Hw-2: Task 2

Taslima Akter

ID: takter

In [1]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

# Imports
import numpy as np
import tensorflow as tf

import os
# os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"   # see issue #152
# os.environ["CUDA_VISIBLE_DEVICES"]="0"
import tensorflow as tf
import numpy as np
import time
import matplotlib.pyplot as plt
from operator import itemgetter
from sklearn.datasets import fetch_mldata

from keras.models import Model
from keras.layers import Input, LSTM, GRU, Dense, Dropout, Bidirectional
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.np_utils import to_categorical

%matplotlib notebook
```

```
Using TensorFlow backend.
```

In [2]:
```
config = tf.ConfigProto()
# config.gpu_options.allow_growth = True
# config.gpu_options.per_process_gpu_memory_fraction = 0.33
```

Read file names for training data

In [3]:
```
### File names for Training Data

import glob
import librosa
fname_trn=[]
for filename in glob.glob('/opt/e533/timit-homework/tr/trn*.wav'):
    (fname_trn.append(filename))
fname_trn.sort()
# fname_trn

fname_trs=[]
for filename in glob.glob('/opt/e533/timit-homework/tr/trs*.wav'):
    (fname_trs.append(filename))
fname_trs.sort()
# fname_trs

fname_trx=[]
for filename in glob.glob('/opt/e533/timit-homework/tr/trx*.wav'):
    (fname_trx.append(filename))
fname_trx.sort()
fname_trx[100]

```

Out[3]: '/opt/e533/timit-homework/tr/trx0100.wav'

Read file names for validation data

In [4]:
```
### File names for validation Data

fname_val_n=[]
for filename in glob.glob('/opt/e533/timit-homework/v/vn*.wav'):
    (fname_val_n.append(filename))
fname_val_n.sort()
# print(fname_val_n)

fname_val_s=[]
for filename in glob.glob('/opt/e533/timit-homework/v/vs*.wav'):
    (fname_val_s.append(filename))
fname_val_s.sort()
# fname_trs

fname_val_x=[]
for filename in glob.glob('/opt/e533/timit-homework/v/vx*.wav'):
    (fname_val_x.append(filename))
fname_val_x.sort()

```

Read file names for test data

In [5]:
```
### File names for Test Data

fname_test=[]
for filename in glob.glob('/opt/e533/timit-homework/te/tex*.wav'):
    (fname_test.append(filename))
fname_test.sort()

```

Write data into txt file

In [6]:
```
### Function For writing file
import librosa
def write_file(file_name, fname_list):
    ### Writing training data S

    with open(file_name, 'wb') as fs:
        for i in range(len(fname_list)):
            sn, sr=librosa.load(fname_list[i], sr=None)
            Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
            mag_Sn=np.abs(Sn)
    #         trn_arr=np.concatenate((trn_arr, mag_Sn), axis=1)
            np.savetxt(fs, mag_Sn, fmt='%.5f')
            fs.write(b'\n')
    fs.close()


```

Writing complex data X into file

In [7]:
```
### Function For writing just X into file
import librosa
def write_file_X(file_name, fname_list):
    ### Writing training data S

    with open(file_name, 'wb') as fs:
        for i in range(len(fname_list)):
            sn, sr=librosa.load(fname_list[i], sr=None)
            Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
#             mag_Sn=np.abs(Sn)
    #         trn_arr=np.concatenate((trn_arr, mag_Sn), axis=1)
            np.savetxt(fs, Sn, fmt='%.5f')
            fs.write(b'\n')
    fs.close()


```

In [8]:
```
### Writing training data S
Train_complx_X=[]

def read_complex_data(fname_trx):
    Train_complx_X=[]
    for i in range(len(fname_trx)):
        print(i),

        sn, sr=librosa.load(fname_trx[i], sr=None)
        Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
        Train_complx_X.append(np.array(Sn))
    return Train_complx_X


#             mag_Sn=np.abs(Sn)
#         trn_arr=np.concatenate((trn_arr, mag_Sn), axis=1)


```

In [9]:
```
Train_complx_X=read_complex_data(fname_trx)
```

```
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
```

In [11]:
```
val_complx_X=read_complex_data(fname_val_x)

```

