进展报告 ——场景文本识别

霍超凡

2019 年 10 月 8 日

1 工作总述

一个月前，老师布置两篇论文，这两篇论文是关于场景文本识别的，刚开始接触这类场景文本识别问题，读论文的过程中出现很多问题。这两篇论文不是关于单一技术论述，它融合了前人的很多研究结果，如果不对这类问题的学术环境熟悉，很难读懂。我从第一篇论文的参考文献中找到几篇比较具有代表性的文章来读，读完之后再会到原文章，发现读起来轻松许多，大概第一个星期开始逐步看懂网络结构，第二个星期开始看源码，并尝试跑通源程序，CRNN网络中用到CTC算法，需要安装warp-ctc，我在Windows上尝试了很多方法，没有成功。我放下代码，开始看第二篇论文，第二篇论文相比第一篇的结构复杂些，比较难以理解的是前面的文本定位网络和图片矫正算法，我试图把TPS算法搞明白，这个算法原理涉及到其他领域的知识较多，最终我放弃搞明白这个算法。第三个星期我还在纠结warp-ctc的配置，warp-ctc 按照一篇博客的步骤已经编译成功，在将它绑定到pytorch中出现了问题，经过数次倒腾，我最终下了安装Ubuntu 双系统的决心，从Ubuntu的安装到NVIDA显卡驱动的配置，再到CUDA安装、python环境的重新搭建，我又花费了一个星期。临至月末我把第一篇论文的程序跑通并测试了几个数据集，由于自己相关参数设置不正确，网络始终维持在一个损失值较大的状态，原因是由于网络中图片尺寸变换设置问题和忽略大小写（统一将a和A视为a）问题。我国庆假期在几个数据集跑了跑程序，测了些数据，得到的结果不太好，可能是数据集的原因或者是我相关参数设置不正确。第二篇论文网络的结构大致明白，前面的文本检测网络稍有糊涂，相对应的程序规模相对较大，需要再花费些时间来读程序。经过一个月后仅仅完成一篇论文，速度太慢，需要加紧对自己的督促，还有多和老师联系，即使没做完，向老师汇报自己的进展也是对自己的一种督促。最后将自己的想法和实验结果形成如下报告。

在第二节中主要简述论文中所用到的技术和我对这些技术的理解，初入这个领域，难免一叶障目，这些想法或许不太成熟。这些技术包括RNN 的结构及其工作机制，RNN的其他复杂结构LSTM、GRN，还有CTC算法和结合Attention机制的序列矫正对齐网络。第三节简述场景文本检测的两种方法。第四节介绍实验环境以及实验结果。

2 从RNN到LSTM

2.1 RNN

基于MP模型所构建的神经网络具有非时序性，MP模型假设当前神经元状态不受之前状态影响，神经元是否兴奋只与输入和神经元的自身特性有关，这一假设使其在处理序列化数据方面存在缺陷。这种神经网络还具有静态特性，一旦网络构建完成之后，其输入、输出被局限在固定大小，这种限制使其在实际应用中需要增添预处理步骤，通过裁剪、压缩、插值等方法将输入数据变换为所需大小，符合规定尺寸的数据才能被网络接受、处理。不同于传统神经元模型的RNN宽松了这些限制，RNN将网络历史输出信息重新输入网络，使当前输出与历史输入建立联系，适合处理

序列化数据。而且输入RNN网络的数据可以在时间维度上无限延伸，支持处理不定长的序列数据。这些优势使RNN在处理诸如文本、声音等序列化数据方面具有独到之处。

2.1.1 RNN结构

不同文献对RNN结构作出不同的阐述，文献[2]中用式(1)对RNN 结构作出概括性说明。

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{1}$$

这里f是一个非线性的激活函数，它可能是简单的$sigmoid$函数，抑或是更加复杂的LSTM单元。网络中包含若干隐含状态$\mathbf{h}$，$\mathbf{h}$与当前输入和上一状态有关，输入是序列$\mathbf{x}$，输出可以是网络的内部状态$\mathbf{h}$，也可以是它投影到另外一个空间的向量$\mathrm{W_{hy}}\mathbf{h}$。RNN可以将一个序列映射到另外一个序列，适合处理序列化的数据。

不同文献在描述RNN时，所使用的符号千差万别，RNN的结构也稍有不同，我将我所看到的几种描述总结如下。

[13]中将网络的输入分成两部分，一部分是从外界的输入$\mathbf{x}^{net}(t)$，另一部分是内部输入$\mathbf{y}(t)$，$\mathbf{x}^{net}(t)$是一个m维向量，t表示在t时刻输入，设RNN内部节点个数为n，$\mathbf{y}(t)$是t时刻的输出，$\mathbf{y}(t)$向量有n个分量，每一个分量表示RNN内部单个节点的输出。为了采用同一个记号表示网络输入，定义

$$x_k(t) = \begin{cases} x_k^{net}(t) & if\ k \in I \\ y_k(t) & if\ k \in U \end{cases} \tag{2}$$

将$\mathbf{x}^{net}(t)$和$\mathbf{y}(t)$连接形成一个m+n维向量$\mathbf{x}(t)$，I是$\mathbf{x}(t)$的分量取自于$\mathbf{x}^{net}(t)$的下标集合，U是$\mathbf{x}(t)$的分量与$\mathbf{y}(t)$对应的下标集合，不妨取$I = \{1, 2, \cdot, m\}, U = \{m+1, m+2, \cdot, m+n\}$。网络中第$k$个节点在$t$时刻的输入定义为[1]

$$s_k(t) = \sum_{l \in I} w_{kl} x_l^{net}(t) + \sum_{l \in U} w_{kl} y_l(t-1) = \sum_{l \in U \cup I} w_{kl} x_l(t) \tag{3}$$

该节点在t时刻的输出为

$$y_k(t) = f_k(s_k(t)) \tag{4}$$

函数f是激活函数，通常取$sigmoid$函数将输入压缩到0和1之间，有时也使用$tanh$函数将输入压缩到-1和1之间。将式(3)(4)写成矩阵的形式得到

$$\mathbf{y}(t) = f(\mathrm{W}_{xh}\mathbf{x}^{net}(t) + \mathrm{W}_{yh}\mathbf{y}(t-1)) \tag{5}$$

当输出向量的维度和神经网络内部隐藏节点数目不同时，需要修改上式，引入隐藏状态以区分输出和内部迭代

$$\mathbf{h}(t) = f(\mathrm{W}_{xh}\mathbf{x}^{net}(t) + \mathrm{W}_{hh}\mathbf{h}(t-1)) \tag{6}$$

输出为内部状态的一个线性组合

$$\mathbf{y}(t) = \mathrm{W}_{hy}\mathbf{h}(t) \tag{7}$$

有些文献喜欢使用$\mathbf{y}_t$来表示在t时刻的输出，这和向量的第t个分量记号相冲突，但是将t标记为下标会带来式子的简洁表达，根据上下文或字母是否是粗体来确定$\mathbf{y}_t$说明的是在t时刻的输出向量

[1]原文使用的是$s_k(t+1) = \sum_{l \in I} w_{kl} x_l^{net}(t) + \sum_{l \in U} w_{kl} y_l(t) = \sum_{l \in U \cup I} w_{kl} x_l(t)$，原作者将数据通过神经元节点作为主要延迟，输入向量$\mathbf{x}(t)$经过网络后得到输出$\mathbf{y}(t+1)$，时间戳在经过节点后增1。我们这里和原作者稍有不同，我们认为输入$\mathbf{x}(t)$对应的输出为$\mathbf{y}(t)$，将输入经过网络得到输出整个过程视为一个时间单元。这样是为了强调输入$\mathbf{y}(t-1)$来自过去的输出。

