REPORT for NLP-Beginner task 2

基于深度学习的文本分类

叶栩冰

1 Task & Data

1.1 Task

实现基于logistic/softmax regression的文本分类

1. 参考
    a. 文本分类
    b. 《神经网络与深度学习》 第2/3章
2. 数据集：Classify the sentiment of sentences from the Rotten Tomatoes dataset
3. 实现要求：NumPy

4. 需要了解的知识点：
    a. 文本特征表示：Bag-of-Word，N-gram
    b. 分类器：logistic/softmax regression，损失函数、（随机）梯度下降、特征选择
    c. 数据集：训练集/验证集/测试集的划分
5. 实验：
    a. 分析不同的特征、损失函数、学习率对最终分类性能的影响
    b. shuffle 、batch、mini-batch
6. 时间：两周

1.2 Data

烂番茄电影评论数据集是用于情感分析的电影评论语料库，最初由 Pang 和 Lee [1] 收集。在他们关于情绪树库的工作中，Socher 等人。[2] 使用 Amazon 的 Mechanical Turk 为语料库中的所有已解析短语创建细粒度标签。本次比赛提供了一个机会，可以在烂番茄数据集上对您的情绪分析想法进行基准测试。您被要求在五个值的范围内标记短语：消极、有些消极、中性、有些积极、积极。句子否定、讽刺、简洁、语言歧义等许多障碍使这项任务非常具有挑战性。训练集共有15万余项，语言为英文，情感分为以下五种情感：

> *0* - 消极
> *1* - 有点消极
> *2* - 中性
> *3* - 有点积极
> *4* - 积极

Example:

> *Input： A positively thrilling combination of ethnography and all the intrigue , betrayal , deceit and murder of a Shakespearean tragedy or a juicy soap opera .*
> *Output： 3*

2 Environment

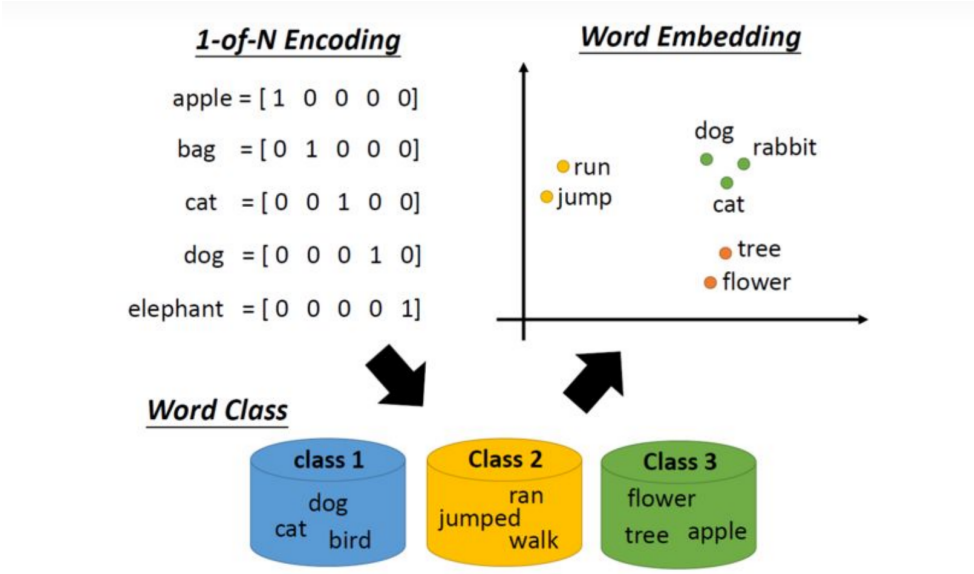
> *Python ~= 3.6*
>
> *IDE : Jetbrains Pycharm*
>
> *torch ~= 1.10.0*
>
> *pandas ~= 0.19.2*
>
> *torchtext ~= 0.12.0*
>
> *argparse ~= 1.0*

