chain model 翻译

sky_186 (/u/93ee89e278fe) ＋关注
2018.12.14 09:26* 字数 6051 阅读 90 评论 0 喜欢 1
(/u/93ee89e278fe)

翻译http://kaldi-asr.org/doc/chain.html (http://kaldi-asr.org/doc/chain.html) 时间2018年12月13日
基于前人翻译的结果,结合自己的理解进行了修改.

lattice翻译为词图
phone 翻译为音素

学习chain model建议参考资料:

- "Purely sequence-trained neural networks for ASR based on lattice-free MMI", Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahrmani, Vimal Manohar, Xingyu Na, Yiming Wang and Sanjeev Khudanpur, Interspeech 2016, (pdf (https://www.danielpovey.com/files/2016_interspeech_mmi.pdf))
- Daniel Povey, "Discriminative Training for Large Vocabulary Speech Recognition," PhD thesis, Cambridge University Engineering Dept, 2003 (pdf (https://www.danielpovey.com/files/phd_2003.pdf))

理解chain model你需要具备的知识:

- MMI目标函数
- LF-MMI的计算
- WFST的计算

“链”模型

“链”模型简介

“链”模型是一种使用nnet3实现的DNN-HMM模型，它在很多方面都和传统的模型不同。你可以将它们看作声学模型空间中的不同设计点。

- 我们在神经网络的输出端使用了3倍小的帧率，这大大减少了测试时间所需的计算量，使实时解码变得更加容易。
- 模型从一开始就用序列级目标函数 - 即正确序列的对数概率进行训练。它的本质是MMI目标函数实现的,通过在音素 n-gram 语言模型导出的解码图上做一个完全的前向-后向，在GPU上没有词图的实现。
- 由于帧速率的降低，我们需要使用非常规的HMM拓扑（允许在一个状态下遍历HMM）。
- 我们在HMM中使用固定转移概率，并且不训练它们（我们可能决定将来训练它们;但是在大多数情况下，神经网络输出概率可以与转换概率完成相同的工作，具体取决于拓扑）。
- 目前，仅支持nnet3 DNN（参见“nnet3”设置），并且尚未实现在线解码。(预计2016年4月到6月实现)(译者注:已实现在线解码)
- 目前结果比传统DNN-HMM的结果好一点（约5％相对更好），但系统解码速度提高了约3倍; 训练时间可能也快一些，但我们还没有完全比较。

在哪里可以找到“链”模型的脚本

目前用于'链'模型的最佳脚本可以在egs/swbd/s5c中的Switchboard设置中找到; 脚本local/chain/run_tdnn_2o.sh (http://2o.sh)是当前最好的脚本。目前可在官方github存储库(https://github.com/kaldi-asr/kaldi.git (https://github.com/kaldi-asr/kaldi.git))的“chain”分支中找到它，它最终将合并到主分支。(译者注:已合并)

该脚本使用TDNN作为神经网络（我们一直在使用TDNN进行开发，因为它们更容易调整LSTM），并提供比基线TDNN更好的WER（词错误率）：11.4％，而最佳TDNN基线为12.1％（在Switchboard的eval2000部分）。

链模型

The chain model itself is no different from a conventional DNN-HMM, used with a (currently) 3-fold reduced frame rate at the output of the DNN.
链模型本身与传统DNN-HMM模型并无大的差别,(当前)使用3倍减少的帧率作为DNN的输出。DNN的输入特征是原始帧速率为每秒100帧; 这是有道理的，因为我们当前使用的所有神经网络（LSTM，TDNN）都有某种经常性的循环连接或内部拼接，即它们不是纯粹的前馈网络。

与通常模型的不同之处在于用于训练它的目标函数：我们使用正确音素序列的对数概率作为目标函数，而不是帧级目标。训练过程在原理上与MMI训练非常相似，其中我们计算分子和分母的“占领概率”，在导数计算中使用两者之间的差异。不再需要将DNN输出归一化为每帧的总和为1 - 这样的归一化没有什么作用。

由于帧速率降低（每30 ms一帧），我们需要使用修改后的HMM拓扑。我们希望HMM在一次转换中是可遍历的（而不是模型的3个转换与正常帧速率相对）。当前偏好的拓扑具有只能出现一次的状态，另一种状态可以出现零次或者多次。使用与基于GMM的模型相同的过程获得状态聚类，尽管具有不同的拓扑（我们将对齐转换为新的拓扑和帧速率）。

“链”模型的训练步骤

链模型的训练过程是MMI的无词格版本，其中分母状态后验概率是通过由音素级解码图形成的HMM上的前向-后向算法获得的，并且分子状态后验是通过类似的向前-向后算法来获取的，但限于对应于转录文本的序列。

