独家|七大NLP文本分类算法评测效果

原创 方城 技术客知音 2018-08-29

点击上方蓝字
关注客知音官方技术平台!

导语:

文本分类（Text Classification)是自然语言处理的一个基本任务。以客知音的外呼机器人为例，我们的机器人之所以能够在不同语境下做出精准的回应，离不开对文本进行准确、高效地分类。选择效果最佳的文本分类算法模型对于机器人的性能至关重要。

客知音作为国内领先的企业语音智能平台，现主要服务于金融、教育、电商、汽车、物流及政府产业，落地产品外呼机器人和销冠AI教练已应用于多家呼叫中心系统，但我们从未停止过更新技术、完善产品的步伐。

近期，我们在对多种文本分类算法模型调研后，择优进行了统一评测。本文主要介绍了以下7种算法模型的基本结构和特点，并通过实验得出不同模型的F1指标，发现FastText模型速度最快，SWEM

模型简单却效果很好，最后我们决定采用情景训练效果最佳的HAN算法模型。

1 TF-IDF+Xgboost

TF-IDF的全称是Term Frequency-Inverse Document Frequency，即词频-逆文本频率。

TF即词频的概念很容易理解，就是将所有文本分词后的结果组成一个vocabulary，每一个*文本的表示*就是一个和这个vocabulary相同大小的向量，在对应的位置放入的是对应的词在这个文本之中出现的频率归一化后的结果。

IDF也就是逆文本频率，基本定义为：

$$IDF(x) = log(N/N(x))$$

其中N表示语料库中文本的总数，而N(x)表示语料库中出现了词x的文本的总数。然而对于实际问题中N(x)可能为0，且IDF(x)等于零会影响算法过程，所以对IDF(x)的值做平滑调整如下：

$$IDF(x) = log(\frac{N+1}{N(x)+1}) + 1$$

这样就解决了上述两个问题，并且可以计算某个词的TF-IDF值了：

$$TF - IDF(x) = TF(x) * IDF(x)$$

经过计算，我们得到的是每个词的重要性，并且避免了常见词比重过大的问题。最终每个文本呈现的是一个len(vocabulary)的向量。

训练文本以此形式作为输入，用xgboost的原理讲起来的话就太复杂了。虽然它和GBDT很相似，两者最大的区别就在于xgboost的目标函数定义用了一下泰勒展开，展开了三项做了一个近似。

2 SWEM

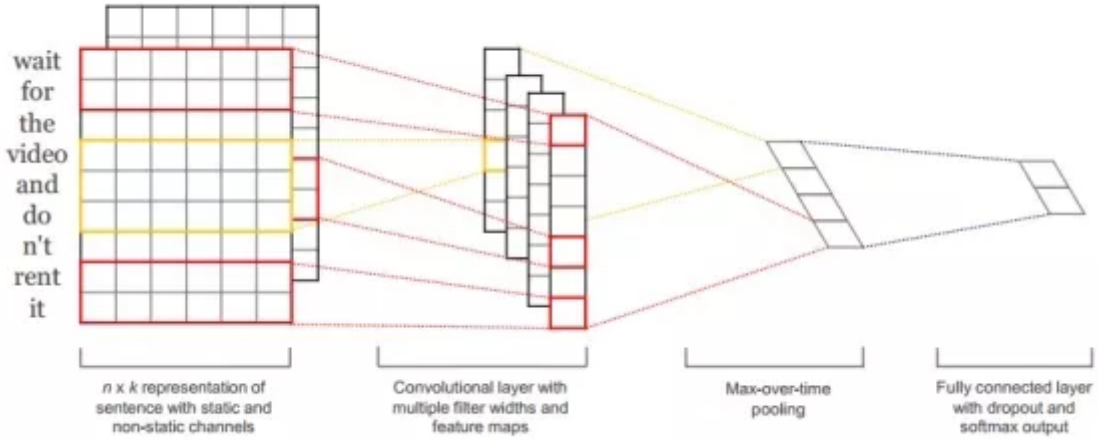
实验中最奇怪的一个现象就是，当模型结果十分简单，只有一层LSTM的时候，效果就已经非常好了。