还是$\mathbf{y}$的第t个分量。下面的记号采用这两种方式，当需要表示第t时刻的输出向量时，使用$\mathbf{y}_t$表示，当需要强调其第k个分量时，使用$y_k(t)$表示。

如果网络输出向量$\mathbf{y}$的维度和内部节点$\mathbf{h}$的维度相同，让W_{hh}和W_{hy}共享参数，得到

$$\mathbf{y}(t) = W_{hy} f(W_{xh}\mathbf{x}^{net}(t) + \mathbf{y}(t-1)) \tag{8}$$

式(8)和[4]所提出的万能逼近器的结构比较相似，写成式(8)形式将会为我们后面分析RNN的工作机理提供便利。式(8)减少了参数数量，$\mathbf{y}(t-1)$ 重新输入到网络中又不会改变模型的时序性，可以达到和原来相同的性能。

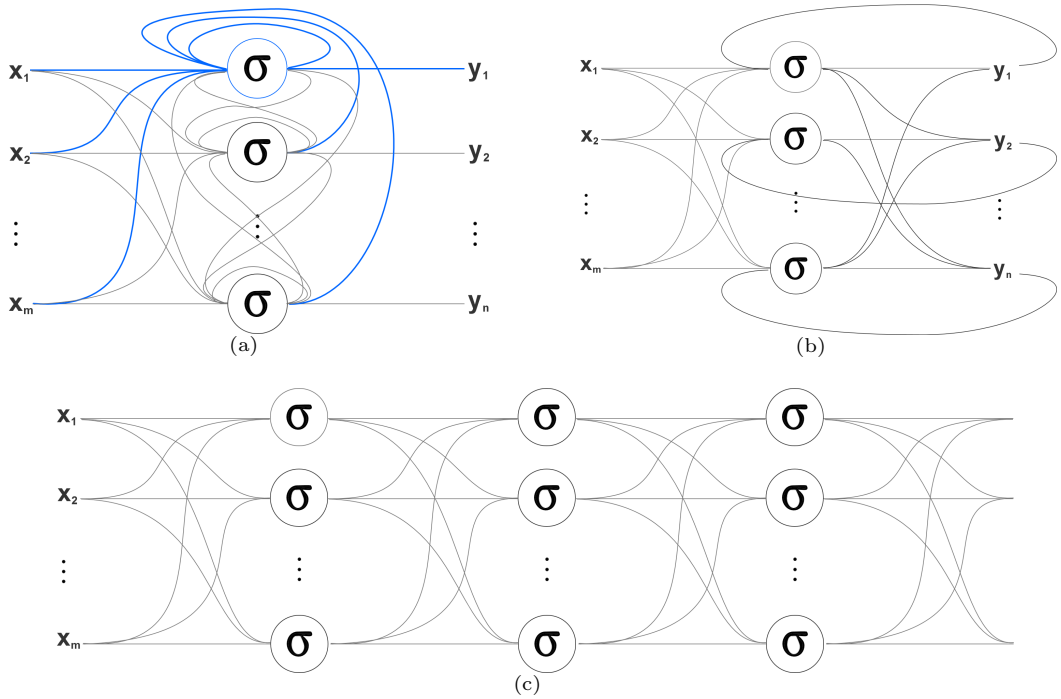


图 1: 1(a)结构中，隐藏层直接作为输出，这与式(5)相符合，1(b)结构与式(8)相对应，无论是1(a)还是1(b)按照时间展开都可以表示成1(c)结构

2.1.2 RNN工作机理

RNN的静态稳定性和动态时序性备受关注，静态稳定性是指当RNN没有接收到信号输入或输入一个恒定信号时，在迭代若干次后，总能达到一个稳定状态。RNN的动态时序性是指当连续序列输入后，RNN能够产生与之相匹配的序列输出。下面结合图形分析一维情形的RNN内部工作机理，稍加修改可以扩展到更高维度。

静态稳定性 当RNN没有接受到输入信号或者结构恒定的输入信号时，在一定条件下总能达到一个稳定状态。考虑只含有一个内部神经元节点的网络结构，在时刻t网络的输出为

$$y(t) = \sigma(wy(t-1) + b)$$

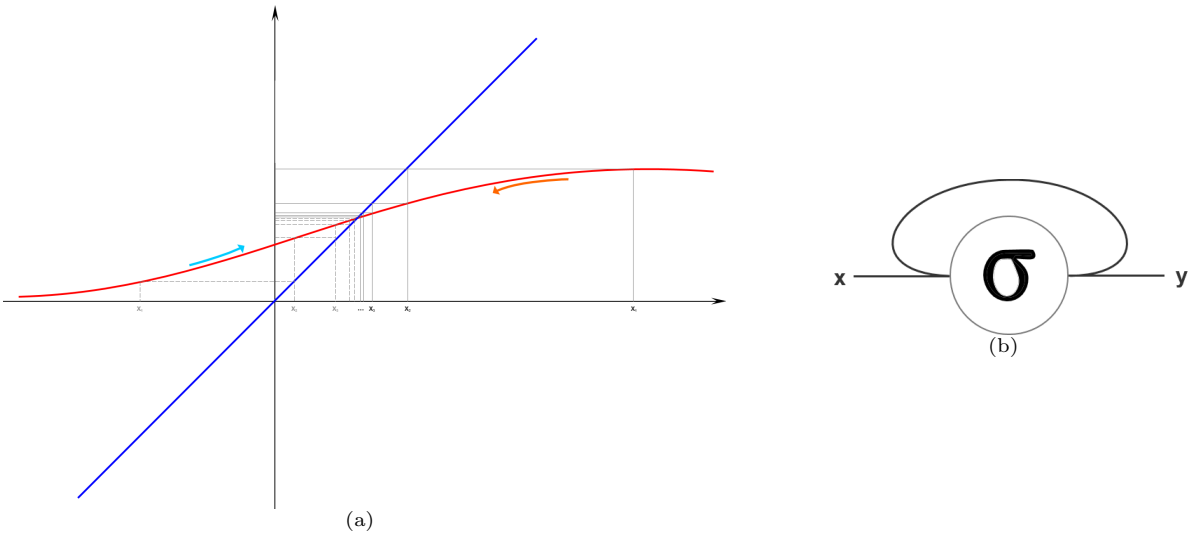
为了方便讨论，将$\sigma(wx + b)$记为$\sigma'(x)$，$\sigma'(x)$与$\sigma(x)$相比，两者在图像上具有相同的增减趋势，$\sigma'(x)$可在$\sigma(x)$图像的基础上经过横坐标的平移和缩放得到。于是，输入恒定的RNN可用如下式(9)刻画

$$y(t) = \sigma'(y(t-1)) \tag{9}$$

在数学上，习惯称式(9)为一个迭代方程，该时序方程存在稳定点x^*，且这个稳定点满足条件

$$\begin{cases} x^* = \sigma'(x^*) \\ \frac{d\sigma'}{dx}|_{x=x^*} \leqslant 1 \end{cases}$$