```
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
```

In [12]:
```
val_complx_S=read_complex_data(fname_val_s)

```

```
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
```

In [13]:
```
test_complx_S=read_complex_data(fname_test)

```

```
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
```

Calling functions to write data

In [ ]:
```
write_file("train_s.txt", fname_trs)
write_file("train_n.txt", fname_trn)
write_file("train_x.txt", fname_trx)
```

In [6]:
```
### Write validation files

write_file("validation_s.txt", fname_val_s)
write_file("validation_n.txt", fname_val_n)
write_file("validation_x.txt", fname_val_x)

```

In [8]:
```
write_file_X("train_x_tr.txt", fname_trx)
write_file_X("validation_x_tr.txt", fname_trx)

```

In [8]:
```
write_file("test_data.txt", fname_test)
```

In [9]:
```
### Writing training data N
import librosa

count=0
total_train_s=[]
with open('train_n.txt', 'wb') as fn:
    for i in range(len(fname_trn)):
        sn, sr=librosa.load(fname_trn[i], sr=None)
        Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
        mag_Sn=np.abs(Sn)
#         trn_arr=np.concatenate((trn_arr, mag_Sn), axis=1)
        np.savetxt(fn, mag_Sn, fmt='%.5f')
        fn.write(b'\n')
fn.close()
```

In [10]:
```
### Writing training data S

with open('train_s.txt', 'wb') as fs:
    for i in range(len(fname_trs)):
        sn, sr=librosa.load(fname_trs[i], sr=None)
        Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
        mag_Sn=np.abs(Sn)
#         trn_arr=np.concatenate((trn_arr, mag_Sn), axis=1)
        np.savetxt(fs, mag_Sn, fmt='%.5f')
        fs.write(b'\n')
fs.close()

```

In [11]:
```
### Writing training file X

with open('train_x.txt', 'wb') as fs:
    for i in range(len(fname_trx)):
        sn, sr=librosa.load(fname_trx[i], sr=None)
        Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
        mag_Sn=np.abs(Sn)
#         trn_arr=np.concatenate((trn_arr, mag_Sn), axis=1)
        np.savetxt(fs, mag_Sn, fmt='%.5f')
        fs.write(b'\n')
fs.close()

```

Read data from files

In [18]:
```
### Function for Reading file

def read_file(file_name):
    with open(file_name) as f:
        lines=f.readlines()
        print(len(lines))
        sentence_full=[]
        count = 0
        sentence=[]
        for line in lines:

            if count < 513:
                if count ==0:
                    sentence=np.array(np.fromstring(line, dtype=float, s
                    count+=1
                else:
                    myarray = np.array(np.fromstring(line, dtype=float,
                    sentence=np.concatenate((sentence, myarray), axis=0)
                    count+=1
            else:
                sentence_full.append(sentence)
                count=0
                sentence=[]
        return sentence_full

```

Read training data

In [19]:
```
data_train_n = read_file("train_n.txt")
data_train_s = read_file("train_s.txt")
data_train_x = read_file("train_x.txt")

```

```
616800
616800
616800
```

Read validation data

In [20]:
```
data_val_n = read_file("validation_n.txt")
data_val_s = read_file("validation_s.txt")
data_val_x = read_file("validation_x.txt")
```

```
616800
616800
616800
```

Read complex X from training and validation data

In [23]:
```
data_train_xtr = read_file("train_x_tr.txt")
data_val_xtr = read_file("validation_x_tr.txt")
```

```
616800
616800
```

In [29]:
```
len(data_val_xtr)
data_val_xtr[0].shape
```

Out[29]:
```
(513, 1)
```

In [21]:
```
data_test = read_file("test_data.txt")
```

