期中 Project：CNN&RNN

18340002 蔡晨语　　负责 RNN 部分
18340159 唐瑞怡　　负责 CNN 部分

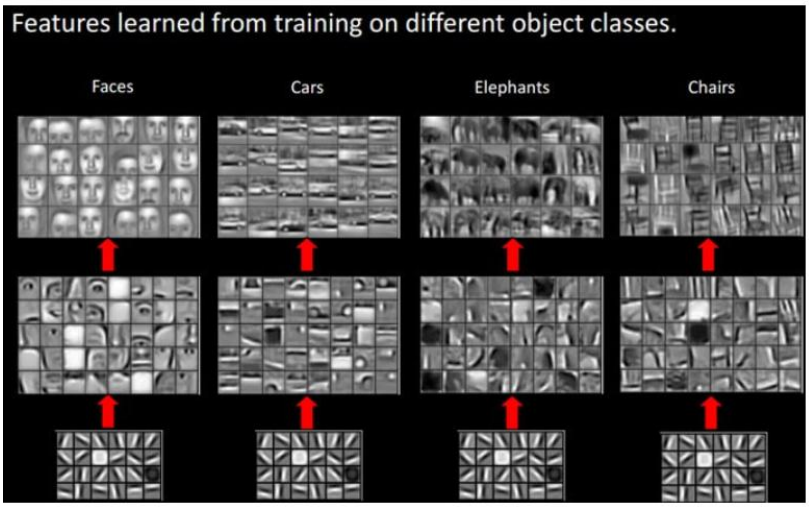
CNN 模型

一、模型原理

1.1 背景

在正式探讨 CNN 原理之前，我们先来看看人类是如何形成视觉以及如何判断出这是一个什么物体的，以便我们更好理解机器学习的过程。

对于不同的物体，人类视觉是通过逐层分级来进行认知的。在最底层特征基本上是类似的，就是各种边缘，越往上，越能提取出此类物体的一些特征（轮子、眼睛、躯干等），到最上层，不同的高级特征最终组合成相应的图像，从而能够让人类准确的区分不同的物体。（具体如下）



容易想到：可以模仿人类大脑的这个特点，构造多层的神经网络，较低层的识别初级的图像特征，若干底层特征组成更上一层特征，最终通过多个层级的组合，最终在顶层做出分类呢。这就是 CNN 的主要思想。

1.2 为什么需要 CNN

图片就是由一个个像素组成的，以 RGB 图像为例，每一个像素都有其对应的数值。如果用普通神经网络其实也可以实现分类功能，但是假如我们处理一张 1000×1000 像素的图片，我们就需要处理 3 百万个参数（1000×1000×3=3,000,000）。这么大量的数据处理起来是非常消耗资源的，而且这只是一张不算太大的图片。更重要的是：我们在大部分场景下，降维并不会影响结果。比如 1000 像素的图片缩小成 200 像素，并不影响肉眼认出来图片中是一只猫还是一只狗，机器也是如此。

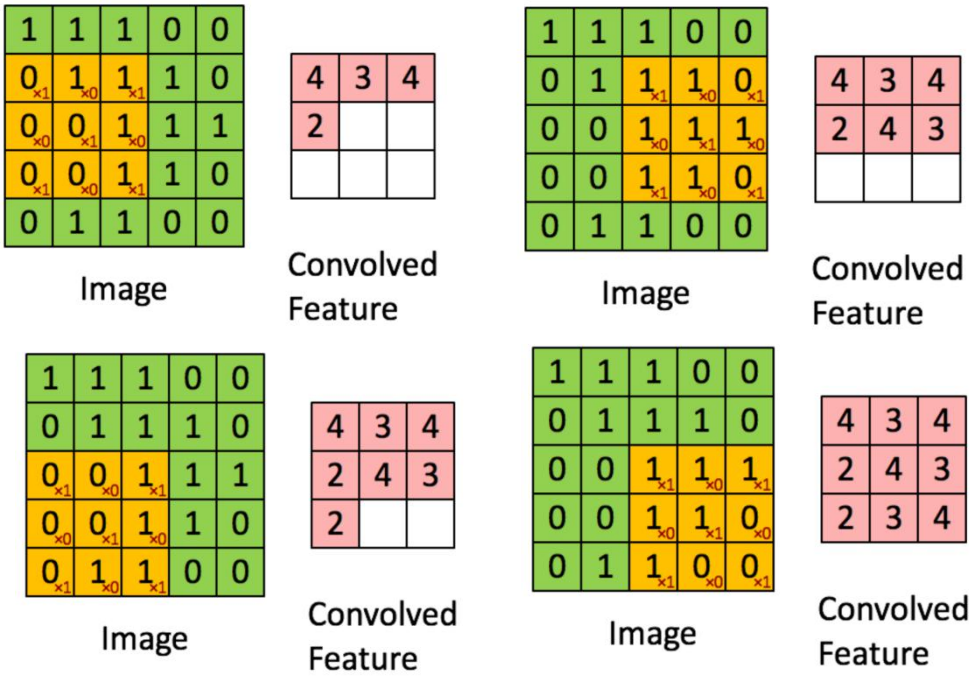
1.3 CNN 基本原理

典型的 CNN 由 3 个部分构成：①卷积层 ②池化层 ③全连接层

如果简单来描述的话：卷积层负责提取图像中的局部特征；池化层用来大幅降低参数量级(降维)；全连接层类似传统神经网络的部分，用来输出想要的结果。

1.3.1 卷积层

通过卷积核的过滤提取出图片中局部的特征，跟上面提到的人类视觉的特征提取类似。下面一组图粗略展示了利用 3*3 卷积核对原图像做卷积的一部分结果：



卷积的公式表达

连续形式

$$f(n)*g(n)=\int_{-\infty}^{\infty} f(t)g(n-t)dt$$

离散形式

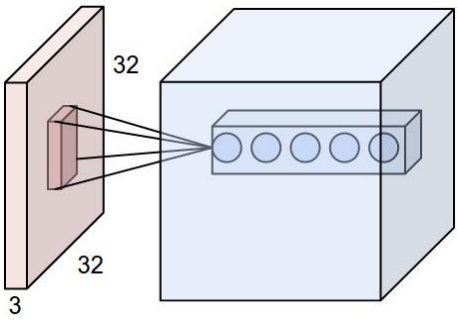
$$f(n)*g(n)=\sum_{-\infty}^{\infty} f(t)g(n-t)$$

应用到图像上就是把一张图像放在另一张图像上滑动，对两张图像重叠的部分对应元素相乘求和。这两张图像分别就是原图像与卷积核。

在上面分析中提到，用普通的全连接神经网络也可以实现图片分类，但参数过多。而 CNN 的解决方法就是每个隐含单元仅仅只能连接输入单元的一部分：每个隐含单元仅仅连接输入图像的一小片相邻区域。每个隐含单元连接的输入区域大小称作神经元的感受野(receptive field)。