在图像上表现为，曲线$y = \sigma'(x)$与直线$y = x$的交点。如图2所示，初始点位于$(x_1, \sigma'(x_1))$，经过若干次迭代后，初始点逐步由$(x_2, \sigma'(x_2)), (x_3, \sigma'(x_3)), \cdots$趋向于$(x^*, x^*)$。函数$\sigma'(x)$具有一个稳定点，且稳定点的收敛域为$(-\infty, +\infty)$，不管从哪个点开始迭代，最终总会达到稳定点周围，从此可以看出这个神经元具有了记忆特性，记忆存储在权重w和偏置b中。



(a)　　(b)

图 2: 2(b)是式(9)对应的网络拓扑结构，2(a)在图形上显示了该结构在实际运转的过程

将激活函数$\sigma'(x)$稍加改造，将会得到一个具有多稳定点的神经元结构，为了得到更加普遍的结论，这里使用含有n个内部节点单隐层感知器代替传统激活函数，网络结构如图3(b)所示，记

$$f(x) = \sum_{i=1}^{n} w_i^{out} \sigma\left(w_i^{in} x + b_i\right) \tag{10}$$

其中w_i^{in}是连接输入节点和第i个内部节点的权重，w_i^{out}是连接第i 个内部节点和外部节点的权重。如果没有外部输入，该网络的时序方程变为

$$y(t) = f(y(t-1))$$

根据[4]通过改变节点数目和内部参数，函数$f(x)$可以以任意精度逼近任意函数。这样我们可以通过所需构造一个具有多个稳定状态的网络，这个网络中，任意点经过若干次迭代后都会达到和自己相邻的稳定点。并且从外界输入一个信号，可以使网络从一个状态转为另一个状态。

通过以上一维情况的讨论，稍加修改可扩展到n维。设f是从$\mathbb{R}^n$到$\mathbb{R}^n$ 的映射，时序方程

$$\mathbf{y}(t) = f(\mathbf{y}(t-1))$$

存在稳定点x^*的条件是方程组

$$\begin{cases} f(\mathbf{x}^*) = \mathbf{x}^* \\ \frac{\partial f}{\partial x_i}|_{x_i=x_i^*} < \mathbf{1}^n, \ i = 1, \cdots, n \end{cases}$$

有解，式中$\mathbf{1}^n$表示全为1的n为向量。

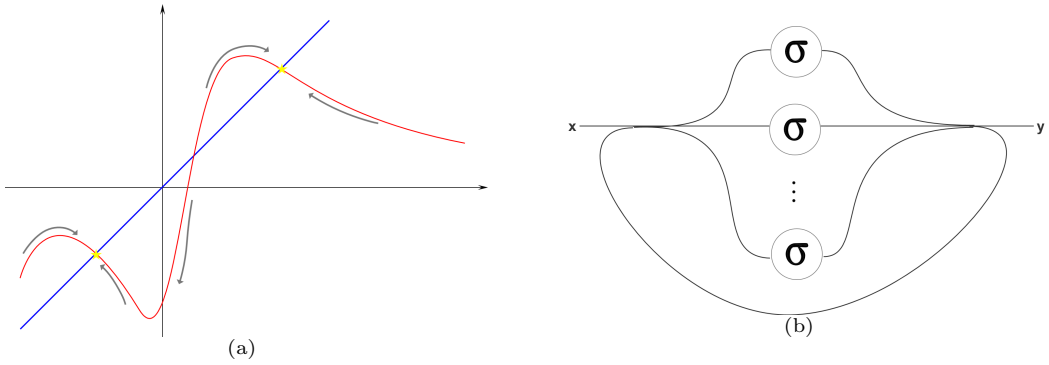


图 3: 图3(b)是式(10)对应的网络拓扑结构，图3(a)是其在图像的动态刻画。

动态时序性 RNN可以处理从序列到序列的映射，即输入序列为$\{\mathbf{x}_t, \mathbf{x}_{t+1}, \cdots, \mathbf{x}_{t+\tau}\}$，目标输出序列为$\{\mathbf{d}_t, \mathbf{d}_{t+1}, \cdots, \mathbf{d}_{t+\tau}\}$。这里我们仍然假设输入序列均为1 维标量。构建RNN结构如下

$$y(t) = f_{net}(x(t) + y(t-1))$$

在图4中画出曲线$y = f_{net}(x)$和$y = x$，标记若干由输入序列和输出序列所构成的点。我们可以看到，RNN内部工作方式在图像上表现为在点$(x_t, y_t), (x_{t+1} + y_t, y_{t+1}), ,, (x_{t+\tau} + y_{t+\tau-1}, y_{t+\tau})$之间的移动。

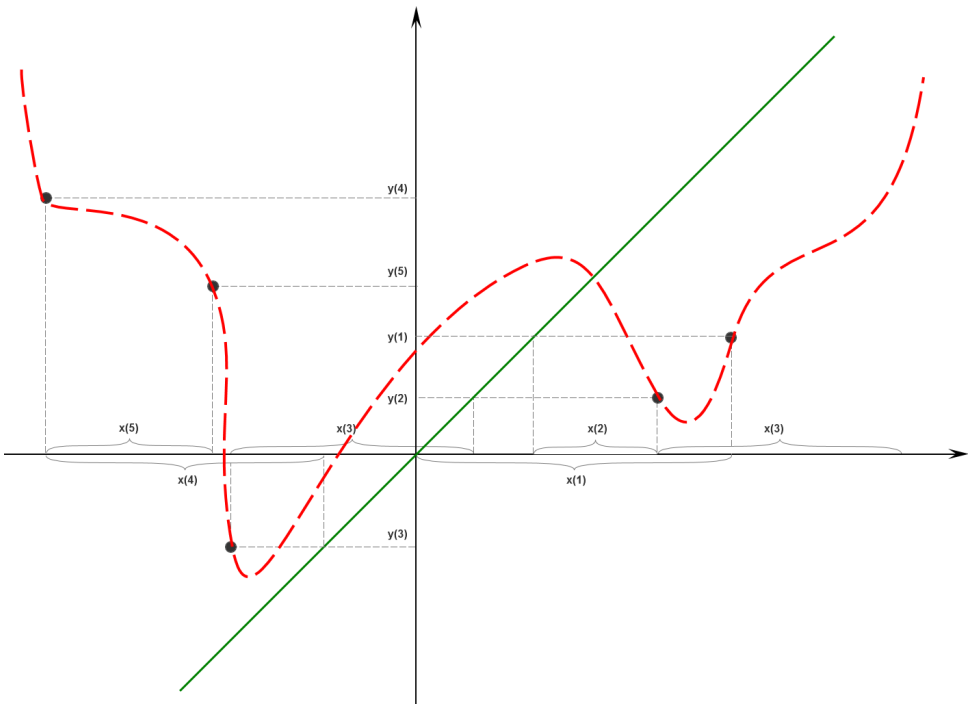


图 4: 红曲线表示曲线$y = f_{net}(x)$，绿实线表示直线$y = x$。从输入点$x = x(1)$ 开始，$x(1)$经过RNN之后得到输出$y(1)$，作曲线$y = y(1)$与直线$y = x$ 相交于点$(y(1), y(1))$，此点横坐标移动$x(2)$个单位后得到下一个输入点$x = x(2) + y(1)$，如此循环。

