

# 人工智能导论：文本情感分类

2021010761 计 13 班 程思翔

## 1 实验配置信息

### 1.1 代码环境信息

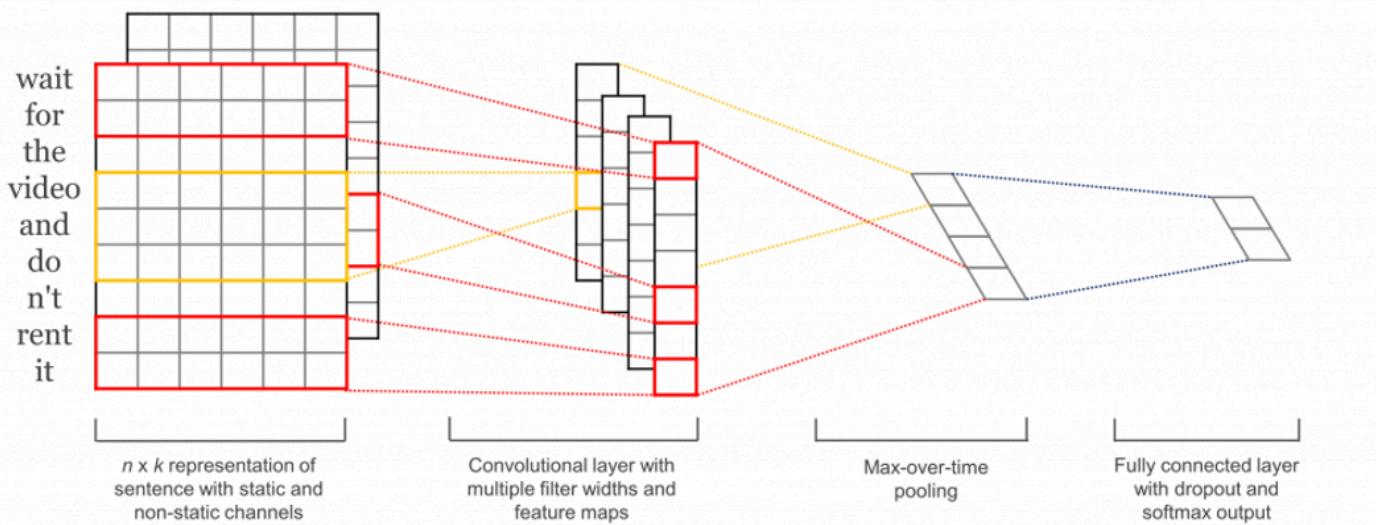
借助 `Pytorch` 框架实现，详见 `requirements.txt` 中提供的配置信息。

### 1.2 实验环境信息

- 个人笔记本电脑, 为 Apple M2 芯片;
- 系内人工智能实验平台, 提供 GPU 运算加速.

## 2 实验模型分析

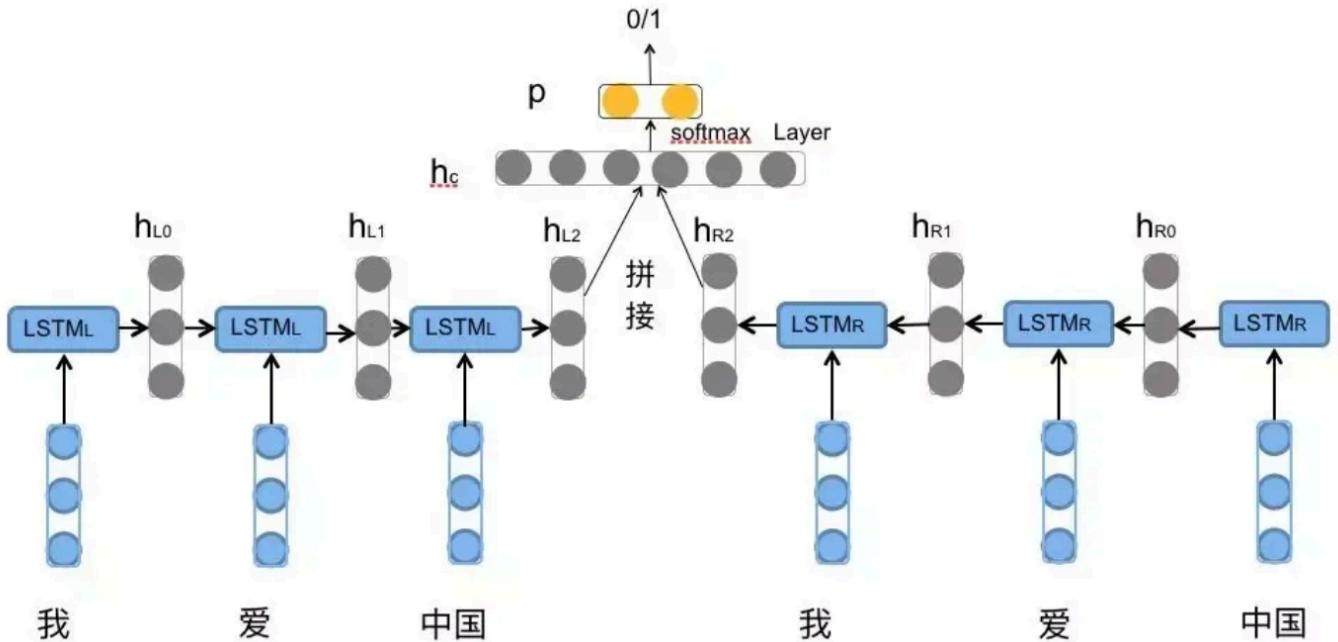
### 2.1 TextCNN



本模型实现参考了[这篇文章](#).