由于卷积层的神经元也是三维的，所以也具有深度。卷积层的参数包含一系列过滤器（filter），每个过滤器训练一个深度，有几个过滤器输出单元就具有多少深度。

具体如下图所示，样例输入单元大小是 32×32×3，输出单元的深度是 5，对于输出单元不同深度的同一位置，与输入图片连接的区域是相同的，但是参数（过滤器）不同。



虽然每个输出单元只是连接输入的一部分，但是值的计算方法是没有变的，都是权重和输入的点积，然后加上偏置，这点与普通神经网络是一样的。

一个输出单元的大小有以下三个量控制：

· **深度**(depth)：顾名思义，它控制输出单元的深度，也就是 filter 的个数，连接同一块区域的神经元个数。

· **步幅**(stride)：它控制在同一深度的相邻两个隐含单元，与他们相连接的输入区域的距离。如果步幅很小（比如 stride = 1）的话，相邻隐含单元的输入区域的重叠部分会很多；步幅很大则重叠区域变少。

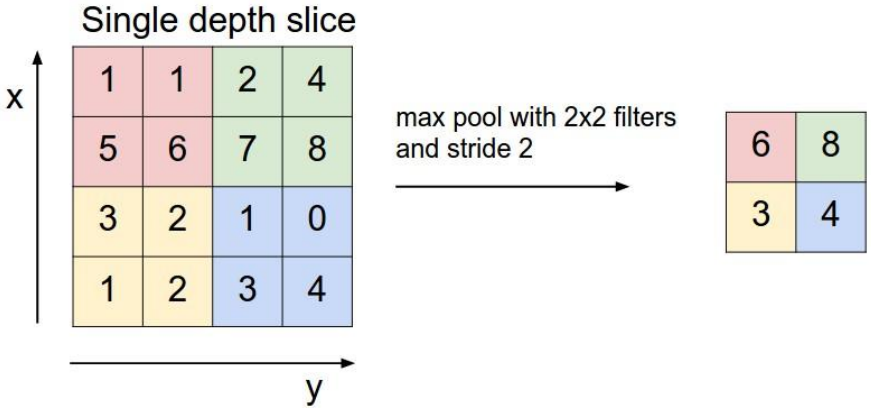
· **补零**(zero-padding)： 我们可以通过在输入单元周围补零来改变输入单元整体大小，从而控制输出单元的空间大小。

1.3.2 池化层

即下采样(down samples)，目的是为了减少特征图。池化操作对每个深度切片独立，规模一般为 2＊2，相对于卷积层进行卷积运算，池化层进行的运算一般有以下几种：

* 最大池化（Max Pooling）：取 4 个点的最大值。这是最常用的池化方法。
* 均值池化（Mean Pooling）：取 4 个点的均值。
* 高斯池化：借鉴高斯模糊的方法。不常用。
* 可训练池化：训练函数 ff ，接受 4 个点为输入，输出 1 个点。不常用。

以最大池化为例：



池化操作将保存深度大小不变。

注意：如果池化层的输入单元大小不是二的整数倍，一般采取边缘补零（zero-padding）的方式补成 2 的倍数，然后再池化。

1.3.3 全连接层

连接所有的特征，将输出值送给分类器（如 softmax 分类器）
在堆叠几个卷积层和池化层之后，重复这个模式知道图片已经被合并得比较小了，然后再用全连接层控制输出。

二、网络结构图

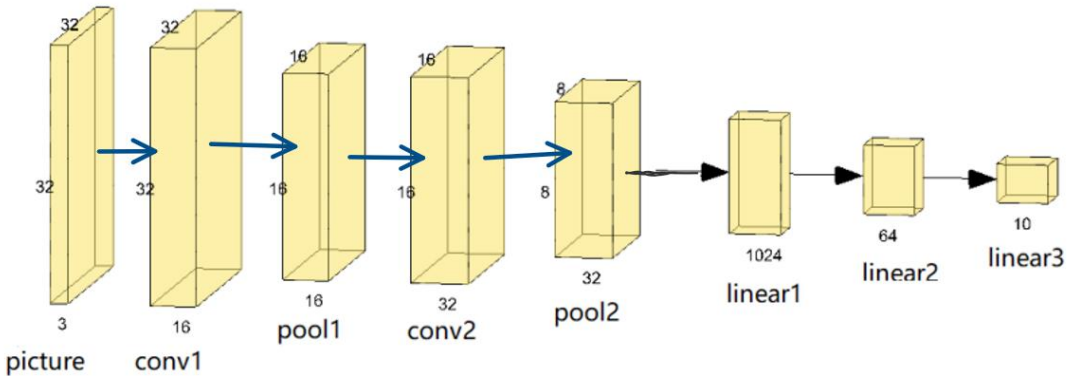


图 1，Vanilla CNN 结构

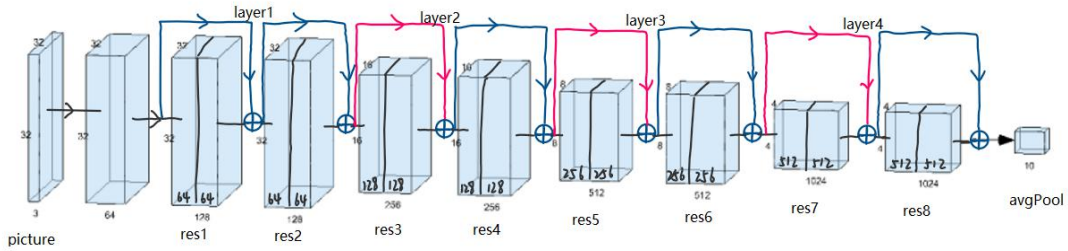


图 2，自定义的 ResNet18 结构图。其中蓝色线表示可直接相加的恒等残差块（通道数相同）；粉色线表示不可直接相加的残差块（通道数不同），需先用 1*1 的卷积核进行卷积，对原始项 x 进行降维（即增加通道数），使得残差项 F(x) 和原始项 x 的通道数相同再相加。

三、创新点

3.1 残差网络 ResNet

在本次实验中，使用了残差网络 ResNet 进行比较。

3.1.1 深度网络的退化问题

从经验来看，网络的深度对模型的性能至关重要，当增加网络层数后，网络可以进行更加复杂的特征模式的提取，所以当模型更深时理论上可以取得更好的结果。但是更深的网络其性能一定会更好吗？实验发现深度网络出现了退化问题（Degradation

problem）：网络深度增加时，网络准确度出现饱和，甚至出现下降。这个现象可以在图 3 中直观看出来：56 层的网络比 20 层网络效果还要差。这不会是过拟合问题，因为 56 层网络的训练误差同样高。

