情感分类任务实验报告

石大川 <sdc17@mails.tsinghua.edu.cn>

2020 年 5 月 28 日

目录

1 介绍

在本次实验中我实现了 MLP，CNN 和 RNN 三类网络模型用于情感分类任务，其中 MLP 是一个三层感知机，CNN 是在 TextCNN 模型上进行修改得到的，RNN 包括带注意力的双向 LSTM 和 GRU 两个模型。在从训练集中随出划出 6% 构成的验证集上，准确率最高的是 CNN 模型，为 0.654514，在正式的测试集上，准确率最高的也是 CNN 模型，为 0.572527。

2 数据预处理

Word2Vec 模型我用的是实验说明文档给的链接中的 300d 模型 sgns.sogounews.bigram-char。该模型的训练语料是搜狗新闻，与本次实验数据语料的特征分布是较为接近的。

在 embedding 之前，对于不同样本中长度不一的句子，我统一将它们的长度固定为 768 词，超过这个长度的词会被截断，短于这个长度的词会被 padding 到这个长度。

对于训练集 sinanews.train，我随机划分出了 6% 的样本来构成验证集。

3 网络模型

3.1 MLP

Linear->BatchNorm1d->ReLU->Linear->BatchNorm1d->ReLU->Linear，模型图见MLP

3.2 CNN

基本框架使用 TextCNN，相同窗口的卷积核数目设为了 128，保持窗口大小为 2/3/4。

(Conv1d->BatchNorm1d->ReLU->MaxPool1d)*3->Linear->BatchNorm1d->ReLU->Linear，模型图见CNN

3.3 RNN

3.3.1 LSTM

双向 LSTM->Attention->Linear->BatchNorm1d->ReLU->Linear，模型图见LSTM

3.3.2 GRU

双向 GRU->Attention->Linear->BatchNorm1d->ReLU->Linear，模型图见GRU

4 实验结果

图例见Legend

训练集上的 Mean Acc 见Train Acc

验证集上的 Mean Acc 见Val Acc

验证集上的 F-score 见Val F-score，使用的是宏平均

验证集上的 Correlation Coefficient 见Val Correlation Coefficient

测试集上的 Metrics 见表 1：

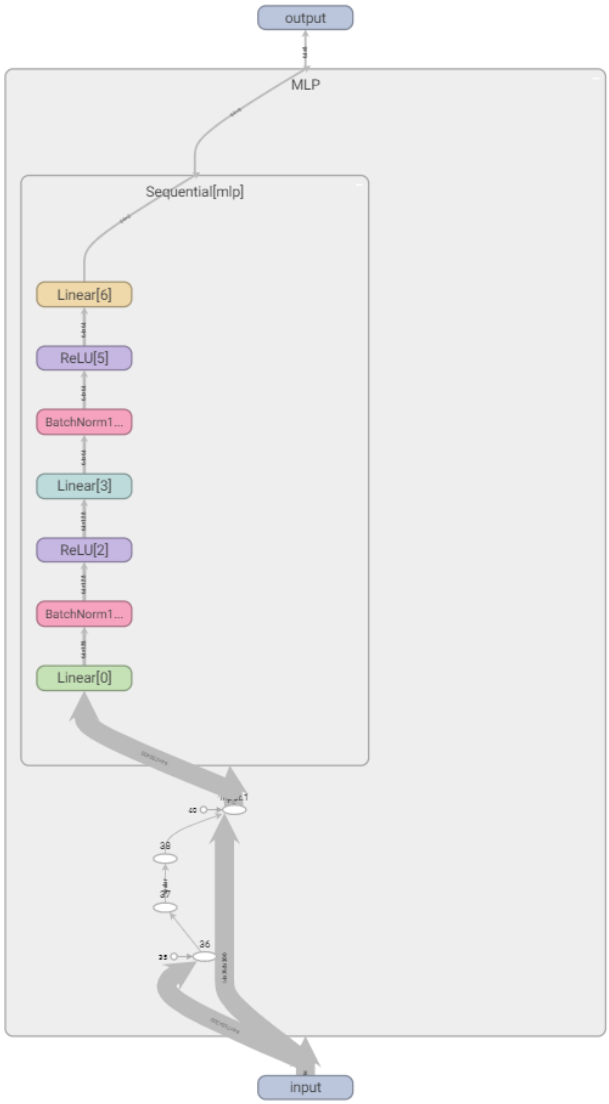


Figure 1: MLP 结构

Model	Acc	F-Score(Macro)	Corr
MLP	0.523695	0.262582	0.516952
TextCNN	0.572527	0.323348	0.571255
BiLSTM+Attn	0.558139	0.252751	0.527346
BiGRU+Attn	0.513427	0.245764	0.496758