同样，今年也有一篇出现相似现象的论文，标题是“Baseline Needs More Love：On Simple Word-Embedding-Based Models and Associated Pooling Machanisms”。

论文作者仅仅只用了简单的词向量以及LSTM，没有做很复杂的模型，就实现了特别高的精度，而在我们自己的实验数据上的实验显示，SWEM确实比很多相对复杂的模型效果都要好 **（实验结果请见文末）**。

3 TextCNN

Embedding现在在nlp各个领域随处可见，首先在在大语料库上训练获得预训练embedding matrix，具体embedding vector的size可以根据你的训练语料库的大小和你的具体的任务来设置，一般建议为128，256，512这三个尺度（word level而不是character level）。



图一：两个channel的版本

Yoon Kim在2014年在一篇论文“Convolutional Neural Networks for Sentence Classification”中提出了一个用CNN做文本分类的算法。

如图一所示，对于一句话“wait for the video and do n't rent it”(实际任务当中，预处理数据的时候，需要先去停用词，然后再分词)，九个单词的one-hot向量分别对应九个embedding vector，在图中将每个embedding vector横向堆叠在一起。

然后在我们现有的堆叠成的矩阵上做卷积，卷积核的形状大概类似于图中的红色或者黄色的框，然后在竖直方向上移动卷积核从而得到一个左二中的一个（具体的个数取决于你的feature map的数量）竖直的向量，设置了几个卷积核则会得到几个这样的竖直的向量。这里的卷积核的宽度是embedding vector的长度，具体核的高度是一个超参数，个人觉得这里卷一次得出来的值类似于一个n-gram信息。

第三步是进行一个最大池化。如图一，一个竖直的向量会池化为一个值。最终，因为你的输入的feature map是固定的，假设为n，你的卷积核的个数是固定的，假设为m，那么你得到的竖直的向量的个数为n*m个，然后进行最大池化，得到一个维度为n*m的固定大小向量。

所以，这里池化层的作用，一来是提取特征，二来是将输出特征的大小固定下来，以便接下来放到全连接层进行处理。

接下来就简单了，将这个n*m的向量放到全连接层，然后再将全连接层的输出放到softmax层便能得出最终我们的预测了。

值得注意的是，图中所描述的是文中所采用的一个trick，设置了两个channel。

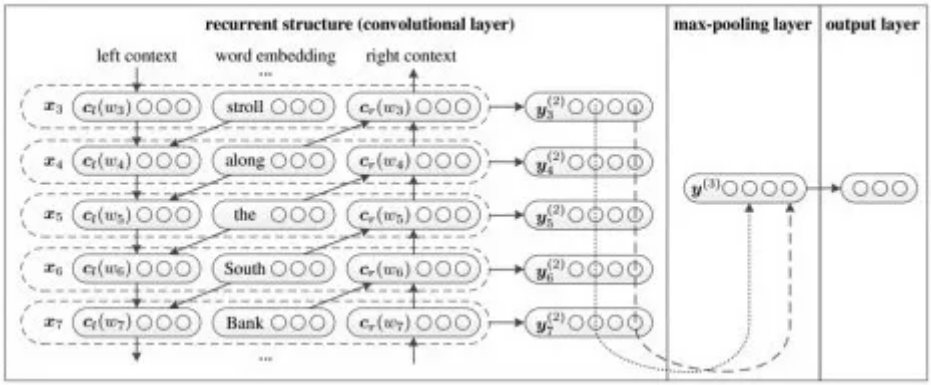
一个是预训练好的embedding矩阵所映射出来的embedding vector，在训练的过程当中，嵌入矩阵的参数是固定的，不会在反向传播的过程中更新。

而另外一个channel和这个channel拥有同样的初始值，只不过在训练过程中，嵌入矩阵的参数是变量而不是常数，也就是说在模型训练的时候这个channel所对应的embedding matrix会更新参数。

4 RCNN

这个用作文本分类的模型在2015年被提出，论文标题是**“Recurrent Convolutional Neural Networks for Text Classification”**。

模型的具体结构如图二：



图二：RCNN结构图（句子 "A sunset stroll along the South Bank affords an array of stunning vantage points"的部分示例）

首先，任何一个输入，都是一段文本，而一般文本分为多个词，词由embedding vector来表示，这是一般的思路，**然而作者在这里改变了词的表示方式，在词的表示中加入了上下文信息。**

具体是怎么加入上下文信息呢?

