Deep Speaker 实验报告

Abduwali

Reference paper：*"Deep Speaker: an End-to-End Neural Speaker Embedding System "*
https://arxiv.org/pdf/1705.02304.pdf
Reference code：*"https://github.com/philipperemy/deep-speaker"*（此代码有许多 *bug*，我在实验的过程中大幅度修改了其中的内容，但主题思想还是类似的）

一、代码介绍

每个 .py 文件里面都有一些注释，可以参考。我在这里大致说一下功能。
train.py
这个是主文件，可以运行后进行训练，并且每过一定的步数就进行保存模型和测试等操作。
models.py
这个是创建模型的模块，包含三个模型，分别是 CNN 模型（和论文上的一致），GRU 模型（和论文上的一致），第三个模型是我自己简化的 simple_cnn 模型，修改了一下 Resnet 模块，参数数量从 24M 下降到了 7M，效果应该差不多
random_batch.py
这个是随机选择 batch 的模块。也就是从数据集中随机选一个当成 anchror，再从同一个 speaker 的语音中随机选一个当 positive，再从不同 speaker 的语音中随机选一个当 negative。在训练初期用 random_batch 还是比较好的。
select_batch.py
选择最优的 batch 喂给网络。这个是本实验的核心之一。以一些具体的数字为例，解释如下：
首先选择 640 个 speaker（如果 speaker 数量小于 640，则可以重复选）。然后从每个 speaker 中读取 2 个语音的特征（fbank）放到候选集里面。再把 1280 大小的候选集经过网络进行前向传播，得到 embeddings。然后把这一步得到的 1280 个 embeddings 放到历史 embeds 表格中，比如我们存取历史 20 次的 embedding 数据，那么我们就有 20*1280 个语音的 embedding。然后再 20*1280 个语音中选择 16 个 speaker，计算 16 个 speaker 的语音与其他语音 embeddings 的相似度。然后对于每一个 speaker，选择最不相似的 2 个 positives 和最相似的 2 个 negatives，组成 2 对 pairs。就这样得到了 32 对 anchor-positive-nagative pairs，再把这个送进网络进行训练。
triplet_loss.py
这个是计算 triplet-loss 的模块，用于网络训练，思想是 anchor 与 negative 的相似度减去 anchor 与 positive 的相似度。
test_model.py
这个是检验模型，测试 eer 等参数的模块。models_train.py 会调用它，也可以单独运行测试模型。这个模块的工作流程是这样子的：首先产生测试数据集：1 个 anchor，1 个 positive，99 个 negative 组成 test-pairs，计算 anchor 与其他语音之间的相似度，得到 100 个 similarity。同时创建 labels，positive 对应位置是 1，neagtive 是 0.然后调用 eval_matrics.py 计算 prediction_similarity 与 labels 之间的 eer，acc 等
eval_matrics.py
输入 prediction 与 labels 可以计算，equal error rate, f-measure, accuracy 等指标
pertaining.py
这个是进行 softmax 分类预训练的模块。运行后在原来模型的后端加一个 softmax 分类层，并且进行训练
pre_process.py
这个是读取语音，过滤静音，抽取 fbank 特征，并且存成.npy 格式的模块。
kaldi_form_preprocess.py
这个和 pre_process.py 几乎类似，只是从 kaldi 格式的数·数据 wav.scp, spk2utt 等读取音频再抽特征与保存

silence_detector.py
这个是进行静音检测的模块，用于 pre-process
utils.py
这个里面有一些实用的函数。比如读取模型保存的 checkpoint 的 get_last_checkpoint_if_any 函数，还有画出 loss 曲线的 plot_loss 函数等
constant.py
一些重要的常亮都可以在这里修改。比如数据路径,batch_size 等。其中 若 PRE_TRAIN 为 True 则读取预训练的 softmax 模型在这个基础上进行训练，若为 False，则直接用原始模型用 triplet-loss 进行训练。 若 COMBINE_MODEL 为真，则训练时同时用 cnn 与 gru 模型进行训练，测试时用这两个模型分别得到 score 再融合 score 得到最终的测试结果
其他:
network.txt 保存了网络结构的视图与参数数量；checkpoints 存储模型的参数与 loss，eer 等；
pretraining_checkpoints 存储 softmax 预训练模型的参数与 loss; best_checkpoint 存储测试集上 eer 最好的模型参数。