3 Method & Model

3.1 Word Embedding

3.1.1 Word Embedding定义

词嵌入模型即是把每一个词映射到一个高维空间里，每一个词代表着一个高维空间的向量。Word Embedding的输出就是每个word的向量表示。对于上文中的原始输入，假设使用最简单的one hot编码方式，那么每个word都对应了一种数值表示。此外，词向量本身与词向量间距也有意义。



- 词向量与词向量之间的距离能体现出词与词之间的相似性。相近词义的词语在高维空间中很相近。
- 词向量之间的距离一定程度上反映着单词之间的差异。

3.1.2 Word Embedding与任务一异同

词嵌入与任务一的词袋模型和N-gram稍有不同。词袋模型和N元特征所提取出来的的特征向量都是超高维的0-1向量，而词嵌入模型的向量每一维是实数，即不仅仅是0或1。

也就是说，词袋模型和N元特征所形成的特征矩阵是稀疏的，但是规模又很大，因而信息利用率很低，其词向量与词向量之间的距离也不能体现词间相似性。而词嵌入模型所形成的特征矩阵不是稀疏的，且规模相对较小，因此能更好的利用每一维的信息，并且没有明确的转换规则，因此我们不可能提前知道每一个词对应的向量，就无法定义出分类标准。

3.1.3 Word Embedding初始化

我们需要为上述提到的词向量参数设置一个合理的初始值。如果参数的初始值选的不好，那么优化模型求解的时候就会使参数值难以收敛，或者收敛到一个较差的极值；相反，如果选得好，就能求出一个更好的参数，甚至能起到加速模型优化的效果。因此设置合理的初始值十分重要。

- 随机初始化

  给定一个维度d（比如50），对于每一个词w，我们随机生成一个d维的向量$x \in R^d$。

  ```
  self.embedding = nn.Embedding(num_embeddings, embedding_dim)
  ```

- 预训练初始化

  即用于预训练好的模型进行初始化，如task中提及的GloVe词袋模型。

  ```
  self.embedding = nn.Embedding.from_pretrained(weight,
  freeze=False)
  ```

3.2 GloVe

GloVe 是一种用于获取单词向量表示的无监督学习算法。对来自语料库的聚合全局词-词共现统计进行训练，得到的表示展示了词向量空间的有趣的线性子结构。

GloVe模型就是将LSA和word2vec两种特征合并到一起的，即使用了语料库的全局统计（overall statistics）特征，也使用了局部的上下文特征（即滑动窗口）。为了做到这一点GloVe模型引入了Co-occurrence Probabilities Matrix。算法原理不在此进行赘述。

kaggle提供了GloVe的词向量库，本作业直接进行使用。利用到了如下内容：glove.6b.50d.txt、glove.6b.100d.txt、glove.6b.200d.txt、glove.6b.300d.txt。给出的demo格式如下：

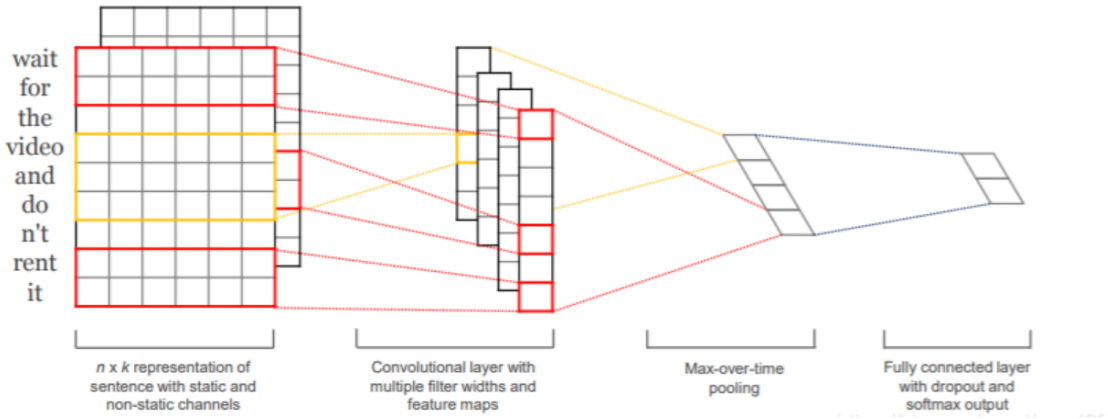
> *to 0.68047 -0.039263 0.30186 -0.17792 0.42962 0.032246 -0.41376 0.13228 -0.29847 -0.085253 0.17118 0.22419 -0.10046 -0.43653 0.33418 0.67846 0.057204 -0.34448 -0.42785 -0.43275 0.55963 0.10032 0.18677 -0.26854 0.037334 -2.0932 0.22171 -0.39868 0.20912 -0.55725 3.8826 0.47466 -0.95658 -0.37788 0.20869 -0.32752 0.12751 0.088359 0.16351 -0.21634 -0.094375 0.018324 0.21048 -0.03088 -0.19722 0.082279 -0.09434 -0.073297 -0.064699 -0.26044*

