

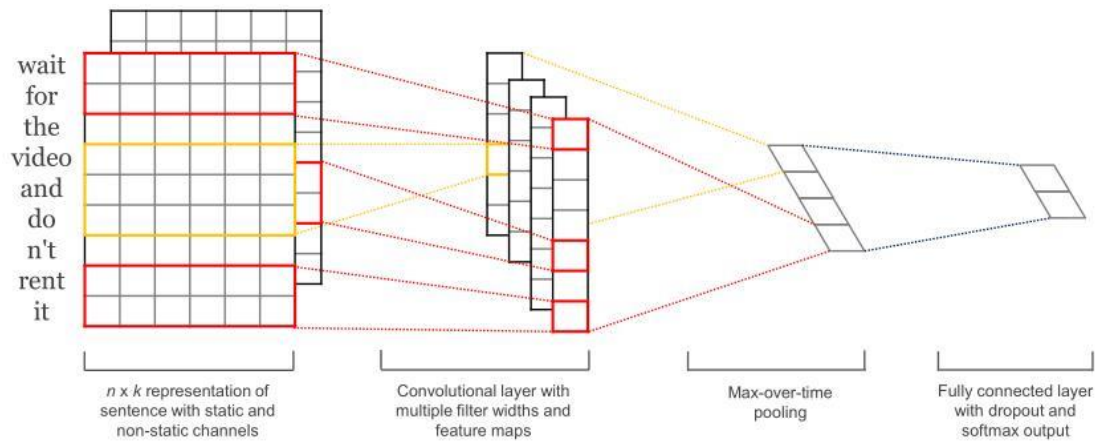
# 文本情感分析

陈新 2018013443

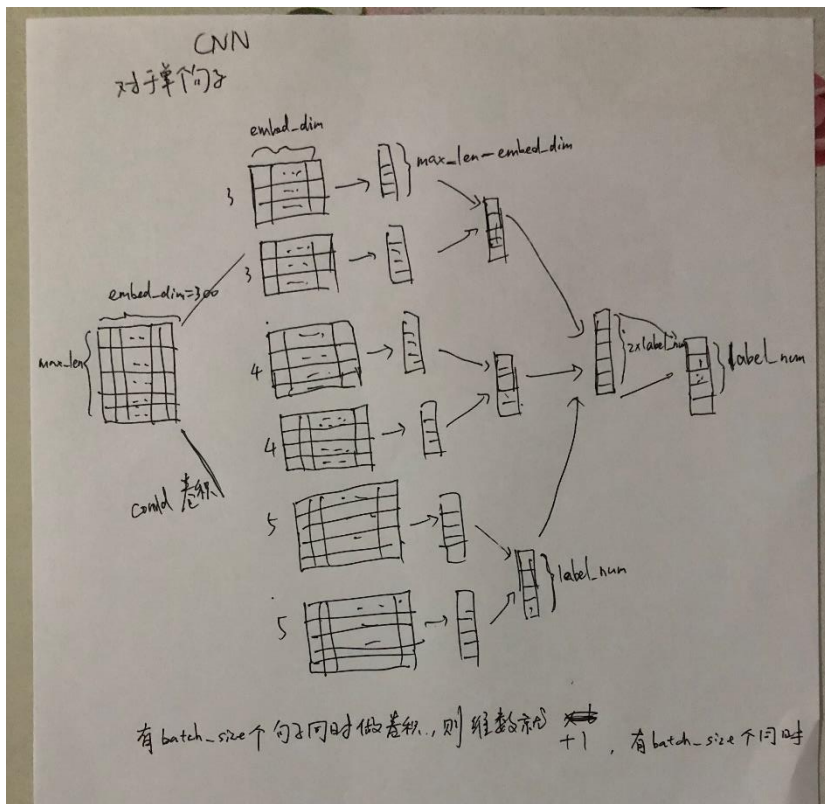
## 1 模型、流程

词向量模型均使用下载的 300 维 word2vec

### CNN



上图为 CNN 模型的网图。与之类似，详细些将中间步骤画出，则我的模型图如下：



画的是单个 sample 的图，若考虑一个 batch，只须添加一维 batch\_size 即可

```

self.convs = nn.ModuleList([nn.Conv1d(in_channels=1,
                                       out_channels = config.kernel_dim,
                                       kernel_size = (K, config.embed_dim)) for K in config.kernel_sizes])
self.dropout = nn.Dropout(config.dropout)
self.fc = nn.Linear(len(config.kernel_sizes) * config.kernel_dim,
                    out_features=config.label_num)

self.loss_fn = nn.MSELoss(reduction='mean')

def forward(self, input, is_training=True):
    # originally, inputs = (batch_size, sentence_len, embed_dim) word2vec
    input = self.embed(input) # inputs = (batch_size, sentence_len, embed_dim) word2vec
    input = input.unsqueeze(1) # get (batch_size, 1, sentence_len, embed_dim)

    input = [F.relu(conv(input)).squeeze(3) for conv in self.convs]