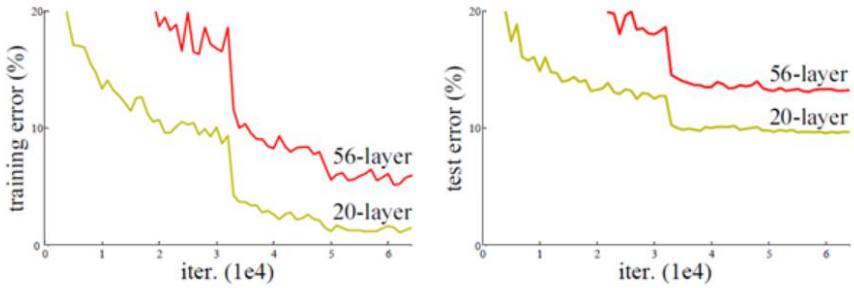
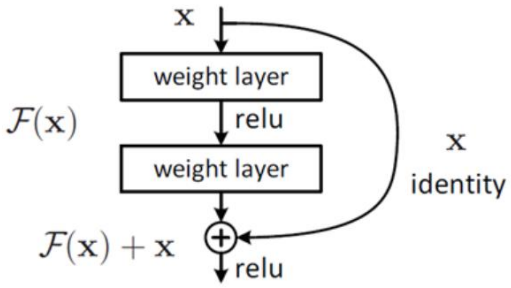


图3 20层与56层网络在CIFAR-10上的误差

3.1.2 残差学习

深度网络的退化问题至少说明深度网络不容易训练。但是，考虑这样一个事实：现在有一个浅层网络，你想通过向上堆积新层来建立深层网络，一个极端情况是这些增加的层什么也不学习，仅仅复制浅层网络的特征，即这样新层是恒等映射（Identity mapping）。在这种情况下，深层网络应该至少和浅层网络性能一样，也不应该出现退化现象。因此，不得不承认肯定是目前的训练方法有问题，才使得深层网络很难去找到一个好的参数。

而残差学习正是用来解决退化问题的手段。对于一个堆积层结构（几层堆积而成）当输入为 **x** 时其学习到的特征记为 **H(x)**，现在我们希望其可以学习到残差 **F(x)=H(x)-x**，这样其实原始的学习特征是 **F(x) + x**。之所以这样是因为残差学习相比原始特征直接学习更容易。当残差为 0 时，此时堆积层仅仅做了恒等映射，至少网络性能不会下降。实际上残差不会为 0，这也会使得堆积层在输入特征基础上学习到新的特征，从而拥有更好的性能。残差学习单元的结构如下图所示。



3.1.3 ResNet 网络结构

ResNet 网络是参考了 VGG19 网络，在其基础上进行了修改，并通过短路机制加入了残差单元。变化主要体现在 ResNet 直接使用 stride=2 的卷积做下采样，并且用 global average pool 层替换了全连接层。ResNet 的一个重要设计原则是：当 feature map 大小降低一半时，feature map 的数量增加一倍，这保持了网络层的复杂度。下表说明了不同深度的 ResNet 的参数设置：

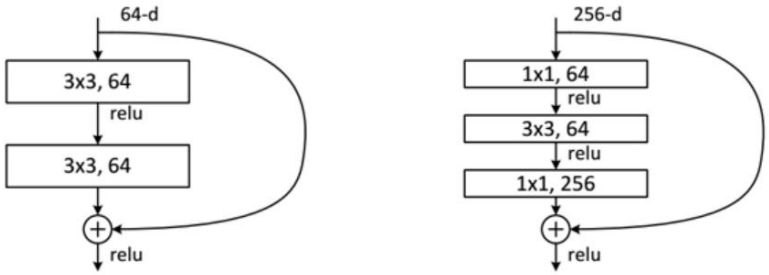
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet 中有两种残差单元（如下图所示）。左图对应的是浅层网络，而右图对应的是深层网络。对于短路连接，当输入和输出维度一致时，可以直接将输入加到输出上。但是当维度不一致时（对应的是维度增加一倍），这就不能直接相加。

有两种策略解决此问题:

（1）采用 zero-padding 增加维度，此时一般要先做一个 down samp，可以采用 stride=2 的 pooling，这样不会增加参数;

（2）采用新的映射（projection shortcut），一般采用 1x1 的卷积，这样会增加参数，也会增加计算量。短路连接除了直接使用恒等映射，当然都可以采用 projection shortcut。（在本实验中，使用了此方法解决）



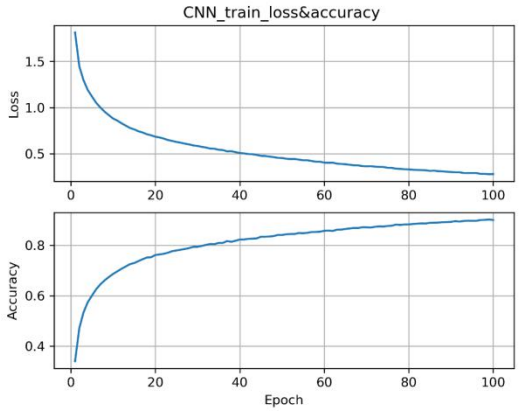
3.1.4 代码编写

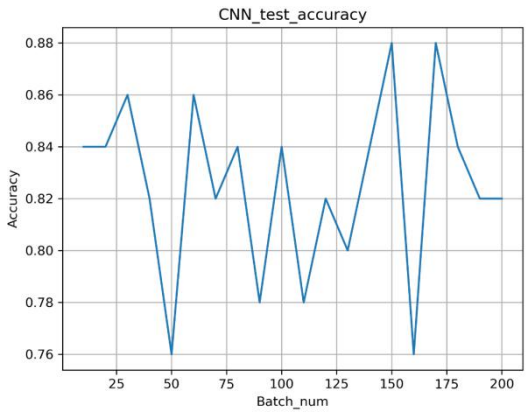
在实验中，我参考 ResNet 的官方源码，改变了卷积核大小等参数，搭建了一个针对 32× 32 输入的 ResNet18 网络，结构如“二、网络结构图”中的图 2。其中，每一个 res 表示一个残差学习单元（代码中的 Basicblock，包含两个卷积层），然后两个 res 表示一个 layer。

四、结果分析

4.1 基本效果呈现

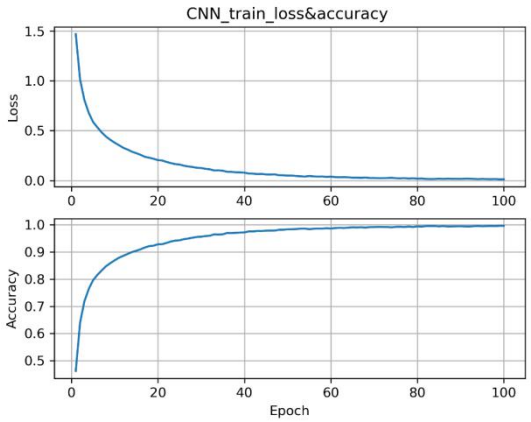
4.1.1CNN

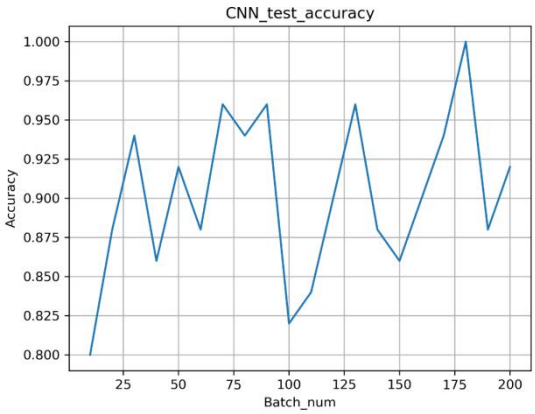




以上两图的条件是 CNN 在有归一化操作，batch_size=50，优化器为 SGDM。其中，左图是 CNN 模型在训练过程中训练集的 loss 和 accuracy 随着 epoch 的变化曲线，可以发现随着 epoch 的增加，loss 在逐渐减小，accuracy 在逐渐增加，且在 epoch=80 左右开始趋于平衡。右图是在训练结束后，应用训练的参数来预测测试集的分类的准确率，可以看出准确率在各个 batch 的数值在浮动，总准确率为 0.835 左右。

