# deep learning 算法

HengGeZhiZou
1018676477@qq.com

2019 年 3 月 31 日

**摘要**

本文总结目前学习的深度学习算法，主要用于 text classification 方向

## 1 Introduction

文本分类作为 nlp 中基础部分，也是许多算法的出发点，在一段时间的学习后，我们总结目前的一些基本算法。

## 2 文本分类算法

按顺序总结 cnn,fasttext,rnn,rcnn,seq2seq with attention,Hierarchical Attention Network,transform,ELMO, BERT 等算法。

### 2.1 CNN: 卷积神经网络

利用 CNN 作文本分类的想法来自于 Yoon Kim[1]。在他的文章中，将卷积运算对文本的信息进行提取，取得了非常不错的效果。在文本分类中，上下文词汇一般具有关系，我们在分类时注意上下文的关系往往能够提升算法的效果。

一般 cnn 分成输入层，卷积层，池化层，全连接层的多层结构。

输入层作为 cnn 的第一层，将句子中的词嵌入到向量空间中。一般我们用 input=[batch size ,sequence length] 作为整个 cnn 的输入，其中 batch size 代表我们输入训练集句子的数量。sequence length 代表这些句子的最大长度。输入构成一个 2 维的 tensor，在 axis=0 方向上是 batch size，在 axis=1

方向上是 sequence length 个词，即为每一句话的词。通常我们将每一个字用数字表示（建立词汇表）。对于第一步将句子嵌入到向量空间，我们有两种做法。第一种是我们初始化一个大小为 [vocabulary size ,word embedding size] 的矩阵，一般我们用随机生成一个均匀分布的映射矩阵 W。第二种我们使用已经预训练好的嵌入矩阵，他的大小和第一种方式相同。从实验的结果来看，第二种方法的损失函数收敛速度更快，但是对最后的结果影响不大（两个词嵌入矩阵会在训练过程中更新）。我们通过查找的方法，将 input 中的词在词汇表 W 找到对应的词向量，得到一个大小为 [batch size,sequence length,word embedding size] 的输入 tensor 叫做 embeddedchar。

卷积层作为卷积神经网络的核心，我们使用不同的卷积核对原始的 embeddedhar 进行卷积运算，我们一般选择不同 filter window size ，在图中选择 [2,3,4]。在文本分类任务中，卷积核的宽度和词向量的维度相等。于是我们的卷积核大小为 [filter window size，word embedding size，channel，num filters] 其中 channel 为卷积的通道数，一般 embeddedhar 的通道数为 1，num filters 为相同卷积窗口大小卷积核的数量。使用不同大小的卷积窗口计算的得到的 feature map 长度也 h 不同（sequence length - filter window size + 1）。计算结果的大小为 [batch,h,1,num filters]，同时我们需要对 feature map 进行一个 relu 的 activation function。

池化层对 feature map 中的结果进行计算，取出每一个 feature map 中的最大值，代表当前卷积核运算的结果，将相同卷积窗口大小的结果进行 concat，然后将所有的不同卷积窗口的计算机结果再次进行 concat。得到最终大小为 [batch,num filter size * 选择不同窗口的卷积核的数量]。

全连接层对池化层的结果进行计算，我们初始化一个大小为 [num filter size * 选择不同窗口的卷积核的数量, num classes] 的权值矩阵，同时使用截断正太分布。通过全连接层后 softmax 得到最终的结果。[batch size,num classes]。最后我们通过损失函数来梯度下降更新全部参数。

有一点需要注意的是我们在文本分析中，没有图像 RGB 的多通道，但是我们可以在词嵌入时使用不同的词汇表作为多个通道（word2vec，glove 等等）。

## 2.2 FASTTEXT

fasttext 的网络结构相对简单，输入一个词的序列或者是一句话，最后的得到这个词序列的类别标签。

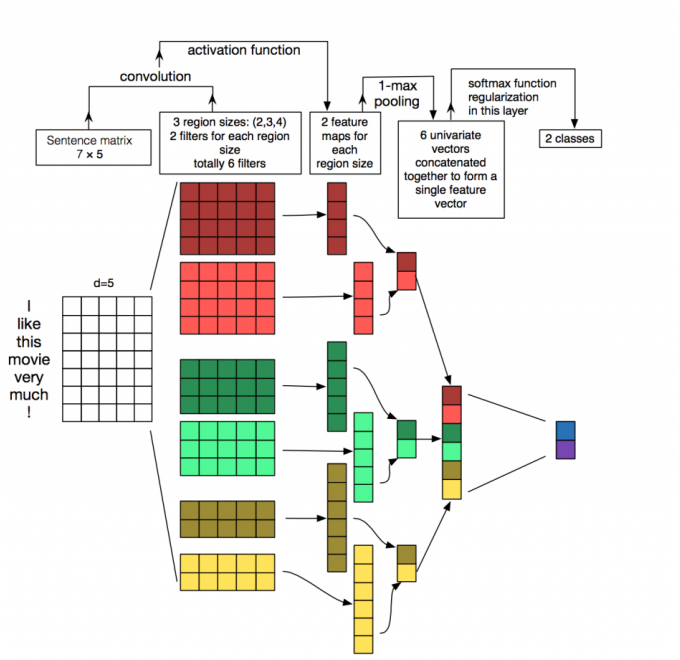



图 1: cnn 文本分类结构

在 fasttext 中，我们不仅使用每一个词的嵌入向量，为了不漏掉一些词意的微小变化，例如 un，dis 这些词缀，我们同时可以使用字符级的 n-gram 来增加一些的新的特征提高整个模型的准确率。训练的输入 [batch size,sequence length ]，通过输入层进行词嵌入后，得到 [batch size,sequence length ,embedding size]，在 hidden layer 我们将上一层的输出向量进行平均，得到一个大小为 [batch size,embedding size] 的 tensor，其中每一个 tensor 都代表开始输入的词序列。然后通过一个全连接层 [embedding size,num classes ] 得到类别结果。

为了改善运算的速度，模型使用了层级 softmax，和传统的 softmax 会计算所有的元素值不同，层级 softmax 在生成词向量中，由于词汇表具有大量单词，而通过 [embedding size, vocabulary size] 的全连接层和 softmax 后需要耗费大量的计算资源，于是我们需要层次 softmax。而在普通文本分类任务中，类别一般较少，没有必要使用层次 softmax。

## 2.3 RNN

rnn 循环神经网络是深度学习中必不可少的一种算法。从人类的语言角度出发，上下文之间存在着强烈的关联，往往每一个单词都不会单独只有本身的意义。和上下文之间的联系使得 rnn 的结构和句子序列结构十分契合。我们首先看看一般基本的 rnn 结构：

其中 x 为序列输入的词，一般输入为一个完整的句子。输入大小为 [batch size,sequence length]，首先经过一个词嵌入层，将所有的句子中的词映射到线性空间，生成的输入为 [batch ,sequence length , embedding size]。让后将句子序列输入到 rnn 中。其中每一层的 rnn 都共享权重参数，即 U，W，V 全部相同，每一个时刻 hidden state 的输入为当前词的输入和上一个 hidden state 的输出，在 cell 中经过一个 activation function 为当前 cell 的输出，当前的输出一部分作为当前时间步的输入 o。另一部分传输到下一个神经单元。对 o 使用激活函数后就为最终的输出。大小为 [batch ,sequence length, hidden state size] , 在 sequence 方向上依次为每一个词输入的输出。通常我们在进行文本分类时一般选取最后一个序列的输出作为最后的输出。虽然这样有一定的局限，比如过长的句子会使得 rnn 遗忘掉开始的内容。选择最后一个输出则大小为 [batch size,1,hidden state size]，在通过一个权值为 [hidden state size,num classes] 的全连接层，我们得到最后的分类结果。我们在进行梯度下降更新参数的时候又遇到了另外一个关键的问题。