    input = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in input]
    concated = torch.cat(input, 1)
    concated = concated.view(-1, concated.size(1))

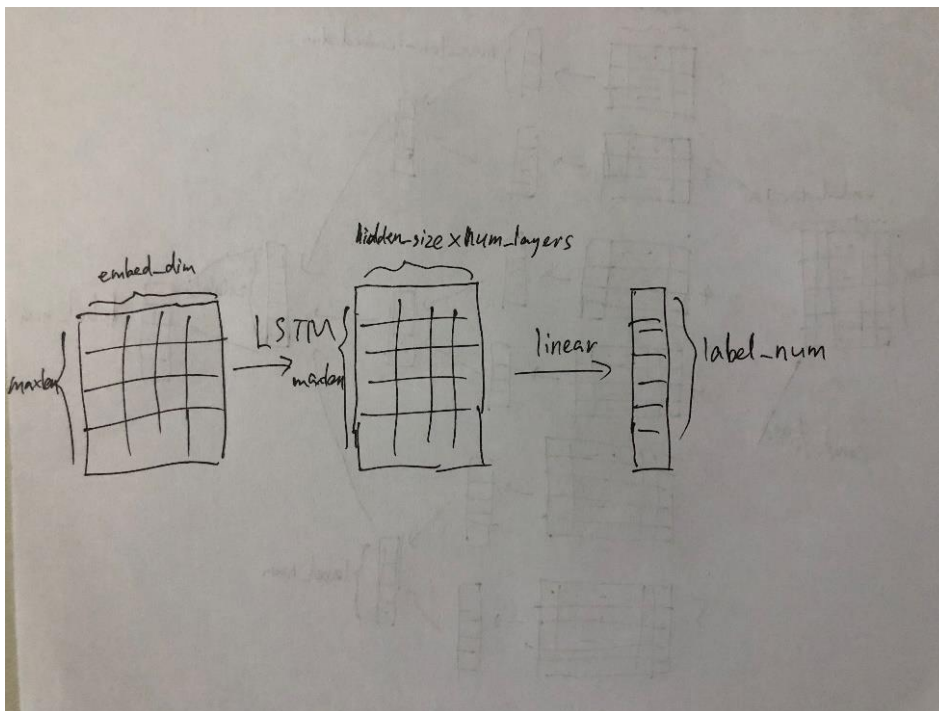
    if is_training:
        concated = self.dropout(concated)
    out = self.fc(concated)

```

CNN 网络中，情感标签我使用的是归一化情感分布  
依次进行卷积、激活函数、最大池化、concat、丢弃(dropout)

## RNN

采取 LSTM



```

self.lstm = nn.LSTM(config.embed_dim,
                    config.hidden_size,
                    config.num_layers,
                    bidirectional=True,
                    batch_first=True,
                    dropout=config.hidden_dropout)
# self.fc_dropout = nn.Dropout(config.fc_dropout)
self.linear = nn.Linear(config.hidden_size * 2,
                        config.label_num)

self.loss_fn = nn.CrossEntropyLoss(reduction='mean')

def forward(self, input, is_training=True):
    # originally, inputs = (batch_size, sentence_len, embed_dim) word2vec
    input = self.embed(input)      # inputs = (batch_size, sentence_len, embed_dim) word2vec

    out, _ = self.lstm(input)
    out = self.linear(out[:, -1, :])

    return out