表 1: Metrics on Test Dataset

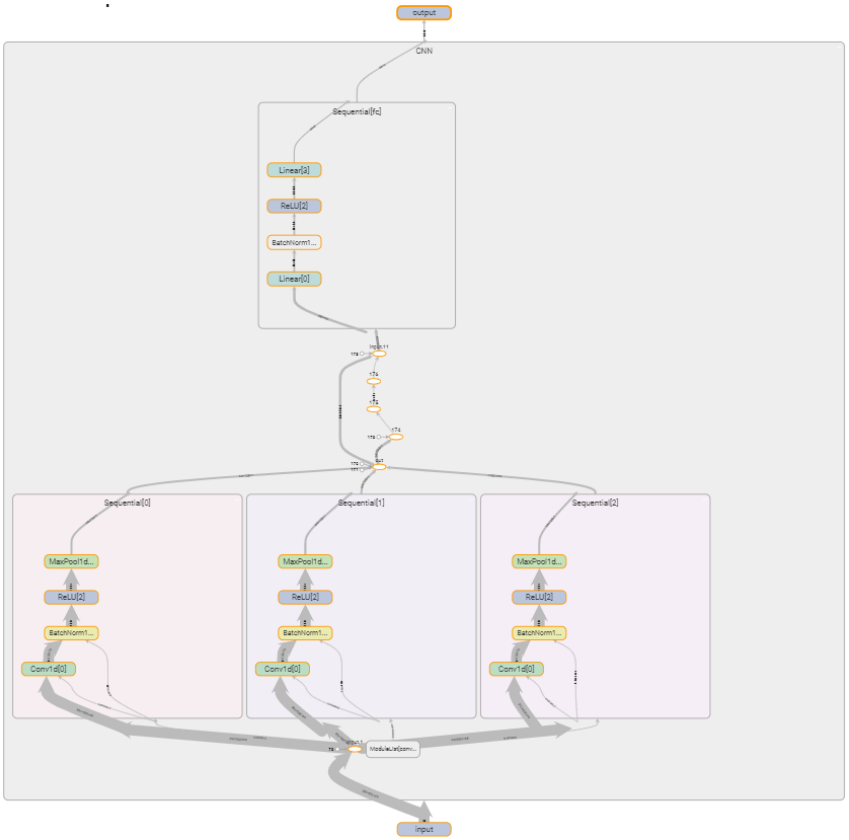


Figure 2: TextCNN 结构

5 模型效果比较

5.1 Metrics

从 Acc, F-Score 以及 Corr 的评价指标来看，在验证集和测试集上综合效果最好的都是 TextCNN，效果最差的是 BiGRU+Attn。

5.2 收敛速度

从收敛速度来看 MLP 和 TextCNN 很接近，且相对较快，而 BiLSTM+Attn 与 BiGRU+Attn 都相对较慢。

5.3 收敛稳定性

从收敛稳定性来看，MLP 在训练过程中 Loss 和 Metrics 曲线的抖动幅度最小，而 TextCNN, BiLSTM+Attn 与 BiGRU+Attn 均有较大程度的抖动。

6 模型参数调整

以 TextCNN 为例进行对比，以下所有图表中，深红色的曲线均代表原始的 TextCNN 设置得到的结果，其他颜色的曲线则代表分别在原始 TextCNN 的基础上只进行下列参数修改得到的结果。