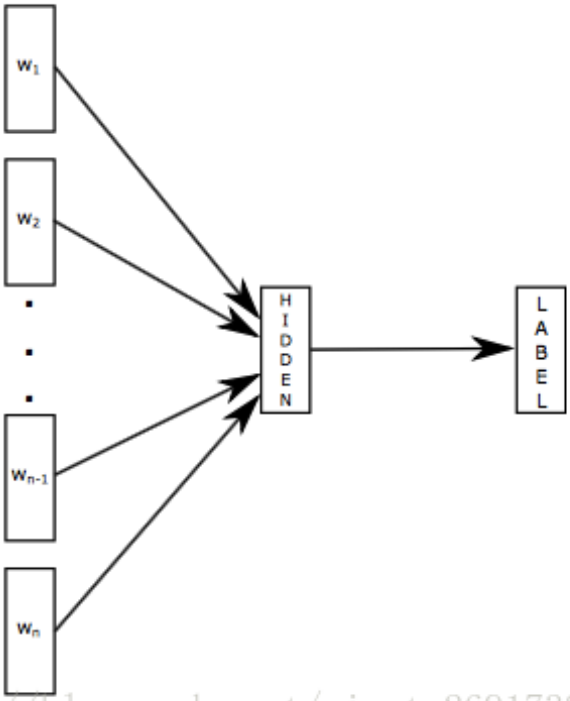




图 2: fasttext 结构

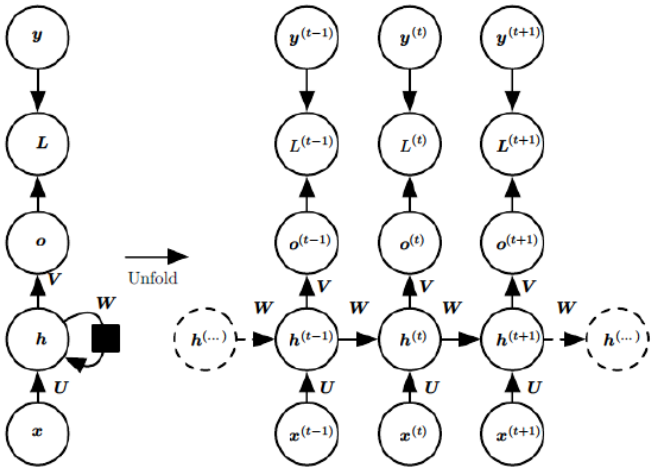



图 3: rnn 文本分类结构

$$\frac{\partial L^{(t)}}{\partial V} = \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

图 4: v

首先我们更新最近的参数 V

v 的参数更新比较容易，我们只需要关注目前的输出，将损失函数对 v 求导。由于时序列化的结构，我们会得到多个单元的不同输出，于是我们求解每一个时刻输出后进行损失的求和：

然而在对于 W 和 U 的求导过程中，由于显示序列化的数据，会链式对一个参数求导：

同时对 U 的求导过程和 W 相同，我们得到一个通式：

其中我们把中间的一项导数求得结果为：

其中我们使用了不同的激活函数，tanh 和 sigmoid 两种不同的激活函数。然而在对这两种激活函数的求导过程中我们发现了问题，

两种激活函数的导数大小都在 0 和 1 之间，当 rnn 中 hidden state 的

$$L = \sum_{t=1}^{n} L^{(t)}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{n} \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

图 5: derivative

rule.png rule.bb

$$\frac{\partial L^{(3)}}{\partial W} = \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial W} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial W} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial W}$$

图 6: chain rule w

$$\frac{\partial L^{(t)}}{\partial W} = \sum_{k=0}^{t} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial W}$$

$$\frac{\partial L^{(t)}}{\partial U} = \sum_{k=0}^{t} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial U}$$

图 7: rule

$$\prod_{j=k+1}^{t} \frac{\partial h^{j}}{\partial h^{j-1}} = \prod_{j=k+1}^{t} tanh' \cdot W_s$$

$$\prod_{j=k+1}^{t} \frac{\partial h^{j}}{\partial h^{j-1}} = \prod_{j=k+1}^{t} sigmoid' \cdot W_s$$

