, size=num_features, wi
                           min_count=1, sample=downsampling, workers=cp
```

In [52]:
```
# Embedding weights
embedding_weights = [np.array([embedding_model[w] if w in embedding_moc
                                                  else np.random.
                                                  for w in vocabu
```

CNN using keras

In [53]:
```
from keras.models import Sequential
from keras.models import Model
from keras.layers import Activation, Dense, Dropout, Embedding, Flatten
import keras

np.random.seed(1507)
```

In [54]:
```
# Model hyperparameters
sequence_length = 66 # length of paded sentence
embedding_dim = num_features
filter_sizes = (3, 4)
num_filters = 150
dropout_prob = (0.25, 0.5)
hidden_dims = 150

# Training parameters
batch_size = 32
num_epochs = 3 # test on 3 epochs
val_split = 0.1
```

In [55]:
```
# Shuffle data
shuffle_indices = np.random.permutation(np.arange(len(y_train)))
x_shuffled = X_train[shuffle_indices]
y_shuffled = y_train[shuffle_indices]
```

In [56]:
```
print x_shuffled.shape
print y_shuffled.shape
```

```
(14048, 66)
(14048,)
```

In [57]:
```
# graph subnet with one input and one output,
# convolutional layers concateneted in parallel
graph_in = Input(shape=(sequence_length, embedding_dim))
convs = []
for fsz in filter_sizes:
    conv = Convolution1D(nb_filter=num_filters,
                         filter_length=fsz,
                         border_mode='valid',
                         activation='relu',
                         subsample_length=1)(graph_in)
    pool = MaxPooling1D(pool_length=2)(conv)
    flatten = Flatten()(pool)
    convs.append(flatten)

if len(filter_sizes)>1:
    out = Merge(mode='concat')(convs)
else:
    out = convs[0]

graph = Model(input=graph_in, output=out)
```

In [43]:
```
# main sequential model
model = Sequential()
model.add(Embedding(len(vocabulary), embedding_dim, input_length=sequen
                    weights=embedding_weights))

model.add(Dropout(dropout_prob[0], input_shape=(sequence_length, embedd
model.add(graph)
model.add(Dense(hidden_dims))
model.add(Dropout(dropout_prob[1]))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='rmsprc

# Training model: ReLU -> Sigmoid
model.fit(x_shuffled,
          y_shuffled,
          batch_size=batch_size,
          nb_epoch=num_epochs,
          validation_split=val_split, verbose=2)
```

```
Train on 12643 samples, validate on 1405 samples
Epoch 1/3
20s - loss: nan - acc: 0.0000e+00 - val_loss: nan - val_acc: 0.0000
e+00
Epoch 2/3
22s - loss: nan - acc: 0.0000e+00 - val_loss: nan - val_acc: 0.0000
e+00
Epoch 3/3
21s - loss: nan - acc: 0.0000e+00 - val_loss: nan - val_acc: 0.0000
e+00
```

Out[43]:
```
<keras.callbacks.History at 0x11cf69810>
```

In [58]:
```
# main sequential model
keras.optimizers.SGD(lr=0.1, momentum=0.75, decay=0.0, nesterov=False)

model = Sequential()
model.add(Embedding(len(vocabulary), embedding_dim, input_length=sequen
                    weights=embedding_weights))

model.add(Dropout(dropout_prob[0], input_shape=(sequence_length, embedd
model.add(graph)
model.add(Dense(hidden_dims))
model.add(Dropout(dropout_prob[1]))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('tanh'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd',


# Training model: ReLU -> tanh
model.fit(x_shuffled,
          y_shuffled,
          batch_size=batch_size,
          nb_epoch=num_epochs,
          validation_split=val_split, verbose=2)
```

```
Train on 12643 samples, validate on 1405 samples
Epoch 1/3
63s - loss: nan - acc: 7.9095e-05 - val_loss: nan - val_acc: 0.0000
e+00
Epoch 2/3
65s - loss: nan - acc: 0.0000e+00 - val_loss: nan - val_acc: 0.0000
e+00
Epoch 3/3
62s - loss: nan - acc: 0.0000e+00 - val_loss: nan - val_acc: 0.0000
e+00
```

Out[58]:
```
<keras.callbacks.History at 0x13b5cd610>
```

In [59]:
```
# main sequential model
keras.optimizers.SGD(lr=0.1, momentum=0.75, decay=0.0, nesterov=False)

model = Sequential()
model.add(Embedding(len(vocabulary), embedding_dim, input_length=sequen
                    weights=embedding_weights))

model.add(Dropout(dropout_prob[0], input_shape=(sequence_length, embedd
model.add(graph)
model.add(Dense(hidden_dims))
model.add(Dropout(dropout_prob[1]))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('softmax'))
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['acc

# Training model: ReLU -> softmax
model.fit(x_shuffled,
          y_shuffled,
          batch_size=batch_size,
          nb_epoch=num_epochs,
          validation_split=val_split, verbose=2)
```

```
Train on 12643 samples, validate on 1405 samples
Epoch 1/3
63s - loss: 59.1601 - acc: 0.0033 - val_loss: 59.2363 - val_acc: 0.
0050
Epoch 2/3
60s - loss: 59.1601 - acc: 0.0033 - val_loss: 59.2363 - val_acc: 0.
0050
Epoch 3/3
56s - loss: 59.1601 - acc: 0.0033 - val_loss: 59.2363 - val_acc: 0.
0050
```

Out[59]:
```
<keras.callbacks.History at 0x145268850>
```

In [50]:
```
score = model.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=2)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
('Test score:', 59.793553770234368)
('Test accuracy:', 0.0030564045568213394)
```

In [ ]: