人智导·情感分析实验报告

计94 周奘 2019011301

2021.6.5

1. 模型分析

1.1 总体思路

情感分析是一个典型的 NLP 任务。本次实验的任务，可以看作一个单标签（label）的多分类任务（具体来讲是7分类，7种不同的情感）。通过词嵌入（Word-Embedding）的方法可以把一个单词看作一个 k 维的向量 $x_i \in \mathbb{R}^k$。那么，一个由 n 个单词构成的句子即可表示为一个 $n \times k$ 的矩阵：

$$X = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^{n \times k}$$

从而，可以将原句子（sentence）处理为这样的矩阵，情感标签（label）处理为 k 向量，通过神经网络建立一套对应关系。

本次实验所有的机器学习框架为 **PyTorch** ，数据集使用 v2 。没有采用现有的预训练语言模型。

具体的代码包说明及程序运行方法见 README 。

1.2 数据预处理

STEP 0

数据清洗。由于原始 csv 文件中存在许多与文本语义无关的奇怪字符，因此去除了这些特殊字符，并统一了大小写。

```
,label,sentence
6069,shame,"This is an event I will never forget.  I am considered a good á mimic.  Th
4175,fear,"Fights that I see in the middle of the street, robberies, fear of á darknes
225,joy,When I met an old schhol friend (school love) who works in á Sk+vde.
1385,anger,"I wanted to go to a friend of mine who had a party; at the moment á I want
6713,joy,"When I passed standard 8 exams I jumped, rolled over the ground."
5239,guilt,When I was scolded by my dad for bad behaviour.
774,guilt,"I had planned a trip to the South during Easter, together with a á friend.
1242,guilt,The cat of my landlady escaped through the window which I had á left open.
1808,fear,I was afraid before I went to the doctor's.
4995,sadness,The death of my guardian with whom I had stayed when I did my á grade six
2201,disgust,I was on a walk with my child when I met a drunk woman.
7261,joy,Talking with a nice girl.
6893,anger,South Korea cheating for gold medals.
2679,sadness,I saw a friend of mine who had suffered a very heavy contusion á after an
648,shame,The day after the night I drank too much at a party.
3530,anger,Most recently when the chairman at the New Zeeland Rugby Union á announced
959,anger,When a girl left me.
7049,anger,I wanted to borrow lecture notes from a friend and he did not á lend me the
7331,joy,When I read the newspaper that morning I found my candidature á number on the
```

STEP 1

构造语料库。将 train 与 valid 两个数据集的词语集中在一起，构建语料库。

STEP 2

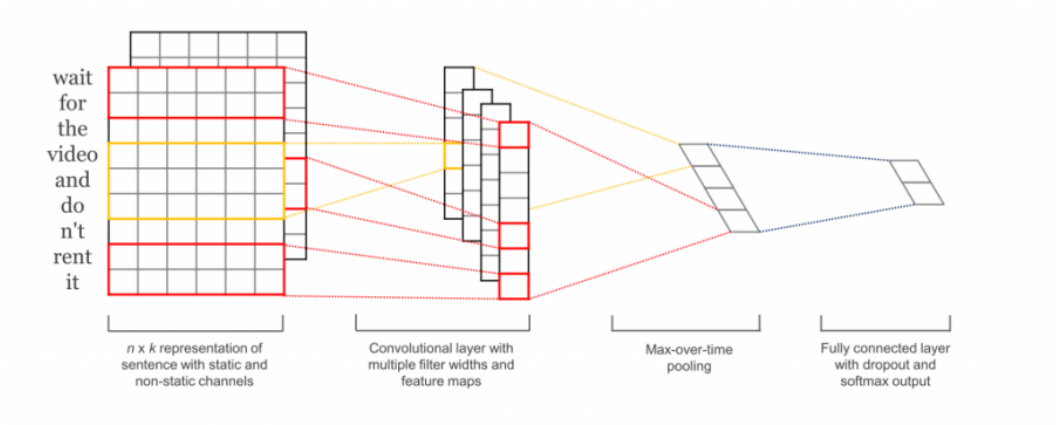
通过语料库构建 Word2Vec 。采用 `gensim` 库的 `word2vec` 进行词向量的构建。本次实验中，训练的词向量维数为 128 。

vec > my_w2v_128.w2v

```
6719 128
i -0.13074553 0.07231826 0.1548066 -0.15718101 0.06969588 0.056221884 -0.071833566 -0.0830640
the -0.112068154 0.12812665 0.08266078 0.079041384 -0.07013923 0.08414 -0.119941935 -0.070626
a 0.080039285 -0.025908235 0.0040478064 -0.100082055 0.12652542 -0.019626454 0.09611658 -0.10
to 0.06320667 -0.14585112 0.02702192 -0.09687348 -0.12277502 -0.028326714 -0.102828704 0.1459
was 0.12743907 0.090669595 -0.02172505 0.09796018 -0.1048096 0.0918583 0.06064493 -0.04961562
my -0.06919431 0.123639844 -0.06319045 -0.04210607 -0.013301077 0.011130889 0.08466584 -0.009
and 0.12338549 -0.037148338 0.01467597 -0.084085636 0.100513846 0.13411124 0.021424446 -0.098
, 0.030032137 -0.14512363 -0.114529796 0.11624443 0.106951684 0.12077815 -0.032019276 0.06299
of -0.0075046676 0.07530516 -0.033351347 0.031155227 -0.052691482 0.11667848 0.06325536 0.078
in 0.045108303 0.066770665 0.04681081 -0.018414069 0.05815711 0.0075108535 0.024987211 -0.073
had -0.013006962 -0.047319893 -0.08516388 0.11034862 -0.000121361205 -0.12037892 -0.07678285
when -0.14150621 0.13664216 0.07512865 0.12575595 -0.097530514 -0.118885495 -0.029893897 0.04
me -0.083542876 0.0014570053 -0.05385383 -0.026671011 0.023588976 0.09522098 0.026278341 -0.0
that -0.09758009 0.052675396 0.050043285 0.14439772 0.15606605 0.07742057 -0.03150305 -0.0008
for 0.12612256 0.11580164 0.04323103 0.12918225 -0.13711552 0.00959535 -0.049704924 -0.042340
with -0.1540265 -0.020449817 -0.07395158 -0.14824928 0.106019795 -0.15050454 0.032807995 0.05
not 0.102592304 0.1093094 0.09075969 -0.006318691 -0.082911186 0.07870124 -0.013783346 0.0607
at -0.120102875 -0.018787352 -0.039176818 -0.057485286 -0.092016414 0.05479648 -0.08500498 -0
it 0.10189846 -0.1266037 -0.10555873 -0.11436862 0.0024962476 0.04095333 -0.04126273 0.090715
```

1.3 神经网络模型

1.3.1 CNN (卷积神经网络)



CNN 示意图

上图展示了 CNN 的大致结构。第一层为输入层；第二层为卷积层，CNN 中定义了多个一维卷积核，用这些卷积核对输入进行卷积计算得到输出，再让输出的所有通道经过一个时序最大池化层（第三层），这些池化输出连接成向量，最后经过全连接层（可含有 dropout 丢弃层以及 softmax 层）输出最后的结果。

设 embedding 的维数为 k，句字长度为 n，则输入层即为 $n \times k$ 的矩阵。

对矩阵 X，定义一个窗口为

$$X_{i:i+h-1} \in \mathbb{R}^{h \times k}$$

表示从第 i 个词开始，连续 h 个词组成的部分。

使用一个卷积核 (kernel)

$$\mathrm{w} \in \mathbb{R}^{hk}$$

卷积核的宽度是与词向量的维数一致的。

对这个窗口 $X_{i:i+h-1}$ 进行点乘计算，可以得到一个特征

$$c_i = f(\mathrm{w} \cdot X_{i:i+h-1} + b)$$

其中 b 为偏执参数，f 为非线性函数如 `ReLU` 函数等。

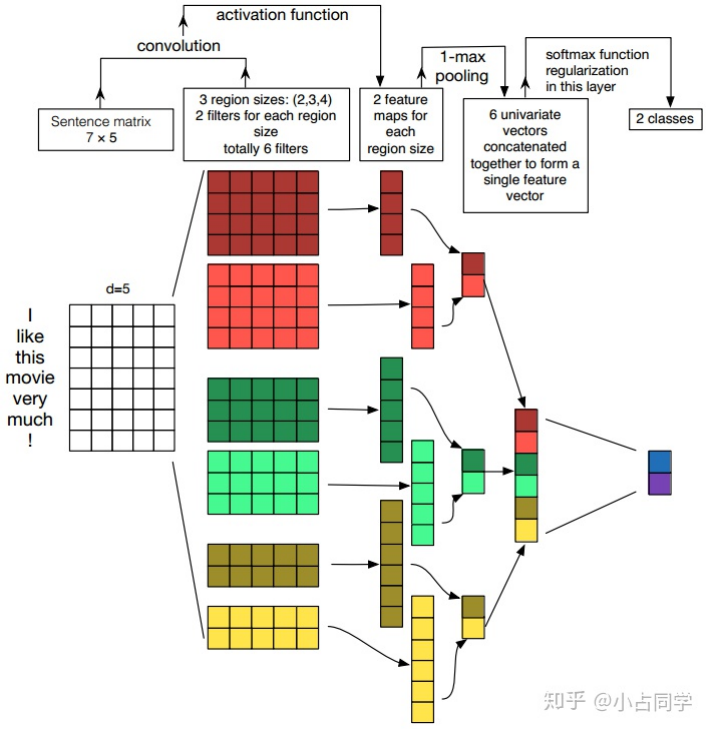
这个窗口把整个 X 从上到下遍历一遍（求卷积），把得到的 c_i 拼接到一起就可以得到一个特征图：

$$c = (c_1, c_2, \ldots, c_{n-h+1})^t \in \mathbb{R}^{n-h+1}$$

然后队齐进行 1-max pooling 池化，即从每个窗口的特征向量 c_i 中选出值最大的特征，再将它们重新拼接起来。

最后再经过带有 dropout 的全连接层，得到结果。

下图是上述过程一个大致的示例：

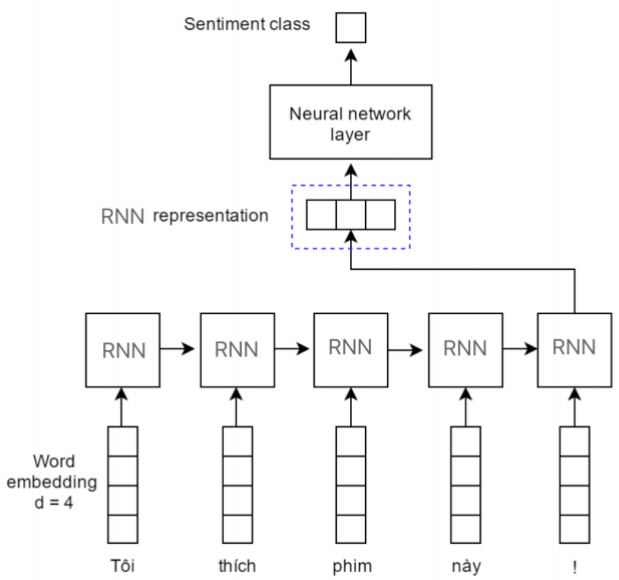


在本次实验实现的 CNN 中，采用了如下的结构：

```
TextCNN(
  (word_embeddings): Embedding(8114, 128)
  (dropout): Dropout(p=0.5, inplace=False)
  (decoder): Linear(in_features=384, out_features=7, bias=True)
  (pool): GlobalMaxPool1d()
  (convs): ModuleList(
    (0): Conv1d(128, 128, kernel_size=(3,), stride=(1,))
    (1): Conv1d(128, 128, kernel_size=(4,), stride=(1,))
    (2): Conv1d(128, 128, kernel_size=(5,), stride=(1,))
  )
)
```

从中可以看到：输入为 128 维词向量，卷积核的大小分别为 [3, 4, 5] 个词数，卷积核的数量均为 128 个。

1.3.2 RNN (循环神经网络)

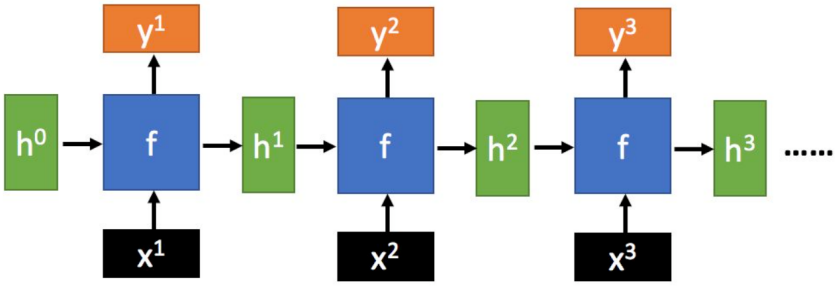


RNN 示意图

1.3.1.1 Naïve RNN

朴素的 RNN 将词向量按顺序循环输入，而后通过最后的输出（out）以及隐藏层（h）来得到最终的特征提取向量。

以下说明普通的 RNN 。考虑 RNN 中的某一状态。设当前状态输入数据为 x^i ， h^{i-1} 表示接收到的上一个节点的隐藏输出； y^i 表示当前节点的结果输出（out），h^i 为传递到下一节点的隐藏输出。即如下图：



这里的 f 为转移函数：

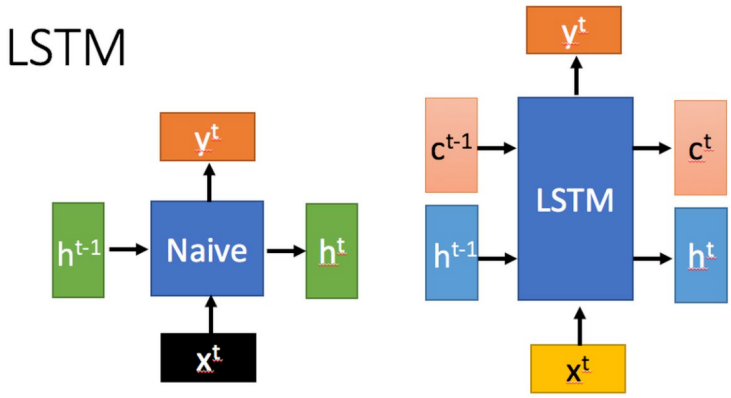
$$f : h^i, y^i = f(h^{i-1}, x^i)$$

对于最终的结果输出，可以将最后的隐藏层通过线形层（Linear）进行维度映射，再经 `Softmax` 函数后进行输出。

本次实验并没有构建 Naïve RNN 作为 baseline，而是直接采用了下文提到的几种网络。

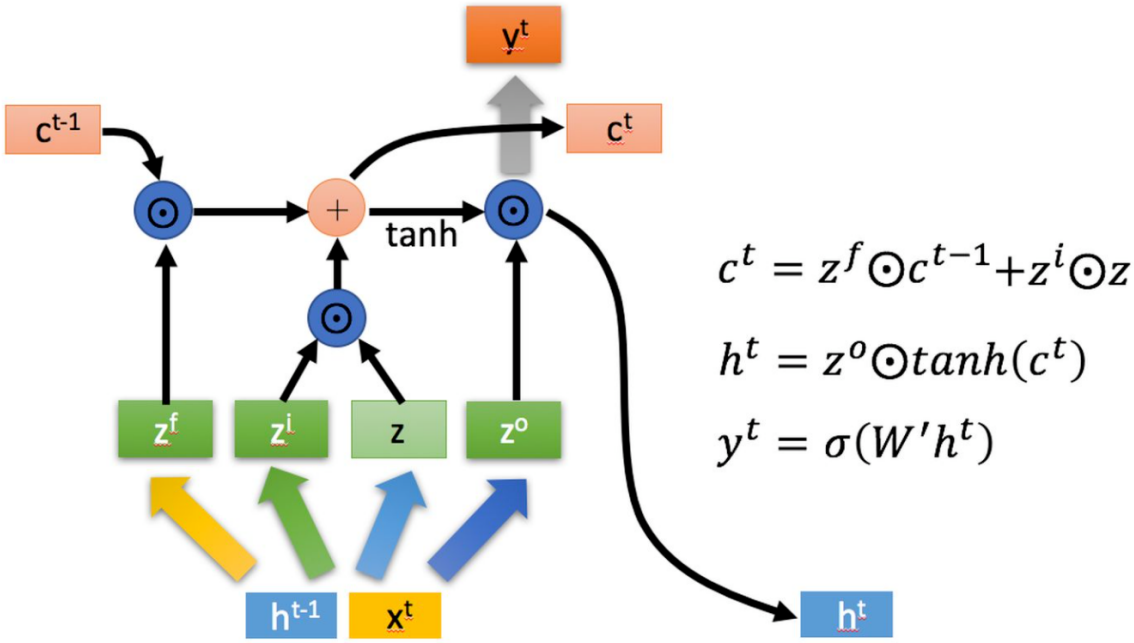
1.3.1.2 LSTM (长短期记忆)

LSTM 是一种特殊的 RNN，可以在更长的序列上有更好表现。普通 RNN 只有一个传递的状态 h^t，而 LSTM 有两个传递状态 h^t (hidden state) 和 c^t (cell state) 。



而为了实现长短期的记忆，需要有“选择性忘记”阶段，以及“选择性记忆”阶段，并且还要决定哪些作为输出。于是需要引入门控信号 z^f (遗忘门)，z^i (输入门) 以及 z^o (输出门) 。还有一个输入数据 z 。这些门由各自的权重矩阵 m 点乘上 x^t 和 h^{t-1} 的拼接，再经过 σ 函数（Sigmoid）得到的。

遗忘门 z^f 控制 c^{t-1} 哪些需要忘记，输入门 z^i 对输入 x^t 进行选择性记忆。以上两步结果相加，即可得到传给下一状态的 c^t。输出门 z^o 则控制了当前状态的输出，点乘上 c^t 的反正切函数 (tanh) 后得到 h^t ，y^t 也通过 h^t 得到。



而LSTM还可以采用正反向双向，这样可以更好地处理上下文信息。

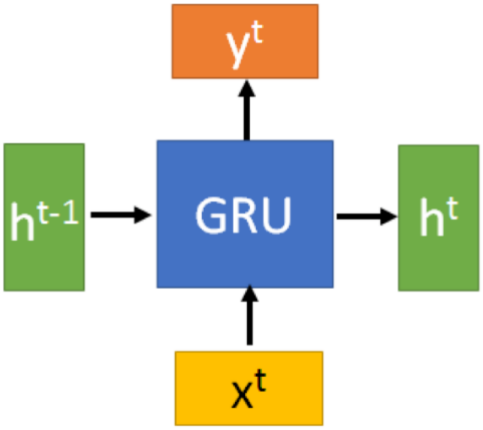
在本次实验中实现的 LSTM，采用了如下的结构：

```
LSTM(
  (word_embeddings): Embedding(8114, 128)
  (lstm): LSTM(128, 128, bidirectional=True)
  (linear): Linear(in_features=256, out_features=7, bias=True)
)
```

采用了双向的结构，输入层后进入双向 LSTM 层，最后是线形层输出。

1.3.1.4 GRU (门控循环单元)

GRU 的内部结构是与 LSTM 相似的，只不过少了隐藏层 c。事实上，GRU 就类似于参数简化版的 LSTM，可以更加节省成本提高效率。



GRU 一个状态的示意图

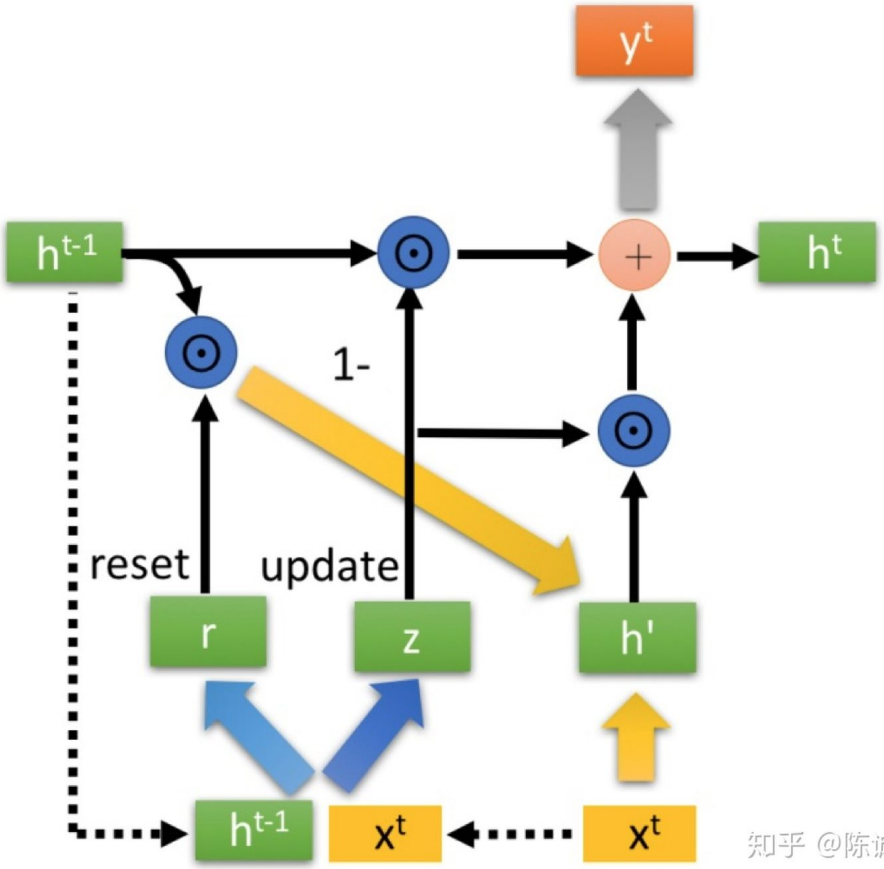
GRU 的门控为重置门 r 以及更新门 z。其计算方法与 LSTM 中的门控类似，都是各自的权重矩阵 m 点乘上 x^t 和 h^{t-1} 的拼接，再经过 σ 函数。

重置门 r 点乘（矩阵对应元素相乘）上 h^{t-1}，得到充值后的输入信号，再与 x^t 拼接、和权重矩阵 m 点乘、取 tanh 值后得到 h' 这一类似于"选择记忆"状态的值。

而更新门则通过类似加权的方式，把 h^{t-1} 和 h^t 加起来，得到了隐藏层输出 h^t：

$$h^t = (1-z) \odot h^{t-1} + z \odot h'$$

这样就使用同一个门 z，实现了"选择性遗忘"和"选择性记忆"两个功能，减少了参数和训练成本，提高效率。



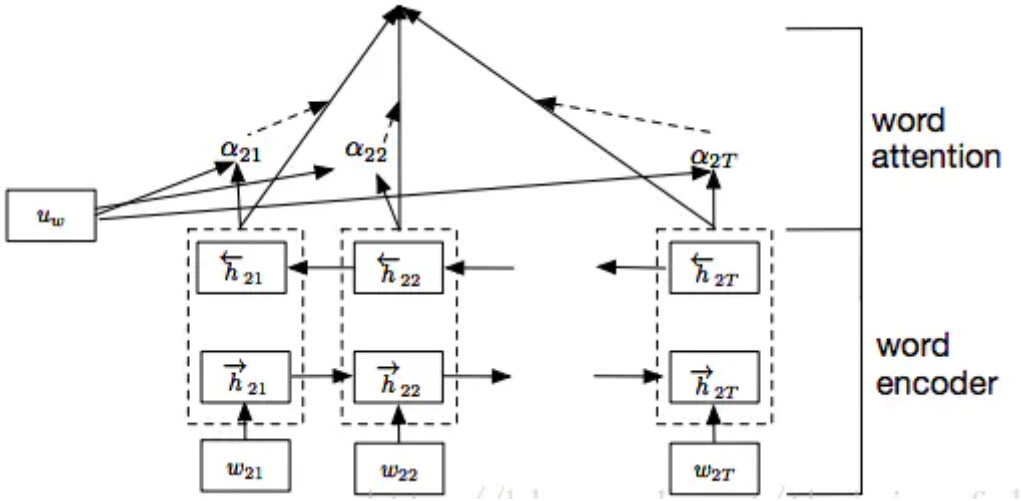
本次实验中使用的 GRU，采用了如下的结构：

```
GRU(
  (word_embeddings): Embedding(8114, 128)
  (gru): GRU(128, 128, bidirectional=True)
  (linear): Linear(in_features=256, out_features=7, bias=True)
)
```

同样是双向的。

1.3.1.4 RNN+Attention

RNN 还可以结合 Attention 方法进行。本次实验采用的 Attention 方法仅对 RNN 的 output 作了权重处理。



本次实验实现了 LSTM_ATT 以及 GRU_ATT 两个带有 Attention 机制的 RNN 模型。

1.3.3 MLP (多层感知机，baseline)

每个词向量都经过线性层以及激活函数（本应使用 `ReLU`，后改用 `Softmax` 发现效果更好）。

本次实验实现的 MLP 模型中，有三个线性层，隐藏层大小取了 128 。结构如下：

```
MLP(
  (word_embeddings): Embedding(8114, 128)
  (linear1): Linear(in_features=128, out_features=128, bias=True)
  (linear2): Linear(in_features=128, out_features=128, bias=True)
  (linear3): Linear(in_features=128, out_features=7, bias=True)
)
```

2. 实验结果

实验采用 PyTorch 框架，数据集使用的是 v2 。

词向量均为自己通过 train 与 valid 两个数据集构造的 word2vec ，维度 128。

CNN 采用大小 [3, 4, 5] 的卷积核，每种 128 个，含 0.5 的 dropout 。RNN 的隐藏层大小均为 128 。

优化器选择的是 Adam ，损失函数使用交叉熵损失函数（`CrossEntropyLoss`）。

CNN 学习率 0.001；RNN 和学习率均为 0.01 。训练至多 16 轮（MLP 为上限 60 轮），并采用 early-stopping 机制，验证集 (valid) 上的准确率连续 5 次上升后将提前终止训练。训练时将数据分 batch 输入，batch_size 统一为 32 。

