author: Zhu Benjin date: 2017.10.09

1. Channel Classification

> /data1/zhubenjin/workshop_channal

data.conf, conv_2xlstm_channal.conf, channel_updated.lst

```
./iml_dnn_train conv_2xlstm_channal.conf > train-1.log 2>&1 &
```

其中，data.conf 和 conv_2xlstm_channal.conf 参照 base_tmp_lstm_5529600_epoch_0.model 的配置文件（位于 ./base_conf 中），进行相应的修改；channel_updated.lst 是由新加入 zjlt.lst 的数据之后，生成新的 channel_label 后 shuffle 生成，具体信息可以进入相应文件查看。

data.conf's modifications:

```
    <Transform>
#        Type = ReorderDeltas
        Type = Splice
#        Context = 11
        opKeys = feat
        LeftContext = 5
        RightContext = 5
#        DeltaOrder = 2
    </Transform>
    <Transform>
        Type = Delay
        opKeys = label
        Time = 5
    </Transform>
```

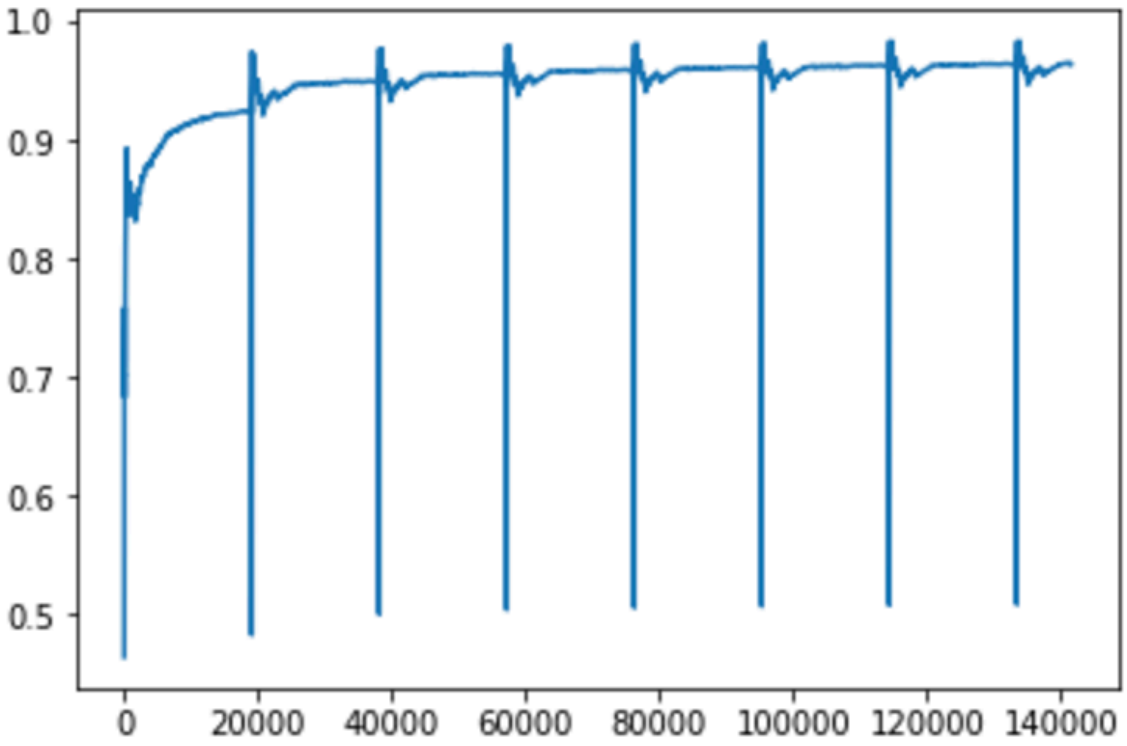
After trying to connect the 2xlstm to cnn_0 and lstm_0, it turned out that last_0 is the better choice. During training, fix learing rate of the base, and read them from base file;

```
type = conv
name = conv_0
inputs = feat
actType = sigmoid
learnRate = 0.000
```

When using the output models to train, comment `read=false` in conv.conf, it means read them from output model, rather than train from beginning again.

```
read = false
```

Traing with adam can gain likelihood up to 0.965, while only approximately get 0.65 with sgd with momentum.



If you need to change the model from base to newly generated ones, except changing the modelFile path, comment `read=false` in each layer of the channel classification branch(2xlstm+fc) is also needed.

```
inDim = 512
outDim = 4
read = false
[end]
```

It means in the beginning, parameters needs to be trained from random state. When you use generated models to train the network, read previously stored parameters from model file.

