CSLT TECHNICAL REPORT-20170004 [2017 年 10 月 30 日]

Neural Sparseness in Speech Recognition Based on Kaldi ASR Toolkit (in Chinese)

Yanqing Wang[1,2*], Zhiyuan Tang[1,3] and Dong Wang[1,2]

*Correspondence:
wangyanqingchn@163.com
[1]Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China
Full list of author information is available at the end of the article

Abstract

本组实验作为“语音识别中的稀疏性深度学习”项目（国家自然科学基金面上项目）的一部分，主要对 DNN 中全连接层的网络神经元（node）进行神经元裁剪（node-pruning），从而对 DNN 全连接层的神经元进行探究。本实验以 onorm 为指标进行 node-pruning，选用 Sigmoid 激活函数（2000维），对经过 node-pruning 和重训练的模型的性能进行考察，最终得到了一些初步结论。

Keywords: speech recognition; deep neural network; sparse neural network; neural sparseness; prune; retrain

1 介绍

本报告将首先于第2简要描述业内的相关工作和笔者之前的工作。在第3部分介绍实验中所使用的工具包（Kaldi）、数据集并进行必要的说明。在第4部分中从全局的角度介绍实验方案时候，在第5部分详细介绍整个实验的脉络、流程和具体操作，读者可依照此部分（并利用提供的文件[1]）复现整套实验，并进行相关的改进。第6部分介绍了提供的命令、代码、文件（夹）的接口和使用方法，这些文件能够方便读者进行实验的复现。第7 部分给出了实验的具体结果，并据此在第8部分对实验结果进行了制图、分析，并给出了相应的结论。第9部分简要说明了实验未完成的部分和下一步的工作。

2 Related work

因业内有关神经网络的裁剪的工作[1]主要分为 connection sparseness[2]和 neural sparseness[3]。笔者在之前做过关于 connection sparseness 的工作[4, 5]，但尚未实现 neural sparseness 。

[1]https://github.com/wyq730/CSLT-Sparse-DNN-Toolkit/tree/master/CSLT_Node_Sparseness_Toolkit

3 实验工具包及数据集

本部分介绍用于本次实验的数据（集），并进行必要的说明。

1. 工具包（Kaldi）介绍

Kaldi（语音识别工具包）官网：http://www.kaldi-asr.org/。

Kaldi 代码下载地址：https://github.com/kaldi-asr/kaldi。

2. 数据集介绍

本实验采用 Kaldi 中的 wsj 数据集。

另外，为了方便实验，将 dev93 和 eval92 两个数据集按 dev93 在前， eval92 在后的顺序进行合并，取名为 dev93_eval92 数据集。因此，在“kaldi/egs/wsj/s5/data”文件夹下，生成 Fbank 特征后，按照数据格式准备好训练数据集 train_si284_hires 和测试数据集 test_dev93_eval92_hires ，两个文件的目录为：

```
/work7/wangyanqing/kaldi/egs/wsj/s5/data/train_si284_hires
/work7/wangyanqing/kaldi/egs/wsj/s5/data/test_dev93_eval92_hires
```

4 实验方案

1. 采用onorm进行node-pruning

因 onorm 被认为是进行 node-pruning 的最佳指标[3]，本套实验均采用 onorm 方案。

2. 采用Pct-Prune方案

Pct 代表 Percentage（百分比）。意思是，笔者的实验中，在设置一个百分比（如：60%）之后，对每一层都会裁减掉该层60%的神经元。这样可以保证神经网络的整体结构保持不变，不会出现某一层裁剪过多而其他层裁剪很少的情况。

5 实验流程

本部分详细介绍实验的基本流程。另外，按照本部分的步骤，读者可以利用所给的代码文件复现各组实验。所使用到的各脚本的接口和功能将于下一部分介绍，在本部分中不着重介绍。

表 1 run_tdnn.sh中需要设置的参数

参数	含义
stage	若设置为8，则脚本实现训练&解码；若设置为9，脚本只实现解码。故在此应设置为8
train_stage	若设置为20，则脚本从20.mdl 开始进行训练
dir	应设置为 nnet_tdnn_a_x（x 可以是1,2,3...）

一、实验流程概述

1. 训练初始模型（pre-train）
2. 进行 node-pruning 操作
3. 重训练
4. 解码

二、实验操作与复现

1. 完成 wsj 的 baseline 和数据准备工作

（1）利用“kaldi/egs/wsj/s5/”下的 run.sh 进行完 default 执行的部分。

（2）对“data/test_eval92_hires”和“data/test_dev93_hires”下的数据进行 Fbank 特征的提取。可利用脚本：

```
/work7/wangyanqing/kaldi/egs/wsj/s5/run_feat.sh
```