3.3 CNN

3.3.1 CNN简述

> 理论部分仅表明学习过程，由于篇幅限制不进行详细阐述。

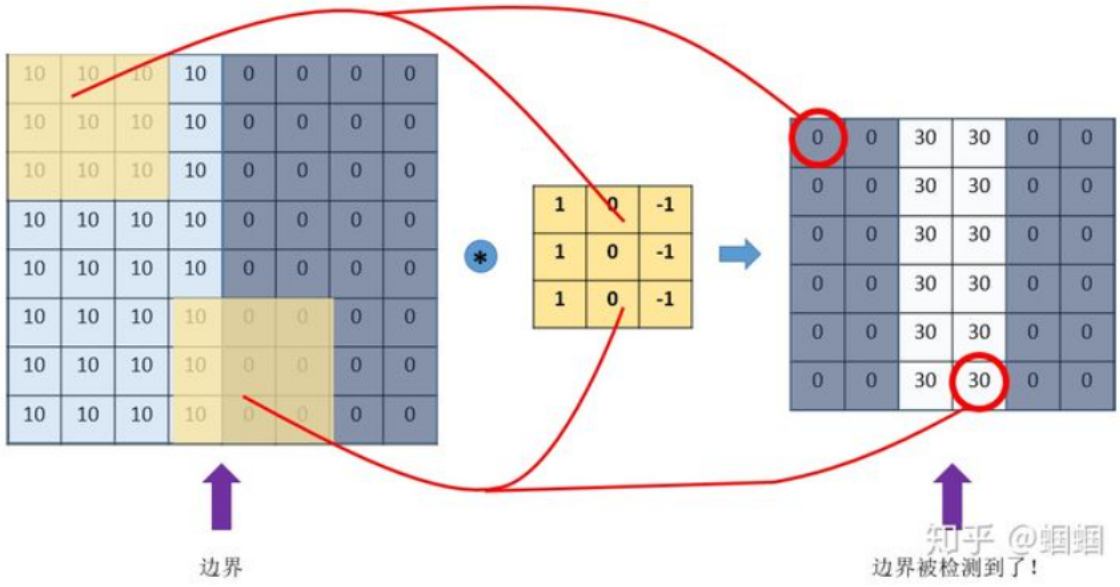
卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。 卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。