- 初始化:** 定义了模型所需要的相关超参数.
- 嵌入层:** 将单词的映射为定长的向量, 并将 `wiki_word2vec_50.bin` 作为预处理的嵌入矩阵输入.
- 一维卷积神经网络层:** 将嵌入层数据视为多通道一维张量, 用不同宽度的卷积核进行卷积操作, 得到多通道的一维输出特征.
- 池化层:** 对每个卷积核的输出进行池化操作, 使用 `dropout` 层来减少过拟合.
- 全连接层:** 将池化后的特征拼接起来, 使用全连接层将其投影到类别空间, 得到表示类别标签预测的向量, 输出一个 `softmax` 函数的 `logit` 作为分类结果.

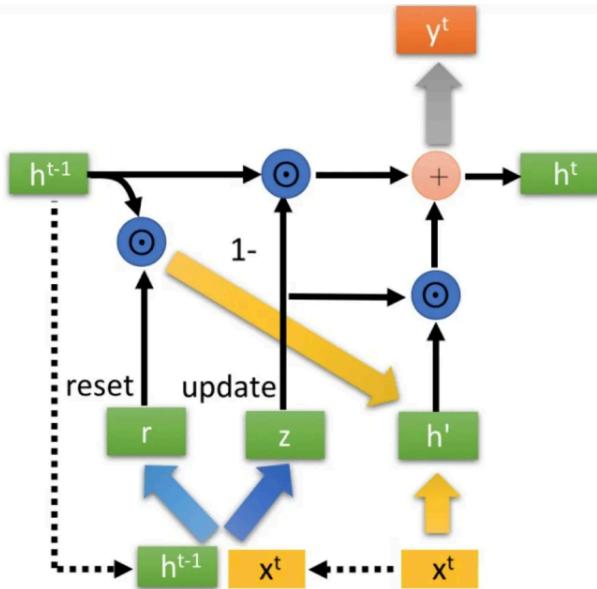
## 2.2 LstmRNN



本模型实现参考了[这篇文章](#).

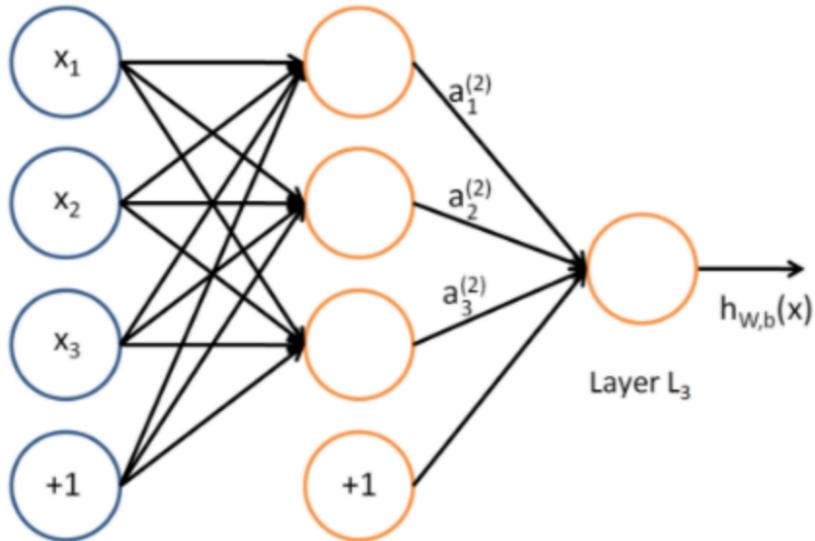
1. **初始化:** 定义了模型所需的相关超参数.
2. **嵌入层:** 将单词的映射为定长的向量, 并将 `wiki_word2vec_50.bin` 作为预处理的嵌入矩阵输入.
3. **双向 LSTM 编码器层:** 两层 LSTM 单元将嵌入的输入序列编码成隐藏状态向量, 在两个方向上分别产生自己的隐藏状态.
4. **解码器层:** 将最后一层 LSTM 单元的前向和后向隐藏状态向量连接在一起, 并投影到 64 维的线性层.
5. **全连接层:** 将解码器层输出的 64 维向量投影到维度与类别空间相同的线性层, 得到表示类别标签预测的向量, 输出一个 `softmax` 函数的 `logit` 作为分类结果.

## 2.3 GruRNN



将上述 LstmRNN 模型中的 LSTM 单元替换为 GRU 单元即得 GruRNN.