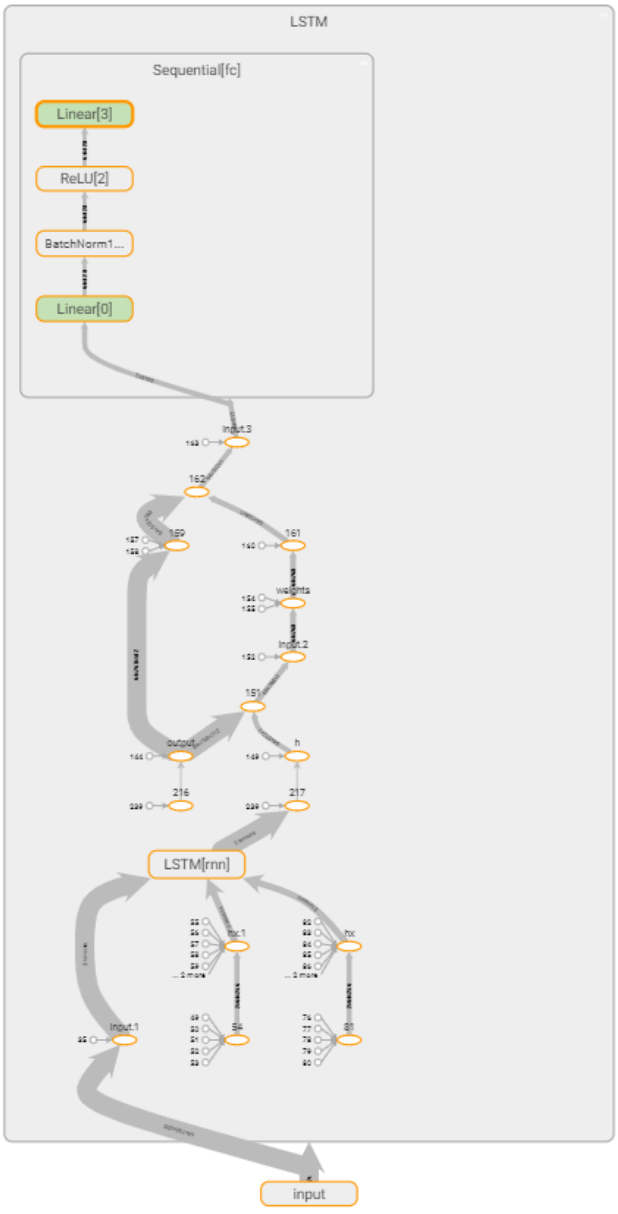


Figure 3: BiLSTM with Attention 结构

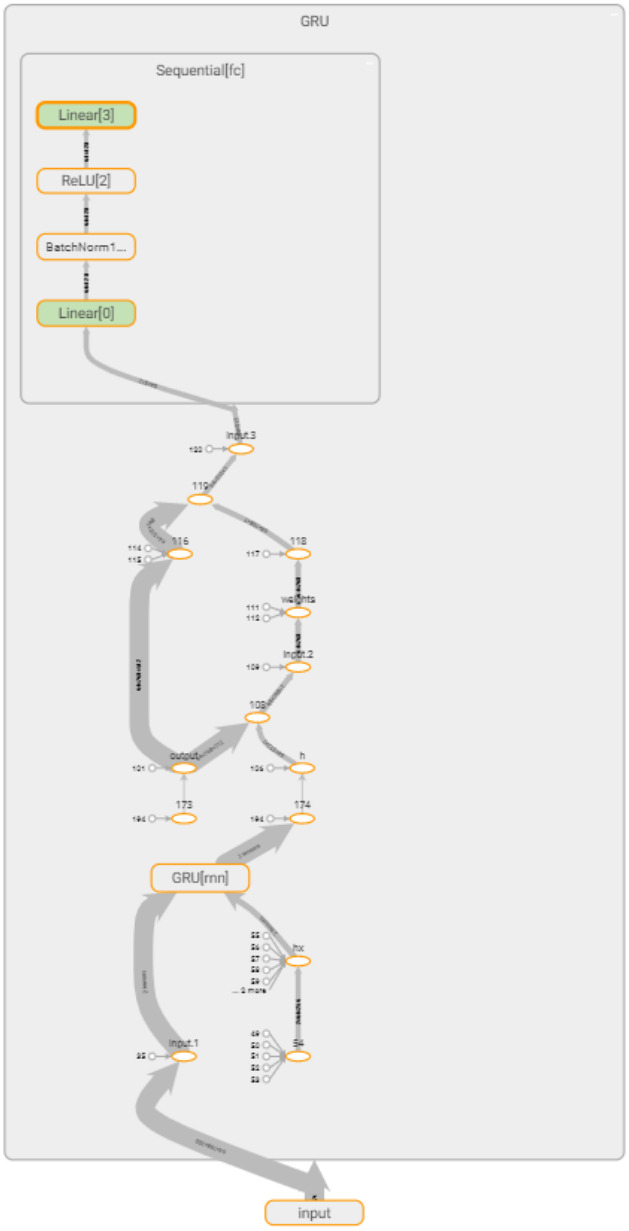


Figure 4: BiGRU with Attention 结构

Figure 5: Lengend

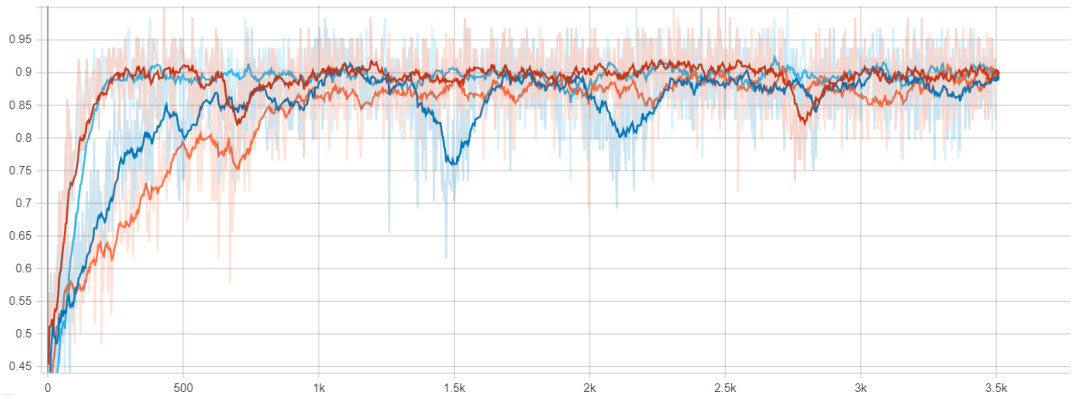


Figure 6: Train Acc

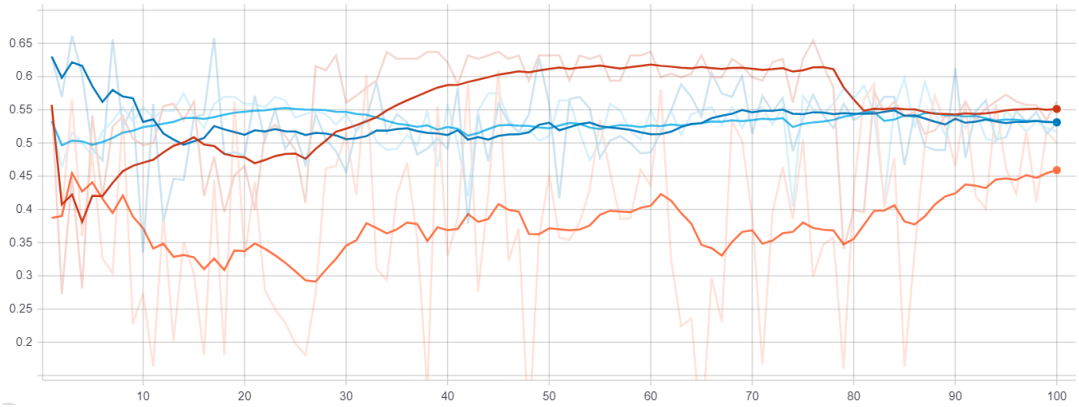


Figure 7: Val Acc

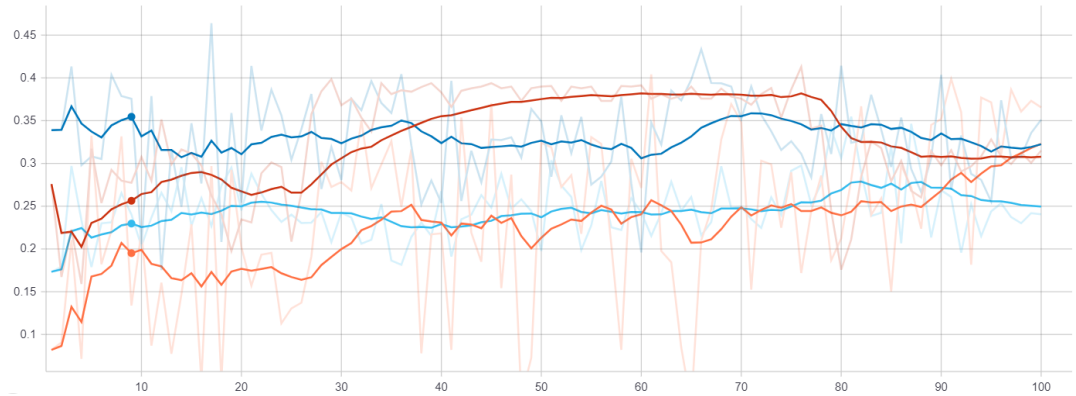


Figure 8: Val F-score

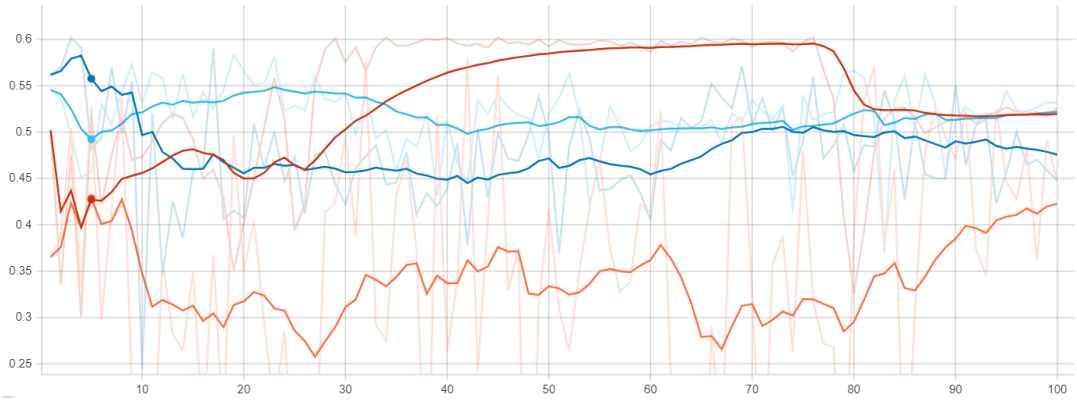


Figure 9: Val Correlation Coefficient

6.1 优化器

分别用 Adam 和带 0.9 动量的 SGD 进行训练，在训练集和验证集上的 Acc 表现见Figure 10