具体配置在同目录下的 `config.ini` 中。

2.1 结果与比较

Model	Accuracy (%)	Macro F1 (%)	Micro F1 (%)	Weighted F1 (%)
MLP	49.18	47.73	49.18	48.01
CNN	**58.71**	**58.57**	**58.71**	**58.76**
LSTM	46.90	47.14	46.90	47.32
GRU	47.49	46.84	47.49	46.98
LSTM + Attention	54.99	54.50	54.99	54.68
GRU + Attention	50.46	49.70	50.36	49.91

在这其中，CNN 模型的整体效果都是最好的，最差的是 GRU 。作为 Baseline 的 MLP 模型处于居中水平，优于 RNN (LSTM，GRU)，次于加上 Attention 机制的两种 RNN 。

可以看到，本应擅长处理文字长序列 LSTM 和 GRU 表现并没有那么理想。可能的原因是数据集中的文本不是那么长，同时数据质量也不高，没办法很好地体现出其优势。

还可以发现，微平均值与准确率几乎一致。但根据 Micro f1 的计算公式，其值并不恒等于准确率，故这只是这两个值十分相近。

此外，这些模型也基本统一了参数（如隐藏层等）。虽然经过调参还可以有略好的结果，但作为用于比较的结果，这样的控制变量也更具说服力，并且实验结果也基本反映出了几个模型基本的特点。

2.2 参数分析

实验中采用了一些超参数，这里着重对 lr (学习率) 以及 hidden_size (隐藏层大小，对于 CNN 为通道数) 进行分析。

2.2.1 CNN 网络:

Model	num_channel	lr	Accuracy (%)	Macro F1 (%)	Micro F1 (%)	Weighted F1 (%)
CNN (**default**)	128	0.001	58.71	58.57	58.71	58.76
CNN	128	0.1	15.00	3.88	15.00	4.09
CNN	128	0.01	46.58	45.63	46.58	45.77
CNN	32	0.001	56.69	56.61	56.69	56.78
CNN	64	0.001	57.34	57.07	57.34	57.27
CNN	256	0.001	58.25	57.98	58.25	58.17

从学习率 (lr) 的角度，可以看到，学习率太大时网络无法收敛，造成欠拟合。学习率小的话训练效率过低。

从卷积核数量 (num_channel) 的角度，可以看到，卷积核数量少时，提取的特征稍少，拟合能力稍差，准确率有所降低。卷积核过多时效果无法显著提升，反而会拉低训练效率，没有必要。

以上结果同理论结果相符合。

2.2.2 LSTM_ATT 网络

Model	hidden_size	Accuracy (%)	Macro F1 (%)	Micro F1 (%)	Weighted F1 (%)
LSTM_ATT (**default**)	128	54.99	54.50	54.99	54.68
LSTM_ATT	32	38.03	37.58	38.03	37.88
LSTM_ATT	64	53.62	53.57	53.62	53.73
LSTM_ATT	256	43.51	43.82	43.51	43.91

从隐藏层大小 (hidden_size) 的角度，可以看到，隐藏层较小时，模型的拟合能力差，准确率低；而当隐藏层过大时，也出现了准确率下降的问题，分析认为可能是过深的 RNN 带来的梯度传递错误问题。

以上结果同理论结果相符合。

3. 问题思考

3.1 训练停止时间

起初，采用了固定轮数 (epoch) 的方式进行训练，这样的后果是需要通过观察 loss 以及 accuracy 来判断拟合程度，工作量较大，有时候即使已经到达最好的一轮时，也无法终止，只好任其过拟合。

因此改进后，对训练轮数 (epoch) 设置了一个不大的上限，同时采用 early-stopping 准则，对 valid_loss 进行计算分析，连续多次上升后提前终止训练。

其中，early-stopping 对限制过拟合有帮助。

3.2 过拟合与欠拟合

为了防止过拟合，使用 3.1 提到的训练轮数上限、early-stopping 准则，在训练轮数上进行预防；此外，还可以使用 torch 自带的 Adam 优化器来控制模型的规模。

为了防止欠拟合，可以在调参时将模型的参数设大，如 embedding 的维数、隐藏层的大小，增大模型的规模，增强拟合能力；也可以如本次实验中实现的一样，增加 Attention 方法。

