- WSJ data processing -

부윤호

서울대학교 신호처리시스템 연구실

code를 돌리기 위해 필요한 file들

1) .wav dataset, wav dataset의 list가 저장된 text 파일, wave dataset의 label이 저장된 text 파일, mean, var가 저장된 .npy 파일

2) 전처리가 된 .npy dataset, dataset의 list가 저장된 text 파일, wave dataset의 label이 저장된 text 파일

- 1)은 .wav 파일에서부터 시작해 전처리를 매 iteration마다 수행하는 방법을 사용합니다.

장점 : 전처리 과정의 코드만 바꾸면 매번 다르게 전처리가 된 data를 훈련시킬 수 있다.

단점 : 매 번 전처리를 수행하므로 느리다. (#epoch 배 만큼의 전처리 overhead가 있음)

- 2)는 전처리를 한 번 쭉 수행한 뒤 이를 따로 저장해 두고 iteration마다 load하는 방법을 사용합니다.

장점 : 전처리 overhead가 적다. (1)방법보다 epoch 속도가 2배정도 빠릅니다)

단점 : 전처리 방법이 바뀌면 다시 data를 처리해 저장해둬야 합니다. (WSJ 기준 모든 data 처리에 30분 ~ 1시간 정도 걸립니다)

Data 처리 과정 3 에서 두 가지 경우 모두 처리하는 과정을 보입니다.

data 처리 과정

1. wv1 -> wav type 변환 및 data list, trans 파일 생성

WSJ corpus는 음성 파일로 .wv1 type의 파일을 제공합니다. 이를 python에서 읽기 위해서는 .wav 파일로 변환해야 합니다.

source code : ./data/wav_gen.py, sph2pipe(실행파일)

a) 코드 설명

내부적으로 sph2pipe command를 실행해 list에 있는 .wv1 파일을 해당 이름의 .wav 파일로 변환해주는 코드입니다.

b) 실행 방법

$python wav_gen.py -i <input_path> -o <output_path>

<input_path> : .wv1 파일 path를 가지고 있는 list 파일이 있는 장소가 input path가 됩니다.

(train_all.list, test_dev93.list, test_eval92.list 파일이 존재해야 합니다. 혹시 wsj corpus를 받으셨을 때 해당 파일들이 없으셨다면 알려주세요.)

<output_path> : 저장하고 싶은 위치의 path를 적으시면 됩니다. **default로 ./data_example/** 폴더로 설정했습니다. 이 폴더가 이후 training 과정에서 base path가 됩니다.

c) 실행 결과 생성되는 파일

- train_all_wav.list / train_all_wav.trans #training data 의 파일 위치 및 label
- eval_wav.list / eval_wav.trans #validation data의 파일 위치 및 label
- test_wav.list / test_wav.trans #test data의 파일 위치 및 label
- wsj0/ / wsj1/ 및 하위 폴더, 파일들 #실제 wav data

d) 실행 결과

```
yhboo@leia:~/skt/data$ python wav_gen.py -h
wavgen.py -i <input_path> -o <output_path>
yhboo@leia:~/skt/data$ python wav_gen.py -i /home/khwang/corpus/wsj/
```

```
drwxrwxr-x 4 yhboo yhboo 4096 Dec 12 00:54 .
drwxrwxr-x 3 yhboo yhboo 4096 Dec 12 00:54 ..
-rw-rw-r-- 1 yhboo yhboo 3100 Dec 12 00:54 test_dev93_wav.list
-rw-rw-r-- 1 yhboo yhboo 9215 Dec 12 00:54 test_dev93_wav.trans
-rw-rw-r-- 1 yhboo yhboo 3100 Dec 12 00:54 test_eval92_wav.list
-rw-rw-r-- 1 yhboo yhboo 9776 Dec 12 00:54 test_eval92_wav.trans
-rw-rw-r-- 1 yhboo yhboo 3000 Dec 12 00:54 train_all_wav.list
-rw-rw-r-- 1 yhboo yhboo 9183 Dec 12 00:54 train_all_wav.trans
drwxrwxr-x 4 yhboo yhboo 4096 Dec 12 00:54 wsj0
drwxrwxr-x 3 yhboo yhboo 4096 Dec 12 00:54 wsj1
yhboo@leia:~/skt/data/data_example$ find wsj1 wsj0 -type f |wc -l
300
```

(train, eval, test 각 100개씩 저장한 예시입니다. github에는 용량 문제로 인해 50개씩만 올려뒀습니다)

2. mean.npy, var.npy 구하기

모든 dataset은 train dataset 기준으로 구한 global mean, global variance으로 normalize를 수행한 뒤 model에 들어가게 됩니다. 이 때 필요한 mean.npy, var.npy를 저장하는 과정입니다.