图 8: result

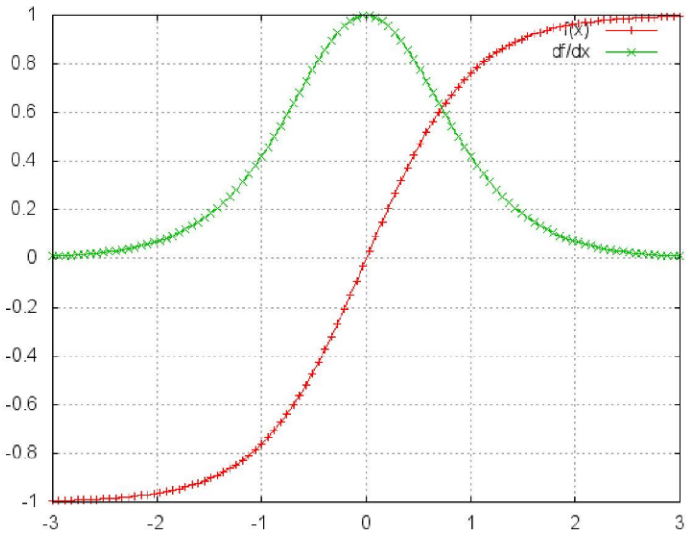



图 9: result

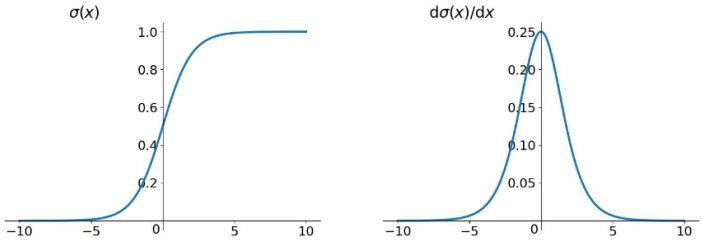



图 10: result

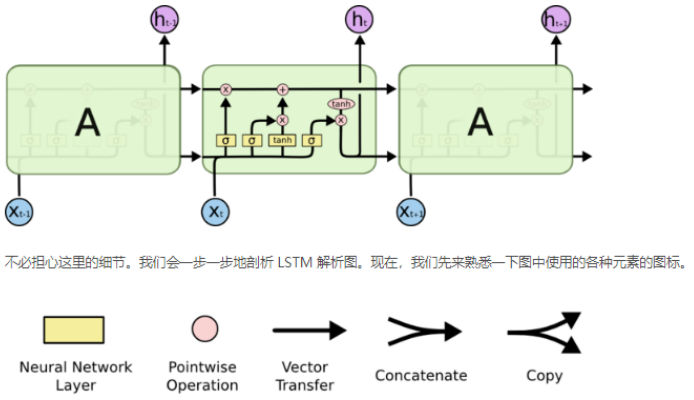



图 11: lstm

层较多时，在不断累乘的过程中，梯度可能会无限接近 0，或者很大，导致我们的无法正常更新参数。也就是‘梯度消失’和‘梯度爆炸’。为了解决梯度更新的问题，以及上文中我们说到的由于长时间的传导，导致最后对前文的内容已经遗忘过多。

## 2.4 LSTM

LSTM 将 rnn 中的普通 cell 更换为另外一种形式：

在 LSTM 中，为了对信息进行选择，共有三个门结构。分别为 forget gate，input gate，output gate。

在 forget gate 中，我们将上一个 cell 的输出和当前时间步的输入进行结合，然后通过一个 sigmoid 激活函数得到 ft（0，1），决定将上一个 cell 传达的信息保留多少，为 0 时全部丢弃，为 1 时全部保留。然后将 ft 和上一个细胞的输出相乘，得到我们需要保留的信息 ft*ct-1。

在 input gate 中，第一步我们同样得到一个 it，决定我们会更新当前的哪些值，同时将上一个 cell 的 hidden state 和当前时间步的输入结合通过一个 tanh 激活函数得到 ct，然后将 it 和 ct 相乘，决定将要保留多少重要的信息。然后我们将当前的结果和 it*ct-1 相加得到 ct。这是作为时间步向下一个 cell 的输出。并且如果在 forget gate 中的值接近 0，那门可能 drop 掉许多信息。