4.1.2 ResNet

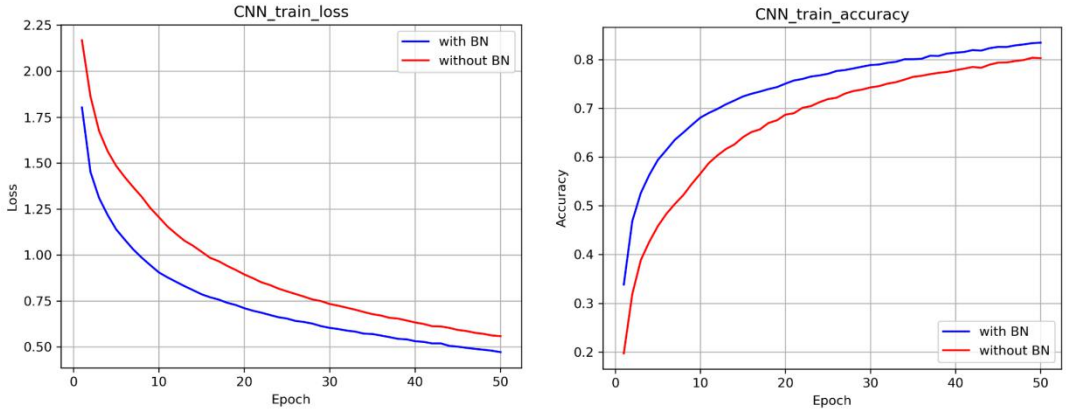




以上两图的条件是 ResNet 在有归一化操作，batch_size=50，优化器为 SGDM。其中，左图是 ResNet 模型在训练过程中训练集的 loss 和 accuracy 随着 epoch 的变化曲线，可以发现随着 epoch 的增加，loss 在逐渐减小，accuracy 在逐渐增加，且在 epoch=40 左右开始趋于平衡，速度明显快于纯 CNN 模型。右图是在训练结束后，应用训练的参数来预测测试集的分类的准确率，可以看出准确率在各个 batch 的数值在浮动，总准确率为 0.902 左右，也要由于 CNN。

4.2 有无 Batch Normalization（归一化）

使用 Vanilla CNN（简易 CNN）来作对比试验，其中一个在每个卷积层的后面都接一个 BN 层，而另一个则完全不使用 BN 层，在 batch_size=50，优化器为 SGDM 的条件下，对同样的数据进行训练后，测试结果如下：



从结果显然可以看出，加入了 BN 后使得网络准确率更高、loss 更小了，所以可以确定我们的模型应该给每层都加上 BN。但此图仍无法看出 BN 对收敛速度的影响，我猜测是网络过于简单的原因。

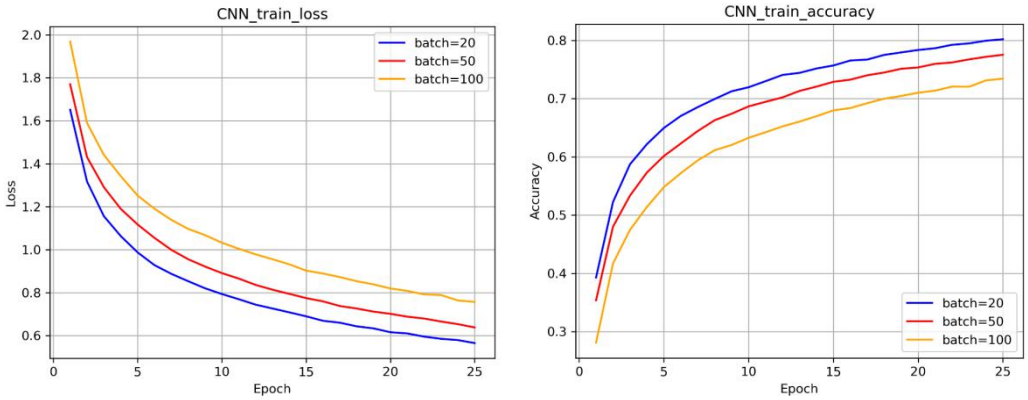
Batch Normalization 的原论文[1]作者给了 Internal Covariate Shift 一个较规范的定义：在深层网络训练的过程中，由于网络中参数变化而引起内部结点数据分布发生变化的这一过程被称作 Internal Covariate Shift。一旦输入的分布发生了变化，那么该层就要重新调整参数使得该层的输出结果可以维持与标签值尽量相同。而 BN 是用了解决“Internal Covariate Shift”的，作用就是加快神经网络训练速度，并且实验发现 BN 还可以增强模型的表达效果。为解决这个问题，提出了对输入数据进行“白化”操作，就是让各层的输入数据尽量符合均值为 0，方差为 1 的高斯分布，这样就可以同时解决上面所说的问题。给出的算法是每次对一个 mini-batch 操作，该 minibatch 里各个样本逐个分量分别计算：

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$

其中：$Var[x^{(k)}]$为 mini-batch 中样本在第 k 维分量上的方差，$E[x^{(k)}]$为 minibatch 中样本在第 k 维分量上的均值。

