3.4 简单RNNs和GRUs的比较

到目前为止，我们所展示的模型都是简单的RNNs，它具有双向递归层，且在时间方向上正向递归和逆向递归都是由公式3所建模的。目前在语音和语言处理方面的研究表明，更复杂的递归可以让网络记住更多的时间步长的状态，同时使它们的计算训练更加昂贵。尽管还存在许多其他变体，两种常用的递归结构是长短时记忆(LSTM)单元和门控递归单元(GRU)。最近对LSTM和GRU架构的数千种变体进行的全面研究表明，GRU与具有适当初始化的忘记门偏的LSTM相当，并且它们的最佳变体彼此竞争。我们决定检查GRUs，因为在更小的数据集上的实验表明，对于相同数量的参数，GRU和LSTM达到了相似的精度，但GRU训练更快，发散的可能性更小。

我们使用的GRU由下式计算

（10）

其中σ（·）是S形函数，z和r分别代表更新和复位门，为简单起见，我们删除了层上标。 我们与标准GRU的不同之处在于，我们先将隐藏状态ht-1乘以Uh，然后再通过复位门进行缩放。这允许对ht-1的所有运算都可以在单个矩阵乘法中进行计算。 输出的非线性函数f（·）通常是双曲正切函数tanh。但是，我们发现tanh和削波的ReLU非线性具有相似的性能，并且选择剪裁的ReLU可以简化网络的简单性和均匀性。

GRU和简单的RNN架构都受益于批处理规范化，并在深度网络中显示出强大的结果。但是，表3显示，对于固定数量的参数，GRU体系结构可在所有网络深度实现更好的WER。这清楚地证明了语音识别任务中固有的长期依赖性，这既存在于单个单词中，也存在于单词之间。正如我们在3.8节中讨论的那样，由于具有大量的训练数据，即使简单的RNNs也能够隐式地学习一种语言模型。有趣的是，具有5个或更多递归层的GRU网络不会显著提高性能。我们将其归因于从1个循环层的每层1728个隐藏单元到7个循环层的每层768个隐藏单元的稀疏化，以保持参数总数不变。

GRU网络的性能优于表3中的简单RNN。但是，我们在以后（第6节）的结果中发现，随着模型规模的扩大，对于固定的计算预算，简单的RNN网络的性能会稍好一些。鉴于此，其余大多数实验都使用简单的RNN层而不是GRUs。

3.5频率卷积

时间卷积通常用于语音识别中，以有效地建模可变长度话语的时间平移不变性。这种卷积是在25年前的语音神经网络中被首次提出的。许多神经网络语音模型的第一层处理带有某些上下文窗口的输入帧。这可以看作是跨度为1的时间卷积。

此外，为了使递归神经网络在计算上具有高采样率音频易于处理，子采样是必不可少的。DS1系统通过使用频谱图作为输入并且在第一层使用时间跨度参数进行时间卷积以减少时间步数来实现这一目标。

在进行任何其他处理之前将频域和时域中的卷积应用于频谱输入特征时，可以稍微改善ASR的性能。与大型全连接网络相比，频率卷积试图更精确地模拟由于说话人可变性而导致的频谱方差。由于特征的频谱排序被完全连接和递归的层删除，因此频率卷积作为网络的第一层能够更好地工作。

我们尝试添加一到三层卷积进行实验。这些都在时频域（2维不变）和仅时域（1维不变）中。在所有情况下，我们都使用相同的卷积，在频率和时间上都保留输入特征的数量。在某些情况下，我们会指定跨任何一个跨度，以减小输出的大小。由于卷积层向我们的网络中添加了一小部分参数，因此我们没有明确控制参数的数量。表4所示的所有网络都有大约3500万个参数。

我们报告了两个数据集的结果：一个是2048个语音的开发集（“常规Dev”），另一个是从CHiME 2015开发数据集中随机抽取的2048个语音的嘈杂数据集（“ 嘈杂Dev”）。我们发现，一维不变卷积的多层仅提供了很小的好处。二维不变卷积极大地改善了嘈杂数据的结果，同时为干净数据提供了一点好处。从一维不变卷积到三层二维不变卷积的变化使嘈杂的开发集上的WER提高了23.9％。

3.6步幅

在卷积层中，我们使用更长的步幅和更宽的上下文来加快训练速度，因为对给定发声进行建模所需的时间步长更少。对输入声音进行下采样（通过FFT和卷积步长）可减少后续层所需的时间步数和计算量，但会降低性能。