```

RNN 中，使用的情感标签为最大值的单标签预测

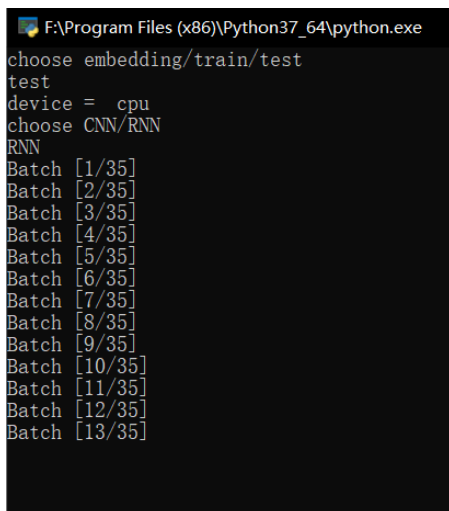
由于所剩时间不多，RNN 写得非常简单。

一个 LSTM 层，外加一个全连接

## 运行

运行时 python run.py

此后在两次 input() 中依次输入对应指令。第一次 train 或者 test，第二次 CNN 或者 RNN，如下图：



```

F:\Program Files (x86)\Python37_64\python.exe
choose embedding/train/test
test
device = cpu
choose CNN/RNN
RNN
Batch [1/35]
Batch [2/35]
Batch [3/35]
Batch [4/35]
Batch [5/35]
Batch [6/35]
Batch [7/35]
Batch [8/35]
Batch [9/35]
Batch [10/35]
Batch [11/35]
Batch [12/35]
Batch [13/35]

```

需要注意的是，config.py 修改中 USE\_GPU 可决定是否使用 cuda

由于在 RNN 的 LSTM 函数中需要 cuDNN，这在 CNN 中是不需要的，所以 CNN 正常跑完了，但我在 RNN 部分遇到了报错

cuDNN error: CUDNN\_STATUS\_EXECUTION\_FAILED

且经检查 cudNN、cuda 和显卡驱动版本均对应，找不出问题所在，故只能用 CPU 训练，导致迟交了作业；同时 RNN 只能保证 CPU 能正常运行程序，而 cuda 版未知。希望助教测试时能够帮我试一下 RNN 的 cuda 是否能够正常运行，

如果不能则使用 `cpu` 运行。

## 2 实验结果

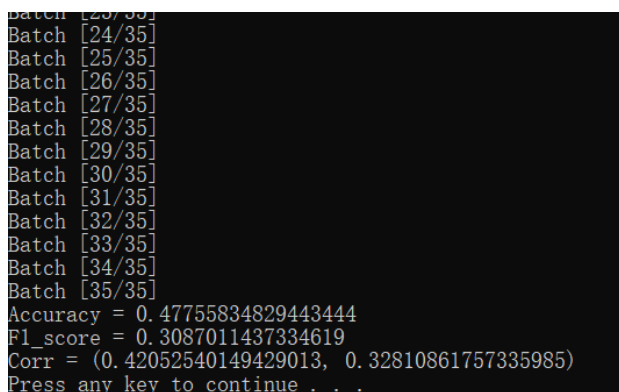
我计算 `F1_score` 是利用 `weight-averaging` 的宏平均计算的  
取最佳

### (1)CNN, dropout=0.6

```
accuracy = 0.5884201077199281  
F1_score = 0.5435800930492517  
corr = (0.6515339651293006, 0.1752778813012905)
```

### (2)RNN, dropout=0

```
Accuracy = 0.47755834829443444  
F1_score = 0.3087011437334619  
Corr = (0.42052540149429013, 0.32810861757335985)
```



```
Batch [23/35]  
Batch [24/35]  
Batch [25/35]  
Batch [26/35]  
Batch [27/35]  
Batch [28/35]  
Batch [29/35]  
Batch [30/35]  
Batch [31/35]  
Batch [32/35]  
Batch [33/35]  
Batch [34/35]  
Batch [35/35]  
Accuracy = 0.47755834829443444  
F1_score = 0.3087011437334619  
Corr = (0.42052540149429013, 0.32810861757335985)  
Press any key to continue . . .
```