从序列到序列的映射在图像上表示为对多个点的拟合。为了得到从序列$\{\mathbf{x}_t, \mathbf{x}_{t+1}, \cdots, \mathbf{x}_{t+\tau}\}$到序列$\{\mathbf{d}_t, \mathbf{d}_{t+1}, \cdots, \mathbf{d}_{t+\tau}\}$ 映射的网络，我们需要构造一个函数$f_{net}(\mathbf{x})$，使其能够穿过点$(\mathbf{x}_t, \mathbf{d}_t), (\mathbf{x}_{t+1} + \mathbf{d}_t, \mathbf{d}_{t+1}), \cdots, (\mathbf{x}_{t+\tau} + \mathbf{d}_{t+\tau-1}, \mathbf{d}_{t+\tau})$，由此我们将序列映射问题转换为回归问题。序列映射关键在

于函数$f_{net}(\mathbf{x})$ 的选择，根据[4] 对万能逼近器讨论，使用含有单隐层的感知器来构建$f_{net}(\mathbf{x})$，即

$$f_{net}(\mathbf{x}) = \sum_{i=1}^{n} w_i^{out} \sigma(\sum_{j=1}^{n} w_{ij}^{in} x_j + b_i)$$

2.1.3 RNN训练

[13]提出BPTT算法用于训练RNN，算法的基本思想是将RNN网络按照时间展开成一个深层网络，之后用传统的BP算法对这个网络反向传播误差进行计算。文章中作者区分连续式训练和阶段式训练，并对应每一种情况提出实时BPTT算法和阶段式BPTT算法，这里我们不考虑这么复杂的情况。这里讨论的BPTT算法针对的是序列到序列映射的RNN，误差在全部序列经过网络一遍后再计算误差和梯度并反向传播。

将长度为T的序列$\{\mathbf{x}_\tau\}_{\tau=1}^{T}$输入网络，期望得到指定的输出序列$\{\mathbf{d}_\tau\}_{\tau=1}^{T}$，将实际输出$\{\mathbf{y}_\tau\}_{\tau=1}^{T}$和目标输出对比，定义误差序列$\{\mathbf{e}_\tau\}_{\tau=1}^{T}$

$$e_k(t) = y_k(t) - d_k(t) \tag{11}$$

在时刻t的误差

$$J(t) = \frac{1}{2} \sum_k [e_k(t)]^2 \tag{12}$$

总误差

$$J^{total}(w) = \sum_{t=1}^{T} J(t) \tag{13}$$

训练网络的目标是通过调整参数w最小化$J^{total}(w)$。

不妨设$I = \{1, 2, \cdots, m\}, U = \{m+1, m+2, \cdots, m+n\}$，那么式(3)(4)变为

$$y_k(t) = \sigma\left(\sum_{i=1}^{m} w_{ki} x_i^{net}(t) + \sum_{j=m+1}^{m+n} w_{kj} y_j(t-1)\right) \tag{14}$$

由于权值共享的缘故，同一个w_{ij}在不同时刻均会对$J(t)$造成不同的影响。我们记$\frac{\partial J(t)}{\partial w_{ij}^{\tau}}$为$t$时刻的误差函数$J(t)$对$\tau$时刻的权重$w_{ij}$求偏导,它反映了$w_{ij}$在时刻$\tau$对$J(t)$的影响。$w_{ij}$对$J(t)$的总影响是从初始时刻到时刻$t$的累积，即

$$\frac{\partial J(t)}{\partial w_{ij}} = \sum_{\tau=1}^{t} \frac{\partial J(t)}{\partial w_{ij}^{\tau}} \tag{15}$$

根据式(12)，权重在t_1时刻对$J(t_2)$的影响

$$\frac{\partial J(t_2)}{\partial w_{ij}^{t_1}} = \sum_{k=1}^{n} e_k(t_2) \frac{\partial y_k(t_2)}{\partial w_{ij}^{t_1}} \tag{16}$$

由式(14)，

$$\frac{\partial y_k(t_2)}{\partial w_{ij}^{t_1}} = \sum_{\ell=m+1}^{m+n} y_k(t_2)(1 - y_k(t_2)) w_{k\ell} \frac{\partial y_\ell(t_2 - 1)}{\partial w_{ij}^{t_2}} \tag{17}$$

如果$i = \ell, j \in I$，那么

$$\frac{\partial y_\ell(t_1)}{\partial w_{ij}^{t_1}} = y_\ell(t_1)(1 - y_\ell(t_1)) x_j^{net}(t_1) \tag{18}$$

如果$i = \ell, j \in U$，那么

$$\frac{\partial y_\ell(t_1)}{\partial w_{ij}^{t_1}} = y_\ell(t_1)(1 - y_\ell(t_1)) y_j(t_1) \tag{19}$$

如果$i \neq \ell$，那么

$$\frac{\partial y_\ell(t_1)}{\partial w_{ij}^{t_1}} = 0 \tag{20}$$

式(18)(19)(20)是迭代方程(17)的终止条件，通过不断迭代式(17)可以求得权重w_{ij}在时刻t_1对第k个节点在时刻t_2的输出$y_k(t_2)$的影响，再带到式(16)求得从时刻t_2传播到时刻t_1的误差。式(15)说明，时刻t的误差会反向传播到之前的任何时刻；而在t_1时刻的权重会影响到之后任何时刻的输出，根据式(13)(15)总误差对其求偏导

$$\frac{\partial J^{total}(w)}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial J(t)}{\partial w_{ij}} = \sum_{t=1}^{T} \sum_{\tau=1}^{t} \frac{J(t)}{\partial w_{ij}^{\tau}} \tag{21}$$

最后权重按照下式更新

$$w_{ij} = w_{ij} - \eta \frac{\partial J^{total}(w)}{\partial w_{ij}} \tag{22}$$

式中，η为学习率。

式(11)-(22)完整描述了BPTT算法的过程，这里采用的符号和原文稍有不同，也没有采用矩阵表示方法，我认为比较简洁，容易理解。

2.1.4 传统RNN结构的缺陷

文献[13]分析了当时间跨度较大时，误差很难传播到靠前的状态，如同训练深度网络，会面临梯度爆炸或梯度衰减。另外一方面不同时间状态内部共享参数，这会出现参数飘忽不定的问题，对于同一个权重w，误差传播到t_1状态时，需要w增大，而传播到t_2状态，又需要w减少，这种冲突会降低训练的速度，这并不是主要问题，所有参数共享都会面临这样的问题。传统RNN最大的问题在于，它的灵活性太差，它只能完成从一个序列到另一个序列的精确映射，如果输入序列稍有波动，输出序列就出现大幅度误差，得不到我们想要的结果。我们所处理的数据往往都含有噪声，这些带有噪点的序列可分为如下几类

1. 序列起始点不定，实际应用中，输入的序列前端会存在太多其它无关序列，比如要输入abcd，而实际输入的序列为#&âbcd。
2. 序列内部存在噪声或空缺，比如要输入abcd，而实际输入的序列为abc%d或a _ bcd。
3. 序列内部状态重复，abcd，abbbcd。

这要求我们的网络能够自动对齐序列，能够处理带有噪声的序列。基于门的RNN能够解决这些问题。

2.2 LSTM

传统RNN因其不能够建立长期联系，在训练上存在梯度弥散问题，一种称为LSTM网络结构被提出，LSTM最早在[8]中提出，最初是为了解决误差反向传播中梯度弥散的问题，LSTM 便于建立长期联系，误差能够尽可能的传播到靠前的状态，这使之能够处理类似a#&b之间带有较长噪声的序列，同时对类似abcdefg 和zbcdefg 序列具有较强的辨析能力。LSTM的三个门控制数据的输入、状态的更新和状态的输出。