## 2.4 MLP



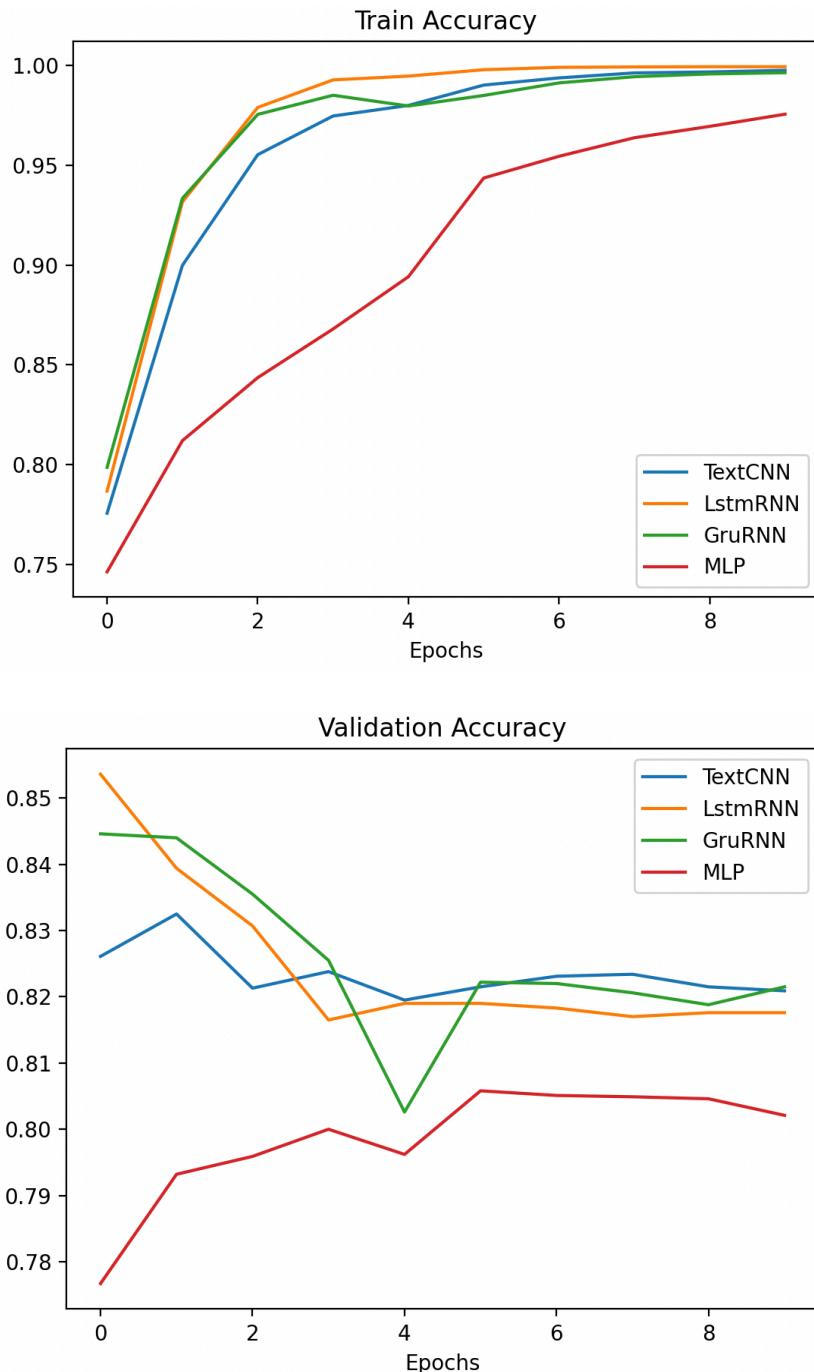
使用 MLP 作为实验的 baseline, 本模型实现参考了[这篇文章](#).

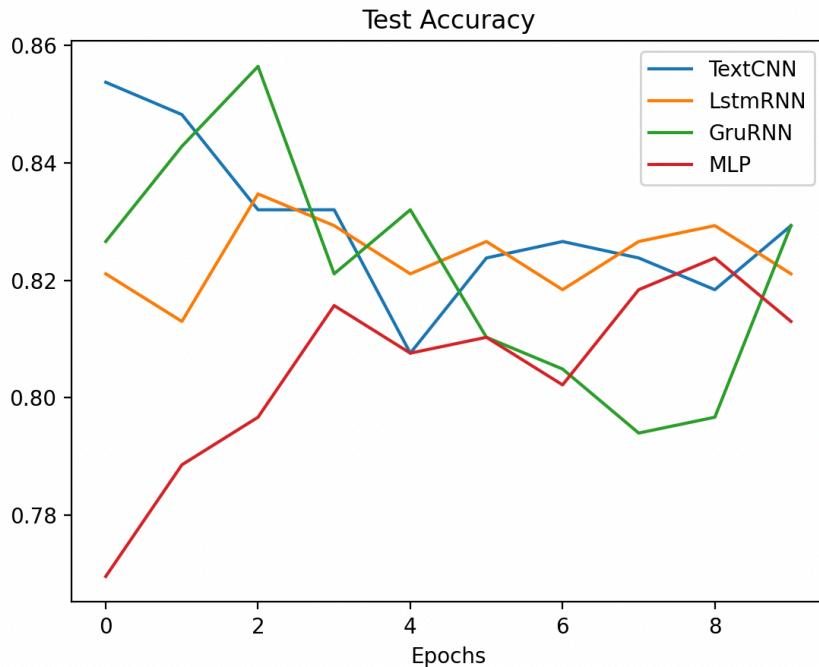
1. **嵌入层:** 将单词的映射为定长的向量, 并将 `wiki_word2vec_50.bin` 作为预处理的嵌入矩阵输入.
2. **多层感知机层:** 对嵌入层的数据进行多层感知机操作, 并使用 **Relu** 激活函数将其投影到一个更高维度的空间.
3. **池化层:** 对多层感知机层的输出进行最大池化操作.
4. **全连接层:** 将池化后的特征投影到类别空间, 得到表示类别标签预测的向量, 输出一个 `softmax` 函数的 `logit` 作为分类结果.