我们看一下图二左边，对于任意一个x，都由三部分组成，第一部分是left context，第二部分是word embedding，第三部分是right context，对于第二部分，应该不需要多说，主要是第一和第三部分究竟是怎么来的。

论文中给出了两个公式：

$$\boldsymbol{c}_l(w_i) = f(W^{(l)}\boldsymbol{c}_l(w_{i-1}) + W^{(sl)}\boldsymbol{e}(w_{i-1})) \quad (1)$$

$$\boldsymbol{c}_r(w_i) = f(W^{(r)}\boldsymbol{c}_r(w_{i+1}) + W^{(sr)}\boldsymbol{e}(w_{i+1})) \quad (2)$$

第一个等式代表了left context， cl(wi) 是左上下文，两个W都是需要训练的参数， wi表示一个文本中的第i个词， cl(wi-1) 表示第i-1个词的左上下文，我们仔细分析第一个等式会发现这是一个递归式，每一个文本都共用一个 cl(w1) ，对于右上下文也是同理，不过计算方向是从最后一个词到目标词，每一个文本都共用一个 cr(wn) ，然后我们有了计算左上下文和右上下文的方式，就可以得到我们的一个新的单词表示方法了。

后面的操作也是非常简单，将我们用上述方式得到的x的序列进行如下计算：

$$y_i^{(2)} = \tanh\left(W^{(2)}x_i + b^{(2)}\right)$$

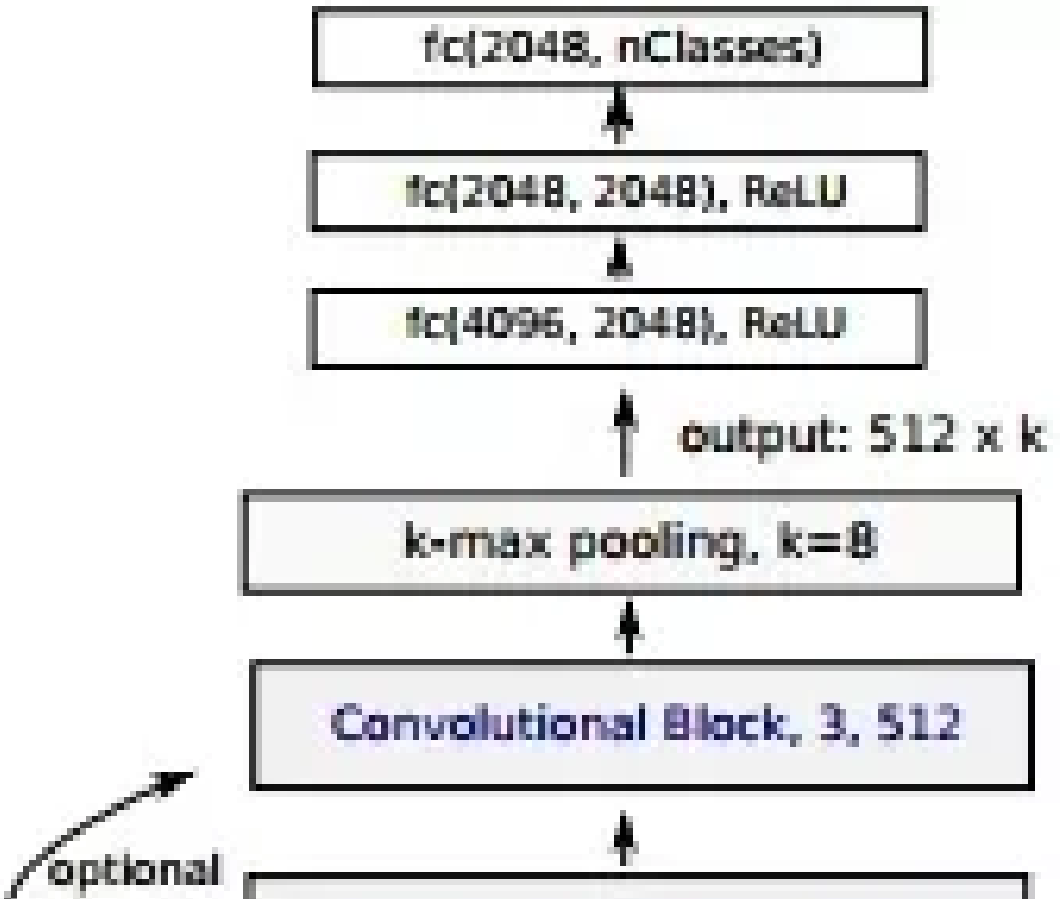
然后我们就得到了一个 y2 的序列，然后再对这个序列沿着图中所显示的竖直方向做一个最大池化，就得到了一个固定形状的 y3 ，而后将它放入全连接层，再将全连接层的输出放入softmax层，从而得到最后的类别预测结果。

