Text Classification Using CNN, LSTM and visualize Word Embeddings: Part-2

Sabber Ahamed Follow
Jan 11, 2018 · 3 min read

This is a part of series articles on classifying Yelp review comments using deep learning techniques and word embeddings. In the last part (part-1) of this series, I have shown how we can get word embeddings and classify comments based on LSTM. In this part, I use one CNN layer on top of the LSTM for faster training time.

Followings are the list of brief contents of different part :

- **Part-1**: In this part, I build a neural network with LSTM and word embeddings were leaned while fitting the neural network on the classification problem.
- **Part-2**: In in this part, I add an extra 1D convolutional layer on top of LSTM layer to reduce the training time.
- **Part-3**: In this part-3, I use the same network architecture as part-2, but use the pre-trained glove 100 dimension word embeddings as initial input.
- **Part-4**: In part-4, I use word2vec to learn word embeddings.

Import Libraries

```
# Keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D,
Dropout, Activation
from keras.layers.embeddings import Embedding

## Plotly
import plotly.offline as py
```

```
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)

# Others
import nltk
import string
import numpy as np
import pandas as pd
from nltk.corpus import stopwords

from sklearn.manifold import TSNE
```

Data processing

Data processing involves the following steps:

1. Remove numeric and empty texts
2. Convert five classes into two classes (positive = 1 and negative = 0)
3. Remove punctuation from texts
4. Convert words to lower case
5. Remove stop words
6. Stemming

Detail code of the data processing can be found on Part-1.

Build neural network with LSTM and CNN

The LSTM model worked well. However, it takes forever to train three epochs. One way to speed up the training time is to improve the network adding “Convolutional” layer. Convolutional Neural Networks (CNN) come from image processing. They pass a “filter” over the data and calculate a higher-level representation. They have been shown to work surprisingly well for text, even though they have none of the sequence processing ability of LSTMs.

```
def create_conv_model():
    model_conv = Sequential()
    model_conv.add(Embedding(vocabulary_size, 100, input_length=50))
```

```
    model_conv.add(Dropout(0.2))
    model_conv.add(Conv1D(64, 5, activation='relu'))
    model_conv.add(MaxPooling1D(pool_size=4))
    model_conv.add(LSTM(100))
    model_conv.add(Dense(1, activation='sigmoid'))
    model_conv.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model_conv

model_conv = create_conv_model()
model_conv.fit(data, np.array(labels), validation_split=0.4, epochs =
3)

### Training output

Train on 1004322 samples, validate on 669548 samples
Epoch 1/3
1004322/1004322 [==============================] - 20429s - loss:
0.2981 - acc: 0.8711 - val_loss: 0.2786 - val_acc: 0.8814
Epoch 2/3
1004322/1004322 [==============================] - 3363s - loss:
0.2657 - acc: 0.8871 - val_loss: 0.2774 - val_acc: 0.8820
Epoch 3/3
1004322/1004322 [==============================] - 11838s - loss:
0.2495 - acc: 0.8954 - val_loss: 0.2768 - val_acc: 0.8841
```

Word embedding visualization

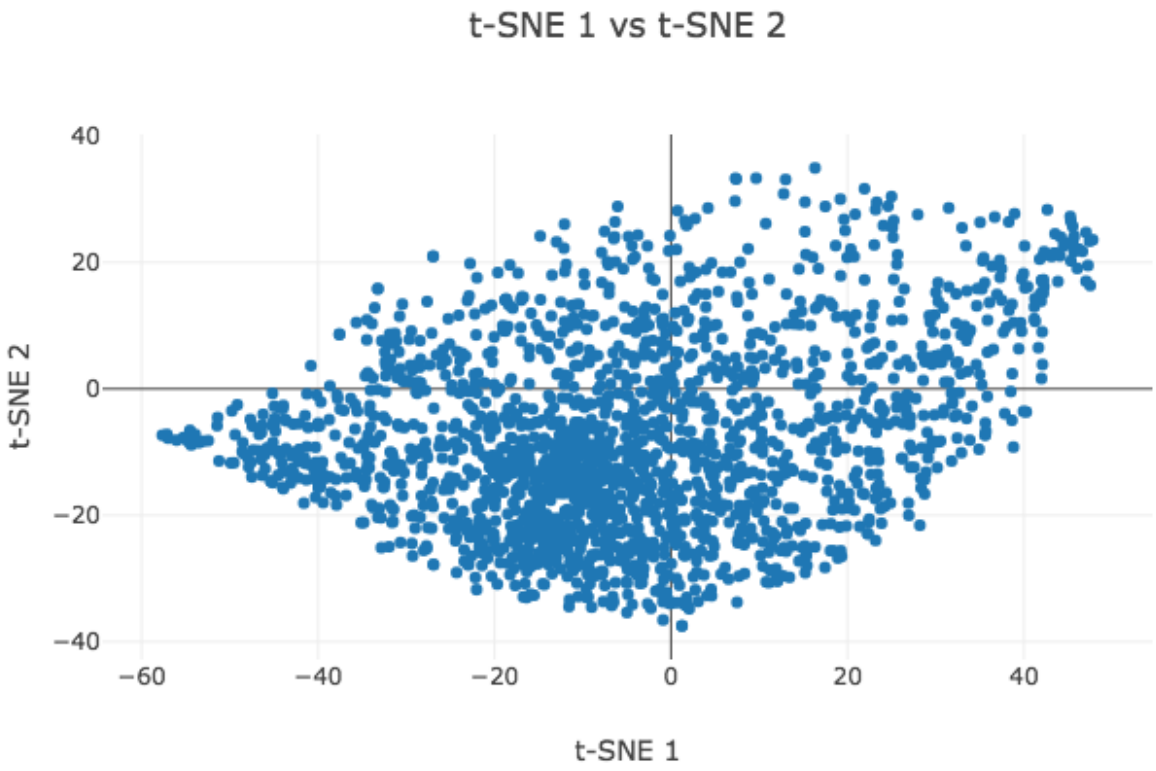
In this subsection, I want to visualize word embedding weights obtained from trained models. Word embeddings with 100 dimensions are first reduced to 2 dimensions using t-SNE. Tensorflow has an excellent tool to visualize the embeddings nicely, but here I just want to visualize the word relationship.

```
## Get weights
conv_embds = model_conv.layers[0].get_weights()[0]

## Plotting function
def plot_words(data, start, stop, step):
    trace = go.Scatter(
        x = data[start:stop:step,0],
        y = data[start:stop:step, 1],
        mode = 'markers',
        text= word_list[start:stop:step]
    )
    layout = dict(title= 't-SNE 1 vs t-SNE 2',
                  yaxis = dict(title='t-SNE 2'),
                  xaxis = dict(title='t-SNE 1'),
```

```
                    hovermode= 'closest')
        fig = dict(data = [trace], layout= layout)
        py.iplot(fig)

## Visualize words in two dimensions
conv_tsne_embds = TSNE(n_components=2).fit_transform(conv_embds)
plot_words(conv_tsne_embds, 0, 2000, 1)
```



Thank you very much for reading. The full code can be found on Github. Let me know if you have any question, or this article needs any correction. In the next part, I will be discussing how we can use pre-trained Glove word embeddings and use them for classification.

Convolution Neural Net Machine Learning Word Embeddings NLP Data Science