可见 100 个 epochs 中两种优化器能达到的最好结果是很相近的，但是在稳定性方面 SGD 是要优于 Adam 的。

6.2 Batch Size

分别用 64 的 batch size 和 16 的 batch size 进行训练，在训练集和验证集上的 Acc 表现见Figure 11

其中因为 batch size 从 64 改成了 16，所以训练时统计的迭代次数变成了之前的 4 倍。从下面验证集上的表现可见 16 的 batch size 过小了，网络收敛速度要明显慢于 batch size 为 64 的情况。

6.3 Learning Rate

分别用 0.01 和 0.001 的 Learning Rate 进行训练，在训练集和验证集上的 Acc 表现见Figure 12

可见当学习率设为较小的 0.001 时模型的 Acc 曲线更加平滑，同时由于较小的学习率使得模型的梯度更新更慢，在相同的 100epochs 下能达到的最好表现也相对更差。

6.4 正则化

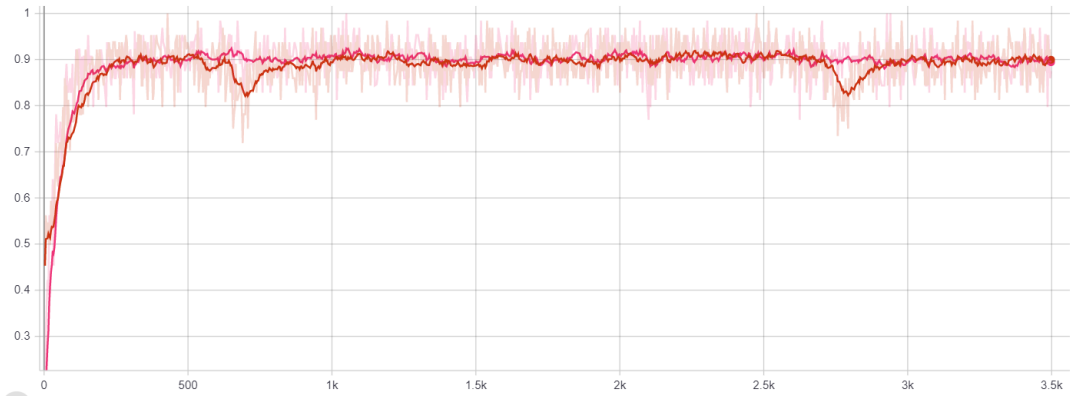
分别用 BatchNorm1d 和概率为 0.5 的 Dropout 对 TextCNN 进行正则化处理，在训练集和验证集上的 Acc 表现见Figure 13