2.2.1 LSTM的结构

不同文献对LSTM的结构的描述存在差异，[8]提出LSTM的鼻祖版本。设输入序列$\{\mathbf{x}_t\}_{t=1}^T$，网络内部状态序列$\{\mathbf{s}_t\}_{t=1}^T$，输出序列$\{\mathbf{y}_t\}_{t=1}^T$。和之前采用的符号一样，$\mathbf{x}_t$和$\mathbf{x}(t)$都表示在t时刻的输入数据，它可能是一个向量也可能是一个标量。定义输入门

$$\mathbf{i}_t = f_{in}(\mathbf{x}_t^{in}) \tag{23}$$

输出门

$$\mathbf{o}_t = f_{out}(\mathbf{x}_t^{out}) \tag{24}$$

我们用$\mathbf{x}_{t-1}^{in}$,$\mathbf{x}_{t-1}^{out}$表示输入门和输出门在时刻t的输入，$\mathbf{i}_t$,$\mathbf{o}_t$表示输入门和输出门在时刻t的输出。f_{in} ,f_{out}分别是输入门和输出门的激活函数，一般选用$sigmoid$函数，$sigmoid$函数默认是从标量到标量之间的函数，如果$sigmoid$函数的输入为一个向量，表示对输入向量中的每一个分量分别应用$sigmoid$函数，这里在符号上宽松是为了得到更加简洁的式子。

作者在原文中说输入单元、输入输出门单元、内部隐藏单元、输出单元都可以作为输入门和输出门的输入，所以

$$\mathbf{x}_t^{in} = W_{xi}\mathbf{x}_t + W_{yi}\mathbf{y}_{t-1} + W_{ii}\mathbf{i}_{t-1} + W_{oi}\mathbf{o}_{t-1} + W_{si}\mathbf{s}_{t-1} \tag{25}$$

$$\mathbf{x}_t^{out} = W_{xo}\mathbf{x}_t + W_{yo}\mathbf{y}_{t-1} + W_{io}\mathbf{i}_{t-1} + W_{oo}\mathbf{o}_{t-1} + W_{so}\mathbf{s}_{t-1} \tag{26}$$

其中W_{xy}为从单元x连接到单元y的权重，下同。

内部状态的更新

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathbf{i}_t \cdot f_{net}(\mathbf{x}_t^s) \tag{27}$$

隐藏状态单元的输入

$$\mathbf{x}_t^s = W_{xs}\mathbf{x}_t + W_{ys}\mathbf{y}_{t-1} + W_{is}\mathbf{i}_{t-1} + W_{os}\mathbf{o}_{t-1} + W_{ss}\mathbf{s}_{t-1} \tag{28}$$

输出

$$\mathbf{y}_t = \mathbf{o}_t \cdot g_{net}(W_{sy}\mathbf{s}_t) \tag{29}$$

实际的LSTM网络在拓扑结构上是十分复杂的，原作者似乎没有仔细思考输入与输出的关系，尽可能将所有单元输出连接到输入，这种暴力式的设计方法可以避免的复杂的分析，但是给后人理解网络结构带来不便。

现在普遍接受的一种LSTM结构在原有的LSTM结构中增加了一个遗忘门，并去掉“多余”的连接。输入门(式25)和输出门(式26)的输入变为

$$\mathbf{x}_t^{in} = W_{xi}\mathbf{x}_t + W_{yi}\mathbf{y}_{t-1} + W_{si}\mathrm{s_{t-1}} \tag{30}$$

$$\mathbf{x}_t^{out} = W_{xo}\mathbf{x}_t + W_{yo}\mathbf{y}_{t-1} + W_{so}\mathrm{s_{t-1}} \tag{31}$$

遗忘门的输入

$$\mathbf{x}_t^{forget} = W_{xf}\mathbf{x}_t + W_{yf}\mathbf{y}_{t-1} + W_{sf}\mathrm{s_{t-1}} \tag{32}$$

遗忘门

$$\mathbf{f}_t = f_{forget}(\mathbf{x}_t^{forget}) \tag{33}$$

内部状态的更新

$$\mathbf{s}_t = \mathbf{f}_t \cdot \mathbf{s}_{t-1} + \mathbf{i}_t \cdot f_{net}(\mathbf{x}_t^s) \tag{34}$$

$$\mathbf{x}_t^s = W_{xs}\mathbf{x}_t + W_{ys}\mathbf{y}_{t-1} \tag{35}$$

输出(式29)不变。

以上各式均省去偏置项，$f_{in}, f_{out}, f_{forget}$激活函数通常选用$sigmoid$函数，而$f_{net}, g_{net}$使用$tanh$函数。在讨论一般网络结构时，LSTM的输出通常不是真正意义上的网络的输出，一般把LSTM的输出$\mathbf{y}_t$ 写作LSTM的隐藏状态$\mathbf{h}_t$，$\mathbf{s}_t$写作$\mathbf{c}_t$。这里我们把LSTM单独从网络中剥离出来，将LSTM网络的输出记为$\mathbf{y}_t$，在讨论其它和LSTM相连的其它网络结构时需要更改以上记号。

2.2.2 LSTM各个门的作用

输入门控制信息的流入，屏蔽无关噪声。

输出门控制信息的流出，当网络还没有达到预期状态时，输出门处于关闭状态，当网络状态达到成熟，输出门打开，信息流出。

遗忘门实时重置网络，当网络连续处理多个序列时，前一个序列输入完成后，遗忘门关闭，为接受新的序列而遗忘之前的状态。

2.3 GRN

文献[2]在LSTM基础上做出简化得到GRN，GRN去掉LSTM的输出门，输入门和遗忘门合并成更新门，增添重置门充当遗忘门的作用。更新门

$$\mathbf{z}_t = \sigma(W_{xz}\mathbf{x}_t + W_{yz}\mathbf{y}_{t-1}) \tag{36}$$

重置门

$$\mathbf{r}_t = \sigma(W_{xr}\mathbf{x}_t + W_{yr}\mathbf{y}_{t-1}) \tag{37}$$

输入

$$\mathbf{x}_t^{in} = \phi(W_{xs}\mathbf{x}_t + W_{ys}(\mathbf{r_{t-1}} \cdot \mathbf{y}_{t-1})) \tag{38}$$

输出

$$\mathbf{s}_t = \mathbf{z}_t \cdot \mathbf{s_{t-1}} + (1 - \mathbf{z}_t) \cdot \mathbf{x}_t^{in} \tag{39}$$

$$\mathbf{y}_t = \mathbf{s}_t \tag{40}$$

更新门控制数据流的流进，遇到噪声GRN的更新门关闭，保持现有状态。重置门在两个两个序列之间重置状态。去掉输出门是有一定原因的，对于序列到序列之间的映射我们需要得到整个内部状态的变化，使用GRN较合适，如果我们不关心这个状态变化过程，而只关心序列识别的结果，加上输出门可以屏蔽不必要的信息。