在 output gate 中，决定当前 cell 的输出 h，同样先经过一个 forget

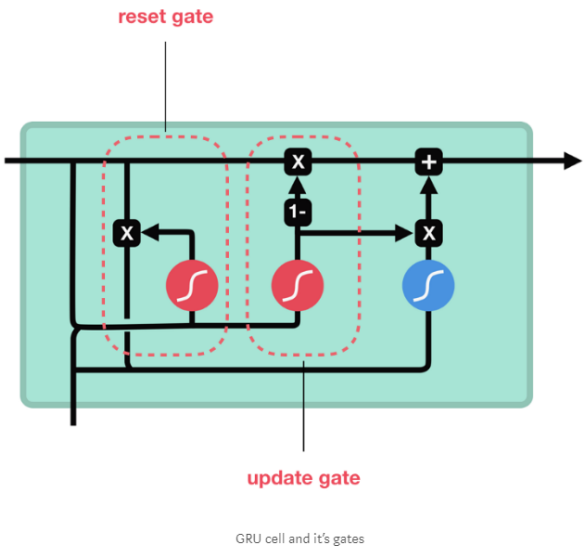



图 12: GRU

gate，然后和 ct 相乘，得到输出 ht，分别输出和传入下一个 cell。

通过以上的三个状态门，我们解决了信息传递丢失的问题，下面我们看看如何解决 gradient vanish 的问题。在对 ct-1 求导的过程中，后一项的数值大于 0 且为加法，我们可以只看主体部分，主体部分为一个值在 0 到 1 之间的数，所以避免了梯度消失。

## 2.5 GRU

作为 rnn 的另外一种变体结构，GRU 具有两个门结构，更新门和重置门。

## 2.6 Bi-rnn

在 rnn 中，往往只有前向序列的句子信息，也就是说一般当前词只包含了前方词语的信息。然而对于一些词来说，后面的部分也会对其产生一定的影响，于是使用双向的 rnn 能够获取到后方传递的信息。双向 rnn 的结构:

同时双向 rnn 中的 cell 可以换成 LSTM 或者 GRU。

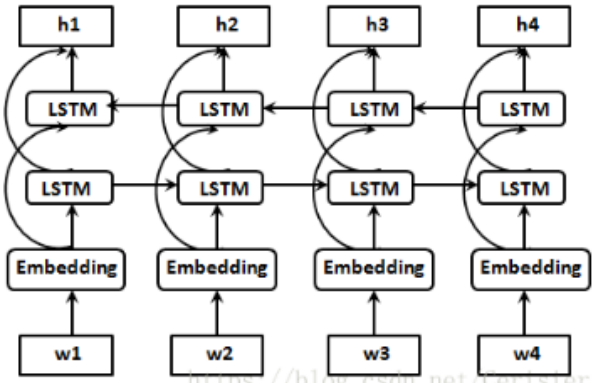



图 13: GRU

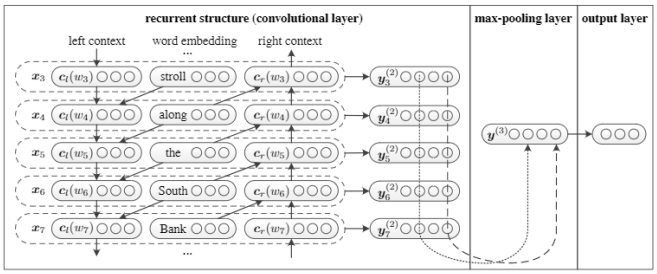



图 14: rcnn

## 2.7 rcnn

为了解决当前输入的词汇和上下文之间的联系问题，一种基于 rnn 和 cnn 的结构产生了。rcnn 能够将一句话中上下文词意融合到每一个词中。使得每一个词都包含更多的信息。rcnn 的结构为：

为了收集上下文对某一个词的影响，利用双向循环网络。在网络的第一层，正向输入句子序列，也就是每一个神经单元都可以获取当前输入以前的句子信息。输入同样为 [batch size,embedding sequence length]，首先将所有的输入嵌入到向量空间得到 [batch size,sequence length,embedding size] 的输入。然后我们对整个输入进行调整，将所有的句子按单词顺序划分，即每一句话的第一个词，第二个词.... 分别放在一组，得到一个长度为 sequence

length 的 [batch size,embedding size ] 大小的 tensor。然后在一开始我们初始化一个随机词向量 [batch size,embedding size] c left word ，作为第一个词的左边的词。同时我们初始化一个权值 W left context 作为在第一层循环神经网络之间传播的参数，它也可以看作我们初始化词左边的文章内容。然后将他们输入到一个神经单元，得到当前输入词左边的内容 [batch size，embedding size]。相反，我们将输入反序输入，得到一个词汇右边的内容 [batch size,embedding size]。然后我们将当前词左边的内容，当前词，当前词右边的内容进行结合。得到一个大小为 [batch size,3*embedding size ] 的 tensor，其中每一个代表了输入词和他的上下文内容的信息。将 tensor 还原，得到一个 [batch size,sequence length,3*embedding size ] 的 tensor，他保存着每一个句子中每一个融合了上下文信息的单词。以上是作为 rcnn 中的双向循环网络。然后将其通过一个最大池化层，将 [batch size,sequence length ,3*embedding size] 在 axis=1 的方向上求最大值，得到一个大小为 [batch size ,1,3*embedding size] 的 tensor，他代表着一个句子的所有信息。通过全连接层后的得到我们最后的计算结果。