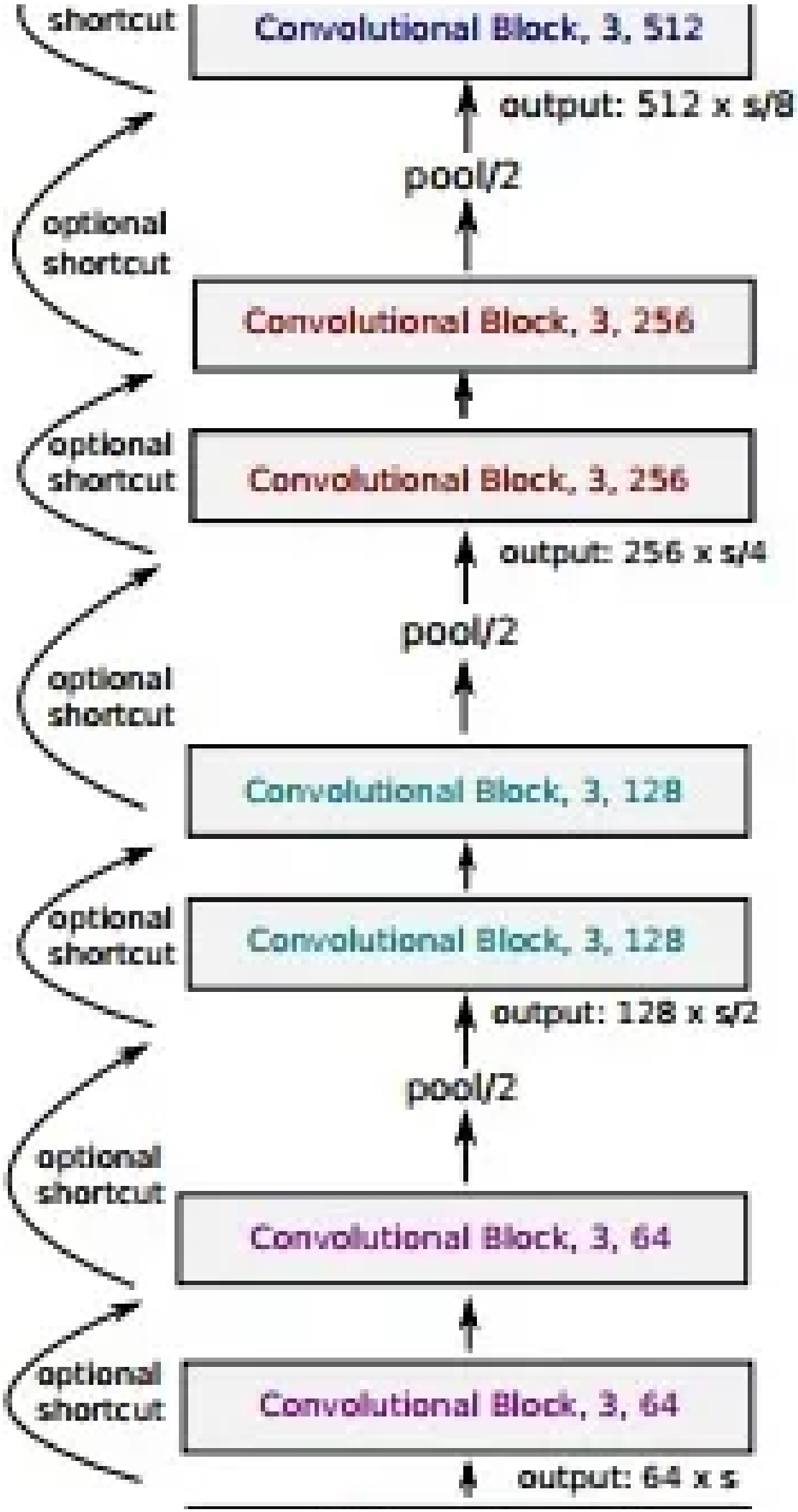
这篇论文的主要的优点是结合了上下文信息来做词的representation，一个很显然的缺点就是运行起来，实在是太慢了（对比大部分其他方法）。