3.3 梯度消失与梯度爆炸

梯度消失与梯度爆炸是指，在根据损失函数计算的误差通过梯度反向传播的方式对深度网络权值进行更新时，得到的梯度值接近 0 或特别大。

梯度消失和梯度爆炸一般来说有两种原因，一是深层网络结构，二是激活函数不合适。网络层数过多时，如果在一部分的梯度大于 1，那么由于链式求导，求出的梯度将指数增加，发生梯度爆炸；小于 1，则指数衰减，发生梯度消失。而激活函数不合适时，比如在某些情况下的 Sigmoid 函数，则会造成梯度消失的情况。

为了解决梯度消失、梯度爆炸的情况，可以采取以下方案：

1. 不用 Sigmoid 激活函数，改用 `ReLU` 等函数。
2. 进行预训练，如采取预训练的词向量，避免参数的初始化不合理。
3. 进行 Batch Normalization (批规范化)，对每一层的输出规范为均值和方差一致。
4. 使用残差网络。
5. 使用梯度剪切、正则化。

3.4 模型优缺点

3.4.1 CNN

优点

1. 参数共享，可以减少模型参数，一定程度上降低计算量，并增加模型泛化能力。
2. 等变表示，即 CNN 神经网络的输出对于平移变换来说是等变的。
3. 稀疏交互，在卷积神经网络中，卷积核尺度远小于输入的维度，这样每个输出神经元仅与前一层特定局部区域内的神经元存在连接权重（即产生交互），这使得 CNN 可以学习局部的特征。

缺点

1. 集中于局部特征，而忽视了与整体的关联。

适用场景

适用于需要提取特征的任务，相比文本任务，更适合于做图像分类、识别的任务。

3.4.2 RNN

优点

1. 能够处理序列变化的数据，兼顾上下文。
2. 在长序列输入的任务上有更好的表现。

缺点

1. 计算量大，模型训练效率较低。
2. 网络容易过深，导致出现梯度消失、梯度爆炸现象。

适用场景

适用于序列检测任务，如各种语义判断的 NLP 问题。

3.4.3 MLP

优点

1. 拟合能力强。
2. 收敛较快，可以快速解决复杂问题。

缺点

1. 容易过拟合。
2. 难以从输入的特点中利用有效的信息。

适用场景

适用于各种模型的连接、输出部分，可以做一个整合。

4. 心得体会

本次实验不是我第一次搭建神经网络，因此整个流程沿用了我自己先前做类似任务的框架（数据清洗 -> 与训练词向量 -> 神经网络搭建），并且 CNN 与 LSTM 也是复刻了我自己先前写过的模型。先前的一些经验让我完成本次任务时更有条理，不会像做四子棋大作业那样毫无头绪。

不过，这次的神经网络搭建让我比先前有了更大的进步。原先对网络的结构理解还很浅，对于参数的作用也没有足够理解，可以说是一知半解；但本次实验过后，对网络的内部实现、参数的效果有了第一手的体会，因此可以说是受益良多。

本次实验的数据集其实还不够理想，一方面是句字长度太短了，没办法很好地体现一些网络的优势；另一方面，里面确实含有一些奇怪的字符，给训练带来了些不良影响。此外，比起英文的语料库，我觉得如果有中文语料库会更加有趣、更有挑战性，对我们来说可能也更加有直观感受。由此来看，数据是十分重要的，对于机器学习来说，不光是模型结构、参数这些，更要有好的数据集，才可以得到好的模型。

最后，再次感谢马老师与助教的悉心指导，感谢！

5. 参考资料

torchtext :

https://pytorch.org/text/stable/index.html

CNN :

https://blog.csdn.net/qsmx666/article/details/105302858?ops_request_misc=&request_id=&biz_id=102&utm_term=Pytorch%20TextCNN&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-105302858

https://zhuanlan.zhihu.com/p/77634533

LSTM :

https://blog.csdn.net/huangqihao723/article/details/105246113?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-4&spm=1001.2101.3001.4242

https://blog.csdn.net/qq_36426650/article/details/105172198

https://zhuanlan.zhihu.com/p/32085405

GRU :

https://zhuanlan.zhihu.com/p/32481747

Early-Stopping :

https://github.com/Bjarten/early-stopping-pytorch

F1-Score :

https://www.jianshu.com/p/9e0caf109e88

梯度消失与梯度爆炸 :

https://zhuanlan.zhihu.com/p/68579467