4.3 不同的 Batch_size

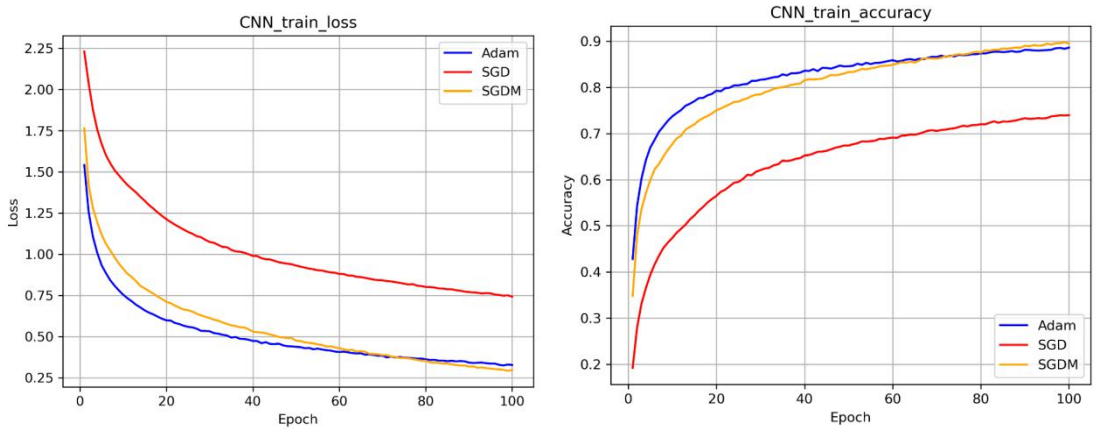
使用图 1 所示的简易 CNN 结构，在使用 BN，优化器为 SGDM 的条件下，改变批训练的 mini-batch 大小，取 batch_size 为 20、 50 和 100，用同样的数据训练后在训练集和测试集的表现如下图：



可以看出 batch_size 越大，在同一个 epoch 中 loss 越大，accuracy 越小。且对应在测试集上的准确率分别为 0.7789（20） 、0.744（50） 、0.7289（100）。因此，可知 batch_size 应取得小一点。但由于 batch_size 取得太小时，每次修正梯度的方向都是小部分样本的方向，这样不同样本集的方向可能不同，且训练速度较慢。因此，在之后的实验分析中，batch_size 取 50。

4.4 不同的优化器

继续使用图 1 所示的简易 CNN 结构，在 batch_size=50，使用 BN 的条件下，改变训练时使用不同的优化器，如 Adam、 SGD 和 SGDM，用同样的数据训练后在训练集合测试集的表现如下图：

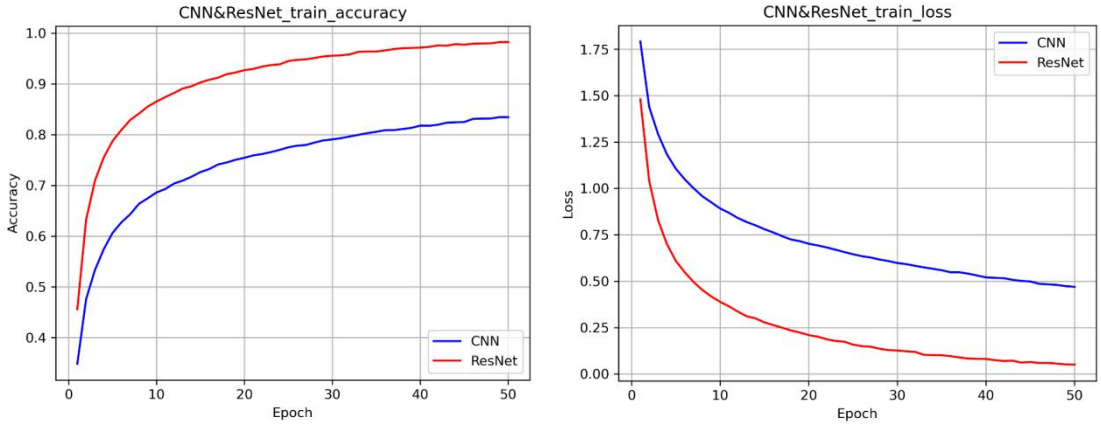


从上面 2 图可以看出，Adam 和 SGDM 明显比 SGD 的收敛速度快，loss 要小且 accuracy 要高，而 Adam 的收敛要比 SGDM 略快一点，但 SGDM 的效果要比 Adam 要略好。且在测试集中的结果为：0.7620(ADAM), 0.64(SGD)，0.7999(SGDM)，可以看出效果是 SGDM > Adam > SGD，与上图相符。

其中，Adam 是自适应学习率的算法，SGD 是随机梯度下降算法，SGDM 是引入了“动量”概念的 SGD 版本。

4.5 CNN 和 ResNet 的比较

在其余条件完全相同下，对比自己搭建的 ResNet 和 Vanilla CNN 的结果如下：



可以看出，使用了残差网络的 ResNet 的效果无论是 loss 还是 accuracy 都比 CNN

要好很多。且在测试集上的准确率也是 0.89(ResNet) > 0.8119(CNN)。但由于 ResNet 的训练速度十分慢，因此其他的调参基于 CNN 进行。

参考文献

[1] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift[C]// International Conference on International Conference on Machine Learning. JMLR.org, 2015:448-456.
[2] Batch Normalization 原理与实践 （https://zhuanlan.zhihu.com/p/34879333）
[3] ResNet 详细解读 （https://blog.csdn.net/csdnldp/article/ details/ 78313087?utm_source=app）
[4] 详解 CNN 神经网络（https://blog.csdn.net/qq_25762497/article/details/51052861）
[5] PyTorch 实现 CNN（https://blog.csdn.net/qq_34714751/article/details/85610966）

RNN 模型

一、模型原理

刚刚实现的卷积神经网络–CNN 和普通的算法大部分都是输入和输出的一一对应，也就是一个输入得到一个输出。不同的输入之间是没有联系的。但是在有一些应用场景中，输出不光和输入有关，也许和上一刻的输出也有关，比如，当我们在理解一句话意思时，孤立的理解这句话的每个词是不够的，我们需要处理这些词连接起来的整个序列；当我们处理视频的时候，我们也不能只单独的去分析每一帧，而要分析这些帧连接起来的整个序列。这种需要处理"序列数据：一串相互依赖的数据流"的场景就需要使用 RNN 来解决了。

以 NLP 的一个最简单词性标注任务来说，将我/吃/苹果三个单词标注词性为 我/nn 吃/v 苹果/nn。

那么这个任务的输入就是：我 吃 苹果 （已经分词好的句子）

这个任务的输出是：我/nn 吃/v 苹果/nn(词性标注好的句子)

对于这个任务来说，我们当然可以直接用普通的神经网络来做，给网络的训练数据格式了就是我-> 我/nn 这样的多个单独的单词->词性标注好的单词。

但是很明显，一个句子中，前一个单词其实对于当前单词的词性预测是有很大影响的，比如预测苹果的时候，由于前面的吃是一个动词，那么很显然苹果作为名词的概率就会远大于动词的概率，因为动词后面接名词很常见，而动词后面接动词很少见。

所以为了解决一些这样类似的问题，能够更好的处理序列的信息，RNN 就诞生了。

RNN 正向传播

传统神经网络的结构比较简单：输入层 – 隐藏层 – 输出层。如图 1 所示

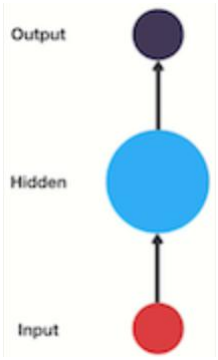


图 1

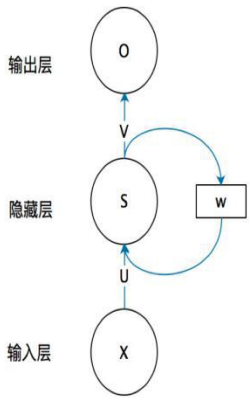


图 2

RNN 跟传统神经网络最大的区别在于每次都会将前一次的输出结果，带到下一次的隐藏层中，一起训练，如图 2 所示。可以看到，循环神经网络的隐藏层的值 s 不仅仅取决于当前这次的输入 x，还取决于上一次隐藏层的值 s。权重矩阵 W 就是隐藏层上一次的值作为这一次的输入的权重。展开如图 3