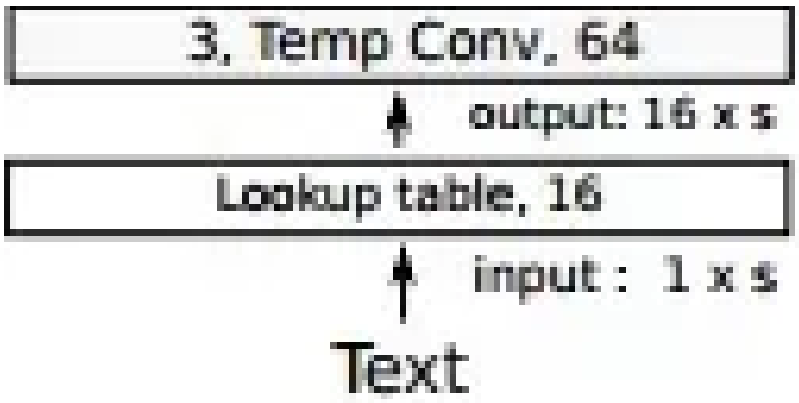
5 VDCNN

VDCNN是facebook实验室Alexis Conneau等人于2016年提出的模型，这个模型将深度卷积神经网络应用到了文本分类上。论文标题为“Very Deep Convolutional Networks for Text Classification”。

模型的整体结构如图三：




shortcut
Convolutional Block, 3, 512
output: 512 x s/8
pool/2
optional shortcut
Convolutional Block, 3, 256
optional shortcut
Convolutional Block, 3, 256
output: 256 x s/4
pool/2
optional shortcut
Convolutional Block, 3, 128
optional shortcut
Convolutional Block, 3, 128
output: 128 x s/2
pool/2
optional shortcut
Convolutional Block, 3, 64
optional shortcut
Convolutional Block, 3, 64
output: 64 x s