(mean, var는 각각 3,40 모양의 numpy array입니다)

실제 training 과정에서 모든 data를 들고 있을 수 없기 때문에 미리 돌려서 mean, var를 저장해 둬야 합니다.

source code : preprocessing.py

* scipy library를 설치해야 합니다.(anaconda로 python을 설치한 경우 기본 library로 같이 설치됩니다)

a)코드 설명

1에서 생성한 wav list파일을 보고 mean과 var를 구해 .npy로 저장해주는 코드입니다.

b)실행 방법

$python preprocessing.py -m <mode> -i <input_path> -o <output_path>

<mode> : meanvar

<input_path> : 1에서의 output_path,

<output_path> : mean, var가 이 path에 저장됩니다. **default로 input_path랑 같게 설정**되어있습니다.

c) 실행 결과 생성되는 파일

output_path/mean.npy, / output_path/var.npy

d) 실행 결과

```
yhboo@vader:~/project/mytensorflow/ctc_am$ python preprocessing.py -h
preprocessing.py -m <mode> -i <input_path> -o <output_path>
```

```
yhboo@vader:~/project/mytensorflow/ctc_am$ python preprocessing.py -m meanvar -i data/data_example
mode     :  meanvar
base path:  data/data_example/
list file:  train_all_wav.list
dst_path :
total file :  50
--------- 0 th-----------
check overflow...
sum
min :  -0.000234406
max :  0.96355
square sum
min :  0.00124082
max :  2.24873
----------final mean nan check ----------
290.127
----------final var nan check-----------
473.211
final result saved at  data/data_example/
```

(50개의 train data에 대한 mean, var가 data/data_example/에 저장됩니다)

3. preprocessing 된 data 저장하기

a) wav 파일을 이용, 매 iteration마다 preprocessing을 수행하는 경우:

1번과 2번에서 수행한 wav와 mean, var 파일을 이용해 훈련을 시작할 수 있습니다. Config.py 에서 cfg['preprocessed'] 을 False로 두면 됩니다. Cfg['base_path]는 *.list와 mean.npy, var.npy 가 있는 폴더로 설정하시면 됩니다.(예시의 경우 data_example 폴더)

```
initial_lr : 0.0003
decay_factor : 0.2
decay_time : 6
patience : 8
max_epoch : 20
train_batch : 20
test_batch : 1
base_path : ./data/data_example/
data_path : ./data/
result_path : ./results/ctc_example/train_results/
model_name : ctc_cnn_online
clip_norm : 400
epoch_small : 0
epoch_mid : 0
epoch_big : 10
kernel_size : (10, 3)
lstm_dim : 512
n_layer : 3
opt_name : Adam
dropout_ratio : 0.5
preprocessed : False
ctc_cnn_online/conv0/kernel:0 : [10  3  3 32]
ctc_cnn_online/bn0/gamma:0 : [32]
ctc_cnn_online/bn0/beta:0 : [32]
ctc_cnn_online/conv1/kernel:0 : [10  3 32 32]
ctc_cnn_online/bn1/gamma:0 : [32]
ctc_cnn_online/bn1/beta:0 : [32]
ctc_cnn_online/multi_rnn_cell/cell_0/lstm_cell/kernel:0 : [ 832 2048]
ctc_cnn_online/multi_rnn_cell/cell_0/lstm_cell/bias:0 : [2048]
ctc_cnn_online/multi_rnn_cell/cell_1/lstm_cell/kernel:0 : [1024 2048]
ctc_cnn_online/multi_rnn_cell/cell_1/lstm_cell/bias:0 : [2048]
ctc_cnn_online/multi_rnn_cell/cell_2/lstm_cell/kernel:0 : [1024 2048]
ctc_cnn_online/multi_rnn_cell/cell_2/lstm_cell/bias:0 : [2048]
ctc_cnn_online/dense0/dense/kernel:0 : [512  31]
ctc_cnn_online/dense0/dense/bias:0 : [31]
-------- 0 / 20 --------
train loss -  359.763  | cer -  0.927107
valid loss -  438.556  | cer -  0.983743
valid wer -  0.925031
status :  save_param , training time :  21.435538053512573
label    : | TWO PREVIOUS WORD PROCESSING LEADERS HAVE SLIPPED <ws> |
predict  : |  <ws> <ws> |
-------- 1 / 20 --------
```

b) preprocessing 결과를 저장하는 경우

wav 파일을 매번 FFT 및 filter bank를 통과시켜 feature를 뽑는 것은 비효율적입니다. Random noise 등을 넣어 data augmentation을 하는 것이 아닌 이상 feature를 미리 뽑아서 저장해두고 훈련 때는 load만 수행하는 것이 효율적입니다. **Data를 전처리한 뒤 저장하는 과정은** 아래의 코드로 수행할 수 있습니다.

Source code : preprocessing.py

실행 방법 :

$python preprocessing.py -m <mode> -i <input_path> -o <output_path>

<mode> : processed

<input_path> : 1에서의 output_path (*_wav.list 파일들 및 mean, var.npy가 존재하는 폴더)

<output_path> : preprocessed된 data들이 저장될 폴더(.npy 파일들의 path를 가진 list도 같은 폴더에 저장됩니다) **default로 ‘./data/data_example_processed/’ 로 설정되어있습니다**.