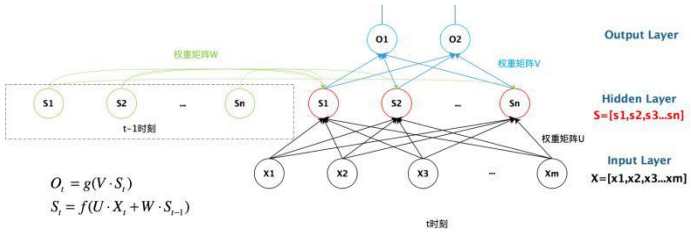


图 3

有了上面的模型，RNN 的前向传播算法就很容易得到了
对于任意一个序列索引号 t，我们隐藏状态 $h^{(t)}$由 $x^{(t)}$和 $h^{(t-1)}$得到：

$h^{(t)} = \sigma(z^{(t)}) = \sigma(Ux^{(t)} + Wh^{(t-1)} + b)$，其中σ为 RNN 的激活函数，b 为线性关系的偏置

序列索引号 t 时模型的输出 $o^{(t)}$的表达式：$o^{(t)} = Vh^{(t)} + c$

最终在序列索引号 t 时我们的预测输出为：$y^{(t)} = \sigma(o^{(t)})$

通常由于 RNN 是识别类的分类模型，所以上面那个激活函数一般是 softmax
通过损失函数 L(t)，比如对数似然损失函数，我们可以量化模型在当前位置的损失

RNN 反向传播

得到输出后，将输出与真实标签值比较得到误差，误差反向传播更新权重值，训练网络为了简化描述，这里的损失函数我们为交叉熵损失函数，输出的激活函数为 softmax 函数，隐藏层的激活函数为 tanh 函数。

对于 RNN，由于我们在序列的每个位置都有损失函数，因此最终的损失 Loss 为：

$L = \sum_{t=1}^{\tau} L^{(t)}$；其中对 V，c 分别计算梯度，得到 $\frac{\partial L}{\partial c} = \sum_{t=1}^{\tau} \frac{\partial L^{(t)}}{\partial c} = \sum_{t=1}^{\tau} \hat{y}^{(t)} - y^{(t)}$ $\frac{\partial L}{\partial V} = \sum_{t=1}^{\tau} \frac{\partial L^{(t)}}{\partial V} = \sum_{t=1}^{\tau} (\hat{y}^{(t)} - y^{(t)})(h^{(t)})^T$

但是 W,U,b 的梯度计算就比较复杂了。从 RNN 的模型可以看出，在反向传播时，在某一序列位置 t 的梯度损失由当前位置的输出对应的梯度损失和序列索引位置 t+1 的梯度损失两部分共同决定。对于 W 在某一序列位置 t 的梯度损失需要反向传播一步步的计算。我们定义序列索引 t 位置的隐藏状态的梯度为：$\delta^{(t)} = \partial L / \partial h^{(t)}$

有了δ(t)，计算 W,U,b 就容易了，这里给出 W,U,b 的梯度计算表达式：

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{\tau} diag(1-h^{(t)2})\delta^{(t)}(h^{(t-1)})^T$$

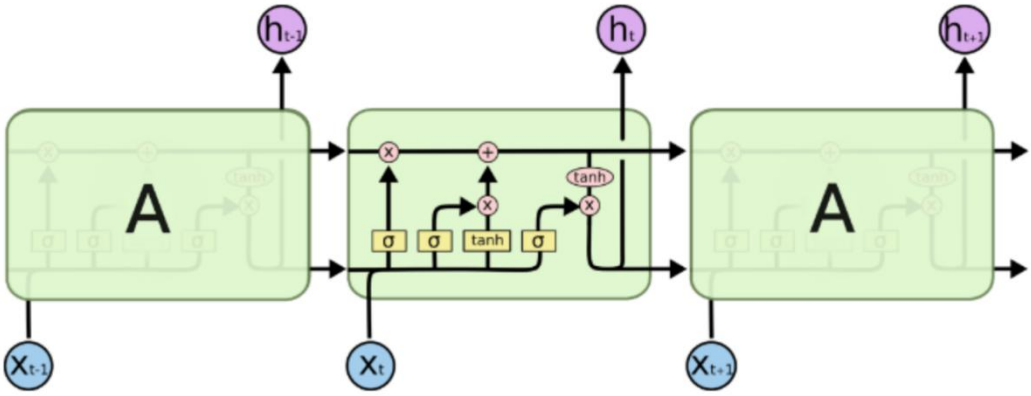
$$\frac{\partial L}{\partial b} = \sum_{t=1}^{\tau} diag(1-h^{(t)2})\delta^{(t)}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{\tau} diag(1-h^{(t)2})\delta(t)(x^{(t)})^T$$

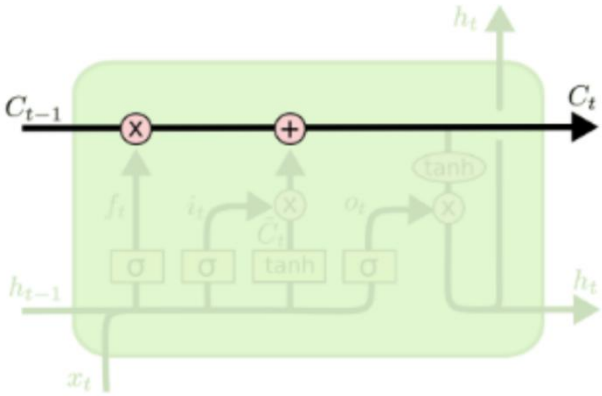
注：其实本次实验不用自己手动实现这些更新公式，但是掌握原理可以更好理解之后的 LSTM 模型。

RNN 改进算法 — LSTM

RNN 是一种死板的逻辑，越晚的输入影响越大，越早的输入影响越小，且无法改变这个逻辑，这就导致无法处理很长的输入序列。LSTM 通过引入门控机制，使得梯度能够进行远距离的流通，缓解梯度消失问题。