不管是LSTM还是GRN所使用的门结构都类似于单层神经元结构，单层神经元只能处理线性可分的数据，这种结构具有较弱的表达能力。将门看成一个分类器，遇到特定的输入输出1，表示门打开，遇到没有预期到的输入时输出0，门关闭。门的引入使RNN具有抗噪能力，能够处理类似于#&abcd，abbbcd，ab__c#d的数据，增加网络的鲁棒性。

2.4 序列转录(transduction)

加入噪声的序列经过RNN得到的序列存在稀疏性，比如&%#abbbc&%d经过基于门的RNN处理得到* * * * a b * * c * * d。我们不知道目标序列在输出的序列中何时开始，序列中间也没有空缺。需要进一步的处理，文献[6]提出了一种处理这种稀疏序列的方法，这种方法将序列出现位置的所有可能性均考虑在内，加和得到总概率。经过RNN得到的序列为$\{\mathbf{y}_t\}_{t=1}^T$，它将作为CTC算法

的输入。目标序列为$\mathbf{z}=(z_1,z_2,\cdots,z_U)$， $\mathbf{z}$的长度至多和输入序列一样长。设序列的字典集$\mathcal{L}$大小为ℓ，取$\mathcal{L}'=\mathcal{L}\cup\varnothing$，向量$\mathbf{y}_t$ 的维度为$\ell+1$，每一个分量代表取对应label的概率。设长度为T的向量$\boldsymbol{\pi}$，$\pi_k\in\{1,2,\cdots,\ell+1\}$，它代表在字典集$\mathcal{L}'$上的一个序列，若$\pi_k=r$表示$\boldsymbol{\pi}$ 所对应的序列的第k个位置为第r个label。将输入序列转译成序列$\boldsymbol{\pi}$ 的概率为

$$p(\boldsymbol{\pi}|\mathbf{y})=\prod_{t=1}^{T}y_{\pi_t}(t) \tag{41}$$

设映射β将长度为T的序列$\boldsymbol{\pi}$映射到长度为U的目标序列$\mathbf{z}$，逆映射β^{-1}将目标序列$\mathbf{z}$映射到稀疏序列$\mathbf{z}$，这个映射β规则如下

1. 重复连续出现$\mathbf{z}$中若干label的$\boldsymbol{\pi}$映射为$\mathbf{z}$，例如$\beta(abbbcd)=abcd$。
2. 在$\mathbf{z}$中连续插入若干空白的$\boldsymbol{\pi}$映射为$\mathbf{z}$，例如$\beta(a_bcd)=abcd$。
3. 把z根据(1)(2)规则有限次组合得到的$\boldsymbol{\pi}$被映射为$\mathbf{z}$，例如$\beta(aaab_cd)=abcd$。

我们定义把$\mathbf{y}$转译成$\mathbf{z}$的概率

$$p(\mathbf{z}|\mathbf{y})=\sum_{\pi\in\beta^{-1}(x)}p(\boldsymbol{\pi}|\mathbf{y}) \tag{42}$$

将$\mathbf{y}$对应的可能性最大的序列作为我们最终转译序列$\mathbf{z}^*$，即

$$\mathbf{z}^*=\arg\max_{\mathbf{z}}\ p(\mathbf{z}|\mathbf{y}) \tag{43}$$

所有可能的序列$\boldsymbol{\pi}$非常多，其数目随着label个数的增长呈指数型增长，对于label数非常多的序列，上面的方法效率会非常低。借助字典可以有效减少枚举$\boldsymbol{\pi}$的数目，或根据最大效应方法选择$\boldsymbol{\pi}$，

$$\boldsymbol{\pi}^*=\arg\max_{\boldsymbol{\pi}}\ p(\boldsymbol{\pi}|\mathbf{y}) \tag{44}$$

$\boldsymbol{\pi}^*$的第k个分量等于向量$y(k)$中响应最大分量的下标，即

$$\pi_k^*=\arg\max_{r}\ y_r(k)$$

训练过程的优化函数为最大化$p(\mathbf{z}|\mathbf{y})$。文献[6]使用动态规划的方法说明了如何反向传播误差，具体细节不再展开。

另外一种序列转译的方法是使用转译网络，文献[11]融合attention机制构建转录网络，这个转录网络依靠另外一个RNN实现，输入为从上一层的RNN的输出，将上一层RNN输出的稀疏序列$\{\mathbf{h}\}_{t=1}^{T}$展开，在不同状态对序列中每一个分量设置一个评分

$$e_{t,i}=Score(\mathbf{s}_{t-1},\mathbf{h}_i) \tag{45}$$

评分最高者输入转录层RNN。转录层RNN从上一层RNN通过基于内容的注意力机制选择有用的信息输入网络，输出自动对齐的序列。输入的信息经过严格的筛选，转录层可以使用结构不太复杂的RNN。基于内容的注意力机制不同时刻对不同分量进行评分，这个评分器也可以视为一个分类器，或者输入门，在指定的状态将特定的分量输入到转录RNN中。

3 场景文本识别

复杂场景文本识别一般分为两个阶段，第一个阶段定位文本，第二阶段是识别文本。

许多文献都提到复杂场景文本的识别不同于传统的OCR，传统的OCR识别打印纸张上面的文字，它没有复杂的背景，背景全白、字体全黑。使用传统的图像处理技术进行阈值分割可以轻松提取出文字，并且字符字体单一、形体较为规范。复杂场景没有这么规律的特征，它具有复杂的背景、多变的字体和扭曲的字形，这些因素给文本识别带来挑战。将文本切分成若干离散字符，再使用CNN识别出字符，最后综合得到文本，这种方法存在以下缺陷。(1)处理的图片大小受限。(2)将文本视为由若干字符组成的集合，而不是把文本视为一条字符序列。(3)没有利用字符中所处的环境。另外一种较为流行的方法是结合RNN，用CNN提取特征，最后使用RNN 综合上下文信息。RNN的引入稍微放松对输入图片尺寸大小的限制，将文本视为一个字符串，利于对单词整体把握。但是RNN也有自身的缺陷，训练RNN的数据集必须尽可能容纳所有的字母组合，否则效果非常差，如果训练RNN仅使用的是英语词汇表的单词，那么当它遇到诸如czm类似的不正确的字母组合，它可能识别错误。相反训练基于字符分割的文本识别只需要保证训练集字符多样，不用考虑字母组合。

关于文本检测与定位，也有许多方法，由些方法结合传统机器视觉方法，而ASTER使用网络从提取特征，识别文字所在位置。

4 实验

4.1 网络结构

在测试CRNN过程中，我对CRNN前面的卷积层做出修改，第一种网络结构使用原作者的VGG结构，以后这种crnn使用VGG-crnn编号。另外，我修改ASTER中使用的残差网络结构，使之和第一种结构具有近乎相同的层数，编号为RES-ℓ-crnn， ℓ表示网络层数。第三种结构使用dense net结构，用DES-n-m-crnn表示，n表示一个dense块中卷积的层数，m表示特征图每经过一次卷积后网络特征图维度的增长数目。

关于最后RNN层，我发现两层LSTM难以训练，和一层LSTM近乎没什么差距，或许是我使用的数据集过于简单。

4.2 单字符识别

CRNN不仅具有处理序列信息的能力，它具有CNN的一些性质，能够综合领域信息识别分类图片，[14]将RNN融入图像识别，底层用卷积层提取特征，高层将提取得到的特征图，按照四个方向的顺序展开，送入RNN后综合得到分类结果，实验结果表明这种方法相对于CNN没有明显的提升，卷积网络具有提取领域信息的特征，底层综合局部信息，高层综合整体信息，卷积运算相当于向量点乘，向量的不同分量均表示不同的意义，在这一点RNN并不占优势，RNN优势在于处理不定长一维序列信息，它相较于CNN而言最大的优势是处理不定长数据。为观察CRNN在图像识别方面的性能设计如下实验。