（3）将上述两个文件夹内的对应文件合并（按照 dev93，eval92 的顺序），放置到一个新的文件夹 data/test_dev93_eval92_hires/ 下。

（4）建立文件夹“data/train_si284_hires”，利用（2）中生成的特征，在该文件夹内按照格式准备好训练 tdnn（nnet3）需要的训练数据。

2. 训练初始模型

使用 nnet3 的 run_tdnn.sh 脚本（位置：/work7/wangyanqing/kaldi/egs/wsj/s5/run_tdnn.sh）训练 tdnn 模型。注意按照表1 设置各参数：

3. 进行 node-pruning 的准备工作

为了使用现有的脚本将后续的工作简化，我们在这里需要做一些准备工作。

（1）完成 tdnn 的训练和解码任务后，在“kaldi/egs/wsj/s5/exp/nnet3/”新生成了“nnet_tdnn_a_x”文件夹（下面将均假设文件夹名为“nnet_tdnn_a_x”）。将文件夹“prune_node”复制到新生成的“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_x”文件夹下。

（2）在“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_x/”下，将解码文件夹“decode_tgpr...”

移动到“decode_baseline”文件夹下。

（3）在“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_x/prune_node”下，将上一层目录的 final.mdl 文件移动到baseline下，并重命名为final_baseline.mdl，并进行格式调整。

（4）在“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_template_x/prune_node/baseline”下，将 final_baseline.mdl 复制为非二进制的形式（final_v_baseline.mdl）。

（5）将下面文件中的 nnet_tdnn_a_3 设置为正确的目录位置 nnet_tdnn_a_x：

```
nnet_tdnn_a_x/prune/run_several_pct.sh
```

（6）统计在“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_x/final.mdl”文件中各层的 linear_params_（转移矩阵）所占的行数，并按照统计结果修改下列文件中对应的各行号：

```
nnet_tdnn_a_x/prune_node/prune_template_pct/sparse_rate_layer.sh
nnet_tdnn_a_x/prune_node/prune_template_pct/sparse_rate_total.awk
```

（7）出于格式上的考虑，需要将 mdl 文件中 final-affine 等三个 component 移动为最后三个 component。所以需要以 final-affine component 的第一行，affine2 的第一行，mdl 文件的倒数第二行为三个分割点，将 mdl 文件分成四个部分。之后再重新拼接，完成顺序的调整。所以，需要按照这个思路，修改下面文件中的几个起止行号：

```
nnet_tdnn_a_x/prune_node/prune_template_pct/split_raw.awk
```

（8）在“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_template_x/prune”下，将 run_several.sh 和 run_several_pct.sh 中“run_tdnn_x.sh”的 x 值改为真实值。

4. 进行node-pruning和解码的任务（支持多组任务同时部署）

（1） 在“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_template_x/prune_node”下，使用如下命令生成 prune 文件夹：

```
bash copy_template_pct.sh n
```

表 2 run_several_pct.sh的参数

参数	含义
max_id	实验 id（即n）的最大值
for x in 后面的部分（两处）	本次部署的实验 id （即：n）

其中，n 是该组实验内的编号。

（2） 进入刚生成的“prune_n”文件夹，修改“strategy_n”文件，在此说明本组实验的 node-pruning 操作的策略。

（3） 修改同目录下的“find_pct.py”文件，在此指定本组实验的 node-pruning 操作的百分比的值。具体方式如下：修改文件首部 pct 参数。

（4） 可以多次重复（1）到（3），同时部署多个实验

（5） 返回上一级目录“kaldi/egs/wsj/s5/exp/nnet3/nnet_tdnn_a_x/prune_node”，按照表2修改run_several_pct.sh中的几个参数。

（6） 运行 run_several_pct.sh 即可进行 node-pruning 操作

（7） node-pruning 完成之后，将文件夹中新生成的 final_new.mdl 复制到“nnet_tdnn_a_x”文件夹下，并重命名为20.mdl，准备进行重训练。

（8） 将 run_tdnn.sh 中的参数 train_stage 设置为20，并运行。此时开始重训练经过 node-pruning 后的模型。监视 nnet_tdnn_a_x 中的 log 文件夹下的 compute_prob_valid.*.log 文件，当 log-likelihood 稳定在一个较大（绝对值较小）的值附近不再大幅度变化时，说明已经充分的进行了重训练。如果重训练尚未充分，由于训练脚本的原因训练已经“完成”，则需要将最后一个模型（如：80.mdl）重命名为20.mdl，再继续进行重训练，直到训练充分为止。