在我们的普通话模型中，我们以简单的方式进行跨步。但是，在英语中，跨步会降低准确性，这仅仅是因为我们网络的输出每个输出字符至少需要一个时间步长，并且每个时步的英语语音中的字符数很高，足以在跨步时引起问题。为了克服这个问题，我们可以用一些符号来丰富英语字母，这些符号代表着完整的单词、音节或不重叠的字格。在实践中，我们使用不重叠的二字或二字组，因为它们很容易构造，不像音节，并且它们与诸如全字词之类的替代词相比有所减少。我们通过一个简单的同构将一元标签转化为二元标签。

不重叠的图会缩短输出转录的长度，从而可以减小展开的RNN的长度。猫坐着这个不重叠的二元组的句子被分割。请注意，对于具有奇数个字符的单词，最后一个字符变为一个字母组合，并且空格也被视为一个字母组合。这种同构性可确保相同的单词始终由相同的双字母组和单字母组合标记组成。双字母组的输出集由训练集中出现的所有双字母组组成。

在表5中，我们显示了在有或没有语言模型的情况下，不同步幅的双字母组和会标系统的结果。我们观察到，二元组允许更大的进步，而不会牺牲单词错误率。这使我们减少了同时受益于计算和内存使用的展开RNN的时间步数。

3.7行卷积和单向模型

双向RNN模型难以在在线，低延迟的环境中进行部署，因为它们是为在整个重新采样上运行而构建的，因此当用户发出话语时，不可能在转录过程中形成拓扑。我们已经找到了性能和双向模型一样好的单向体系结构。 这使我们可以在部署系统中使用单向，仅前向RNN层。

为此，我们使用了一个如图3所示的特殊的层，我们将其称为行卷积。·该层背后的直觉是，我们只需要一小部分未来信息即可在当前时间步长做出准确的预测。假设在时间步长t，我们使用τ步长的将来上下文。现在，我们有一个大小为d×（τ+ 1）的特征矩阵ht：t +τ= [ht，ht + 1，...，ht +τ]。我们定义一个与ht：t +τ大小相同的参数矩阵W。在时间步t处，新层的激活rt为。 由于等式中的卷积运算。对于W和ht：t +τ都是面向行的，我们称此层为行卷积。

我们将行卷积层放置在所有循环层之上。 这有两个优点。首先，这允许我们在行粒度较低的行卷积层下面流式传输所有需要的小上下文。其次，与普通话的最佳双向模型相比，此结果具有更好的CER。我们推测，递归层已经学习了良好的特征表示，因此行卷积层只是收集了适当的信息以馈送到分类器。在第7节“部署”中给出了具有行卷积的单向普通话语音系统的结果以及与双向模型的比较结果。

3.8语言模型

我们通过数百万种独特的话语训练RNN模型，使网络能够学习强大的隐式语言模型。我们最好的模型是拼写熟练的拼写，没有任何外部语言限制。此外，在我们的开发数据集中，有很多情况下我们的模型可能隐含地消除了同音异义词的歧义，例如，“他希望日本代理商能卖到27.58万美元”。 但是，与可用的未标记文本语料库相比，标记的训练数据很小。 因此，我们发现，使用外部文本训练的语言模型来补充我们的系统时，WER会得到改善。

我们使用n-gram语言模型，因为它们可以很好地缩放到大量未标记的文本。对于英语，我们的语言模型是经过修剪的Kneser-Ney平滑5-gram模型，使用KenLM工具包对来自通用抓取存储库的清除文本进行了训练。该词汇表最常用的词是来自2.5亿行软件扩展中的40万个单词，该词产生的语言模型大约有8.5亿个n-gram。对于普通话，该语言模型是经过修剪的KneserNey平滑字符级5-gram模型，该模型在内部80亿行文本的语料库上训练。 这将产生大约20亿个n-gram的语言模型。

在推论过程中，我们将寻找使方程12中所示的Q（y）最大化的转录y。 这是来自CTC训练的网络和语言模型的对数概率与单词插入词的线性组合.