4.2.1 数据集

这里选用的数据集Char75k、ICDAR。数据集具体参数见表1。 Char75K中包含不同类型的字符，手写体字符集（Char75K-Hnd）、不同字体类型的字符集（Char75K-Fnt）还有取自实际场景的字符集，Char75K-gimg中字符清晰完整，Char75K-bimg中字符存在清晰度不高、字符遮挡、不完整等问题。Char75K 包含数字的大小写字母，我手动按照数据的大小将他们按照一定比例划分成不同大小的测试集和训练集。ICDAR是著名的关于字符文本识别的挑战，选取2003年

的RCR(Robust Character Recognition)和2013年的单个数字识别数据集，这两个数据集均采用官方的划分方式，注意到ICDAR2013-single测试集较大而训练集太小，这导致网络在实际测试中，泛化能力不强，我将测试集和训练集颠倒一下，用测试集的图片训练，从训练集的图片检验，效果明显提升。

表 1: 各个数据集相关参数

数据集	Char75K-Fnt	Char75K-Hnd	Char75K-gimg
字符	0-9A-Za-z	0-9A-Za-z	0-9A-Za-z
训练集大小	50406	2728	6190
测试集大小	12586	682	1515
数据集	Char75K-bimg	ICDAR2003-RCR	ICDAR2013-single
字符	0-9A-Za-z	0-9A-Za-z	0-9
训练集大小	3862	6185	7000
测试集大小	937	5430	21780

(a) Char75K-Fnt (b) Char75K-Hnd (c) Char75K-gimg

(d) Char75K-bimg (e) ICDAR2003-RCR (f) ICDAR2013-single

图 5: 单字符数据集

4.2.2 实验结果

在上面6个数据集测试的结果见表2。训练时和测试时，将输入图片统一将大小缩放为(32, 32)。RES-crnn中使用的深度为14，DES-crnn为DEN-4-16-crnn（或者DES-4-12-crnn，具体细节我忘记记录了）。训练的终止条件不是迭代到固定次数，当我看到训练精度达到接近1.00时，我手动终止，所以得到的测试数据或许不准确，因为迭代次数越多越可能过拟合也可能导致测试精度提升。训练使用不同的参数（比如batch size），得到的结果或许也稍有不同，这几个网络测试时间间隔有点

长，我忘记是不是采用了统一的参数。从表2看出，VGG-crnn无论在哪个数据集都保持自己的优势，res-crnn比较逊色，可能是我没有对网络结构调优。

表 2: 单字符数据集测试结果

数据集	VGG-crnn		RES-crnn		DES-crnn	
	训练精度	测试精度	训练精度	测试精度	训练精度	测试精度
Char75K-Fnt	0.97	**0.93**	0.96	0.92	0.99	0.83
Char75K-Hnd	0.99	**0.79**	0.95	0.69	0.99	0.70
Char75K-gimg	0.99	**0.82**	0.99	0.80	0.99	**0.82**
Char75K-bimg	0.99	0.57	0.98	0.57	0.99	**0.58**
ICDAR2003-RCR	0.95	**0.79**	0.99	0.71	0.98	0.71
ICDAR2013-single	0.99	0.95	0.99	0.94	1.00	**0.96**

4.3 序列识别

4.3.1 数据集

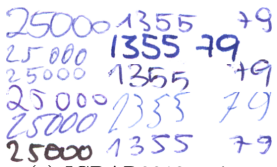
选中IIIT5K和ICDAR数据集，其相关参数如表3。其中ICDAR2003-RWR 和ICDAR2013-WR不仅包含简单字符，而且包含标点符号。注意到ICDAR2013-string数据集测试集大小为训练集大小的6倍。

表 3: 各个数据集相关参数

数据集	IIIT5K	ICDAR2003-RWR	ICDAR2013-string	ICDAR2013-WR
字符	0-9A-Za-z	0-9A-Za-z	0-9	0-9A-Za-z
训练集大小	2000	1156	1262	3568
测试集大小	3000	851	6698	1439

(a) IIIT5K

(b) ICDAR2003-RWR


(c) ICDAR2013-string

(d) ICDAR2013-WR

图 6: 数据集

4.3.2 实验结果

实验结果如表4，训练时没有使用额外的其它数据集，结果自然不像文献中所写的那样好，原因在于数据集太小，前面已经说到训练字符序列时，对字母组合要求较高，如果训练集频繁出现com，crnn已经将com视为一个整体，那么如果o不出现在c和m之间，crnn可能已经无法识别出o。ICDAR2013-string测试集和训练集具有相同的数字组合序列，但是效果却如此差。原因不清楚，我又将测试集和训练集颠倒，测试结果达到0.9+。

表 4: 文本序列识别结果

数据集	VGG-crnn		RES-crnn		DES-crnn	
	训练精度	测试精度	训练精度	测试精度	训练精度	测试精度
IIITK	0.94	**0.53**	0.83	0.42	0.96	0.48
ICDAR2003-RWR	0.98	0.32	0.93	0.26	0.98	**0.36**
ICDAR2013-string	0.93	0.27	0.95	**0.37**	1.00	0.28
ICDAR2013-WR	0.97	**0.71**	0.96	0.69	0.94	0.67

4.4 其它结果

我在训练网络时发现，训练图片尺寸对训练的结果有非常大的影响，对于单字符识别来说，固定大小不仅可以提高训练速度，而且得到的结果也有提升，对于文本序列固定大小得到的结果比较糟糕，我刚开始训练时，没有经验，没有注意到尺寸这件事结果网络始终保持在一个错误的状态，训练数据如果英文字符既有大写也有小写，不能忽略大小写，否则如果数据集太小的话，非常难以训练。我在训练时还发现，双层LSTM相比单层LSTM并没有性能上的显著提升，反而减慢网络的收敛速度。