（9） 将充分训练的模型在该文件夹（nnet_tdnn_a_x）内复制为 final.mdl ，准备进行解码。

（10） 将 run_tdnn.sh 中的参数 stage 设置为9，并运行，开始进行解码工作。

（11） 解码后在 nnet_tdnn_a_x/decode_tgpr_dev93_eval92 中查看WER 后，重命名文件夹 nnet_tdnn_a_x/decode_tgpr_dev93_eval92，为下一个模型的解码做准备。

（12） 可以不断重复（5）到（11），将利用（1）到（3）同时部署的多组实验的 node-pruning 的结果进行重训练。

（13） 可以不断重复（1）到（12），多次部署实验。

6 代码、命令和文件

在本部分中，主要介绍辅助 node-pruning 的主要代码文件（夹）的接口、功能，笔者基于 Kaldi 环境开发的命令（Kaldi-based Command），以及各个生成文件（夹）的含义。

表 3 Kaldi-based Command

命令	含义和用法
nnet3-calc-onorm	按照论文[3]中的方案，实现了 onorm 的计算，并保存在指定文件中。例如，nnet3-calc-onorm –binary=false orig.raw result.txt 这一行命令，计算了 orig.raw 中网络的各个神经元的 onorm 值，并且保存在了 result.txt 文件中。
nnet3-prune-node	根据输入的网络以及各层的 mask 参数（根据 onorm 参数得到），输出一个网络。例如，nnet3-prune-node orig.raw mask_2.txt mask_3.txt mask_4.txt mask_5.txt mask_6.txt mask_7.txt new.raw，利用输入的各层 mask 值，对 orig.raw 中的网络进行 node-pruning，然后输入新的网络到 new.raw

表 4 prune_node/下的核心文件（夹）

文件（夹）	接口/功能/含义
baseline/	存放 baseline 的 mdl 文件 final_v_baseline.mdl，以及其转换格式后的 raw 文件：final_v_baseline_format.raw。
copy_template_pct.sh	使用模板 prune_template_pct/，生成相应的 prune_n 文件夹
prune_template_pct/	prune_n 的模板，通过调用其内部的脚本，可以具体实现 node-pruning 的操作，具体方案见（2）
run_several_pct.sh	用于部署 node-pruning 的一个或多个实验。需在编辑好 prune_n 文件夹之后再使用。

本部分将首先介绍笔者基于 Kaldi 的环境开发的命令（Kaldi-based Command），然后介绍 prune_node 文件夹下的核心文件（夹），最后以一次完整的 node-pruning 的过程为例，详细介绍 prune_node/prune_template_pct/ 文件夹下的文件。

1. Kaldi-based Command 介绍

如表3所示。

注：读者将笔者提供的代码[2] 文件在 Kaldi 环境下编译后，即可使用表3中的命令。

2. prune_node 文件夹[3] 下的核心文件（夹）

如表4所示。

3. 一次完整的 node-pruning 的过程

如表 5所示。

[2]https://github.com/wyq730/CSLT-Sparse-DNN-Toolkit/tree/master/Supplement_for_Kaldi_Source_Code/src/nnet3

[3]https://github.com/wyq730/CSLT-Sparse-DNN-Toolkit/tree/master/CSLT_Node_Sparseness_Toolkit

表 5 一次完整的node-pruning过程

步骤	命令/脚本名	接口/功能/含义
1	nnet3-calc-onorm	输入转换格式后的 baseline 的 raw 文件，输出 onorm.txt（存储网络中每一个 node 的 onorm 值）
2	split.sh	输入步骤1生成的onorm.txt文件，输出 layer{1..7}.txt，即：将 onorm.txt 按照 layer 进行分割（split）操作
3	find_pct.py	输入步骤2生成的 layer{1..7}.txt 操作，输出 trs（存储每一个 layer 的 onorm 阈值）
4	gnrt_node_mask.sh	输入 layer{1..7}.txt 的操作，和 trs，调用 prune.awk，输出 layer{1..7}_mask.txt（即网络的 mask，每一个 node 对应一个值，-1 代表 保存，1 代表要进行 Prune）
5	nnet3-prune-node	输入转换格式后的 baseline 的 raw 文件，和步骤4生成的 layer{1..7}_mask.txt，生成 final_new_format.txt
6	make_raw_from_format.sh	输入步骤5生成的 final_new_format.raw，调用 split_raw_format.awk，生成 final_new.raw
7	change_raw_to_mdl.sh	输入步骤6生成的 final_new.raw，加上 hmm 的部分和最后的 context 部分，输出 final_new.mdl
8	sparse_rate_layer.sh	（可选）进行稀疏度 sparse_rate 的计算，输出 sparse_rate（存储新的模型的 connection sparse rate）