### 3 实验结果与模型差异比较

使用 `Matplotlib` 绘制不同模型训练过程中 (1) `Accuracy`, (2) `Loss`, (3) `F1_score` 指标随训练 `Epoch` 变化的关系.

#### 3.1 Accuracy





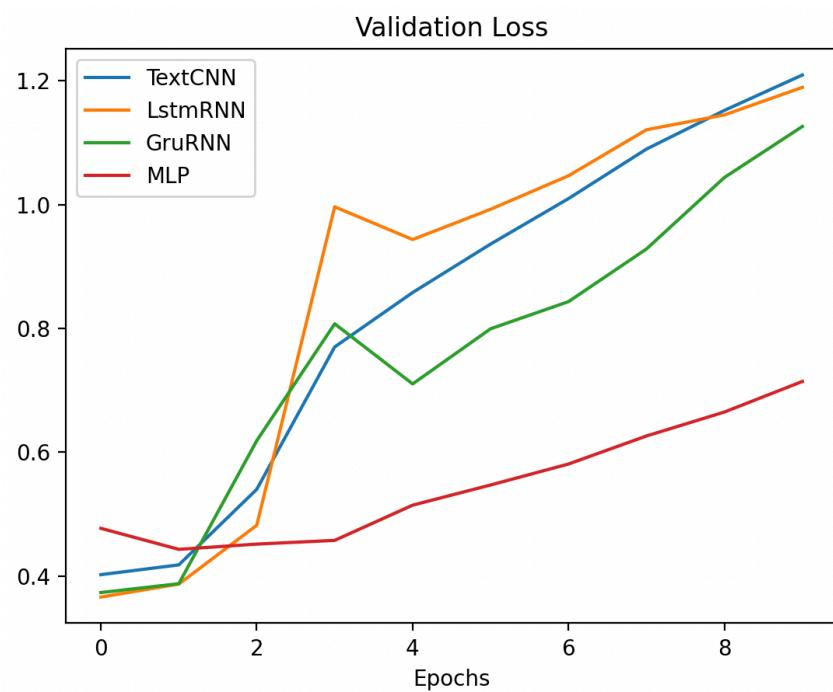
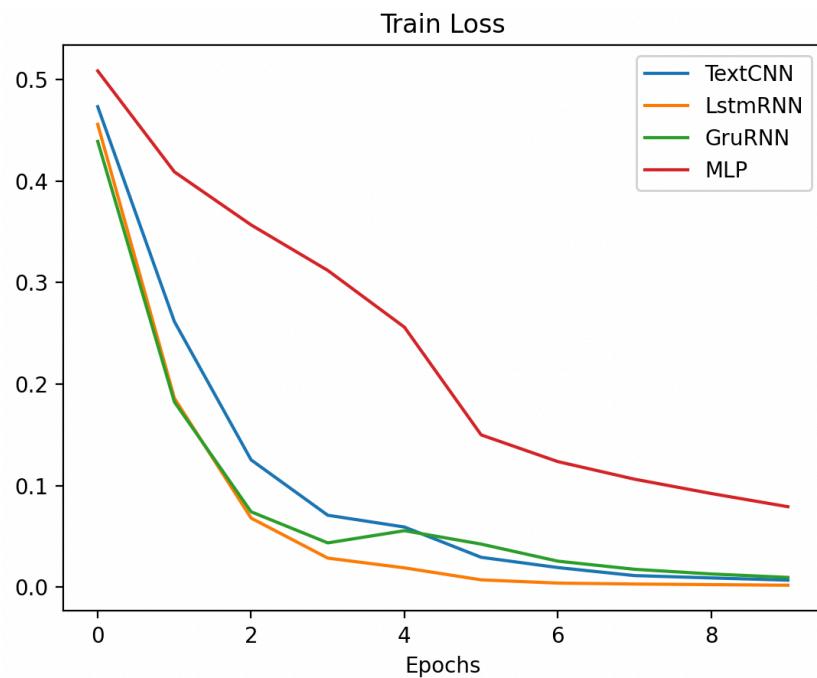
观察到每个模型对应的最佳 `Accuracy` 结果为:

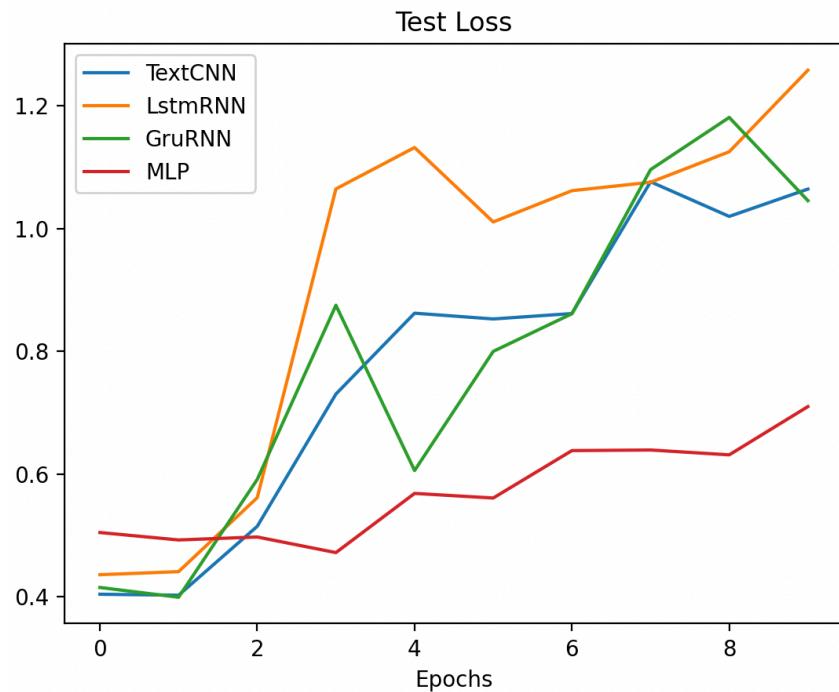
TextCNN	LstmRNN	GruRNN	MLP
0.8537	0.8347	0.8564	0.8238

观察 `Accuracy` 的变化:

- MLP 准确率最低;
- CNN 准确率在 LstmRNN、GruRNN 与 MLP 之间, 过拟合现象不明显;
- RNN 准确率最高, 但是过拟合现象出现较快.