Q(y) = log(pctc(y|x)) + αlog(plm(y)) + β word\_count(y) (12)

权重α控制语言模型和CTC网络的相对贡献。 权重β鼓励转录中的更多单词。 这些参数是在调整开发设置的。 我们使用光束搜索来找到最佳转录。

表6显示了外部语言模型对英语和普通话语音系统都有帮助。 当我们从具有5层和1个递归层的模型变为具有9层和7个递归层的模型时，语言模型所提供的相对改进从英语的48％降至36％，普通话从27％降至23％。 我们假设网络建立了一个具有更多循环层的更强大的隐式语言模型。

与英语相比，语言模型的相对性能改进要高。 我们将其归因于以下事实：中文字符比英文字符代表更大的信息块。 例如，如果我们直接输出到英语的音节或单词，则该模型的拼写错误将更少，而语言模型的帮助可能会更少。

3.9适应普通话

到目前为止，我们已经介绍了这些技术，可以用来构建直接输出汉字的端到端普通话语音识别系统。 这排除了构建语音模型的必要，该模型很容易就可以将语音系统与其他语言的语音系统结合使用。直接输出到字符还排除了显式建模语言特定发音特征的需要。 例如，我们不需要像某些语音系统一样必须对普通话音进行显式建模。

唯一的体系结构更改是我们根据汉字字符集的特性对我们的网络进行评估的结果。 首先，网络的输出层输出约6000个字符，其中包括罗马字母，因为中英文混合成绩单很常见。 如果此集合中不包含字符，则在评估时会出现词汇错误。 这并不是主要的问题，因为我们的测试集仅有0.74％的特征字符。

我们在普通话中使用字符级语言模型，因为在文本中单词通常不分段。等式12中的单词插入项变成了字符插入项。此外，我们发现在解码过程中，波束搜索的性能在较小的波束尺寸下趋于稳定。这使得我们可以使用200的光束，而在CER中的降级可以忽略不计。在第6.2节中，我们展示了我们的普通话语音模型在架构变化方面的改进大致与我们的英语语音模型相同。

4系统优化

我们的网络具有数以千万计的参数，并且训练算法需要数以千计的单精度exaFLOP进行收敛。由于我们评估有关数据和模型的假设的能力取决于快速训练模型的能力，因此我们构建了高度优化的训练系统。 该系统具有两个主要组件-用C ++编写的深度学习库，以及同时用CUDA和C ++编写的高性能线性数据库。我们优化的软件运行在每个节点具有8个Titan X GPU的密集计算节点上，当我们在一个节点上训练单个模型时，可以维持每秒24个单精度teraFLOP。这是每个节点的理论峰值计算吞吐量的45％。如下一小节所述，我们还可以扩展到多个节点。

4.1可伸缩性和数据并行性

我们使用标准的数据并行技术在使用同步SGD的多个GPU上进行训练。 我们最常见的配置是在8个GPU上使用512的最小批量。我们的培训管道将一个过程绑定到每个GPU。 然后，这些过程在反向传播期间使用all-reduce交换梯度矩阵，后者在多个过程之间交换矩阵并将结果求和，以便最后，每个过程都具有所有过程的所有矩阵总和的副本。

我们发现同步SGD很有用，因为它具有可重复性和确定性。 我们发现，系统中不确定性的出现通常预示着严重的错误，因此将可重复性作为目标可以极大地促进调试。 相反，Dean等人发现异步方法，例如带有参数服务器的异步SGD。 通常不提供可重复性，因此难以调试。同步SGD易于理解和实现。当我们在训练过程中添加多个节点时，它可以很好地扩展。

图4显示，随着我们将训练的GPU数量加倍，训练所需的时间减半，从而实现了接近线性的弱缩放。在此实验中，我们将每个GPU的最小批量保持恒定为64，随着我们将GPU数量加倍，有效地将最小批量提高了一倍。 尽管我们有能力扩展到大型迷你批处理，但为了获得最佳结果，我们通常在训练期间使用8或16个GPU（最小批处理为512或1024）。

由于全缩减对于培训的可扩展性至关重要，因此我们编写了自己的环形算法实现，以实现更高的性能和更好的稳定性。我们的实现避免了CPU和GPU之间的多余复制，这对我们的可伸缩性至关重要。我们将OpenMPI与smcuda传输配置在一起，该传输可以使用GPUDirect发送和接收驻留在两个不同GPU的内存中的缓冲区。 当两个GPU位于同一PCI根联合体中时，这可以避免不必要地复制到CPU内存。通过在相邻设备之间同时运行环的多个段，这也利用了树形结构的互连。 我们使用MPI发送和接收以及CUDA内核构建了实施方案，以进行基本操作。

