Deep Speaker Report

Abuduweili

Reference paper: *"Deep Speaker: an End-to-End Neural Speaker Embedding System"*

Reference code: *"https://github.com/philipperemy/deep-speaker"*

1. About the Code

models_train.py

This is the main file, contains training, evaluation and save-model function

models.py

The neural network used for the experiment. This file contains three models, CNN model (same with the paper's CNN), GRU model (same with the paper's GRU), simple_cnn model. simple_cnn model has similar performance with the original CNN model, but the number of trained parameter dropped from 24M to 7M.

random_batch.py

This is for randomly select batches for training. Randomly select one utterance as an anchor, randomly select another utterance from the same speaker as positive-sample, and randomly select another utterance from different speakers as negative-sample. It is better to use random_batch in the early stages of training.

select_batch.py

Choose the optimal batch feed to the network. This is one of the cores of this experiment. I explain as follows with specific number:

1) Select 640 speakers. 2) Select 2 utterance from each speaker, that means we have 640*2=1280 utterance at all. 3) Fed 1280 utterances to the network for inference, get embeddings of 1280 utterances and save it to the history table. 4) If we set history-table size=20, then we store the recent 20 times inference result. That means, we have 20*1280=25600 candidates. We used these candidates to select the best hard-negative samples. You might ask why we choose historical inference results as candidates? if the parameters of the network are updated every iteration. Because, if the end-stage of learning, the weight of networks updated very slightly. So the recent 20 times inference is very similar. To seek efficiency, we do not need to inference all of 25600 results on each iteration. 5) We randomly select 16 anchors (utterance) from 25600 candidates. 6) Calculate the similarity score between anchors w.r.t other candidates. 7) For each anchor, select 2 top similar negative samples (hard negative) and 2 top dissimilar positive samples. 2 anchor-positive-negative pairs for each anchor, 16*2=32 at all. 8) We used this 32 best-samples to backward and optimize the network.

triplet_loss.py

This is a code to calculate triplet-loss for network training. Implementation is the same as paper.

test_model.py

This is to test the model in terms of eer, etc.

The procedure as follows: 1) generate test data sets: (1 anchor, 1 positive, 99 negative), 2) calculating a similarity between the anchor and the other utterances, to get 100 similarity scores. For the label, 1 for positive samples, 0 for negative samples. Then call *eval_matrics.py* to calculate eer, acc, etc.

eval_matrics.py

For calculating equal error rate, f-measure, accuracy, and other metrics

pertaining.py

This is for pre-training on softmax classification loss. In pretraining, there is additional softmax classification layer at the end of the original model.

pre_process.py

Read the utterance, filterer out the mute, extract the fbank feature and save the module in .npy format.

kaldi_form_preprocess.py

Very similar to *pre_process.py,* but the input is Kaldi format data, like wav.scp, spk2utt, etc.

silence_detector.py

This is a module for silent detection for pre-processing

utils.py

There are some useful functions in this. For example, the get_last_checkpoint_if_any, plot_loss function etc.

constant.py

Some important constant for the project. Such as data path, batch_size and so on. If PRE_TRAIN=True, the run the softmax-pretraining, if PRE_TRAIN=False, then the direct train original model with triplet-loss. If COMBINE_MODEL =true, the training is performed simultaneously with the CNN and GRU models. When testing, the two models are used to get the score and then merge the score to get the final test result.

2. Experimental result

Training on the Librispeech dataset

The data used for the experiment as follows:

train-clean-100: 251 speaker, 28539 utterance

train-clean-360: 921 speaker, 104104 utterance

test-clean: 40 speaker, 2620 utterance

1. Pretraining:

used train-clean-100 data sets to pretraining with softmax classified. (I only 5% of the train-clean-100 as a test set for pretraining, remaining for training set).