## 3.2 Loss

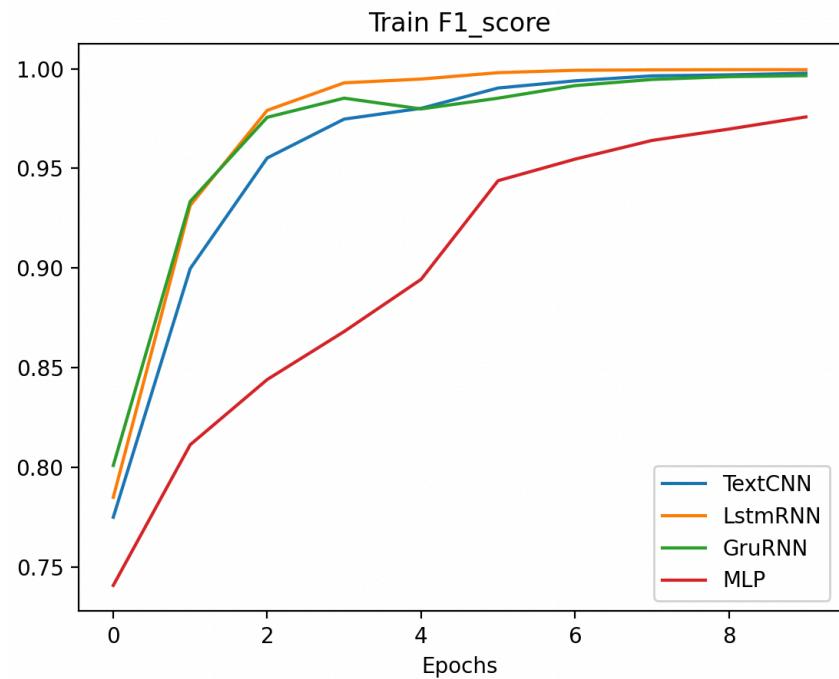


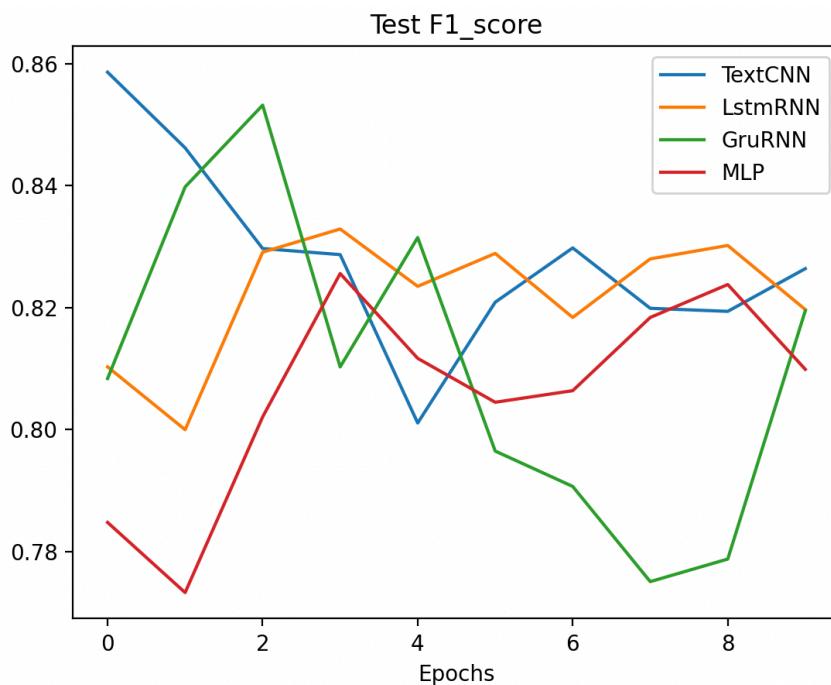
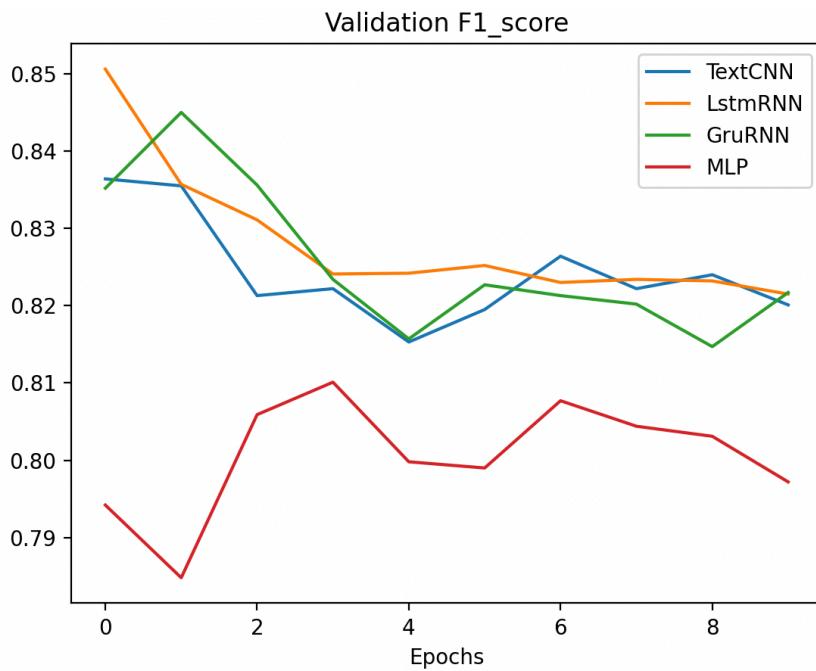


观察训练过程中 `Loss` 的下降速度:

- MLP 损失值下降速度最慢, 模型收敛速度最慢, 稳定程度最低;
- CNN 损失值下降速度居中, 模型收敛速度居中, 稳定程度较好;
- RNN 损失值下降速度最快, 模型收敛速度最快, 稳定程度最好.

### 3.3 F1\_score





`F1_score` 评估了模型的精度和召回率, 观察到每个模型对应的最佳 `F1_score` 结果为:

TextCNN	LstmRNN	GruRNN	MLP
0.8586	0.8329	0.8532	0.8256

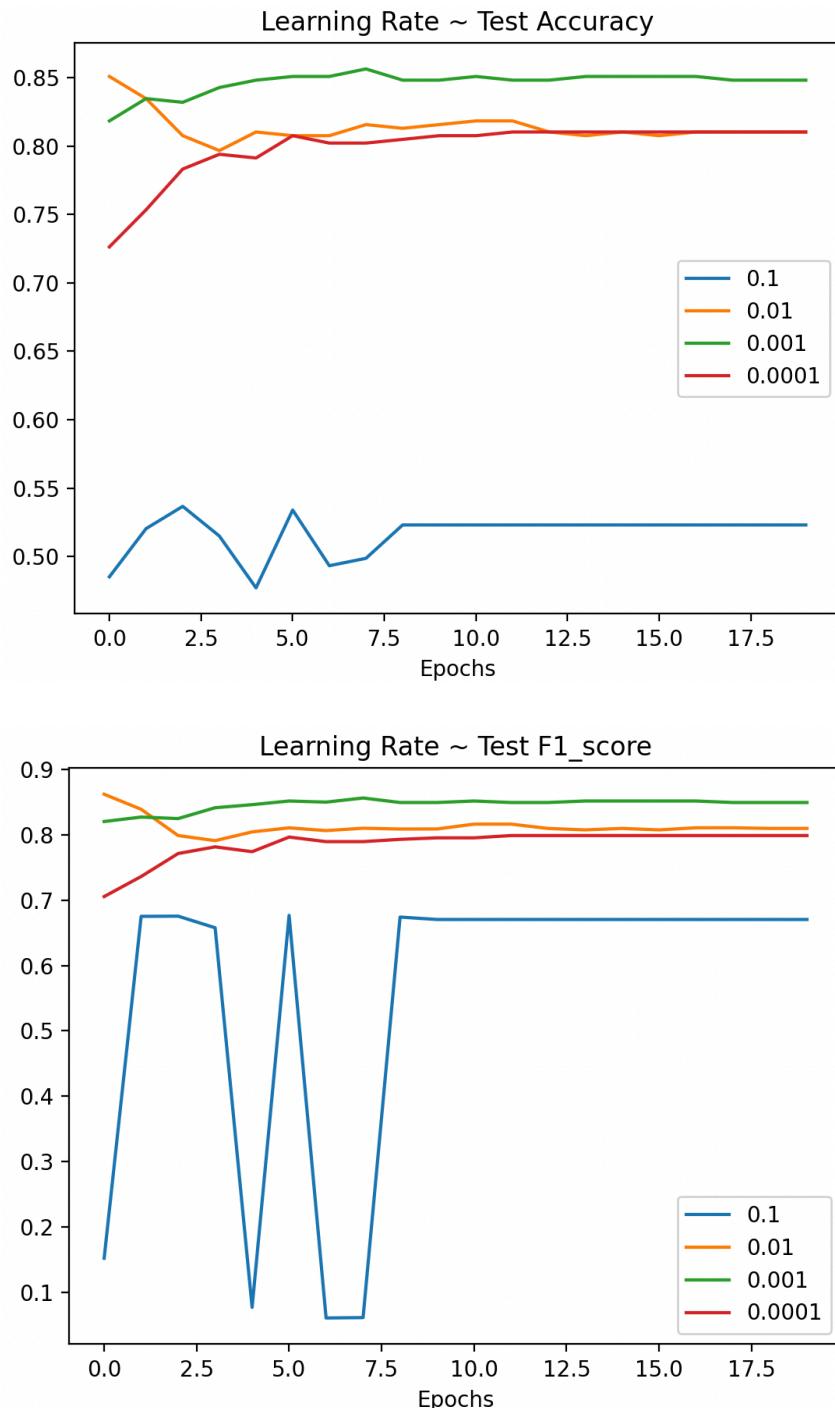
观察 `F1_score` 的变化:

- MLP 的 `F1_score` 最低, 性能相对最差;
- CNN 的 `F1_score` 在 LstmRNN 与 GruRNN 与 MLP 之间;
- RNN 的 `F1_score` 最高, 性能相对最好.

## 4 实验参数分析

### 4.1 Learning Rate

- 以 CNN 为例, `learning_rate` 默认设置为 0.001. 调整 `learning_rate` 的大小, 检测测试集上的 `Accuracy` 与 `F1_score`:

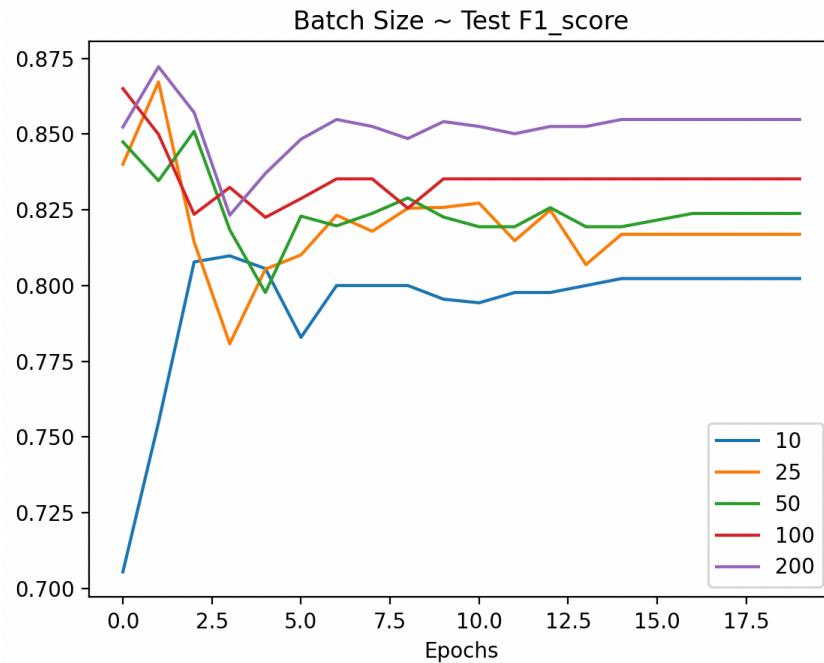
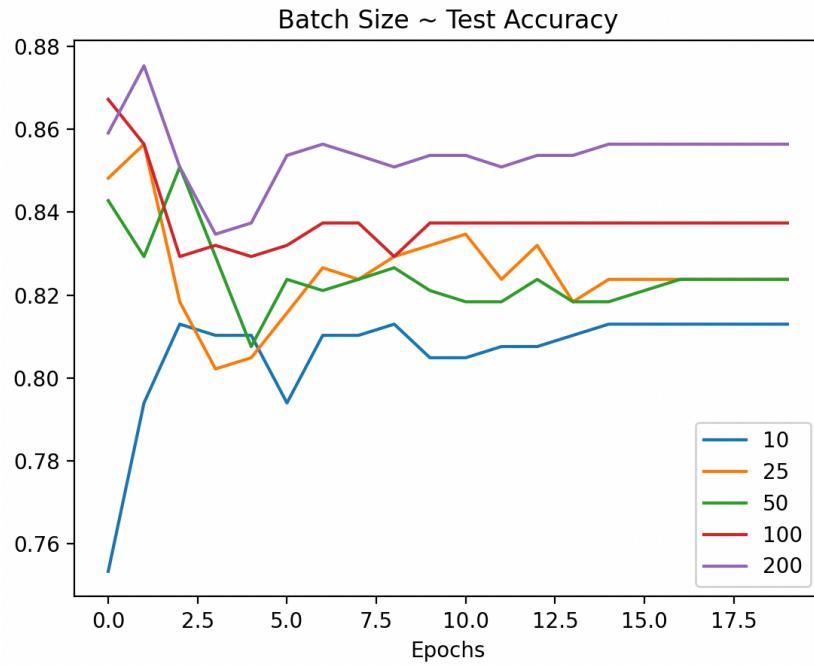