可能由于我写的 RNN 太过简陋，各项指标都显得非常低

此外还发现，在 **dropout=0.5** 的情况下，`word2vec` 词向量的不同会影响正确率。猜测一小部分原因是由于给出的 `train` 和 `test` 语料均来自于新闻，人民日报的语料类型接近，都是新闻，而 `merge` 则综合了百科等各类语料，类型重合度不如人民日报。但是这应该对正确率的影响应该不高，其余大部分原因未明。

对于 CNN（在 **dropout=0.5** 的情况下）：

#### a. 人民网 word2vec

```
accuracy = 0.5628366247755835  
F1_score = 0.5311799570818034  
corr = (0.6275760870212104, 0.19237738102029847)
```

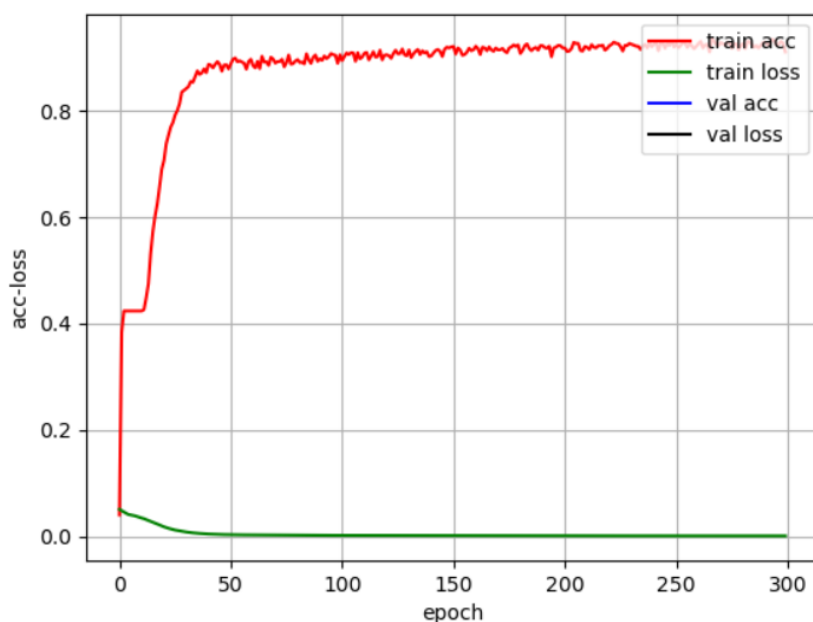
#### b. merge word2vec

```
accuracy = 0.5816876122082585  
F1_score = 0.5222787601930754
```

corr = (0.6431785368823646, 0.18266199850833184)

### 3 参数比较

#### (1) epoch



根据 epoch-acc/loss 图来看，epoch=20 是一个比较合适的值。再大则可能导致过拟合。

#### (2) dropout

CNN:

##### Dropout=0.7

[ accuracy = 0.5906642728904847  
F1\_score = 0.5284639245640056  
corr = (0.6450185719072878, 0.17938383280554107)

##### Dropout=0.6

[ accuracy = 0.5884201077199281  
F1\_score = 0.5435800930492517  
corr = (0.6515339651293006, 0.1752778813012905)

##### Dropout=0.5

[ accuracy = 0.5628366247755835  
F1\_score = 0.5311799570818034  
corr = (0.6275760870212104, 0.19237738102029847)

### Dropout=0.4

```
[ accuracy = 0.531867145421903  
  F1_score = 0.5276767278979894  
  corr = (0.6183038484676653, 0.18839904645905584)
```