(30분 ~ 1시간 정도 소요됩니다. Mean, var로 normalize된 data 및 list가 저장됩니다)

```
yhboo@vader:~/project/mytensorflow/ctc_am/data$ ls
data_example  data_example_processed  fb.npy  sph2pipe  sph2pipe_v2.5  wav_gen.py
yhboo@vader:~/project/mytensorflow/ctc_am/data$ cd data_example_processed/
yhboo@vader:~/project/mytensorflow/ctc_am/data/data_example_processed$ ls -al
total 52
drwxrwxr-x 4 yhboo yhboo 4096 Dec 12 03:23 .
drwxrwxr-x 5 yhboo yhboo 4096 Dec 12 03:22 ..
-rw-rw-r-- 1 yhboo yhboo 1550 Dec 12 03:22 test_dev93_processed.list
-rw-rw-r-- 1 yhboo yhboo 4395 Dec 12 03:22 test_dev93_wav.trans
-rw-rw-r-- 1 yhboo yhboo 1550 Dec 12 03:22 test_eval92_processed.list
-rw-rw-r-- 1 yhboo yhboo 4868 Dec 12 03:22 test_eval92_wav.trans
-rw-rw-r-- 1 yhboo yhboo 1500 Dec 12 03:22 train_all_processed.list
-rw-rw-r-- 1 yhboo yhboo 4193 Dec 12 03:22 train_all_wav.trans
drwxrwxr-x 4 yhboo yhboo 4096 Dec 12 03:22 wsj0
drwxrwxr-x 3 yhboo yhboo 4096 Dec 12 03:22 wsj1
yhboo@vader:~/project/mytensorflow/ctc_am/data/data_example_processed$ find wsj0 wsj1 -type f |wc -l
150
```

위와 같은 파일들이 생성되는 것을 확인할 수 있습니다.

훈련은 config.py만 바꾸면 a)와 동일하게 수행 가능합니다.

Cfg[‘preprocessed’] = True 로 설정

Cfg[‘base_path’] = 위의 output_path로 설정

훈련 실행 결과

```
yhboo@vader:~/project/mytensorflow/ctc_am$ python train.py
/home/yhboo/anaconda3/lib/python3.6/importlib/_bootstrap.py:205: RuntimeWarning: compiletime version 3.5
 'tensorflow.python.framework.fast_tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
training configuration summary
mode : uni
gpu_target : 0
charset : ABCDEFGHIJKLMNOPQRSTUVWXYZ.'

initial_lr : 0.0003
decay_factor : 0.2
decay_time : 6
patience : 8
max_epoch : 10
train_batch : 40
test_batch : 1
base_path : ./data/data_example_processed/
data_path : ./data/
result_path : ./results/ctc_example/train_results/
model_name : ctc_cnn_online
clip_norm : 400
epoch_small : 0
epoch_mid : 0
epoch_big : 0
kernel_size : (10, 3)
lstm_dim : 512
n_layer : 3
opt_name : Adam
dropout_ratio : 0.5
preprocessed : True
# of small dataset :  2
# of mid dataset :  48
# of big dataset :  50
# of all dataset :  50
ctc_cnn_online/conv0/kernel:0 : [10  3  3 32]
ctc_cnn_online/bn0/gamma:0 : [32]
ctc_cnn_online/bn0/beta:0 : [32]
ctc_cnn_online/conv1/kernel:0 : [10  3 32 32]
ctc_cnn_online/bn1/gamma:0 : [32]
ctc_cnn_online/bn1/beta:0 : [32]
ctc_cnn_online/multi_rnn_cell/cell_0/lstm_cell/kernel:0 : [ 832 2048]
ctc_cnn_online/multi_rnn_cell/cell_0/lstm_cell/bias:0 : [2048]
ctc_cnn_online/multi_rnn_cell/cell_1/lstm_cell/kernel:0 : [1024 2048]
ctc_cnn_online/multi_rnn_cell/cell_1/lstm_cell/bias:0 : [2048]
ctc_cnn_online/multi_rnn_cell/cell_2/lstm_cell/kernel:0 : [1024 2048]
ctc_cnn_online/multi_rnn_cell/cell_2/lstm_cell/bias:0 : [2048]
ctc_cnn_online/dense0/dense/kernel:0 : [512  31]
ctc_cnn_online/dense0/dense/bias:0 : [31]
-------- 0 / 10 --------
train loss -  355.132  | cer -  1.22032
valid loss -  521.487  | cer -  0.968035
valid wer -  0.923638
status :  save_param , training time :  7.862609386444092
label    : | SUPERFLUOUS LABOR IN INDUSTRIES SUCH AS COAL AND SHIPBUILDING IS GETTING CUT <#s> |
```

요약 :

1번 과정을 통해 wv1 -> wav type convert

2번 과정을 통해 mean, var 계산 후 저장

3번 과정을 통해 preprocessing된 npy data 저장 후 훈련 시작

*** train.py 함수의 맨 밑에서 config.py 중 어떤 config 함수를 사용할 지 선택합니다 config dict는 result path에 저장되어 후에 어떤 setting이었는지 확인할 수 있습니다.**