"""

Defines a class that is used to featurize audio clips, and provide

them to the network for training or testing.

"""

import json

import numpy as np

import random

from python\_speech\_features import mfcc

import librosa

import scipy.io.wavfile as wav

import matplotlib.pyplot as plt

from mpl\_toolkits.axes\_grid1 import make\_axes\_locatable

from utils import calc\_feat\_dim, spectrogram\_from\_file, text\_to\_int\_sequence

from utils import conv\_output\_length

RNG\_SEED = 123

class AudioGenerator():

def \_init\_(self, step=10, window=20, max\_freq=8000, mfcc\_dim=13,

minibatch\_size=20, desc\_file=None, spectrogram=True, max\_duration=10.0,

sort\_by\_duration=False):

"""

Params:

step (int): Step size in milliseconds between windows (for spectrogram ONLY)

window (int): FFT window size in milliseconds (for spectrogram ONLY)

max\_freq (int): Only FFT bins corresponding to frequencies between

[0, max\_freq] are returned (for spectrogram ONLY)

desc\_file (str, optional): Path to a JSON-line file that contains

labels and paths to the audio files. If this is None, then

load metadata right away

"""

self.feat\_dim = calc\_feat\_dim(window, max\_freq)

self.mfcc\_dim = mfcc\_dim

self.feats\_mean = np.zeros((self.feat\_dim,))

self.feats\_std = np.ones((self.feat\_dim,))

self.rng = random.Random(RNG\_SEED)

if desc\_file is not None:

self.load\_metadata\_from\_desc\_file(desc\_file)

self.step = step

self.window = window

self.max\_freq = max\_freq

self.cur\_train\_index = 0

self.cur\_valid\_index = 0

self.cur\_test\_index = 0

self.max\_duration=max\_duration

self.minibatch\_size = minibatch\_size

self.spectrogram = spectrogram

self.sort\_by\_duration = sort\_by\_duration

def get\_batch(self, partition):

""" Obtain a batch of train, validation, or test data

"""

if partition == 'train':

audio\_paths = self.train\_audio\_paths

cur\_index = self.cur\_train\_index

texts = self.train\_texts

elif partition == 'valid':

audio\_paths = self.valid\_audio\_paths

cur\_index = self.cur\_valid\_index

texts = self.valid\_texts

elif partition == 'test':

audio\_paths = self.test\_audio\_paths

cur\_index = self.test\_valid\_index

texts = self.test\_texts

else:

raise Exception("Invalid partition. "

"Must be train/validation")

features = [self.normalize(self.featurize(a)) for a in

audio\_paths[cur\_index:cur\_index+self.minibatch\_size]]

# calculate necessary sizes

max\_length = max([features[i].shape[0]

for i in range(0, self.minibatch\_size)])

max\_string\_length = max([len(texts[cur\_index+i])

for i in range(0, self.minibatch\_size)])

# initialize the arrays

X\_data = np.zeros([self.minibatch\_size, max\_length,

self.feat\_dim\*self.spectrogram + self.mfcc\_dim\*(not self.spectrogram)])

labels = np.ones([self.minibatch\_size, max\_string\_length]) \* 28

input\_length = np.zeros([self.minibatch\_size, 1])

label\_length = np.zeros([self.minibatch\_size, 1])

for i in range(0, self.minibatch\_size):

# calculate X\_data & input\_length

feat = features[i]

input\_length[i] = feat.shape[0]

X\_data[i, :feat.shape[0], :] = feat

# calculate labels & label\_length

label = np.array(text\_to\_int\_sequence(texts[cur\_index+i]))

labels[i, :len(label)] = label

label\_length[i] = len(label)

# return the arrays

outputs = {'ctc': np.zeros([self.minibatch\_size])}

inputs = {'the\_input': X\_data,

'the\_labels': labels,

'input\_length': input\_length,

'label\_length': label\_length

}

return (inputs, outputs)

def shuffle\_data\_by\_partition(self, partition):

""" Shuffle the training or validation data

"""

if partition == 'train':

self.train\_audio\_paths, self.train\_durations, self.train\_texts = shuffle\_data(

self.train\_audio\_paths, self.train\_durations, self.train\_texts)

elif partition == 'valid':

self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts = shuffle\_data(

self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts)

else:

raise Exception("Invalid partition. "

"Must be train/validation")

def sort\_data\_by\_duration(self, partition):