表7比较了OpenMPI 1.8.5版提供的所有减少实现的性能。 我们报告了使用5层，3递归层架构以及2560个隐藏单元的所有图层在英语数据集上进行一个完整时期的完整训练所花费的全部缩减时间。 在此表中，我们使用每个GPU的64个最小批处理，并随着我们扩展到更多GPU而扩展了算法最小批处理。 我们看到，当通信在一个节点（8个GPU或更少）内时，我们的实现比OpenMPI的实现要快得多。 随着我们增加GPU的数量并增加节点间通信的数量，尽管我们的实现速度仍快2-4倍，但差距在缩小。

我们所有的训练都使用8个或16个GPU运行，在这种情况下，与直接使用OpenMPI相比，我们的全部缩减实施方式导致整个训练运行的速度提高了2.5倍。 因此，优化全约缩减可为我们的实验带来重要的生产力提高，并使我们的简单同步SGD方法可扩展。

4.2 GPU实现CTC丢失功能

计算CTC损失函数比在我们的RNN架构上执行正向和反向传播更为复杂。最初，我们将激活从GPU转移到CPU，然后我们使用CTC的OpenMP并行实现来计算损失函数。但是，由于两个原因，此实现极大地限制了我们的可伸缩性。首先，随着我们提高RNN本身的效率和可伸缩性，它在计算上变得更加重要。 其次，在CPU和GPU之间传输大型激活矩阵要求我们花费用于CTC的互连带宽，而不是在传输梯度矩阵上允许我们使用数据并行性扩展到更多处理器。

为了克服这个问题，我们编写了CTC损失函数的GPU实现。我们的并行实现依赖于轻微的重构以简化CTC计算中的依赖性，以及使用ModernGPU 中优化的并行排序实现。我们将在附录中提供有关此并行化的更多详细信息。

表8比较了两种CTC实施的性能。 GPU实现为我们每个时期节省了95分钟在英语条件下， 25分钟在普通话条件下。这样可以将总体训练时间减少10-20％，这对于我们的实验也是一个重要的生产力。

4.3内存分配

我们的系统频繁使用动态内存分配给GPU和CPU的内存，主要用于存储激活数据以实现可变长度的语音和中间结果。个体分配可能非常大； 超过1 GB的最长语音通话时间。对于这些非常大的分配，我们发现CUDA的内存分配器甚至std :: malloc在我们的应用程序中引入了显着的开销—在某些情况下，使用std :: malloc的速度降低了2倍。这是因为cudaMalloc和std :: malloc都将非常大的分配转发给操作系统或GPU驱动程序以更新系统页表。对于运行多个应用程序，都共享内存资源的系统来说，这是一个很好的优化，但是对于我们的系统而言，编辑页面表是纯粹的开销，因为我们的节点完全专用于运行单个模型。为了解决这个限制，我们为CPU和GPU分配编写了自己的内存分配器。我们的实现遵循jemalloc中最后一级共享分配器的方法：使用伙伴算法从连续的内存块中划分出所有需要分配的内存。为避免碎片，我们在训练开始时预先分配了所有GPU内存，并从该模块中细分了各个分配单元。同样，我们将转发给mmap的CPU内存块大小设置为实质上大于std :: malloc，为12GB。

训练深层循环网络所需的大多数内存都用于存储通过每个层的激活，以供反向传播使用，而不用于存储网络的参数。例如，存储具有9层的70M参数网络的权重大约需要280 MB的内存，但是存储一批64次，七秒钟的话语的激活则需要1.5 GB的内存。TitanX GPU包含12GB的GDDR5 RAM，当处理较长的语音时，有时很深的网络可能会超过GPU的存储容量。 这可能是不可预料的，特别是当发声长度的分布包含异常值时，并且希望避免发生这种情况时发生灾难性故障。当请求的内存分配超过可用的GPU内存时，我们将改为使用cudaMallocHost分配页面锁定的GPU内存映射的CPU内存。 通过在PCIe上以减少的带宽转发单个内存事务，GPU可以直接访问该内存，并且即使遇到异常，模型也可以继续运行。

快速的内存分配与后备机制相结合，使我们在特殊情况下可以稍微超调可用的GPU内存，从而使系统显着更简单，更强大，更高效。