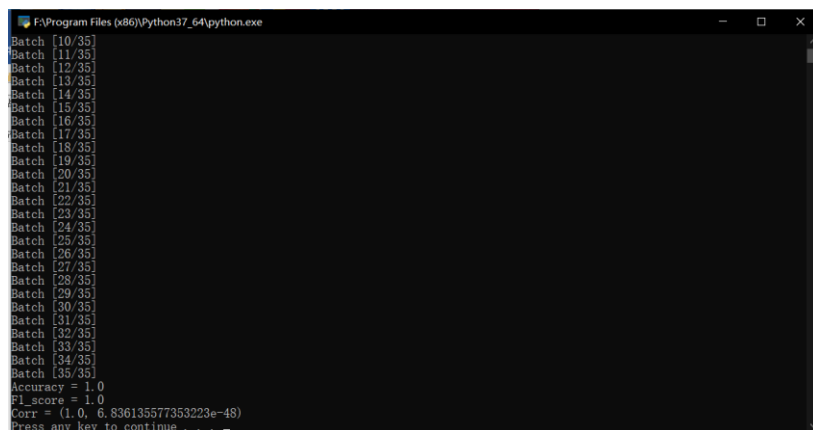
### Dropout=0.2

```
[ accuracy = 0.5498204667863554  
  F1_score = 0.5301647920000986  
  corr = (0.6252655983790485, 0.1942593648195852)
```

经由以上尝试，最终权衡三个参数的总体最大值，选择 CNN 的 dropout=0.6

PS：有一次奇怪的高正确率也值得思考

```
[ accuracy = 1.0  
  F1_score = 1.0  
  corr = (1.0, 6.836135577353223e-48)
```



```
F:\Program Files (x86)\Python37_64\python.exe  
Batch [10/35]  
Batch [11/35]  
Batch [12/35]  
Batch [13/35]  
Batch [14/35]  
Batch [15/35]  
Batch [16/35]  
Batch [17/35]  
Batch [18/35]  
Batch [19/35]  
Batch [20/35]  
Batch [21/35]  
Batch [22/35]  
Batch [23/35]  
Batch [24/35]  
Batch [25/35]  
Batch [26/35]  
Batch [27/35]  
Batch [28/35]  
Batch [29/35]  
Batch [30/35]  
Batch [31/35]  
Batch [32/35]  
Batch [33/35]  
Batch [34/35]  
Batch [35/35]  
Accuracy = 1.0  
F1_score = 1.0  
Corr = (1.0, 6.836135577353223e-48)  
Press any key to continue . . .
```

CNN 调到 0.2 看到结果的时候是吓了一跳的，但后来选取了部分测试 sample 手动计算 max\_label，多次运行程序发现基本全是吻合的，但在重启机器之后就消失了。多次重复的答案基本一样可以用随机 seed 来解释，但是基本全对的正确率却很迷。

### RNN:

#### Dropout=0.2

```
[ accuracy = 0.47755834829443444  
  F1_score = 0.3087011437334619  
  corr = (0.4154294451503728, 0.3296156024386623)
```

#### Dropout=0

```
[ accuracy = 0.47755834829443444  
  F1_score = 0.3087011437334619  
  corr = (0.4154294451503728, 0.3296156024386623)
```

```
F1_score = 0.3087011437334619
corr = (0.42052540149429013, 0.32810861757335985)
```

虽然权衡之后 dropout 选择为 0，但可见 accuracy 和 f1\_score 几乎不变，而且均处于一个低水平，说明大概率不是参数的问题，而是模型建立有问题。但由于遇到了预料外的 bug，跑完发现为时已晚。

### (3)learning\_rate

**CNN（基于 dropout=0.6）：**

**Learning\_rate=1e-2**

```
{ accuracy = 0.5502692998204668
  F1_score = 0.49026331054725286
  corr = (0.6029155778926324, 0.20076657061854025)
```

**Learning\_rate=4e-3**

```
{ accuracy = 0.5884201077199281
  F1_score = 0.5435800930492517
  corr = (0.6515339651293006, 0.1752778813012905)
```

**Learning\_rate=1e-3**

```
{ accuracy = 0.5574506283662477
  F1_score = 0.5492435757636059
  corr = (0.6542122635551353, 0.16291250902818633)
```