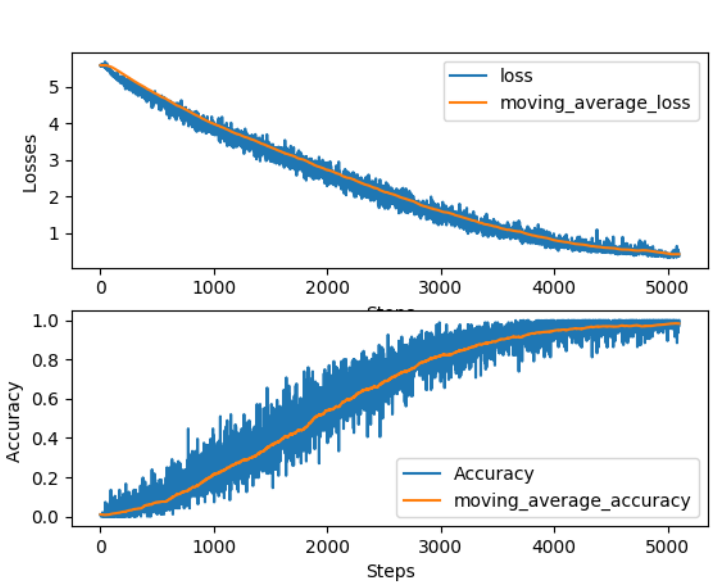
二、 实验结果

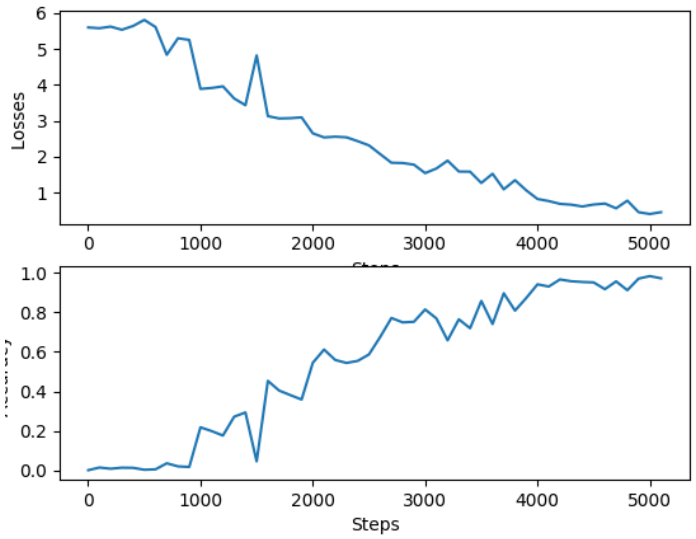
librispeech 数据集上的训练

数据集构造如下：

train-clean-100: 251 speaker, 28539 utterance
train-clean-360: 921 speaker, 104104 utterance
test-clean: 40 speaker, 2620 utterance

1）利用 train-clean-100 数据集进行预训练 softmax 分类的效果如下，因为主要目的是预训练，所以我只抽取了 5%的数据当做测试集。（左边是训练集上的