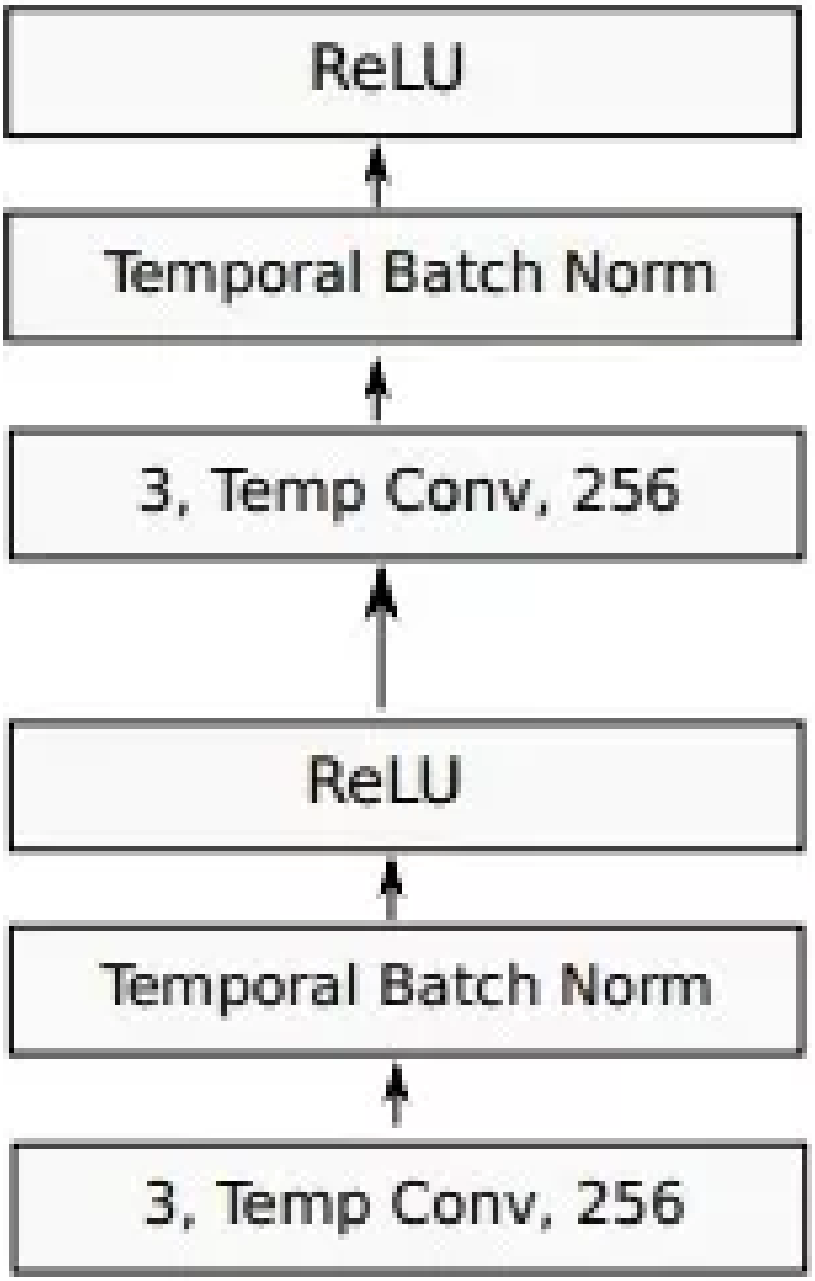


图三：VDCNN结构

首先开始输入的是一系列字母的字母，或者说是对应一系列的字母one-hot vectors，然后通过一个Lookup table将每一个字母都表示为一个16维的向量。这时的输入shape转化为了16×s，然后在s这条轴上进行一个windows size为3的卷积，卷积核的个数是64，得到一个64×s的输出。

接下来用到了一个Convolutional Block和左边的一个optional shortcut。

前者的结构如图四：



图四：Convolutional block

图中的参数不是固定的3，256，具体参数可以看图三。一般来说，这里的3是指的windows size，而256或者其他的数字，指的是卷积核的个数。可以看出，在每一个block之中，feature map的个数（也是卷积核的个数）是不变的，然后在每一次二分之一池化之后feature map的个数则翻倍。**这样做对比常规的深度卷积网络的好处是减少了内存占用。**

另外一个左边的连接，也就是optional shortcut，它的作用是能让卷积网络的深度更深而不用担心梯度弥散或者梯度爆炸的问题。用的是ResNet的思想，和Densenet同出一辙。

回到图三，讲过一系列conv-block和pooling/2之后，我们把得到的固定size的输出接连放入全连接层，最后返回一个长度为nClasses的向量，而后运用softmax得出最大概率的类别，也就是我的预测类别。模型具体的back-propogation以及loss就不详细叙述了，大同小异。

这篇论文将深度卷积神经网络运用在了文本分类任务上，有两大优点。

第一就是用深度网络对于提取文本信息；第二就是CNN对比RNN能够并行计算，加快了运行速度。当然，相对而言层数也变多了。对于我的数据，具体训练时间要比RCNN短。

6 FastText

Word embedding的发明者Mikolov在facebook AI实验室于2016提出了fasttext模型。论文标题为“Bag of Tricks for Efficient Text Classification”。

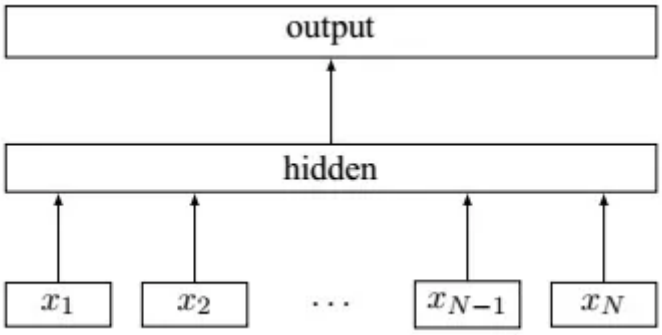
这个模型类似于word2vec中Cbow模型的结构，Cbow是根据contexts预测目标词，而fasttext是根据contexts预测label。

fasttext最大的优点就是快。在公司的分类问题训练数据上，一般其他模型训练需要半小时到半天，同样的训练集，fasttext训练只需要不到一分钟（都忽略文本预处理的时间）。而在速度这么快的同时，它还能和其他模型有差不多的训练指标（F1指数）。