""" Sort the training or validation sets by (increasing) duration

"""

if partition == 'train':

self.train\_audio\_paths, self.train\_durations, self.train\_texts = sort\_data(

self.train\_audio\_paths, self.train\_durations, self.train\_texts)

elif partition == 'valid':

self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts = sort\_data(

self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts)

else:

raise Exception("Invalid partition. "

"Must be train/validation")

def next\_train(self):

""" Obtain a batch of training data

"""

while True:

ret = self.get\_batch('train')

self.cur\_train\_index += self.minibatch\_size

if self.cur\_train\_index >= len(self.train\_texts) - self.minibatch\_size:

self.cur\_train\_index = 0

self.shuffle\_data\_by\_partition('train')

yield ret

def next\_valid(self):

""" Obtain a batch of validation data

"""

while True:

ret = self.get\_batch('valid')

self.cur\_valid\_index += self.minibatch\_size

if self.cur\_valid\_index >= len(self.valid\_texts) - self.minibatch\_size:

self.cur\_valid\_index = 0

self.shuffle\_data\_by\_partition('valid')

yield ret

def next\_test(self):

""" Obtain a batch of test data

"""

while True:

ret = self.get\_batch('test')

self.cur\_test\_index += self.minibatch\_size

if self.cur\_test\_index >= len(self.test\_texts) - self.minibatch\_size:

self.cur\_test\_index = 0

yield ret

def load\_train\_data(self, desc\_file='train\_corpus.json'):

self.load\_metadata\_from\_desc\_file(desc\_file, 'train')

self.fit\_train()

if self.sort\_by\_duration:

self.sort\_data\_by\_duration('train')

def load\_validation\_data(self, desc\_file='valid\_corpus.json'):

self.load\_metadata\_from\_desc\_file(desc\_file, 'validation')

if self.sort\_by\_duration:

self.sort\_data\_by\_duration('valid')

def load\_test\_data(self, desc\_file='test\_corpus.json'):

self.load\_metadata\_from\_desc\_file(desc\_file, 'test')

def load\_metadata\_from\_desc\_file(self, desc\_file, partition):

""" Read metadata from a JSON-line file

(possibly takes long, depending on the filesize)

Params:

desc\_file (str): Path to a JSON-line file that contains labels and

paths to the audio files

partition (str): One of 'train', 'validation' or 'test'

"""

audio\_paths, durations, texts = [], [], []

with open(desc\_file) as json\_line\_file:

for line\_num, json\_line in enumerate(json\_line\_file):

try:

spec = json.loads(json\_line)

if float(spec['duration']) > self.max\_duration:

continue

audio\_paths.append(spec['key'])

durations.append(float(spec['duration']))

texts.append(spec['text'])

except Exception as e:

# Change to (KeyError, ValueError) or

# (KeyError,json.decoder.JSONDecodeError), depending on

# json module version

print('Error reading line #{}: {}'

.format(line\_num, json\_line))

if partition == 'train':

self.train\_audio\_paths = audio\_paths

self.train\_durations = durations

self.train\_texts = texts

elif partition == 'validation':

self.valid\_audio\_paths = audio\_paths

self.valid\_durations = durations

self.valid\_texts = texts

elif partition == 'test':

self.test\_audio\_paths = audio\_paths

self.test\_durations = durations

self.test\_texts = texts

else:

raise Exception("Invalid partition to load metadata. "

"Must be train/validation/test")

def fit\_train(self, k\_samples=100):

""" Estimate the mean and std of the features from the training set

Params:

k\_samples (int): Use this number of samples for estimation

"""

k\_samples = min(k\_samples, len(self.train\_audio\_paths))

samples = self.rng.sample(self.train\_audio\_paths, k\_samples)

feats = [self.featurize(s) for s in samples]

feats = np.vstack(feats)

self.feats\_mean = np.mean(feats, axis=0)

self.feats\_std = np.std(feats, axis=0)

def featurize(self, audio\_clip):

""" For a given audio clip, calculate the corresponding feature

Params:

audio\_clip (str): Path to the audio clip

"""

if self.spectrogram:

return spectrogram\_from\_file(

audio\_clip, step=self.step, window=self.window,

max\_freq=self.max\_freq)

else:

(rate, sig) = wav.read(audio\_clip)

return mfcc(sig, rate, numcep=self.mfcc\_dim)

def normalize(self, feature, eps=1e-14):

""" Center a feature using the mean and std

Params:

feature (numpy.ndarray): Feature to normalize

"""

return (feature - self.feats\_mean) / (self.feats\_std + eps)

def shuffle\_data(audio\_paths, durations, texts):