A 引入 边缘检测

10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0

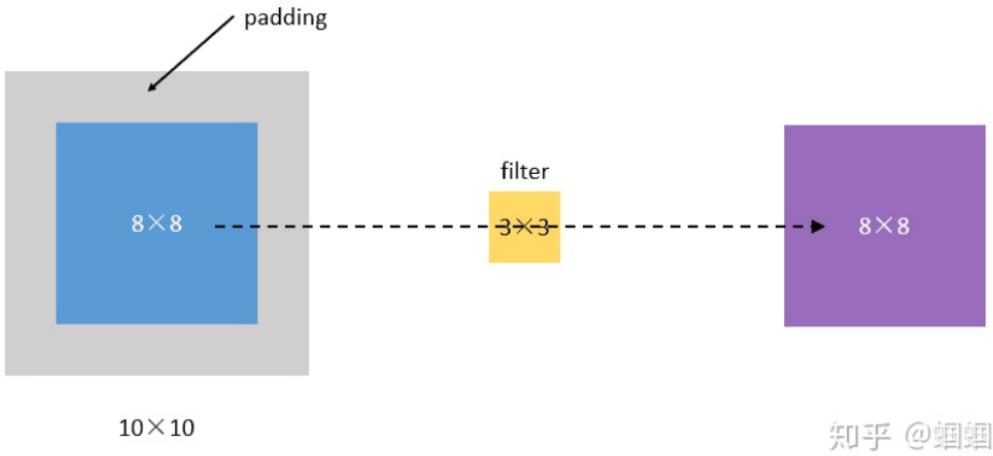
对于这样的一张图片，我们想要找到这张图片的边界，那么我们将如何找到中间这条竖线的边界呢，我们需要filter 滤波器，假设我们的滤波器大小为3x3 如下图中间所示，我们用滤波器与图片的矩阵相乘 得到新的数据，这样就可以检测到边界，我们把这个过程成为卷积，而卷积神经网络就是通过一个个不同特点的filter来对图片进行卷积，这样可以识别不同的特征，由于自主设计filter十分困难，那么机器去自主学习filter的参数便称之为CNN卷积神经网络。



这里有一点是值得注意的，你要知道为什么卷积神经网络是利用filter去进行点积，因为这样能够反应两矩阵的相似度，所以根据你想要的矩阵去设置filter。点积得到的值的大小能够反应相似程度。

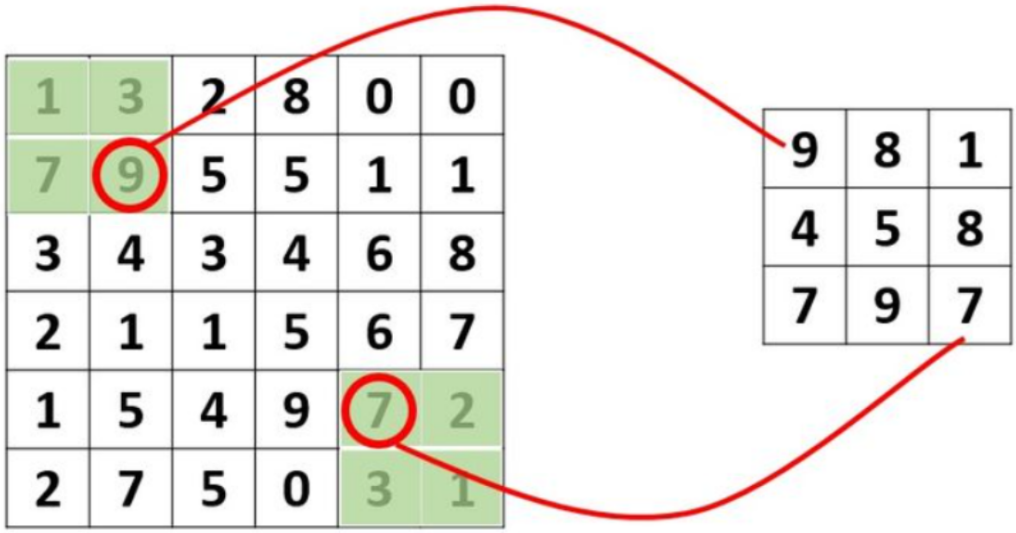
B 其他的**CNN**相关概念

- padding 每次卷积，图像都缩小，这样卷不了几次就没了； 相比于图片中间的点，图片边缘的点在卷积中被计算的次数很少。这样的话，边缘的信息就易于丢失。为了解决这个问题，我们可以采用padding的方法。我们每次卷积前，先给图片周围都补一圈空白，让卷积之后图片跟原来一样大，同时，原来的边缘也被计算了更多次。