不仅如此，现在fasttext模型已经有了API，可以直接调用，更加省去了自己写代码造轮子的过程，可以快乐地做个调包工程师。

那fasttext的模型结构和原理是什么呢?

如图五：



图五： N个ngram的句子 X1,.....,XN的fast text模型

根据论文叙述，fasttext之所以这么快还能达到一个类似于state-of-art的结果是因为用到两个方法。

第一个是hierarchical softmax来减小计算的时间复杂度。这个方法用到了Huffman树的原理，然后通过Huffman树层次结构减小了时间复杂度从而大大减少了训练时间。

第二个是用到了n-gram来保存了语序信息。如图五，图中的输入到隐藏层的所有的x都是一个n-gram。将所有的n-gram的embedding加起来求平均，然后丢到隐藏层，最后再丢到输出层的hierarchical softmax里面来得出最后的label。

7 HAN

我们平时使用的模型认为对每个输入都是一样重要的，然而对于有些问题，一部分词会更加重要。

比如根据老师说的话判断一节课是不是体育课，老师说到“跑步”等专业的体育的词汇就比“似乎”等这些词要更重要，更有助于我们对问题的判断。

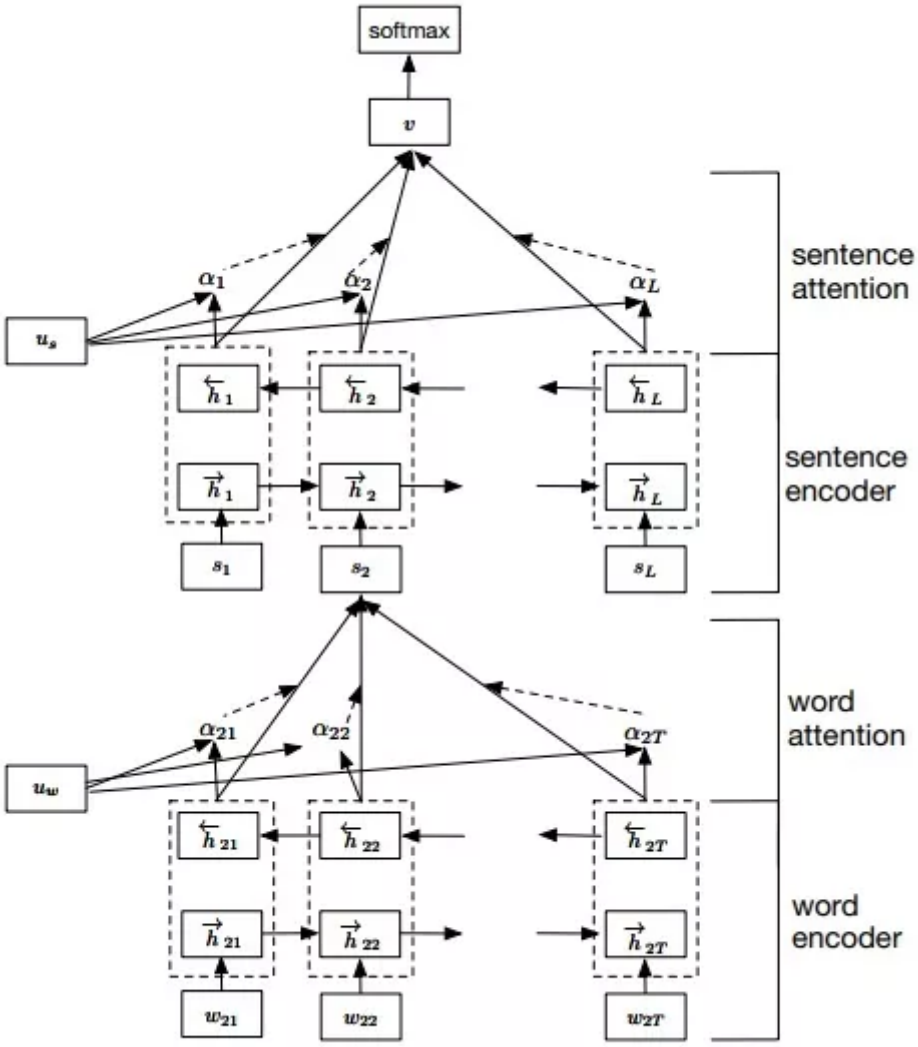
但因为模型不知道哪个词更重要，后来提出的注意力机制，给输入的每个词学习不同的注意力系数，大大提高了预测结果的指标。