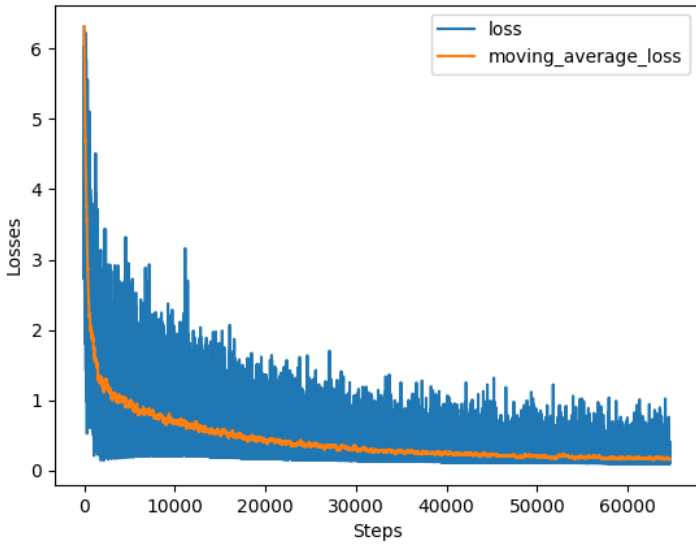


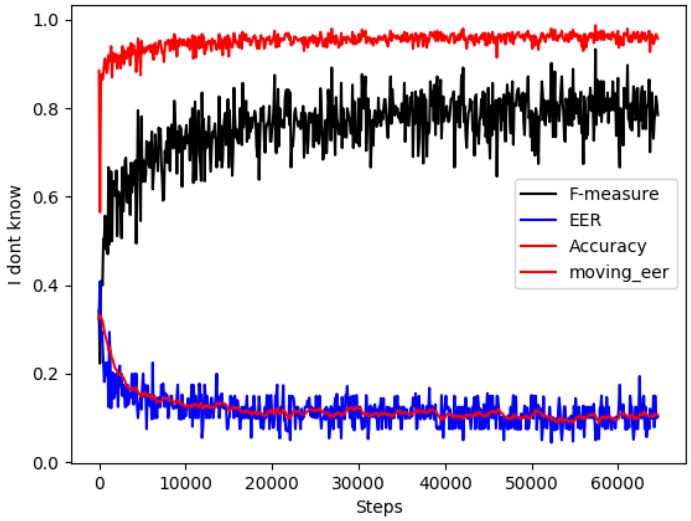


loss 与 acc，右边是测试集上的 loss 与 acc）

最后的结果是 5000 步以后训练集上的准确率达了 99%以上，测试集上是 98%

2) 在 train-clean-100 数据集直接用随机选择的 triplet-loss 进行训练。



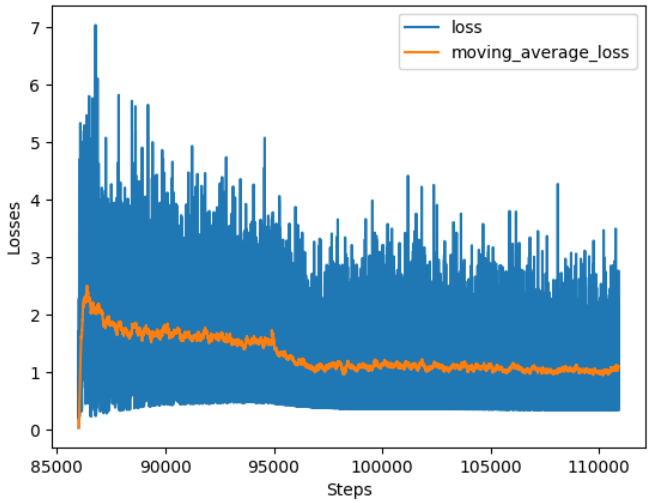


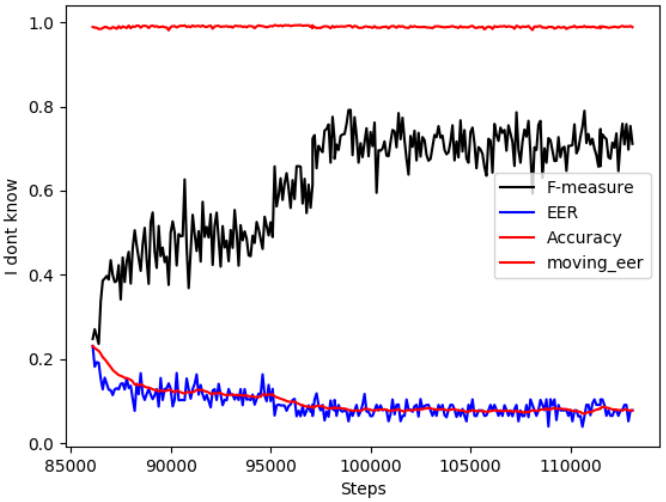
loss 最小达到 0.1 左右。eer 达到了 8%左右，f 值达到了 0.8 左右。

注意这个是从一开始就用随机选择 batch 的 random_batch.py 进行训练的。这说明训练初期随机选择 batch 也是很有效的，可以加快收敛，这说明 softmax 预训练似乎不是很有必要，可以用 random-batch 代替预训练。但是需要注意的是这个只是非常 clean 的数据集，如果是线上数据集可能 random-batch 没那么容易收敛。另一方面，random-batch 到后面 eer 就无法提高，需要用非常严格的 select-batch 紧接着进行训练。