- stride 步长 我们默认步长为1 即代表 88的输入卷积一次得到66 但是将步数调整为2的时候输出变为33 即一次移动2个距离

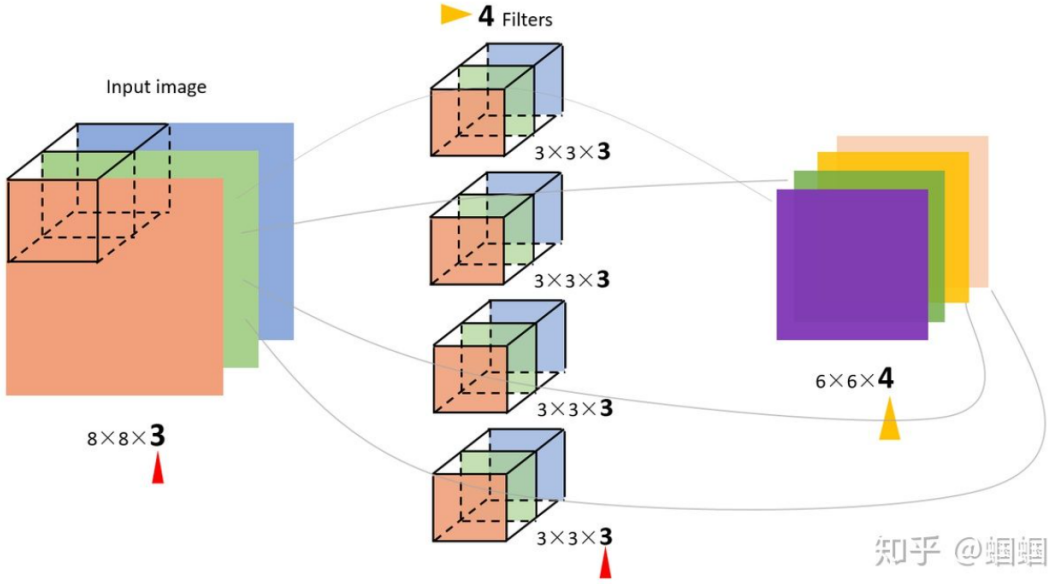
- pooling 池化 减少参数量 防止过度拟合 有不同的方式 例如取最大 取平均值



C 多通道的图片卷积

图片一般为RGB（长x宽x通道）那么同理就需要选取三维的filter

因此生成的最后卷积矩阵就和你的filter数量是有关系的了 如下图



输入参数为883 第一层神经网络参数为四个filter shape=(3,3,3,4)

输出层为最后的664 同样在输出后可能还会有一个激活函数

D CNN结构组成

- 卷积层

  是由filter和激活函数构成的，一般包含filter的数量大小步长以及padding的属性

- 池化层

  正常的参数都是指定好的，需要指定超参数，窗口大小、步长

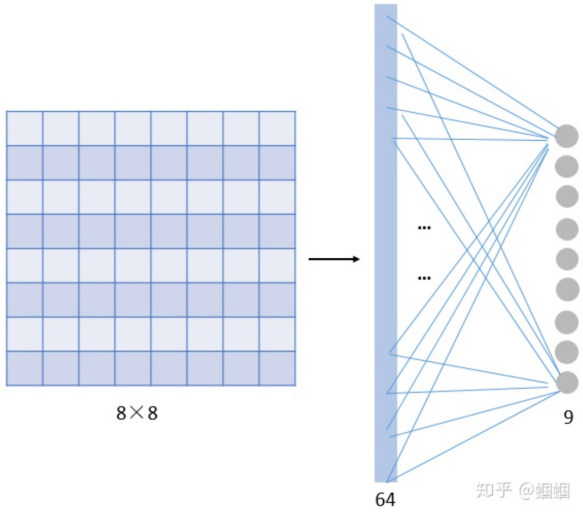
- 全连接层

  这个就是对应正常普通神经网络中的一排神经元，正常的话只需要限定神经元数量以及激活函数种类。可以发现883的图像矩阵在经过一次卷积处理之后变得扁平化，那么多维数据在经过多次卷积和池化后就会变得扁平，即将数据压扁，那么就会变成一维数组，与FC层连接后就会变得和正常神经元一样了。