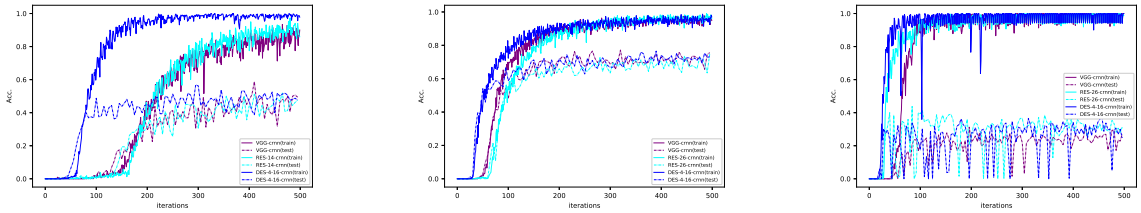
表 5: 结果三

数据集	VGG-crnn		RES-crnn		DES-crnn	
	单层LSTM	双层LSTM	单层LSTM	双层LSTM	单层LSTM	双层LSTM
ICDAR2013-single	**0.95/0.99**	0.92/0.99	**0.94/0.99**	**0.94/0.99**	0.96/1.00	-
ICDAR2013-single	-	-	**0.96/1.00(r)**	**0.96/0.99(r)**	-	-
ICDAR2013-string	**0.27/0.99**	0.25/1.00	**0.37/1.00**	0.24/0.97	0.004/1.00	-
ICDAR2013-string	0.89/0.98(r)	**0.96/0.96(r)**	0.91/0.98(r)	**0.94/0.96(r)**	0.93/1.00	-
ICDAR2013-WR	0.71/0.99	**0.72/0.95**	**0.69/0.97**	0.63/0.93	0.67/0.95	**0.72/0.95**
ICDAR2013-RWR	**0.32/0.98**	0.12/0.80	0.26/0.93	-	0.36/0.98	-
ICDAR2003-RCR	**0.76/0.98**	**0.76/0.98**	0.71/0.99	-	0.71/0.97	-
Char75K-Fnt	**0.93/0.99**	0.92/0.94	**0.92/0.96**	0.89/0.90	0.83/-	-
Char75K-Hnd	**0.79/0.99**	0.78/0.96	**0.69/0.98**	0.63/0.93	0.70/0.98	-
Char75K-gimg	**0.82/0.99**	**0.82/0.99**	**0.80/0.99**	0.79/0.95	0.82/0.99	-
Char75K-bimg	0.57/0.98	**0.58/0.97**	**0.57/0.98**	0.53/0.96	0.58/0.98	-

表5给出了其它的一些结果，为了对比单层和双层LSTM的效果，用粗体标出优胜的一方，（r）表示将数据集的测试集和训练集颠倒。双层LSTM比较难以训练，有时达到最大迭代次数，网络都在训练集上保持较低精度。表中的训练精度仅仅是大概范围，训练时随机从训练集抽数若干照片，本按照长宽比最大的作为统一比例将抽出的图片统一缩放到统一尺寸比例。所以每次测试得到的训练精度均不相同。

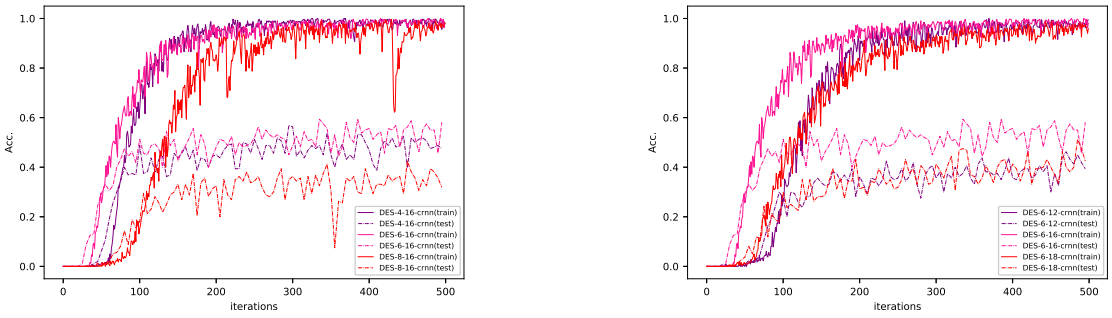
最后我尝试调节网络中结构相关参数，RES-crnn改变残差块的数目，DES-crnn改变Dense块内的卷积层数和扩增特征图维度数目。选取ICDAR2013-string、ICDAR2013-WR和IIIT5K数据集测试，这次除了网络结构不同，其它参数可以保证完全一致。得到图7结果。DES-crnn具有无与伦比的优势。

改变DES-crnn中卷积层的深度和宽度，以期望得到更好的结果，结果得到如图 8结果，加深深度或扩大宽度均不能太性能的提升，这或许与数据集有关。



(a) IIIT5K (b) ICDAR2013-WR (c) ICDAR2013-string

图 7: 不同网络结构在不同数据集的性能



(a) 不同深度 (b) 不同宽度

图 8: 不同深度和宽度DES-crnn在IIIT5K上的性能

另外，DES-crnn具有的另外一个优势，使用少量参数就可以达到较好的性能。几种网络模型的大小如图9所示。

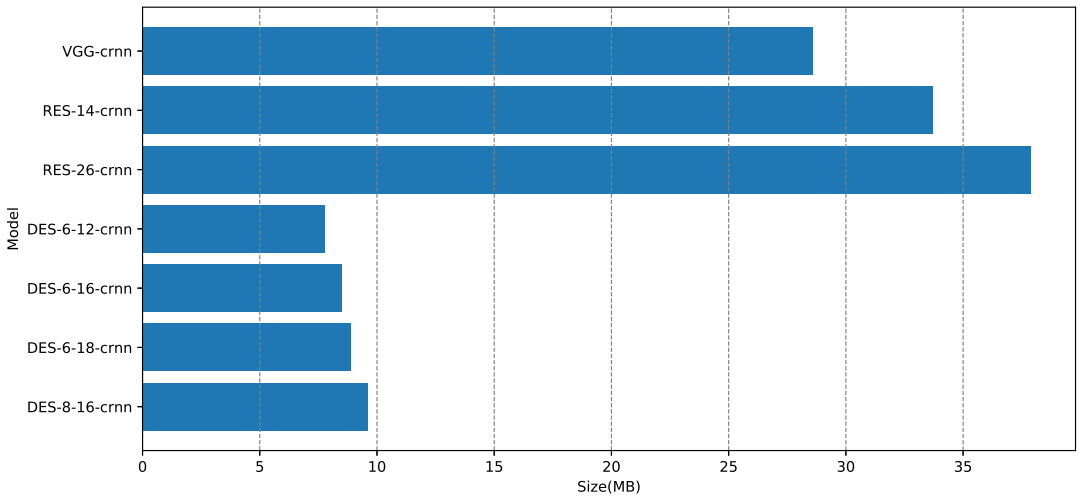


图 9: 模型大小对比

参考文献

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[2] Kyunghyun Cho, Bart van Merri?nboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 06 2014.

[3] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Y. Bengio. Attention-based models for speech recognition. 06 2015.

[4] George Cybenko. Approximation by superpositions of a sigmoidal function. math cont sig syst (mcss) 2:303-314. *Mathematics of Control, Signals, and Systems*, 2:303–314, 12 1989.

[5] Markus Diem, Stefan Fiel, Angelika Garz, Manuel Keglevic, Florian Kleber, and Robert Sablatnig. Icdar 2013 competition on handwritten digit recognition (hdrc 2013). pages 1422–1427, 08 2013.

[6] Alex Graves, Santiago Fernά ndez, Faustino Gomez, and Jü rgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks. volume 2006, pages 369–376, 01 2006.

[7] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 38, 03 2013.

[8] Sepp Hochreiter and Jü rgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[9] A. Mishra, K. Alahari, and C. V. Jawahar. Scene text recognition using higher order language priors. In *BMVC*, 2012.

[10] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 07 2015.

[11] Baoguang Shi, Mingkun Yang, Xinggang Wang, Pengyuan Lyu, Cong Yao, and Xiang Bai. Aster: An attentional scene text recognizer with flexible rectification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 06 2018.

[12] Bolan Su and Shijian Lu. Accurate scene text recognition based on recurrent neural network. pages 35–48, 11 2014.

[13] Ronald Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. 09 1998.

[14] Zhen Zuo, Bing Shuai, Gang Wang, Xiao Liu, Xingxing Wang, Bing Wang, and Yushi Chen. Convolutional recurrent neural networks: Learning spatial dependencies for image representation. pages 18–26, 06 2015.