**Learning\_rate=1e-4**

```
{ accuracy = 0.5816876122082585
  F1_score = 0.45345896830308424
  corr = (0.6078401384740152, 0.2219425716212173)
```

得出 learning\_rate=4e-3 这个数量级时，大致效果最好

## 4 问题思考

### (1) 实验训练的停止

**【方案 1】**在上一节讨论过，通过预先的训练与反馈，根据 epochepoch-acc/loss 图来看，epochs 取 20 是一个比较合适的值，确定一个固定的训练轮数之后停止。是我现在采用的形式。

**【方案 2】**记录每次 loss 的降低值，若连续超过 k 个 batch 的下降程度不大于一个阈值或者反而升高，就终止训练。但这个方案的问题在于，我们训练样本本来就小，连续 batch 不足以达到终止训练的 k；若将 k 设置小了，则暂时

性上涨的 loss 峰也会导致训练提前结束，得不偿失，故不用这种办法。

【方案 3】边训练边抽取 test 样本计算准确率、F1\_score、corr 等，达到极大值时终止训练。但与方案 2 一样，尚未想到好的方法判别是否过了暂时性高峰后面还能有提升，故暂时不采取，但总的来说可行性大于方案 2

## (2) 参数初始化

Pytorch 中用 optimizer 的 Adam 模型初始化参数。尝试了 SGD 等模型，loss 的初始随机性与方差较大，且下降速度有时会非常非常慢，故仍然选择 Adam

## (3) 过拟合

把 learning\_rate 设置为随着训练 epoch 的增长而递减，应该是一个有一点效果的方案，但未做尝试。

其余详见第(1)问

## (4) CNN、RNN、MLP 比较

未写 MLP 模型，但从理论上讲，由于其未关注上下文关系，而是变相地削弱了文本上下文关系变成了一个词袋模型；虽然词袋模型并不差，但从直观上来看应该不如具有严格上下文关系的模型。

从训练过程中 loss 的对比和结果来看，CNN 比 RNN 更准也更快。CNN 卷积窗口宽度就是在考虑上下文关系，然后最大池化就是把其中影响力最大的拿出来作为参考。

但 RNN 把词语的上下文关系严格考虑进去了，双向 RNN 甚至还能正序逆序分别考虑句子。但是问题在于 RNN 模型中越接近最后的单词权重越大，这一点不如 CNN。

从理论上讨论了很多，但是这个样本只有 6MB，结果 CNN 效果总是比 RNN 好很多，因此好像得不出通用结论。

## 5 心得体会

这次作业的学习跨度非常大，给没有接触过机器学习的同学（包括我）来说带来了巨大的挑战。上学期学过的同学/大佬的最大问题在于调参，这也是机器学习最主要研究的地方；而我则花了一周半的所有课余时间学习写 CNN、RNN 框架，最终留给我调参的时间便所剩无几了。

这次作业最大的戏剧性在于 dropout 从 0.5 调到 0.2 之后的正确率奇怪飙升现象。但我的基础比较薄弱，学艺不精，只能猜测是过拟合的问题。

此外，不同的初始化模型、不同的参数也会有不同的效果，有时差距还非常大，需要我们尝试，黑盒和白盒的差距就在此，这也是我第一次接触黑盒算法，这次作业对我理解两种算法的差异算是一个启蒙。但仅仅接触了一会儿工夫，所



以我对黑盒算法的不透明度是感到不太满意的，因为写完了框架最终得出的结果好坏仅仅由参数的选择决定，而这是技术含量极低的，并且极具随机性的过程、

最后，由于莫名的原因，训练 `cuDNN` 时一直报错，让我不得不用 `CPU` 训练完了一小部分的 `RNN` 网络，这才真切体会到没有显卡的难处。希望以后能提供类似远程 `GPU` 给没有显卡的同学使用（性能也不用太强，让有显卡的同学不必卷来卷去抢资源），缩小硬件差距，同时也给像我这种出临时 `bug` 的同学提供一个应急手段。