E 对比

其实我们可以发现 卷积神经网络和普通神经网络之间的差别是先将三维矩阵经过CONV和POOl扁平而已，好处是什么呢？

- 参数共享（也可以说是减少参数）



这里我们可以看到如果使用最普通的神经网络要设置9*64*个参数，因为每个神经元都要确立权值和激活函数。但如果利用*filter*，*33*的矩阵只需要设置一个filter参数，这样不仅简化了参数设置的困难，还会有效避免过度拟合。（平移不变性）

- 链接稀疏性

  部分相关，而非全连接。而传统神经网络中，由于都是全连接，所以输出的任何一个单元，都要受输入的所有的单元的影响。这样无形中会对图像的识别效果大打折扣。比较，每一个区域都有自己的专属特征，我们不希望它受到其他区域的影响。

3.3.2 架构

本作业设置的卷积神经网络整体架构如下：

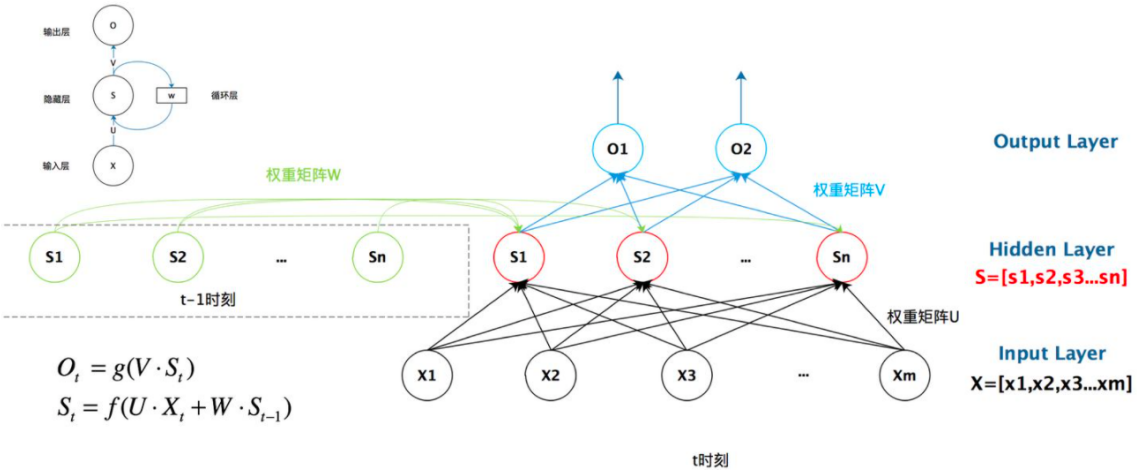
```
# Embedding
self.embedding = nn.Embedding.from_pretrained(weight, freeze=False)
# Convlution
self.convs = nn.ModuleList([nn.Conv2d(in_chaneels, kernel_num, (ks,
self.embedding_dim)) for ks in kernel_size])
# Drop
self.dropout = nn.Dropout(args.dropout)
# Full connection
self.fullconnection = nn.Linear(len(kernel_size) * kernel_num,
label_num)
```

3.4 RNN

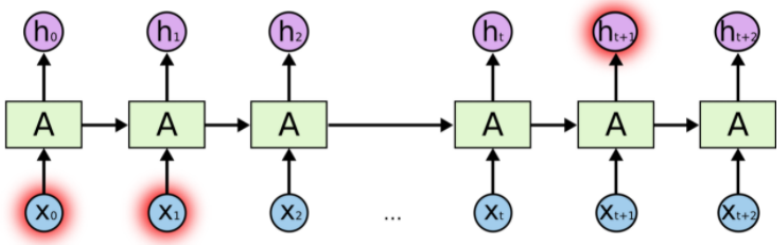
3.4.1 RNN简述

> 理论部分仅表明学习过程，由于篇幅限制不进行详细阐述。

循环神经网络（Recurrent Neural Network, RNN）是一类以序列（sequence）数据为输入，在序列的演进方向进行递归（recursion）且所有节点（循环单元）按链式连接的递归神经网络。