可见使用 Dropout 时，模型收敛速度明显变慢，且在测试集上的表现也远低于 BatchNorm1d。

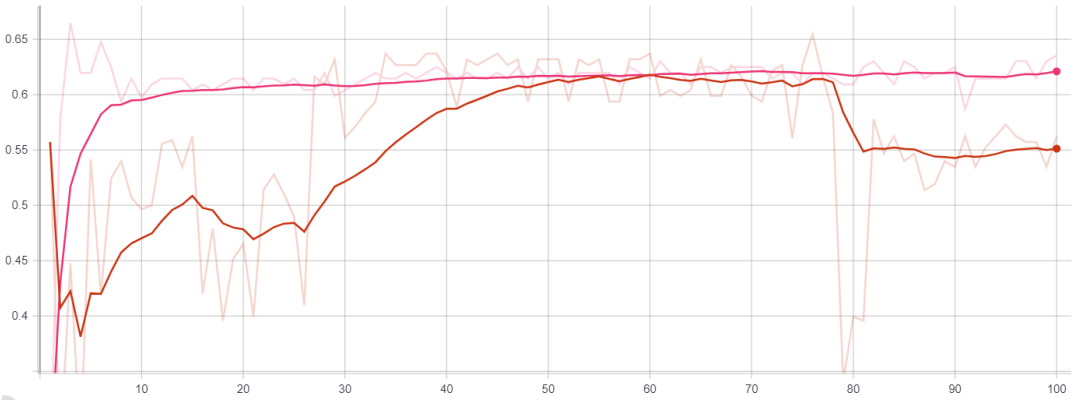
6.5 窗口大小/数量

分别使用 [2, 3, 4] 和 [3, 4, 5, 6] 的窗口大小和数量设置，在训练集和验证集上的 Acc 表现见Figure 14

可见当窗口变得更大，且数量变得更多时，模型的训练稳定性有所下降，但是在测试集上最好的表现是相近的。其实在第一次拼音输入法的大作业中就了解到过中文句子内

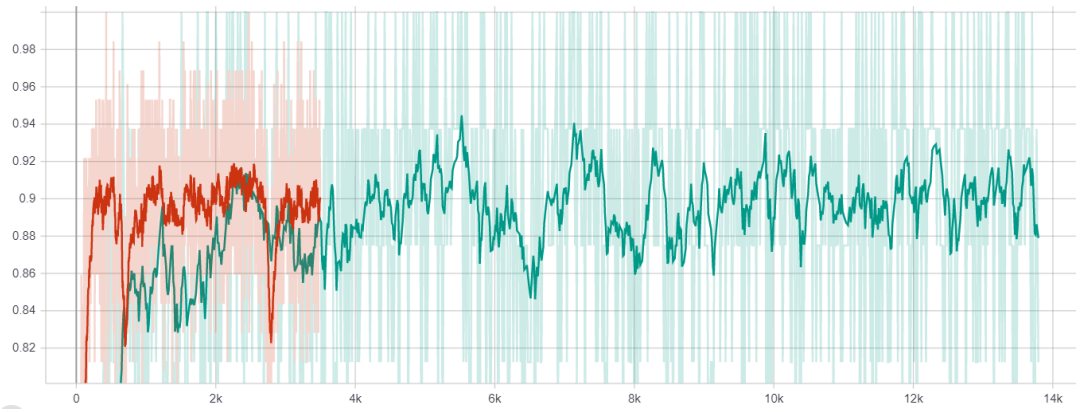


(a) Train Acc

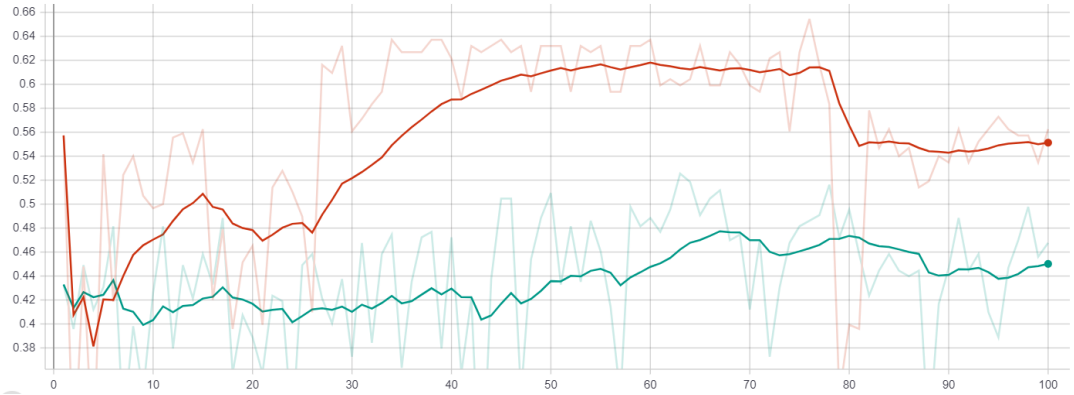


(b) Val Acc

Figure 10: Adam(Red) vs SGD(Pink)