对于神经网络的每个输出索引（即每个pdf-id），我们计算（分子占用概率 减去 分母占用概率）形式的导数， 并将它们传播回网络。

分母FST

对于计算的分母部分，我们在HMM上做向前-向后。实际上，因为我们将它表示为有限状态接受器，所以标签（pdf-id）与弧而不是状态相关联，因此在通常的公式中它并不是真正的HMM，但是我们更容易将其视为HMM，因为我们使用前向前-向后算法来获得后验。在代码和脚本中，我们将其称为“分母FST”。

分母FST的音素级别的语言模型

构建分母FST的第一个阶段是创建音素语言模型。该语言模型是从训练数据音素对齐中学习的。这是一种不平滑的语言模型，这意味着我们永远不会退回到低阶n-gram。但是，某些语言模型状态会被完全修剪，因此转换到这些状态会改为低阶n-gram状态。我们避免平滑的原因是减少语音环境扩展后编译图中将出现的弧数。

我们选定的配置是用来估算一个4-gram语言模型和不修剪低于trigram的LM状态（因此我们始终保持至少2-phone历史记录）。除了由不剪枝的trigram规则决定的状态数量之外，我们还有一个可指定数量（例如2000）的4-gram语言模型状态，这些状态将被保留（所有其余的都用相应的trigram状态标识），我们选择保留的那些是以最大化训练数据似然的方式确定的。所有概率是以最大化训练数据似然的方式估计。不修剪三元组的原因是任何允许三元组的稀疏性将倾向于最小化编译图的尺寸。请注意，如果我们的音素LM只是一个简单的音素循环（例如：一个unigram），由于语音的临近序列效应，它会扩展为三音素，但它会有所有可能的三元组的弧。因此，使用未修剪的三元模型得到的任何稀疏性都是一个奖励。根据经验，未平滑的trigram LM可扩展到尽可能小的FST; 并修剪一些trigrams，虽然它增加了编译的FST的大小，导致很少或没有WER改善（至少300小时的数据扩展3倍速度扰动;在较少的数据可能有帮助）。

在Switchboard设置中，我们尝试的各种模型的phone-LM复杂度在5到7之间; 我们所选配置的phone-LM复杂度（4-gram，除了2000个状态以外都被修剪为trigram）大约是6个。并不是因为较低的phone-LM复杂度总是使训练系统有更好的WER; 对于传统的（基于单词的）MMI训练，一个中间强度的语言模型似乎效果最好。

分母FST的汇编

将上一节中描述的音素语言模型扩展为FST，其中把'pdf-id'作为弧，在这个过程中反映了正常Kaldi解码中的解码图编译过程（参见解码图创建配方（测试阶段） (http://kaldi-asr.org/doc/graph_recipe_test.html)），除了没有涉及词典，最后我们将transition-id转换为pdf-id。

一个区别在于我们如何最小化图的大小。正常的配方包括确定化和最小化。我们无法使用此过程减少图的大小，或者使用消歧符号来减小图形的大小。相反，我们的图形最小化过程可以紧凑地描述如下：“重复3次：推动，最小化，反向;推动，最小化反转。”。'推'指的是权重推重(weight-pushing); “反向”是指弧的方向反转，并交换初始和最终状态。

初始和最终概率，以及'标准化FST'

上面提到的图形创建过程自然地给出了一个初始状态，以及每个状态的最终概率; 但这些不是我们在向前-向后中使用的那些。原因是这些概率适用于话语边界，但我们训练分裂的是固定长度（例如1.5秒）的话语块。在这些任意选择的切割点上限制HMM的初始状态和最终状态是不合适的。相反，我们使用从“running the HMM”得到的初始概率进行固定次数的迭代并对概率求平均值; 每个状态的最终概率等于1.0。我们有理由这样做，但现在没有时间解释它。在分母向前-向后过程中，我们将这些初始和最终概率应用于初始和最终帧作为计算的一部分。但是，我们还写出了具有这些初始和最终概率的分母FST版本，我们将其称为“归一化FST”。（使用epsilon弧模拟初始概率，因为FST不支持初始概率）。这种“标准化FST”将用于向分子FST添加概率,使用的方法下面介绍。

分子FST

作为我们准备训练过程的一部分，我们为每个语料生成一份称为“分子FST”（“numerator FST ”）的东西。分子FST编码了监督转录文本，并且还编码该转录本的对齐（即，它强制与从基线系统获得的参考对齐相似），但是它允许稍微的“摆动空间”与该参考不同。默认情况下，我们允许音素在词图对齐中分别在其开始和结束位置之前或之后0.05秒发生。合并对齐信息非常重要，因为我们不是对整个话语进行训练，而是对分裂的固定长度的话语进行训练（这对于基于GPU的训练很重要）：如果我们将话语分成几部分知道转录本对齐的位置。