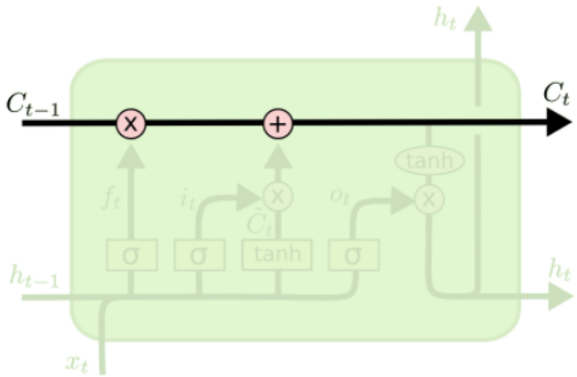


这张图就说的很清晰了，为什么要引入循环神经网络，是因为上一次的输入会影响到本次的输出结果，因此隐藏层需要一个权重矩阵来记录前n-1此输入后的权重作为一个影响作用到第n次输入的隐藏层。



然而，RNN有时候也会遇到些问题，当文章过长，产生联系的跨度非常大的时候，例如上图，当在文末出现的词语需要与文章开头产生联系与作用的时候，那么RNN将无法做到这一点，就需要LSTM网络的引入。

LSTM网络的关键在于增加了访问输出的能力，未来任意时刻都可以访问当前节点的输出。这样就可以颠覆以往的只能记住最后状态的普通循环神经网络模型。



3.4.2 架构

本作业设置的循环神经网络整体架构如下：

```
# RNN
```

```
self.rnn = nn.RNN(input_size=embedding_dim,
                      hidden_size=self.hidden_size,
                      num_layers=self.num_layers,
                      batch_first=True,
                      bidirectional=self.bidirectional)
# LSTM
self.lstm = nn.LSTM(input_size=embedding_dim,
                      hidden_size=self.hidden_size,
                      num_layers=self.num_layers,
                      batch_first=True,
                      bidirectional=self.bidirectional)
# Full connection
if self.bidirectional:
        self.fullconnection = nn.Linear(self.hidden_size * 2,
label_num)
else:
        self.fullconnection = nn.Linear(self.hidden_size,
label_num)
```

4 Train & Result

4.1 Train

根据上述模块所述的Model构建模型。由于数据量较为可观，因此将数据集按照train: crossValidation: test=6:2:2比例进行划分，以便于及时对模型进行验证与保存，防止过拟合。我们共训练了五组模型，分别是RNN+Glove、RNN+Random、CNN+Glove、CNN+Random、(LSTM+Random)，这样我们可以分析CNN与RNN、使用GloVe和随机初始化之间的优劣关系，构成对比模型。