当然按照论文说的如果预训练会稍微提高最终的 eer，但最关键的还是选择 hard negative 的 select-batch。

3) 在 train-clean-100 数据集上 softmax 预训练模型后，再用 select_bacth 选择最优 batch 进行训练





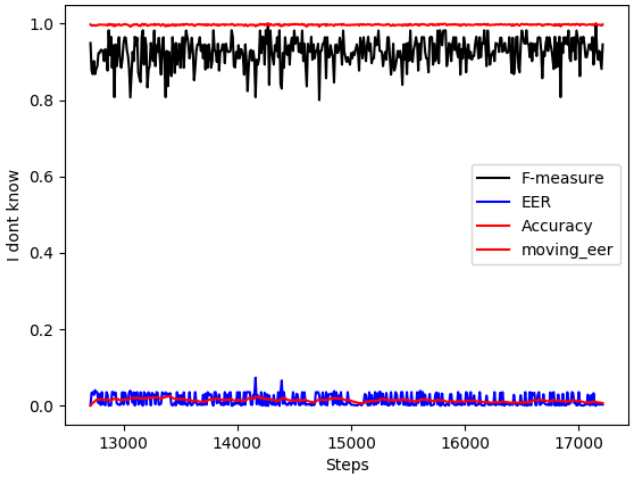
我们发现一开始 loss 会上升，随后下降，这可能是 softmax -loss 到 triplet-loss 转变导致的。对于 eer 而言，用 softmax 预训练完成后直接测 eer，大概在 23%左右，随后用 select-batch 选择最优的 batch 进行训练后，最好的 eer 可以达到 5%~6%左右。

4）其他说明：

（1）加上 VAD：删除掉静音后模型会有比较好的改进。在 train-clean-100 数据集上直接训练 eer=8%，而去掉静音后再利用纯语音训练，eer=5%~6%，大约提高了 2%以上。这还只是比较 clean 的 librispeech 数据集而言的，如果是线上数据集则加入 VAD，删除静音是必须的。

（2）增加数据集：在 train-clean-100 数据集上训练好以后，紧接着加入 train-clean-360 数据集进行训练。最好的参数从 eer=5%~6% 来到了 eer=3%~4%，也有了一些提高，验证了论文上说的，deep-speaker 对数据的利用很好，数据多则训练效果会持续改进。

（3）在训练集上的 eer 一直有非常好的表现。下图是训练集中的 eer，最好甚至可以达到 0.3%，几乎达到了绝对准确的地步，但这个是非常 clean 的训练集，所以过拟合也是可以理解的。



5）有关模型运行时间的说明：

（1）数据预处理时间：读取一个语音，进行 VAD 删掉静音，抽取 fbank 特征，再保存成 npy 格式。对于一个语音这个过程大致需要 0.8s，还比较慢，所以 pre_process.py 用了 10 个进程来处理数据。

（2）选择 batch 的时间：从所有 utterance 中读取选择 1280 个，并且读取其 npy 文件再截一段需要 1.8s 左右的时间，前向传播 1280 个 utts 需要 2.8s 时间，从包括历史数据的 20*1280 个候选集中选择 32 对 pairs 需要 0.9s 的时间。第一步读文件并且截取与后面的步骤是多线程运行。所以 select-batch time = 2.7s

（3）batch-size=32 的网络训练时间=0.9s

—----------—综合起来，从选择 batch 到训练网络，一步大概需要 3.6s 时间————————-

（4）PS：在这里吐槽一下一个大坑！那就是 pickle！请看下面的数据

local: load pkl time 0.00169s
server: load pkl time 0.0389s
也就是服务器上读 pickle 居然是龟速的，比本地慢 20 倍？what？！！！！！

emmmm….这就是我用 numpy 的 npy 格式存数据的原因：
local: save as npy 0.001s,load npy audio in 0.00084 s
server: save as npy 0.001s, load npy audio in 0.00090 s

(5) CNN 学习率最初可以选择 0.001 或者 0.01 然后降到 0.0001，最后可以试试 2e-5

三、 总结与其他说明

1. deep speaker 在 clean 数据集上的表现还是很的，但是复杂数据集上的表现不够好，有待进一步研究。
2. 我试了一下 gru 模型，但我训不出来，似乎 gru 对学习率的要求很严，我很容易就发散了。但是 gru 得 softmax 预训练倒是没太大问题，可以进行，只是 triplet loss 我没训出来
3. 按照论文上说的融合 cnn 与 gru 模型的打分会有比较明显的改善，我在代码中已经写好了融合模型，但由于 gru 迟迟没训出来，所以没有用到，下一步可以研究一下这一点。