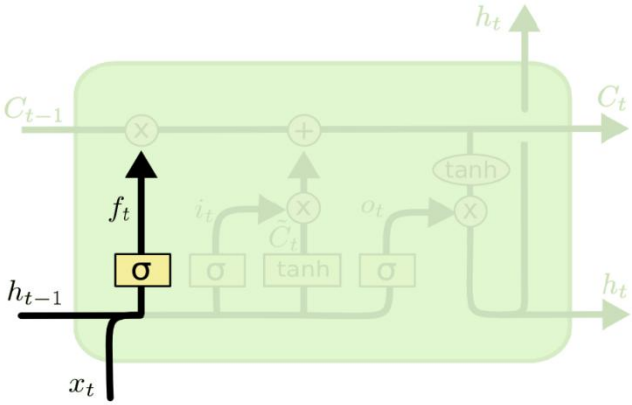


从上图中可以看出，在每个序列索引位置 t 时刻向前传播的除了和 RNN 一样的隐藏状态 $h^{(t)}$，还多了另一个隐藏状态，如图中上面的长横线。这个隐藏状态我们一般称为细胞状态(Cell State)，记为 $C^{(t)}$。如下图所示：



遗忘门

顾名思义，是控制是否遗忘的，在 LSTM 中即以一定的概率控制是否遗忘上一层的隐藏细胞状态。遗忘门子结构如下图所示：



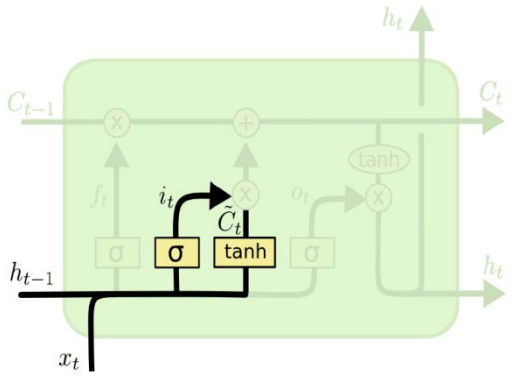
图中输入的有上一序列的隐藏状态 $h^{(t-1)}$和本序列数据 $x^{(t)}$，通过一个激活函数，一般是 sigmoid，得到遗忘门的输出 $f^{(t)}$。由于 sigmoid 的输出 $f^{(t)}$在[0,1]之间，因此这里的输出 $f^{(t)}$代表了遗忘上一层隐藏细胞状态的概率。用数学表达式即为：

$$f(t) = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)$$

其中 W_f，U_f，b_f是线性关系的系数和偏移，和 RNN 中的类似。σ为 sigmoid 激活函数

输入门

负责处理当前序列位置的输入，它的子结构如下图：

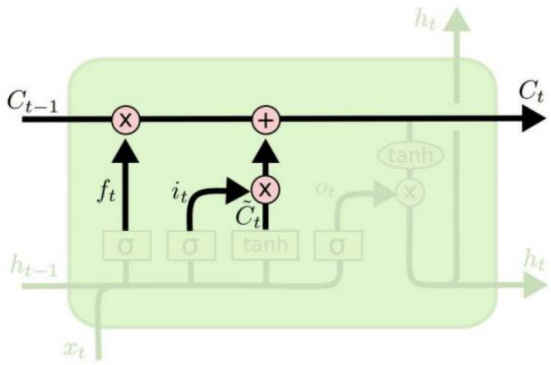


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

从图中可以看到输入门由两部分组成，第一部分使用了 sigmoid 激活函数，输出为 i(t)，第二部分使用了 tanh 激活函数，输出为 C(t)，两者的结果后面会相乘再去更新细胞状态。

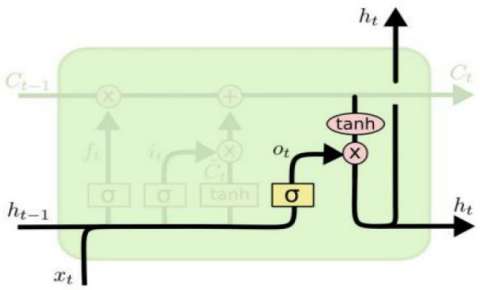
细胞状态更新

在研究 LSTM 输出门之前，我们要先看看 LSTM 之细胞状态。前面的遗忘门和输入门的结果都会作用于细胞状态 $C^{(t)}$。我们来看看从细胞状态 $C^{(t-1)}$如何得到 $C^{(t)}$。如下图所示：



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

输出门



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

从图中可以看出，隐藏状态 $h^{(t)}$的更新由两部分组成，第一部分是 $o^{(t)}$，它由上一序列的隐藏状态 $h^{(t-1)}$和本序列数据 $x^{(t)}$，以及激活函数 sigmoid 得到，第二部分由隐藏状态 $C^{(t)}$和 tanh 激活函数组成，即上图所示的公式

本次实验目的

本次实验需要实现的是关键词提取，给定具有预先识别的实体（例如，手提电脑）的一组句子，识别句子中存在的关键词，并返回包含所有不同关键词的列表。例如：

- I liked the service and the staff, but not the food

可以提取出关键词是：service、staff、food

- The hard disk is very noisy

可以提取出关键词是：hard disk

需要注意的是：1. 有几个关键词就需要提取几个，返回一个去重后的列表；
2. 方面术语（aspect term）一定是在原句子中出现过的

数据预处理

本次实验数据是以 xml 文件形式给出的，为了实现实验目的我们需要将其中的 text 和 term 提取出来，存放在 dataframe 中，并且需要借助 jieba 库来分词。并且通过分析句子和关键词之间的关系可以知道，关键词不太可能是'the'/'a'/'and'这类停用词，我们也可以进行处理将停用词去掉。接下来的步骤就是将每个分好词之后的单词转化成向量并输入网络进行训练。

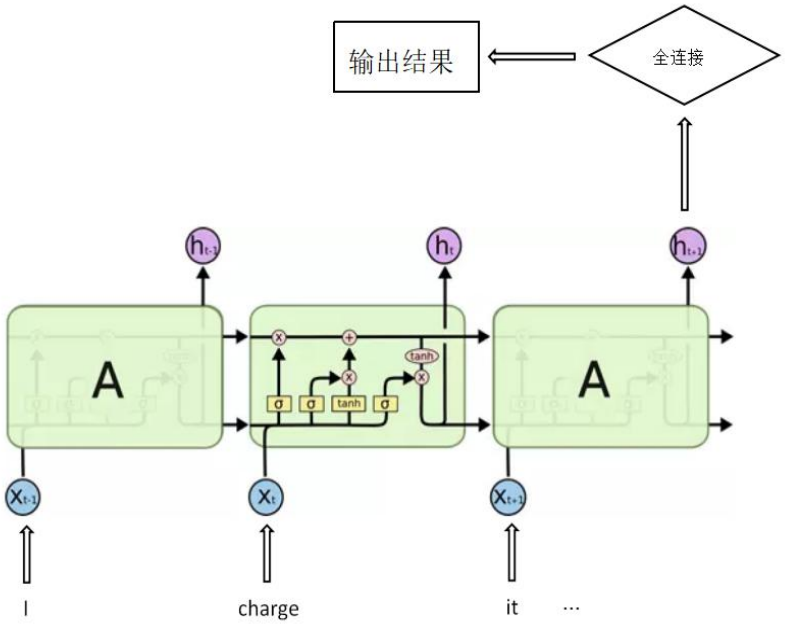
还存在一个问题就是，Lstm 模型的输入，不接受不定长度的输入，一个矩阵一定要对齐才能作为输入进入模型，因此我们要设置一个允许的句子最长长度作为截止，超过这个长度的句子单词舍弃不要，小于这个的做一个 padding 操作来补齐，对齐了数据之后组成一个 nparray 就可以作为输入进入 Lstm。