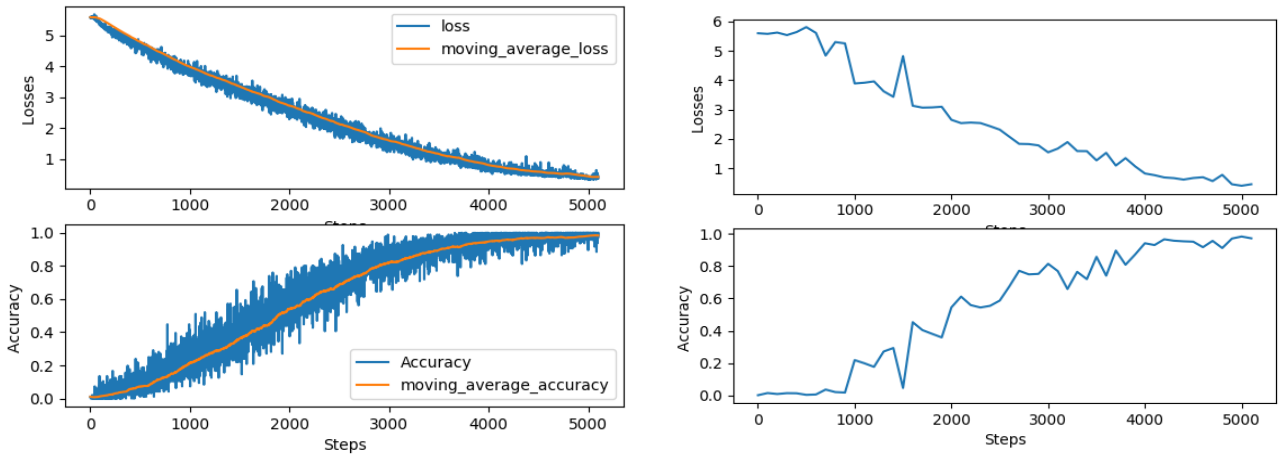

Figure 1 Pretraining Result. The left figure: loss and accuracy for the training set. The right figure: loss and accuracy for the testing set.

After 5000 steps, the training-accuracy = 99 %, and the testing-accuracy = 98 %.

2. triplet-loss training on train-clean-100 data sets with randomly selected batch.

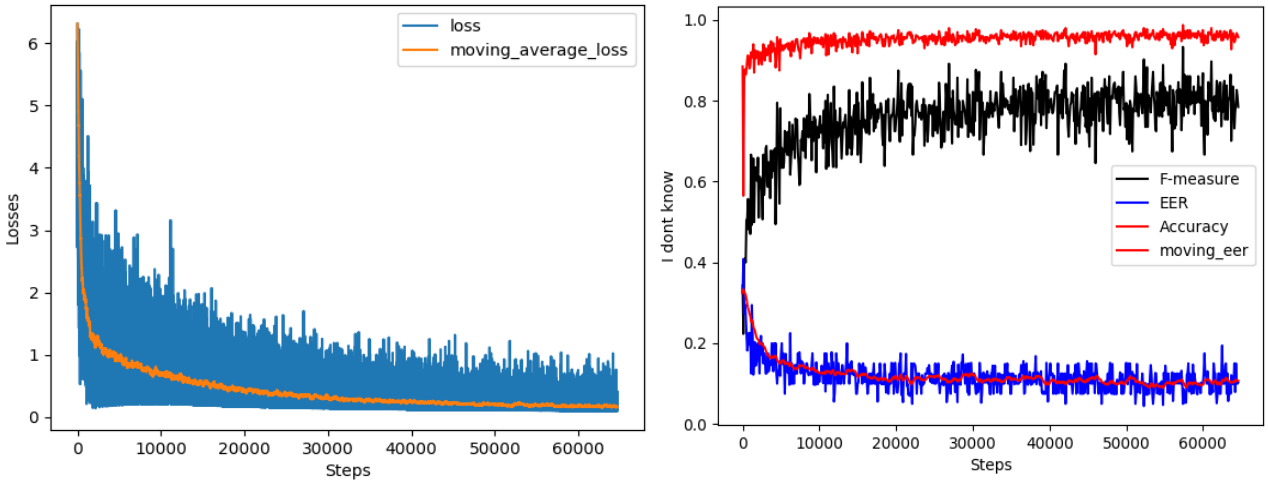

Figure 2 Testing loss and testing metrics for triplet-loss random-batch training

After 60000 steps training, the testing triplet-loss ~ 0.1, testing EER=0.8% and testing F1=0.8.

Note that this is all training with random selection of batch 's using “random_batch.py”. The result shows, random selection is also very effective at the beginning of training, and can speed up the convergence. It seems to be softmax pre-training may not necessary, instead of pretraining, you can use random-batch. However, it should be noted that this is only for very clean data set. If it is an online noisy data set, random-batch may not be so easy to converge.

(Of course, according to the paper, softmax-pretraining will slightly improve the final eer, but select hard negative play a more important role on it).

3. triplet-loss training on train-clean-100 data sets with selected hard-negative samples after softmax-pretraining.