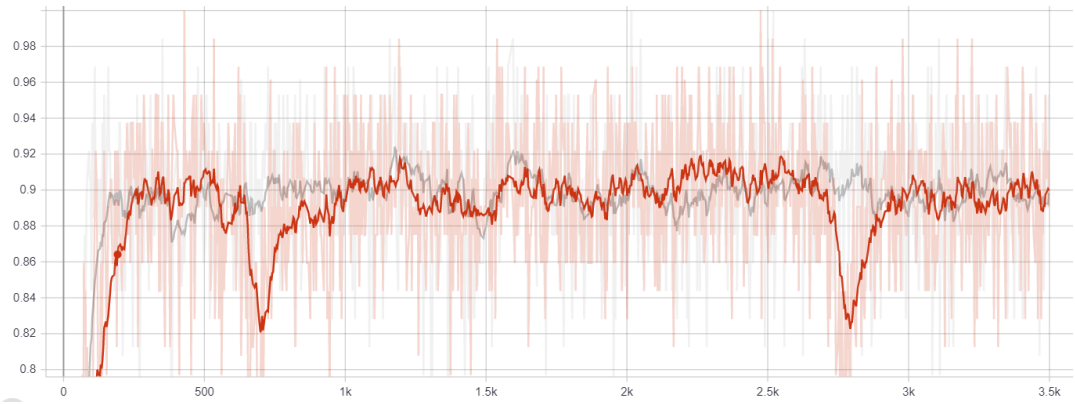


(a) Train Acc

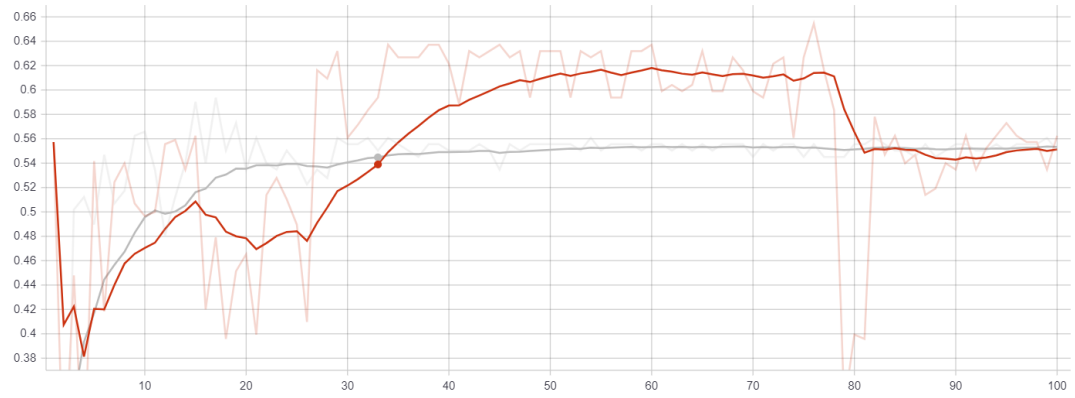


(b) Val Acc

Figure 11: Batch Size 64(Red) vs 16(Green)