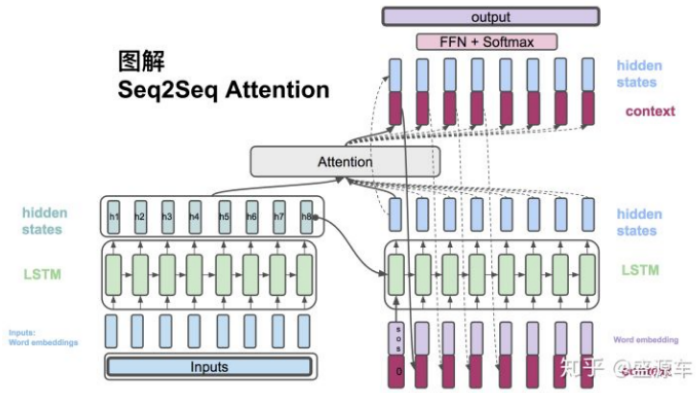
rcnn 能够收集到上下文的信息对某个词的影响，同时他还融合了 cnn 中的最大池化层，作为句子信息的提炼。经过实验，效果良好

## 2.8 seq2seq with attention

作为在翻译领域提出的模型，seq2seq 分为 encoder 和 decoder 两个部分。用一个循环神经网络将句子编码以后，通过另一个循环神经网络解码，这样可以翻译两种语言的不同长度句子。在 seq2seq 的基础上进一步加入了 attention 机制，使得整个文本中重要的部分得到提升，不重要的地方弱化。seq2seq with attention 的具体结构为：

本来模型是作用在翻译领域，但是经过改造也可以用在多文本分类上。作为文本的输入，同样是 [batch size,sequence length ] 代表 batch size 个长度为 sequence length 的句子输入，然后将输入嵌入到词向量空间 [batch size, sequence length ,embedding size].

encoder 部分: 使用一个双向 GRU 来作为句子的编码层：我们会得到两个输出。每一个 GRU 的双向单元会输入一个 hidden state，大小为 [batch size,hidden size]ht，同时将上一个 GRU 单元输出 ht 和当前的词作为下一个 GRU 单元的输入，得到 GRU 单元的输出。由于时双向 GRU 网络，每一层都会得到一个大小为 [batch size,sequence length,hidden size ] 的输出，将两




$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a [\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

层 GRU 的输出 concat，得到一个大小为 [batch size,sequence length,hidden size *2] 的 tensor。将两层最后一次的输出进行结合，得到一个 [batch size, hidden size *2] 的 encoder 结果。

decoder 部分：作为多分类任务，每一个句子都有多个属性标签。label 为 [batch size ,decoder length size ]。为了将 label 嵌入到向量空间，首先初始化一个 [ num classes,embedding size *2] 大小的标签 tensor，其中每一个 label 都初始化一个 tensor。在训练阶段，将我们的结果先嵌入到向量空间中，得到一个大小为 [batch size,decoder length size,embedding size *2] 的 tensor，代表每一个类别的 tensor。也是作为解码阶段的输入。同样，我们将类别 tensor 按顺序划分，即第一类，第二类...，得到一个长度为 decoder length 的 [batch size,embedding size*2] 的 label。在解码部分我们同样使用一个双向的 GRU 解码网络，解码网络的输出为，上一个 cell 的输出，当前句子的类别标签，经过 attention 后的 context。其中 attention 有多种方式：

我们选择第三种作为 attention 的方式。首先初始化一个大小为 [hidden size ,hidden size] 的权重 tensor 为 Wa，将上一个 cell 的输出结果 hs 和 Wa 相乘，然后将在解码层的全部输出和 Wa 相乘，对两者相求和后通过激活函数 tanh，然后和大小为 [hidden size ,1] 的 Va 相乘，从而得到一个 attention 的打分。attention 的方式也就是上文中的 concat。将计算得到的 score 和