HAN模型的全名是Hierarchical Attention Networks，也叫分层注意力模型，是Zichao Yang等人于2016年提出来的，论文标题是“Hierarchical Attention Networks for Document Classification”。

之所以叫做分层注意力模型，是因为模型使用了两层的注意力机制。第一层作用在word层，第二层作用在sentence层。

具体的做法是将文本分成多个sentence，每个sentence再分成多个word。模型先根据word学习sentence representation，在学习中加入了注意力机制。而后根据得到的多个sentence表示学习document representation，最后再根据document representation来进行分类。

模型结构如图六:



图六: Hierarchical Attention Network

模型的具体做法是，首先将每个句子表示为多个词的one-hot向量序列，然后先放入一个embedding层得到对应的embedding vector。

这里采用的是预训练的embedding matrix。如果实验数据特别多，并且有足够的时间和耐心去跑模型的话可以考虑将embedding matrix设置为变量，在训练过程中学习。

然后将这个word embedding sequence放入一个biLSTM，将前向和反向的输出concat得到隐藏表示。接着根据隐藏序列学得注意力系数，结合注意力系数和隐藏层得到句子的表示 s1 到 sm (假设有m个句子)。

得到每个句子的表示之后，将这个句子表示序列 s1 到 sm放入一个biLSTM中，同样还是将前向和反向结果concat得到对应的h1 到 hm。然后根据这个序列来学习第二层注意力系数，最后得到document表示v，紧接着将v放入全连接层和softmax层得到最后的预测label。

HAN的优势在于模型简单，并且效果非常好，胜过之前的模型，在结果上非常充分的体现出了注意力机制的优势。

实验结果：

我们将上述模型应用在公司内部数据上（只关注F1指数），通过在不同的字数上逐步实验，得到实验结果如下：

算法名称\字数	100	150	200	250	300	350	400	450	500
tf-idf+xgboost	0.53	0.56	0.58	0.6	0.61	0.62	0.62	0.63	0.63
swem	0.58	0.61	0.65	0.68	0.69	0.7	0.71	0.71	0.71
textcnn	0.56	0.59	0.61	0.63	0.63	0.64	0.62	0.64	0.64
rcnn	0.57	0.6	0.62	0.64	0.66	0.67	0.67	0.67	0.68
vdcnn	0.57	0.6	0.62	0.64	0.64	0.65	0.64	0.65	0.66
fasttext	0.55	0.6	0.61	0.64	0.64	0.65	0.67	0.66	0.67
han	0.6	0.63	0.66	0.68	0.69	0.71	0.72	0.72	0.73

图七：实验中的各模型F1指标

由于实验文本的长度问题，VDCNN模型的层数被修改为7层，而不是论文中的9层或更多。

根据最终得出的实验数据，fastText模型的训练速度最快；SWEM没有复杂的模型，却得到了很好的效果；但相比较而言，HAN模型在我们产品情景下的训练结果最佳。

参考文献：

[1] D. Shen, G. Wang, W. Wang, M. Renqiang Min, Q. Su, Y. Zhang, C. Li, R. Henao, and L. Carin. 2018. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In ACL.

[2] Yoon Kim. 2014. Convolutional neural networks for sentence classification. EMNLP.

[3] Lai, S., Xu, L., Liu, K., & Zhao, J. (2015, January). Recurrent Convolutional Neural Networks for Text Classification. In AAAI (Vol. 333, pp. 2267-2273)