```
205600
```

Calculating M for training and validation data

In [22]:
```
### Calculating M:
data_train_M=[]
data_val_M=[]
for i in range(len(data_train_s)):
    data_train_M.append(1*(data_train_s[i]>data_train_n[i]))
    data_val_M.append(1*(data_val_s[i]>data_val_n[i]))


```

In [64]:
```
data_train_M[2].shape
```

Out[64]:
```
(513, 65)
```

Creating Batch

In [23]:
```
def next_batchXSCmplx(X_, S_, X_cmplx, S_cmplx):

    batch_x = None
    batch_s = None
    batch_x_cmplx = None
    batch_s_cmplx = None

    for e,(x, s, x_cmplx, s_cmplx) in enumerate(zip(X_, S_, X_cmplx, S_c
        batch_x = np.array(x.T) if batch_x is None else np.concatenate(
        batch_s = np.array(s.T) if batch_s is None else np.concatenate(
        batch_x_cmplx = np.array(x_cmplx.T) if batch_x_cmplx is None els
        batch_s_cmplx = np.array(s_cmplx.T) if batch_s_cmplx is None els

        if e>0 and (e+1)%10==0:
            temp_x, batch_x = batch_x, None
            temp_s, batch_s = batch_s, None
            temp_x_cmplx, batch_x_cmplx = batch_x_cmplx, None
            temp_s_cmplx, batch_s_cmplx = batch_s_cmplx, None

            temp_x = temp_x.reshape((-1,Max_RNN,513))
            temp_s = temp_s.reshape((-1,Max_RNN,513))
            temp_x_cmplx = temp_x_cmplx.reshape((-1,Max_RNN,513))
#             temp_s_cmplx = temp_s_cmplx.reshape((-1,Max_RNN,513))

            yield temp_x, temp_s, temp_x_cmplx, temp_s_cmplx
```

In [24]:
```
def next_batch(X_,Y_):

    batch_x, batch_y = None, None

    for e,(x,y) in enumerate(zip(X_,Y_)):
#         print(e)

        batch_x = np.array(x.T) if batch_x is None else np.concatenate(
        batch_y = np.array(y.T) if batch_y is None else np.concatenate(

#         print('batch_x',batch_x.shape)
#         print('batch_y',batch_y.shape)

        if e>0 and (e+1)%10==0:
            temp_x, batch_x = batch_x, None
            temp_y, batch_y = batch_y, None

            temp_x = temp_x.reshape((-1,Max_RNN,513))
            temp_y = temp_y.reshape((-1,Max_RNN,513))

#             print('temp_x',temp_x.shape)
#             print('temp_y',temp_y.shape)

            yield temp_x,temp_y

```

RNN implementation

In [25]:
```
Max_RNN=5
model = Sequential()

model.add(Bidirectional(GRU(Max_RNN, return_sequences=True), input_shape
model.add(Dropout(0.2))
model.add(Bidirectional(GRU(Max_RNN, return_sequences=True)))
# model.add(GRU(output_dim = 513, input_length = 5, input_dim = 513, ret

# model.add(Activation('relu'))
# model.add(TimeDistributed(Dense(513, activation='sigmoid')))
model.add(Dense(513, activation='sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics
print(model.summary())

for e in range(10):
    for (b_x,b_y), (v_x,v_y) in zip(next_batch(data_train_x, data_train_
        model.fit(b_x, b_y, validation_data=(v_x,v_y), shuffle=True, bat

#     model.fit( , epochs=20, steps_per_epoch=700, validation_data=next_
```

```
Epoch 1/1
190/190 [==============================] - 0s 488us/step - loss: 0.6724 -
acc: 0.5777 - val_loss: 0.6817 - val_acc: 0.5427
Train on 186 samples, validate on 186 samples
Epoch 1/1
186/186 [==============================] - 0s 548us/step - loss: 0.6754 -
acc: 0.5757 - val_loss: 0.6755 - val_acc: 0.5804
Train on 218 samples, validate on 218 samples
Epoch 1/1
218/218 [==============================] - 0s 658us/step - loss: 0.6739 -
acc: 0.5831 - val_loss: 0.6751 - val_acc: 0.5993
Train on 202 samples, validate on 202 samples
Epoch 1/1
202/202 [==============================] - 0s 676us/step - loss: 0.6719 -
acc: 0.6133 - val_loss: 0.6718 - val_acc: 0.5977
Train on 148 samples, validate on 148 samples
Epoch 1/1
148/148 [==============================] - 0s 593us/step - loss: 0.6700 -
acc: 0.6251 - val_loss: 0.6761 - val_acc: 0.5945
```

Calculating Accuracy

In [26]:
```
scores = []
for v_x,v_y in next_batch(data_val_x, data_val_M):
    scores.append( model.evaluate(v_x, v_y,verbose=0)[1] )
scores=np.mean(scores)
print("Accuracy: ",  scores*100)
```