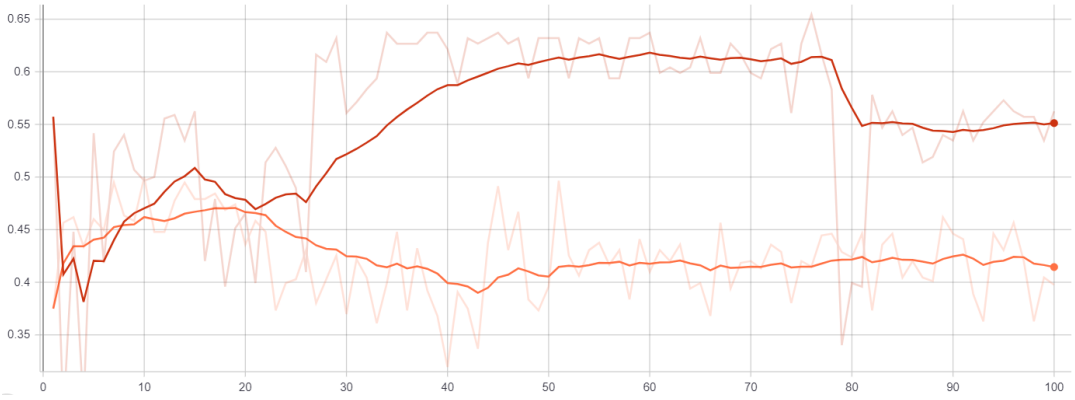
(a) Train Acc



(b) Val Acc

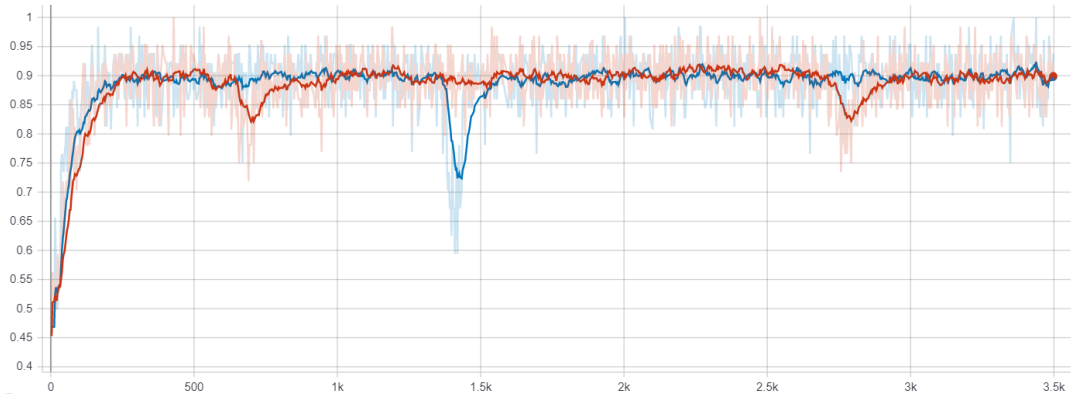
Figure 12: Learning Rate 0.01(Red) vs 0.001(Gray)

(a) Train Acc

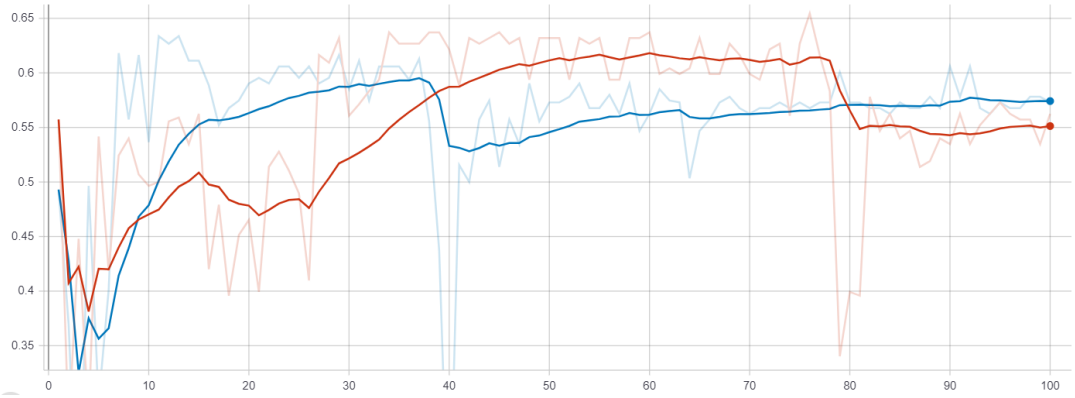


(b) Val Acc

Figure 13: BatchNorm1d(Red) vs Dropout(Orange)



(a) Train Acc



(b) Val Acc

Figure 14: Windows: [2, 3, 4](Red) vs [3, 4, 5, 6](Blue)

词的关系很少会超过四元，因此添加四以上的窗口大小几乎不会带来模型效果的提升，与此同时去掉了大小为 2 的窗口又会使得模型少学习了很多相对更常见的二元词关系，使得模型的性能波动较大。

7 问题思考

① 实验训练什么时候停止是最合适的？简要陈述你的实现方式，并试分析固定迭代次数与通过验证集调整等方法的优缺点。

最合适的停止时机是模型的 Loss/Metrics 在验证集上不再降低了，此时如果继续训练，得到的模型会有较高的过拟合风险。我的实现方式是固定最大迭代次数为 100，并在每个 epoch 检查当前模型的 Metrics 是否优于之前的模型，如果是则用保存当前模型并更新最优 Metrics，否则进入下一个 epoch。

固定迭代次数的优点在于能进行足够轮数的 epoch，更有可能得到更接近全局最优的模型。缺点在于时间成本大，不够灵活。

通过验证集调整的方法优点在于能尽快地得到一个效果较好的模型，当发现验证集上效果不再提升后即可结束训练，节省训练时间。缺点在于模型可能在训练过程中暂时陷入了局部最优，此时验证集上效果不再提升了就会导致训练过程结束，但是在更多的 epochs 之后模型可能会跳出局部最优，达到更接近全局最优的解。

② 实验参数的初始化是怎么做的？不同的方法适合哪些地方？（现有的初始化方法为零均值初始化，高斯分布初始化，正交初始化等）

初始化我使用的是 Pytorch 的默认初始化方法。具体来说查看 Pytorch 源码可知 nn.Linear，nn.Conv1d/nn.Conv2d 的初始化方式为 kaiming 初始化：

```
def reset_parameters(self):
  init.kaiming_uniform_(self.weight, a=math.sqrt(5))
  if self.bias is not None:
    fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
    bound = 1 / math.sqrt(fan_in)
    init.uniform_(self.bias, -bound, bound)
```

nn.BatchNorm1d/nn.BatchNorm2d 的初始化方法为 weight 置 1，bias 值 0：

```
def reset_running_stats(self):
  if self.track_running_stats:
    self.running_mean.zero_()
    self.running_var.fill_(1)
    self.num_batches_tracked.zero_()

def reset_parameters(self):
  self.reset_running_stats()
  if self.affine:
    init.ones_(self.weight)
    init.zeros_(self.bias)
```