2. Training Jointly

> /data1/zhubenjin/workshop_ad_train_2/iml-train-platform

Select the best model from above training, and copy it to `workshop_ad_train/iml…/` .

- Use cut model to cut the parameters of the 2xlstm and the last fc_layer.

  ```
  ./cut_model path_to_channel_classification_model saved_cut_model_name
  ```

- Merge the base model and the saved cut model to `./new_model`

  ```
  ./merge_model path_to_base_mode path_to_saved_cut_model
  ```

- Modify `modelFile` in `./ce_ad_train.conf` to the new_model path for the first training, then

```
./iml_dnn_train ce_ad_train.conf > train.log 2>&1 &
```

According to the test result, it doesn't matter which optimization method is used during training the channel classification model (achieve 0.965 and 0.65 for adam and sgd respectively), the base model is trained use sgd with momentum. It's config file is under dir `./base_conf`.

Dynamically changing the `modelFile` to newly generated model files according to test result. My results shows that best performance is approximately 16.000% WER.

3. Test Model Performance

use `t.sh` to test model file's WER.

```
/data1/futianxiao/dynamic-wfst-call-ctc/ctc_model_test/zhuanche_test_8k/run_compile_model_2.py
```

when run this script, you need to change the thread id in it correspond to the model file:

```
os.system("mkdir -p %s\n"%(targetdir));

for i in layerID:
    stahash = {}

    for stat_file in stat_files(dirname, 'Statistics_Layer_%d_thread_258*_part_*.sta'%i):

        tt = time.ctime(os.path.getmtime(stat_file))
        st = time.strptime(tt, '%a %b %d %X %Y');
        dt = datetime.datetime(*st[:5])
```

to run test of zhuance, tengxun or driv, uncomment its specific part in the script:

```
'''
werfile = "wer_driv_test.lst"
scpfile = "testset/zhuanche_driver/test_ronglian.lst"
#reffile = "testset/zhuanche_driver/test_ronglian.ref"
reffile = "testset/zhuanche_driver/new_ronglian.ans"
'''


'''
werfile = "wer_tengxun_test.lst"
scpfile = "testset/zhuanche_driver/test_tengxun.lst"
reffile = "testset/zhuanche_driver/test_tengxun.ref"
'''


#'''
werfile = "wer_zhuanche_test.lst"
scpfile = "/data1/futianxiao/dynamic-wfst-call-ctc/ctc_mo
reffile = "/data1/futianxiao/dynamic-wfst-call-ctc/ctc_mo
#'''
```

the test result is stored in `./DECODER`, for example:

```
/data1/zhubenjin/workshop_ad_train_2/iml-train-platform/tmp_lstm_153600_epoch_0.model.bigdata-gpu-server29.xg01-25880
[WER: 16.015%] [SUB: 12.527%] [INS: 1.841%] [DEL: 1.647%]
/data1/zhubenjin/workshop_ad_train_2/iml-train-platform/tmp_lstm_204800_epoch_0.model.bigdata-gpu-server29.xg01-25880
[WER: 16.002%] [SUB: 12.531%] [INS: 1.825%] [DEL: 1.647%]
/data1/zhubenjin/workshop_ad_train_2/iml-train-platform/tmp_lstm_870400_epoch_0.model.bigdata-gpu-server29.xg01-25880
[WER: 16.180%] [SUB: 12.641%] [INS: 1.885%] [DEL: 1.654%]
```

4. Analysis

It turns out this method seems to be noneffective.

According to the paper, domain label classifier is used when gaining bad performance on using colored MNIST as test set of a MNIST-Trained model. So it is used to remove additive noises, while in our scenario, differences between recorders, compression methods etc. are more likely to be multiplicative noise. Apart from that, it's intuitive to connect the LSTM classifier to the CNN feature attractor, than the 1st lstm layer in the base model. But connect to CNN shows great decrease. The `negScalar` is also a problem, it is set between -0.01~-0.1, this will leads to gradients flow back propagate to previous layer slower, while changing to -1 will crash. And also the convolution is operated on filter banks, I think maybe it's not a good choice for learning the additive noises.

I spent about one and a half months to jointly train base and channel model, it never outperform the base model.

Reference

- Domain-Adversarial Training of Neural Networks
- Domain Sepration Networks
- Pixel Domain Adaptation with GAN
- Deep Domain Confusion
- A Deep Relationship Network with shared convolutional and task-specific fully connected layers with matrix priors
- The widening procedure for fully-adaptive feature sharing
- Cross-stitch networks for two tasks

Thanks for the past 3 months.