实验基本框架

本次实验相当于一次有监督的学习过程，在训练集中提供了句子和它对应的关键词。在之前我们已经调用了 gensim 训练 word2vec; 并且把句子中的关键词标记出来（如果是两个或多个单词组成的关键词，标记的时候可以借助 NLP 中所学的序列标注的方式：B-keyword，I-keyword），搭建模型训练。

在本次实验中我用到了 BiLSTM+CRF 模型，模型的输入是由训练好的词向量模型 embding 得到的句子（注意要进行 padding 补齐操作），输出是一个词向量文件，bin 格式，用于确定输出的 aspect term。

网络结构图



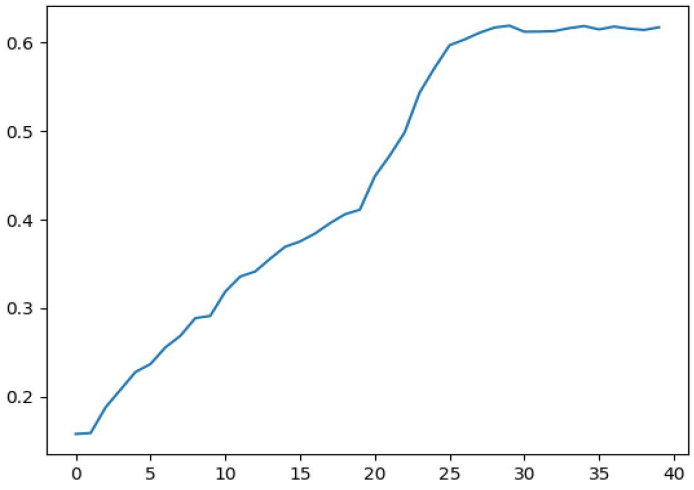
二、结果分析

2.1 运行时间

检测模型的性能是否优秀，其中一个很重要的指标就是迭代完全部次数之后所花费的总时间。可以看到在全部迭代了 20 次之后，所花费的时间只有 20 多分钟，其性能还是非常可观的。

```
2020-11-12 22:43:08 epoch 1, step 1, loss: 14.64, global_step: 1
2020-11-12 22:43:08 epoch 1, step 48, loss: 4.134, global_step: 48
```

```
2020-11-12 23:02:00 epoch 20, step 1, loss: 0.267, global_step: 913
2020-11-12 23:02:00 epoch 20, step 48, loss: 0.1853, global_step: 960
```



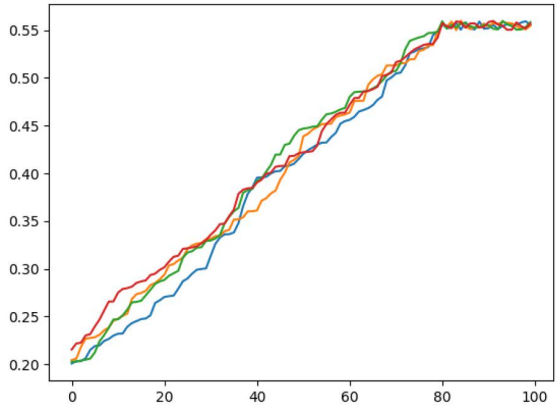
2.2 batch_size

上图的 epoch 代表的是第几轮，step 是第几个 batch，每一轮有多个 batch；而 global_step 就是花费的总迭代次数，也就是 step 的和。可以通过调整 mini_batch 的值来调整 step 的数目，从而达到提升准确率的效果。而自己调参后发现 mini_batch 在 65 左右的时候性能最好，故实验结果均展示 mini_batch 为 64 时的准确率等指标

```
关键词准确率：0.6764705882352942
关键词回率：0.6571428571428571
F值：0.6666666666666666
```

mini_batch 为 60，此时 step=51

```
2020-11-12 23:02:00 epoch 20, step 1, loss: 0.267, global_step: 913
2020-11-12 23:02:00 epoch 20, step 48, loss: 0.1853, global_step: 960
===========validation / test===========
关键词准确率：0.8857142857142857
关键词回率：0.8857142857142857
```



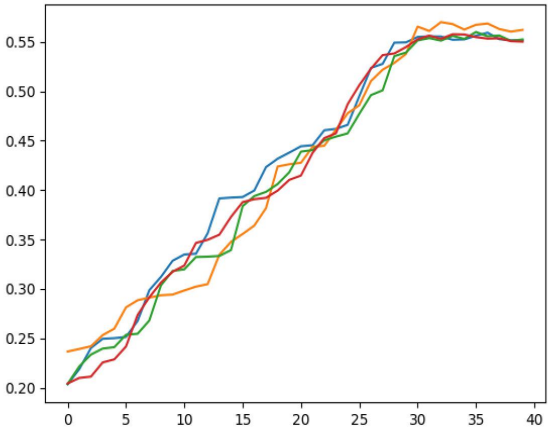
可以看到，因为这个模型对于这个数据是可以收敛的，35，50，100和500在80次batch后都大概收敛到相近的位置，由此可以推断，对于可以收敛的情况，batch_size对实验的精度不会有很明显的提升。

2.3 epoch

epoch 在上面提到，即总的迭代次数。从常理来看迭代次数越高效果越好，在经过模型训练之后发现总的迭代次数达到 30 次后 F 值就稳定在 65%左右，所以也无需迭代过多次数。

2.4 隐藏层个数

讨论在 lstm 在不同隐藏层节点下的实验效果，将 lstm 隐藏层节点从 32，64，128，256 和 512 逐步递增，运行 100 次迭代的实验效果：



红色：hidden_dim=64
蓝色：hidden_dim=128
绿色：hidden_dim=256
黄色：hidden_dim=512

可以看出当hidden_dim=512时F值是明显偏大的，但是其余情况的隐藏层维度其实对结果没有什么太大改变
所以本次实验最终版本为hidden_dim=512

2.5 模型的相对最优结果

参数	取值
Mini_batch	64
Hidden_dim	512
lr	0.001
embedding_dim	300

注：这不一定是这个模型在这个数据集上的最优情况，只是我在实验过程中调整参数得到的相对最优结果。通过图像可以观察得出，在迭代过程中 loss 还是一直在下降的，所以如果提高从迭代次数或者调整参数，可能可以达到更优的水平。

三、参考资料

[1]爱奇艺 NLP：BiLSTM_CRF 的关键词自动抽取（https://www.sohu.com/a/341792748)
[2]BiLSTM 模型构建及序列标注(https://blog.csdn.net/vivian_ll/article/details/93894151)
[3]条件随机场 CRF(https://zhuanlan.zhihu.com/p/29989121)