常用的一些初始化方法中：

零均值初始化　使得参数值为常数 0，包括 Xavier 初始化，Kaiming 初始化等。

高斯分布初始化　使得参数值服从 $N(0,1)$ 的正态分布，包括 Xavier 正态和 kaiming 正态初始化等。

正交初始化　使得 tensor 之间是正交的。主要用于解决梯度消失和梯度爆炸的问题，常用于 RNN 网络的参数初始化

Xavier 均匀分布/正态分布　Xavier 均匀分布初始化方法服从均匀分布 $U(-a,a)$，$a = gain \times sqrt(\frac{6}{fan_{in}+fan_{out}})$，其中 gain 参数依据激活函数的类型来设定；Xavier 正态分布则在初始化方法中服从正态分布 $N(0,std)$，$std = gain \times sqrt(\frac{2}{fan_{in}+fan_{out}})$。Xavier 初始化常配合 tanh 激活函数使用。

Kaiming 均匀分布/正态分布　Kaiming 均匀分布初始化方法服从均匀分布 $U(-bound, bound)$，$bound = sqrt(\frac{6}{(1+a^2)\times fan_{in}})$，其中 a 为激活函数的负半轴的斜率，对于 ReLU 来说即为 0，对于 LeakyReLU 等一些 ReLU 变体来说非 0；Kaiming 正态分布则在初始化方法中服从正态分布 $N(0,std)$，$std = sqrt(\frac{2}{(1+a^2)\times fan_{in}})$。Kaiming 初始化常配合 ReLU 激活函数来使用。

③ 过拟合是深度学习常见的问题，有什么方法可以方式训练过程陷入过拟合。

Batch Normalization　批量归一化。维持网络每一层输出的特征分布差别不过大，且加在激活函数前让能其输入更集中在梯度有效变化的区域。

Dropout　随机失活一些神经元。防止模型对于某些参数的过分依赖，从而提升泛化性能。

Architecture　合理地设计网络结构，一般来说网络层数越多，参数量越大的模型越容易过拟合。

Weight Decay　在损失函数中添加 L_2/L_1 正则化项可以有效控制模型复杂度，降低过拟合的风险。Adam 等优化器中也有自带的 weight_decay 选项。

Data Augmentation　引入更多的相关的数据或者在原始数据上做数据增强。对于 NLP 来说可以随机交换/复制/删除句子中的一些单词，对于 CV 来说可以做图片的随机裁剪，镜像，旋转，形变，resize 以及随机噪声等操作。

Network Compression/Pruning　对网络进行压缩或者剪枝，能在一定程度上降低模型复杂度，提升泛化性能。

Mutil-Task Learning　通过多任务学习引入相关任务的信息提升泛化性能。但是如果任务之间不是强关联的反而有可能损害模型表现。也可以用一些类似于 bagging 的方法组合几个网络，通过加权平均或者多数表决方法降低整个网络群过拟合的风险。

Learning Rate　取一个合适的学习率，太小的学习率容易卡到局部最优，过拟合风险更大。

Batch Size 小批量的情况下方差大，也相当于在训练过程中引入更多的噪声，能起到一定的正则化效果。不过实际训练的过程中小批量收敛慢，可以在训练的初期用大批量快速收敛，训练到后期再换成小批量 fine-tuning.

④ 试分析 CNN，RNN，全连接神经网络（MLP）三者的优缺点。

MLP

优点 模型结构简单。

缺点 参数量巨大，大型一些的 CNN 中模型参数量很大一部分就来自于全连接层，例如一个 1*512*512 的 feature map 连到包含 1024 个神经元的全连接层上参数量就是 512*512*1024，考虑 channels 的维度参数量就更大，显存开销很大。此外 MLP 表达空间结构和时序结构的能力也较差，且要求大小固定的输入，对于变长数据的处理不够灵活。

CNN

优点 网络开销小，这得益于卷积层的参数共享。对局部信息有更好的表达能力，因为卷积的操作只涉及到卷积核大小范围内的数据，提取出的特征是更精细的局部特征。全卷积的网络，也即没有全连接层的网络如 SegNet/UNet 等可以处理大小不一的数据，虽然训练时考虑到效率会把训练数据 resize 到相同的尺寸，但是在做预测时用训练出的网络是可以处理大小不同的数据的，而这一点固定了神经元和连接数量的 MLP 是做不到的。

缺点 模型结构相对复杂。相比于 MLP 来说连接关系更少（MLP 是全连接，而 CNN 只和卷积核的感受野内的特征连接），求解的空间更小，因此层数较少时模型表达能力较差。

RNN

优点 可以处理变长数据，适合于处理时序信息。

缺点 序列太长的情况下还是不能避免梯度消失的问题。相比于其他网络更难做并行。

8 总结

总的来说这是这是一次关于人工神经网络基本结构的，比较全面的大作业，现在绝大部分更复杂，更先进的人工神经网络在结构上也是 MLP，CNN 和 RNN 三种基本模型的组合。

对我个人来说，我在课上讲到 TextCNN 之前还没有了解过 CNN 也能用在 nlp 领域，正好写报告时又看到 Facebook 把 Transformer 拿去做目标检测了还取得了不错的效果，再者我之前做过语义分割的任务时也读过一些 LSTM 结合 CNN 的文章，我越来越觉得做神经网络相关的工作，跳出 NLP/CV/RL 等人为框定的 Subfields，去别的领域找一些成功的方法结合到自己的研究方向还是大有可为的。

最后有个小的建议，因为人智导的作业一和三都是 NLP 相关的，而且三次作业里也没有涉及到 CV 的部分，所以我想或许以后能给实验三安排两个任务选其一完成：第一个就是现在的情感分类任务，第二个可以拿一些小的数据集比如 MNIST，VOC2012 等做一些图像分类或者目标检测的小任务。