两模型具体参数如下：

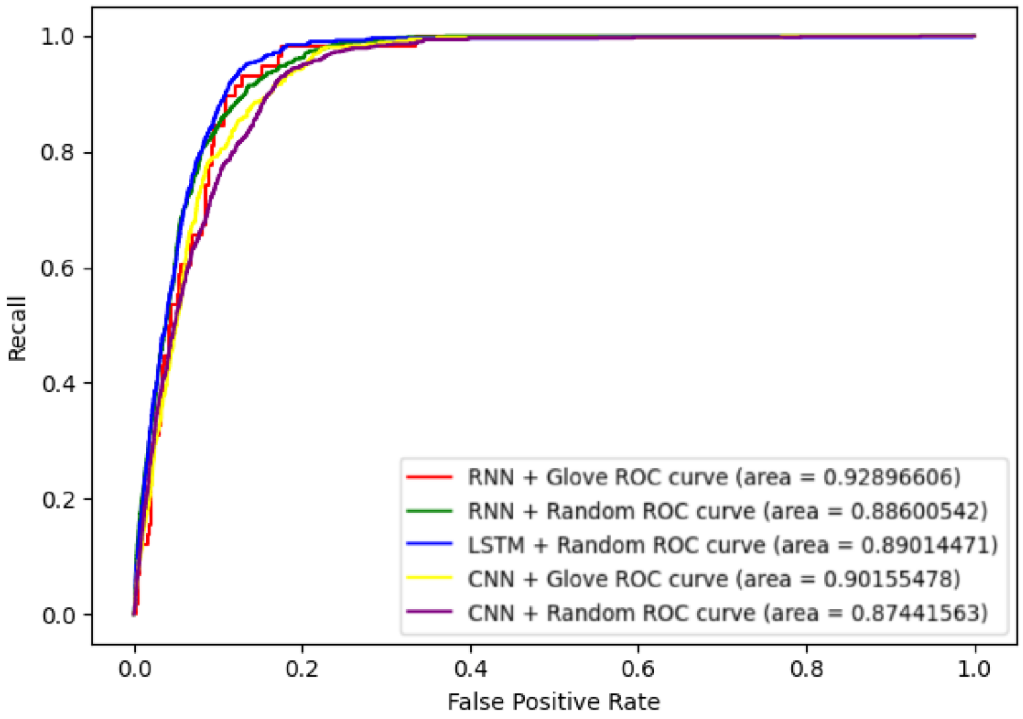
- 样本量：150000
- train: crossValidation: test=6:2:2
- 初始化：随机初始化 / GloVe预训练模型初始化
- l_h, d : 50
- epoch : 20
- kernal_size : 2, 3, 4
- kernal_num : 100
- batch_size：500
- loss : cross_entropy 交叉熵损失函数
- learning rate：$10e-3$

4.2 Result

模型训练结果如下表，由于准确率并未达到十分令人满意的水准，因此摄入AUROC等参数作为模型评判与对比的标准，从而更好地分析模型准确率、敏感度、稳定性等因素。

MODEL	ACCURACY	F1-SCORE	RECALL	AUROC
RNN+Glove	0.672541	0.684412	0.699478	0.928966
RNN+Random	0.597702	0.602247	0.754412	0.883005
CNN+Glove	0.654423	0.666554	0.700182	0.901555
CNN+Random	0.584771	0.591477	0.810024	0.874416

下图为五种模型的Classification ROC Curve。



4.3 Analysis

1. RNN&CNN

   我们对随机初始化时两种模型的预测结果进行横向对比分析，可以看到RNN在测试集的准确率、F1分数、AUROC值以及RECALL比CNN都要高，在训练时我也发现其损失值更低。因此得出结论RNN模型的准确性、稳定性与敏感度方面要比CNN表现更好。

2. GloVe&Random

   从结果可分析GloVe初始化要比随机初始化的结果表现要显著更好。随机初始化是不遵循任何规律进行初始化，因此使用GloVe更具有优势。

3. 准确率~=66-70 AUROC~=90

虽然模型准确率并没有较作业一有很大的提升，但是总体来说两种网络模型的ROC曲线下面积都达到了一个很可观的水平，因此无法将准确率低的原因归结于模型于训练参数。我认为主要有以下几点参考因素：

a. 数据集标签本身的准确性。
b. 解析文本整体的弊端。虽然RNN可以很好的构建起前后文之间的语义联系，但是归根结底还是无法从根源解决解析文本的坏处。Log-based_Anomaly_Detection_Without_Log_Parsing一文中分析了解析文本的主要方法（如Drain等），因此利用无解析文本（如BERT）的方法能够更好的解决语义关系一问题。
c. 词袋模型的局限性。我们无法保障所有词语都是存在于词库当中的，因此对生僻词预测时会出现不准确的问题也就是OOV问题。

以上为我第二次作业报告的全部，请您批评指正。

顺祝学业顺利，心情愉快！

叶栩冰