我们上面解码网络中的 out 进行 multiply，对每一个 hidden state 的结果进行打分，然后求解在 axis=1 上的最大值，得到最后经过 attention 后的 context。然后放入 GRU 计算单元中，得到最新的 hidden state 的输出。

解码完成后，我们通过一个全连接层得到最后计算结果。

### 2.9 Hierarchical Attention Network

多层注意力网络是做为文档分类的一种重要算法。他收集每一个句子的信息，然后将句子的信息汇集成文档的信息。从而对文档进行分类，他的模型结构为：

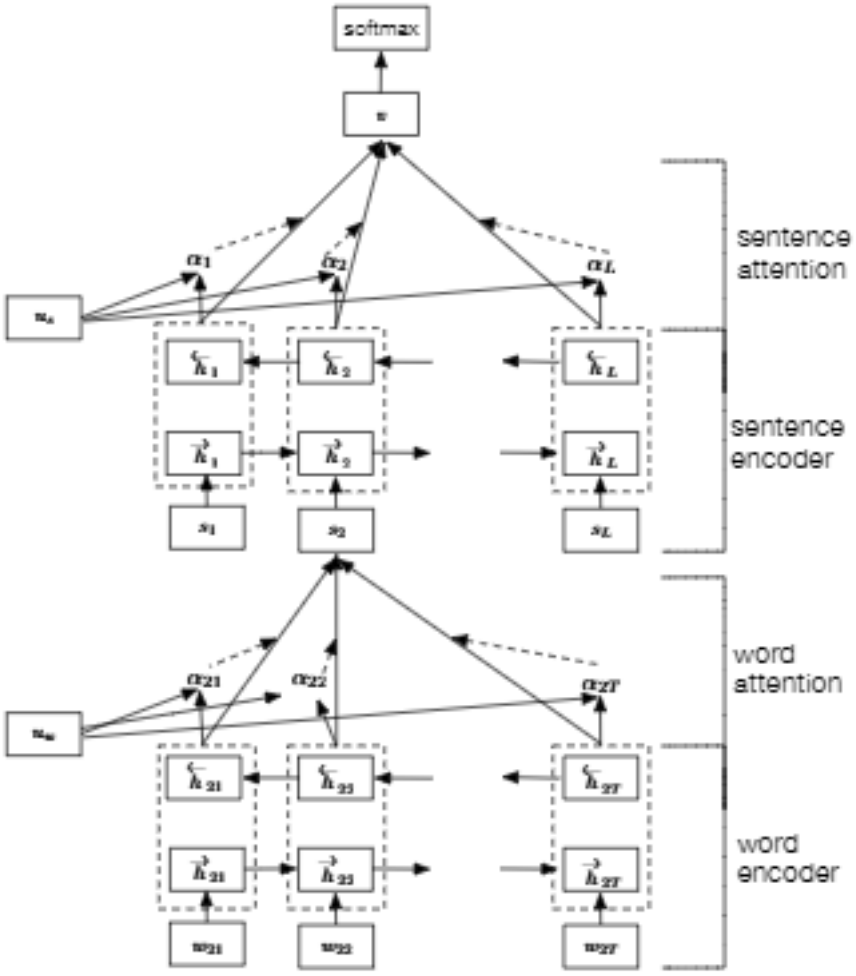
作为文档，输入为 [batch size, article length ,sentence length]，其中 article length 为每一篇文档的句子数量，sentence length 为每一句话的长度。同样我们要将这些文章的词嵌入到向量空间，得到一个大小为 [batch size,article length, sentence length ,word embedding size ] 的输入 tensor。我们对文章进行分割,得到一个长度为 article length 大小为 [batch size,sentence length,word embedding size] 的序列，其中每一组数据代表着每篇文章中的一句话。先对每一句话使用一个双层的 LSTM 或 GRU 进行信息选取，在 LSTM 的输出层后加入一个 attention 层，得到每一句话经过 attention 后的结果。然后在进行句子信息的提炼，得到最后的文章信息结果。

## 参考文献


[1] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.


softmax
sentence attention
sentence encoder
word attention
word encoder