译者注:这段翻译的不好

我们使用特定于话语的图解作为解码图，通过词图生成解码过程生成训练数据的替代发音的网格，而不是强制执行训练数据的特定发音。这会生成发音的所有对齐, 在最佳得分发音的束内。

拆分分子FST

如上所述，我们训练固定长度的话语片段（例如长度为1.5秒）。这要求我们将分子FST分成固定大小的碎片。这并不难，因为分子FST（记住，编码时间对齐（time-alignment）信息）自然具有我们可以识别具有特定帧索引的任何FST状态的结构。注意：在我们进行此拆分的阶段，分子FST中没有损失 - 它只是被视为编码路径上的约束 - 因此我们不必决定如何分割路径上的损失。

归一化分子FST

上图（分母FST的汇编）提到了如何计算分母FST的初始和最终概率，以及我们如何在“归一化FST”（ 'normalization FST'）中对这些概率进行编码。我们用这个“归一化FST”组成分子FST的分割部分，以确保分母FST的损失反映在分子FST中。这确保了目标函数永远不会是正的（这使得它们更容易解释），并且还防止分子FST可能包含分母FST不允许的状态序列的可能性，这原则上可以允许目标函数在不受约束的条件下增加。这钟情况可能发生的原因是音素LM缺乏平滑，并且从1-best对齐估计，因此词图可能包含有在训练中看不到的音素n-gram序列。

偶尔（但很少）会发生这种归一化过程产生空FST的情况：这种情况可能发生在当词图包含用于训练音素语言模型的1-best对齐中不存在的三音素时，并且没有任何可选路径的格可以弥补由此产生的“失败”路径。这是有可能发生的，因为1-bes和词图产生对齐选择了单词的不同发音。这些语料就被丢弃了。

分子FST的格式

分子FST是加权接受器，其中标签对应于pdf-id加1。我们不能使用pdf-id，因为它们可能为零; 并且零是由OpenFST专门处理的（如epsilon）。当我们形成小批量时，我们实际上将它们放在一起以形成更长的FST，而不是存储一系列单独的分子FST; 这使我们能够对小批量中的所有话语进行单向向前-向后，直接计算总的分子对数概率。（这不是一个重要的特性，它只是一个软件细节，我们在这里解释，以免产生混淆）。

固定长度的块和微批量处理

为了训练小批量，我们将我们的话语分成固定长度的语音块（在我们当前的脚本中长度为1.5秒）。比这更短的话语被丢弃; 那些更长的，被分成块，块之间有重叠，或块之间有小间隙。请注意，我们的声学模型通常需要左或右框架用于声学环境; 我们对其进行了补充，但这是一个单独的问题; 在确定块之后添加了环境。

我们的小批次(minibatch)大小通常是2的幂，它可能受GPU内存因素的限制。我们的许多示例脚本每个小批量使用128个块。GPU内存的最大单个使用者是向前-向后计算中的alpha概率。例如，对于1.5秒的块，我们在3倍二次抽样后有50个时间步长。在我们的Switchboard设置中，典型的分母FST中有30,000个状态。我们对alpha使用单精度浮点数，因此以千兆字节为单位的内存为（ 128 * 50 * 30000 * 4 ）/ 10 ^ 9 = 0.768G。

这不会耗尽所有GPU内存，但还有其他内存来源，例如我们在内存中保留两个nnet输出副本，这根据配置需要相当多的内存 -- 例如，将30000替换为10000，它将为您提供在合理配置中用于一个nnet输出副本的内存量。

对帧移位数据进行训练

在神经网络训练中，我们已经有了生成扰动数据的方法来人为地增加我们训练的数据量。我们的标准nnet3神经网络训练示例脚本通过0.9, 1.0和1.0的因子对原始音频进行时间扭曲，以创建3倍的增强数据。这与'链'模型正交，我们这样做（或不这样做）就像我们对基线一样。但是，有一种额外的方法可以通过移动帧来增加链模型的数据。这些模型的输出帧速率是常规帧速率的三分之一（当然是可以配置的），这意味着我们只评估nnet输出t值为3的倍数，因此我们可以通过将训练样例移动0,1和2帧来生成不同版本的训练数据。这是在训练脚本中自动完成的，当我们从磁盘读取训练样例时，它“即时”完成 - 程序nnet3-chain-copy-egs有一个由脚本设置的选项–frame-shift（帧-移位）。这会影响解释epochs(训练次数)。如果用户请求例如4个epochs，那么我们实际上训练了12个epochs; 我们只是对3个不同移位版本的数据这样做。选项–frame-shift=t选项实际上做的是将输入帧t移位并将输出帧移动最接近的3到t。（通常它可能不是3，它是一个名为–frame-subsampling-factor的配置变量）。