""" Shuffle the data (called after making a complete pass through

training or validation data during the training process)

Params:

audio\_paths (list): Paths to audio clips

durations (list): Durations of utterances for each audio clip

texts (list): Sentences uttered in each audio clip

"""

p = np.random.permutation(len(audio\_paths))

audio\_paths = [audio\_paths[i] for i in p]

durations = [durations[i] for i in p]

texts = [texts[i] for i in p]

return audio\_paths, durations, texts

def sort\_data(audio\_paths, durations, texts):

""" Sort the data by duration

Params:

audio\_paths (list): Paths to audio clips

durations (list): Durations of utterances for each audio clip

texts (list): Sentences uttered in each audio clip

"""

p = np.argsort(durations).tolist()

audio\_paths = [audio\_paths[i] for i in p]

durations = [durations[i] for i in p]

texts = [texts[i] for i in p]

return audio\_paths, durations, texts

def vis\_train\_features(index=62):

""" Visualizing the data point in the training set at the supplied index

"""

# obtain spectrogram

audio\_gen = AudioGenerator(spectrogram=True)

audio\_gen.load\_train\_data()

vis\_audio\_path = audio\_gen.train\_audio\_paths[index]

vis\_spectrogram\_feature = audio\_gen.normalize(audio\_gen.featurize(vis\_audio\_path))

# obtain mfcc

audio\_gen = AudioGenerator(spectrogram=False)

audio\_gen.load\_train\_data()

vis\_mfcc\_feature = audio\_gen.normalize(audio\_gen.featurize(vis\_audio\_path))

# obtain text label

vis\_text = audio\_gen.train\_texts[index]

# obtain raw audio

vis\_raw\_audio, \_ = librosa.load(vis\_audio\_path)

# print total number of training examples

print('There are %d total training examples.' % len(audio\_gen.train\_audio\_paths))

# return labels for plotting

return vis\_text, vis\_raw\_audio, vis\_mfcc\_feature, vis\_spectrogram\_feature, vis\_audio\_path

def plot\_raw\_audio(vis\_raw\_audio):

# plot the raw audio signal

fig = plt.figure(figsize=(12,3))

ax = fig.add\_subplot(111)

steps = len(vis\_raw\_audio)

ax.plot(np.linspace(1, steps, steps), vis\_raw\_audio)

plt.title('Audio Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.show()

def plot\_mfcc\_feature(vis\_mfcc\_feature):

# plot the MFCC feature

fig = plt.figure(figsize=(12,5))

ax = fig.add\_subplot(111)

im = ax.imshow(vis\_mfcc\_feature, cmap=plt.cm.jet, aspect='auto')

plt.title('Normalized MFCC')

plt.ylabel('Time')

plt.xlabel('MFCC Coefficient')

divider = make\_axes\_locatable(ax)

cax = divider.append\_axes("right", size="5%", pad=0.05)

plt.colorbar(im, cax=cax)

ax.set\_xticks(np.arange(0, 13, 2), minor=False);

plt.show()

def plot\_spectrogram\_feature(vis\_spectrogram\_feature):

# plot the normalized spectrogram

fig = plt.figure(figsize=(12,5))

ax = fig.add\_subplot(111)

im = ax.imshow(vis\_spectrogram\_feature, cmap=plt.cm.jet, aspect='auto')

plt.title('Normalized Spectrogram')

plt.ylabel('Time')

plt.xlabel('Frequency')

divider = make\_axes\_locatable(ax)

cax = divider.append\_axes("right", size="5%", pad=0.05)

plt.colorbar(im, cax=cax)

plt.show()

**CODE RUN CHESEDI**

1)from data\_generator import vis\_train\_features

# extract label and audio features for a single training example

vis\_text, vis\_raw\_audio, vis\_mfcc\_feature, vis\_spectrogram\_feature, vis\_audio\_path = vis\_train\_features()

2)from IPython.display import Markdown, display

from data\_generator import vis\_train\_features, plot\_raw\_audio

from IPython.display import Audio

%matplotlib inline

# plot audio signal

plot\_raw\_audio(vis\_raw\_audio)

# print length of audio signal

display(Markdown('\*\*Shape of Audio Signal\*\* : ' + str(vis\_raw\_audio.shape)))

# print transcript corresponding to audio clip

display(Markdown('\*\*Transcript\*\* : ' + str(vis\_text)))

# play the audio file

Audio(vis\_audio\_path)