```
Accuracy:  74.60236812739446
```

Calculating SNR

In [27]:
```
sum_s = 0.0
sum_s_diff = 0.0

for v_s,v_x,v_x_cmplx,v_s_cmplx in next_batchXSCmplx(data_val_s,data_val

#     print(v_s.shape)
#     print(v_x.shape)
#     print(v_x_cmplx.shape)
#     print('v_s_cmplx',v_s_cmplx.shape)

    mask = model.predict(v_x)
    S_hat = (mask) * v_x_cmplx
    S_hat = S_hat.reshape(-1,513).T
    S = v_s_cmplx.T
#     S=S.reshape(-1,513)

#     print('S.shape',S.shape)
#     print('S_hat.shape',S_hat.shape)

    S_org = librosa.istft(S, hop_length=512)
    S_pred = librosa.istft(S_hat, hop_length=512)

    sum_s += np.sum(S_org*S_org)
    sum_s_diff += np.sum((S_org-S_pred)*(S_org-S_pred))

acc = sum_s/ sum_s_diff
print(acc)

10*np.log10(acc)
```

```
3.631143355294444
```

Out[27]:
```
5.600433949289625
```

Write into audio file

In [29]:

```
def write_audio1(file_name):
    for i in range(len(file_name)):
        audio_fname=file_name[i].replace("/opt/e533/timit-homework/te/",
        print(audio_fname)
        sn, sr=librosa.load(file_name[i], sr=None)
        Sn=librosa.stft(sn, n_fft=1024, hop_length=512)
        mag_Sn=np.abs(Sn)
        print(mag_Sn.shape)
#         mag_Sn=mag_Sn.reshape(-1, 5, 513)
        Stest_hat=model.predict(mag_Sn.reshape(-1, 5, 513))
        Stest_hat=Stest_hat.reshape(-1,513)
        S_hat=(Sn/mag_Sn)*Stest_hat.T

        S_time=librosa.istft(S_hat, hop_length=512)
        audio_fname=audio_fname + "_recons.wav"
        print(audio_fname)
        librosa.output.write_wav(audio_fname, S_time, sr)


```

In [41]:

```
def write_audio(fname_test):
    mags = None
    cmplxs = None
    count=0
    for e, file_x in enumerate(fname_test):
        print(e)

        sn, sr = librosa.load(file_x, sr=None)
        Sn = librosa.stft(sn, n_fft=1024, hop_length=512)
        mag_Sn=np.abs(Sn)

        mags = np.array(mag_Sn.T) if mags is None else np.concatenate( (
        cmplxs = np.array(Sn.T) if cmplxs is None else np.concatenate( (

        if e>0 and (e+1)%10==0:
            temp, mags = mags, None
            temp_cmplx, cmplxs = cmplxs, None

            temp = temp.reshape((-1,Max_RNN,513))
            mask = model.predict(temp)

            mask=mask.reshape(-1,513)
            S_hat = (mask) * temp_cmplx
            S_hat = S_hat.T

            lenght_w = S_hat.shape[1]//10
            print(S_hat.shape[1])
            for clip in range(10):
                start_w = clip*lenght_w
                end_w = (clip+1)*lenght_w

                wav = S_hat[:,start_w:end_w].T
                S_time=librosa.istft(wav, hop_length=512)
#                 fname = PATH_directory+PATH_denoise+ e + "_redoise.wav
#                 audio_fname=file_x.replace("/opt/e533/timit-homework/t
                audio_fname="./data/"+ str(count) + "_recons.wav"
                print(audio_fname)
                count+=1
                librosa.output.write_wav(audio_fname, S_time, sr)

```

In [42]:
```
1 write_audio(fname_test)
```

```
385
386
387
388
389
680
./data/380_recons.wav
./data/381_recons.wav
./data/382_recons.wav
./data/383_recons.wav
./data/384_recons.wav
./data/385_recons.wav
./data/386_recons.wav
./data/387_recons.wav
./data/388_recons.wav
./data/389_recons.wav
390
391
392
393
```

In [ ]:
```
1
```