GPU训练中的问题

特定于 “链”计算的计算部分是分子FST上的向前-向后和分母HMM上的向前-向后。分子部分非常快。向前-向后分母需要相当多的时间，因为分母FST中可能存在很多弧（例如，典型的Switchboard设置中有200,000个弧和30,000个状态）。所花费的时间几乎与计算的神经网络部分所花费的时间一样多。我们非常小心地确保内存的位置。

进一步加快这一步骤的下一步可能是实现向前-向后计算的修剪版本（如修剪的Viterbi，但计算后验）。为了获得加速，我们必须削减很高比例的状态，因为我们需要弥补修剪带来的内存局部性损失。在我们当前的实现中，我们仔细地确保一组GPU线程都处理相同的HMM状态和时间，只是来自不同的块（我们在代码中调用这些不同的'序列'）; 并且我们确保对应于这些不同序列的内存为止在内存中彼此相邻，因此GPU可以进行“合并内存访问”。使用状态级修剪，因为不同序列的内存访问将不再“同步”，我们会失去这个优势。但是，获得向前-向后算法的修剪版本仍然是可行的。

对于速度，我们不在分母图的alpha-beta计算中使用对数值。为了将所有数值保持在合适的范围内，我们将每一帧上的所有声学概率（指数化nnet输出）乘以选定的“任意值”，以确保我们的alpha分数保持在良好的范围内。我们称之为“任意值”，因为算法的设计使得我们可以在这里选择任何值，并且它在数学仍然上是正确的。我们将一个HMM状态指定为“特殊状态”，并且选择“任意常数”是前一帧上该特殊状态的alpha的倒数。这使特殊状态的alpha值保持接近1。作为“特殊状态”，我们选择一个在HMM限制分布中具有高概率的状态，并且它可以进入大部分的HMM状态。

使用'链'模型解码

使用'链'模型的解码过程与基于常规nnet3神经网络的模型完全相同，实际上它们使用相同的脚本（steps/nnet3/decode.sh）。但是也有一些配置差异：

- 首先，使用不同且更简单的拓扑构建图形; 但这不需要用户采取任何特殊操作，因为图构建脚本无论如何都要从“链”训练脚本生成的'final.mdl'中获取拓扑，该脚本包含正确的拓扑。
- 默认情况下，当我们编译图时，我们使用0.1的“自循环标度”(-self-loop-scale)。这会影响如何处理自循环的转换概率（通常效果更好）。但是，对于“链”模型，鉴于它们的训练单方法，我们需要使用与我们训练过的完全相同的转换概率缩放，为简单起见，我们将其设置为1.0。所以我们将utils/mkgraph.sh脚本中的–self-loop-scale 1.0选项设置为1.0。
- 这些模型中没有“先行分割”的必要条件。所以我们根本就没有在.mdl文件中设置先验矢量;我们确保解码器在没有设置先验的前提下，只省略除法。。
- 我们通常在解码中使用的默认声学标度（0.1）不适用于“链”模型，最佳声学标度非常接近1.因此我们提供–acwt 1.0选项给脚本 steps/nnet3/decode.sh。
- 评分脚本只能以1为增量搜索语言模型比例，这在最佳语言模型比例在10到15之间的典型设置中很有效，但在最佳语言模型比例接近1时则不行。（注意：出于当前目的，您可以将语言模型比例视为声学比例的倒数）。为了解决这个问题而不改变评分脚本（这是特定于数据库的），我们提供了一个新的选项–post-decode-acwt 10.0给脚本steps/nnet3/decode.sh，它在转储词图之前将声学概率缩放10。在此之后，最佳语言模型比例将在10左右，如果您不了解此问题，可能会有点混乱，但是对于评分脚本的设置方式来说是很方便的。

一旦使用该–acwt 1.0选项，默认解码和lattice beams适用于“链”模型而无需修改。但是，它们不会显示完全可能的加速，并且您可以通过使用稍微紧一点的束(beam)来获得更快的解码。通过在Switchboard设置中收紧光束，我们能够将解码时间从实时的1.5倍降低到实时的约0.5倍，精度降低仅约0.2%（这是CPU上神经网络评估的结果;在GPU上会更快）。Dan提供的注意事项：在我写这篇文章的时候，这是我最好的回忆; 实际上，退化可能不止于此。请记住，这是在高性能的现代服务器机器（单线程）上。

您可能会注意到当前的示例脚本中,我们使用的是iVectors。我们这样做只是因为它们通常有所帮助而且因为我们正在使用它们作为基线设置。“链”模型没有内在联系，也没有使用它们的基本要求。

小礼物走一走，来简书关注我

赞赏支持