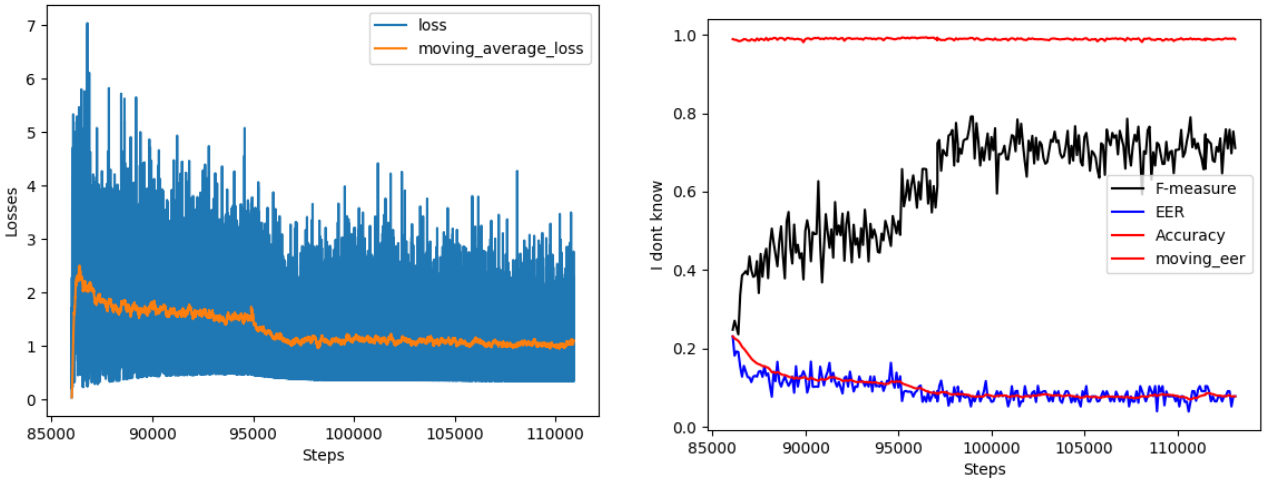

Figure 3 Testing loss and testing metrics for triplet-loss select hard negative batch training after softmax-pretraining

At the begging of triplet-loss training, the loss values will increase. And the loss will decrease after several steps training. Why loss will increase of the beginning? I think the reason is the transformation of loss-function (softmax loss -> triplet loss).

The EER of the pre-trained model (only train with softmax) is about 23%. After triplet-loss training with select best batch, the eer decrease to 5%~6%.

4 . Other Tips

1) VAD (Voice Activity Detection): The model had better performance after removing the mute (VAD).
If we directly training on train-clean-100 dataset, the final eer=8%.
After removing the mute (VAD), the final eer =5%~6 %. VAD can improve performance by 2%.

2) Increase the data set: After trained on train-clean-100 dataset, We continuously trained on train-clean-360 dataset. And the performance is improved to eer=3%~4%. We can get the same conclusion with paper, Deep-Speaker can effectively utilize the large dataset.

3) The eer on the training set has always performed very well. As the figure below, eer to reach 0.3 %, so the model is overfitting.

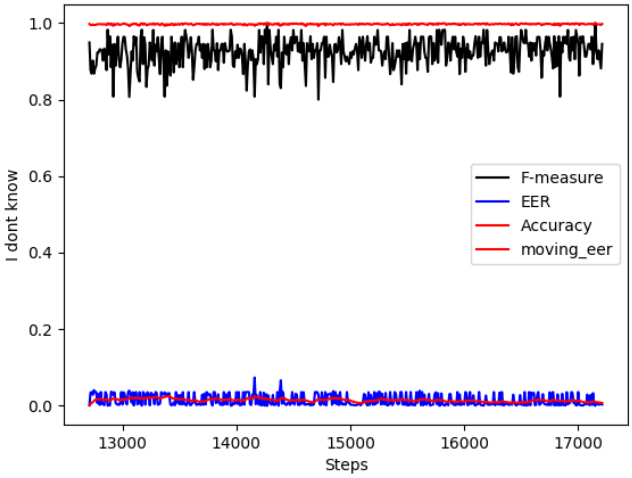

Figure 4 Training-set EER

4) Running time:
(1) Data preprocessing time: read an utterance, run VAD, extract the fbank feature, and save it in npy format. For a voice in this process takes roughly 0.8s, still relatively slow.
(2) select the batch time:
Read randomly selected 1280 utterance: 1.8s
Inference on 1280 utterance: 2.8s
Select best batch (hard negative) from all 20*1280 utterance: 0.9s.
(3) (batch-size=32) network backward and update: 0.9s

5) For CNN mode, learning rate can be selected initially 0.001 or 0.01 then drop to 0.0001, and try final_lr = 2e-5.

6) I failed to train GRU mode…. It never converges on my training.