表 6 未经重训练的node-pruning结果

神经元稀疏度（%）	连接稀疏度（%）	WER（%）
5	8.5	9.82
7	11.9	10.43
10	16.6	12.47
15	24.3	26.23
25	31.7	89.95

注：此过程在脚本 run_several_pct.sh [4] 中得到展现。读者可结合该脚本进行理解，也可利用该脚本进行复现。

7 结果

一、结果

1. 未经过重训练的结果

如表6所示。

需要注意的是，我们需要区分神经元稀疏度（node sparse rate）和连接稀疏度（connection sparse rate）。而事实上我们也可以通过 node sparse rate 计算出每一层（或整体）的 connection sparse rate。如：如果 node sparse rate 是5%，则affine2的 connection sparse rate 是1-95%*0.95%=9.8%。

2. 经过重训练的结果

如表7所示。

[4]https://github.com/wyq730/CSLT-Sparse-DNN-Toolkit/blob/master/CSLT_Node_Sparseness_Toolkit/prune_node/run_several_pct.sh

表 7 经过重训练的node-pruning结果

神经元稀疏度（%）	连接稀疏度（%）	WER（%）
50	68.2	10.00
80	91.6	11.24
90	96.5	11.28
95	98.4	12.68
97	99.1	17.82

表 8 neural sparseness V.S. connection sparseness（经过重训练）

裁剪方式	神经元稀疏度（%）	连接稀疏度（%）	WER（%）
connection-pruning	-	97	10.47
node-pruning	90	96.5	11.28

二、neural sparseness和connection sparseness的对比

通过表8，我们可以看到：对比这两组实验，connection-pruning 不仅得到了更高的 connection sparse rate ，而且仍然有着更好的 WER。我们可以得到初步的结论：connection-pruning 优于 node-pruning。有关 connection-pruning 的工作，可以参见笔者之前的工作[4]。

笔者对node-pruning的劣势的解释如下：事实上，node-pruning 对神经网络的结构没有本质的改变，因为在进行了 node-pruning 之后，神经网络仍然是一个全连接的网络。与直接训练一个全连接的神经网络相比，node-pruning 一方面改变了各层的节点（神经元）数目，另一方面在 pre-train 之后，为神经网络赋予了（或许）更好的初值。但是，“全连接”这个基本的属性，始终没有改变。

三、不进行 random node-pruning 的原因

笔者曾在 connection sparseness 工作[4]的基础上尝试进行了 random connection-pruning [5]，但是在此不再按照类似的思路做 random node-pruning （即：不进行pre-train，随机砍掉各层的一部分神经元）。原因在于：如前文所说，node-pruning 没有改变神经网络的“全连接”属性，如果在没有进行 pre-train 的前提下做 random node-pruning，就等价于直接训练一个窄一些的深度神经网络。读者可对这一点进行验证。

8 总结

笔者基于 Kaldi ASR tookit 进行了语音识别中 neural sparseness 的实现。在实验结果的基础上进行了初步的分析，并提供了相关的代码及其使用方式，供读者复现。

9 下一步的工作

因为我们得到了 connection sparseness 优于 neural sparseness 的初步结论，所以读者可在进一步印证这一结论之后，基于笔者之前你的工作[4, 5]，更多的对 connection sparseness 进行研究。

Acknowledgement

This research was supported by the National Science Foundation of China (NSFC) under the project No. 61371136.

Author details

[1]Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China. [2]Center for Speech and Language Technologies, Division of Technical Innovation and Development, Tsinghua National Laboratory for Information Science and Technology, ROOM 1-303, BLDG FIT, 100084 Beijing, China. [3]Chengdu Institute of Computer Applications, Chinese Academy of Sciences, 610041 Chengdu, China.

References

1. Dong Wang, Qiang Zhou, and Amir Hussain, *Deep and Sparse Learning in Speech and Language Processing: An Overview*, Springer International Publishing, 2016.
2. Anders Krogh and John A. Hertz, "A simple weight decay can improve generalization," in *International Conference on Neural Information Processing Systems*, 1991, pp. 950–957.
3. Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu, "Reshaping deep neural network for fast decoding by node-pruning," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014, pp. 245–249.
4. Yanqing Wang, Zhiyuan Tang, and Dong Wang, "Connection sparseness in speech recognition based on kaldi asr toolkit," Tech. Rep., CSLT, RIIT, Tsinghua University, 2017.
5. Yanqing Wang, Zhiyuan Tang, and Dong Wang, "Attempts on randomly pruning deep neural networks in speech recognition," Tech. Rep., CSLT, RIIT, Tsinghua University, 2017.