- `learning_rate` 太低可能会导致收敛缓慢, 需要更多的迭代次数才能达到较高的准确率; 而 `learning_rate` 太高可能会导致模型在训练的过程中可能会出现震荡或者无法收敛的情况, 准确率无法达到最优.
- Adam 优化算法所提供的 0.001 的 `learning_rate` (也是我设置的默认值) 是使得模型性能

达到最优的一个合适的值.

## 4.2 Batch Size

- 以 CNN 为例, 在笔记本电脑上使用 CPU 计算时出于算力限制, `batch_size` 默认设置为 50. 在使用实验平台提供的 GPU 进行计算加速时, 调整 `batch_size` 的大小, 检测测试集上的 `Accuracy` 与 `F1_score`:



- 在一定范围内, 增加 `batch_size` 可以利用硬件加速器的并行计算能力, 提高训练效率和准确率, 减少梯度计算的时间. 较大的 `batch_size` 也可以提供更多的数据, 有助于模型更好地学习数据的分布特征.

- 然而, 过大的 `batch_size` 可能会导致模型训练过程出现不稳定的情况, 导致梯度计算不准确或陷入局部最优解. 因此在选择 `batch_size` 时需要根据具体的问题和硬件资源进行权衡和调整.

## 5 实验问题思考

### 5.1 训练停止

**实验训练什么时候停止是最合适的?**

- 起初我在每个 `Epoch` 结束后输出验证集上的性能指标并与最好指标进行比较, 如果性能更优则继续训练, 并更新最好性能指标; 如果观察到验证集 `Accuracy` 达到峰值且训练集 `Loss` 几乎不变则手动停止训练, 这说明此时模型已经出现了过拟合情况, 模型性能已经达到最优.
- 经过尝试后我发现对于所实现的模型, 均在前 8 个 `Epoch` 即可达到最优性能, 如果继续训练, 模型性能会逐渐收敛. 最终我选择默认采用 10 个 `Epoch`, 并可通过命令行参数实现手动调节.

**分析固定迭代次数与通过验证集调整等方法的优缺点.**

- **固定迭代次数**的优点是简单直接, 可以确保训练时间的控制; 缺点是无法保证模型性能达到最优, 可能存在欠拟合或过拟合的风险.
- **通过验证集调整**的优点是可以更好地控制模型的性能; 缺点是如果验证集和测试集的数据分布不同, 可能会导致模型在测试集上的性能不佳, 并且可能会因为持续训练而浪费算力.
- 在具体实验中需要根据数据集和任务情况选择合适的停止训练策略.

### 5.2 参数初始化

**实验参数的初始化是怎么做的?**

- CNN 和 RNN 模型中我并未手动指定参数初始化方法. 查阅资料得到, 在 CNN 和 RNN 模型中默认使用了 Pytorch 的 kaiming 初始化方法对 `Linear` 和 `Conv` 层参数进行初始化.
- MLP 模型中使用了均方差为 0.01, 均值为 0 的高斯分布初始化.

```

1 | for _, parameter in self.named_parameters():
2 |     if parameter.requires_grad:
3 |         torch.nn.init.normal_(parameter, mean=0, std=0.01)

```

**不同的方法适合哪些地方?**

- **零均值初始化**: 适用于深度神经网络, 尤其是包含 ReLU 激活函数的网络. 通过设置所有参数的初始值为 0, 可以缓解梯度消失的问题.
- **高斯分布初始化**: 适用于深度神经网络. 通过将参数随机初始化为高斯分布, 可以控制网络权

重的大小和方差, 控制信息的流动和保持稳定的梯度, 避免了死神经元的问题.

- **正交初始化**: 适用于循环神经网络和卷积神经网络. 通过利用 **SVD** 分解将权重矩阵分解为正交矩阵和对角线矩阵的乘积, 保证了梯度的稳定性和权重的均衡性, 避免了梯度爆炸或消失的问题.
- **kaiming初始化**: 我还查阅了关于 **kaiming初始化** 的相关信息. 它是一种针对 **ReLU** 等类似的非饱和激活函数的初始化方法, 可以提高网络的收敛速度和准确性.

### 5.3 过拟合

有什么方法可以防止训练陷入过拟合?

- **增加数据量**: 收集更多的数据, 让模型更好地泛化到未见过的数据.
- **Data Augmentation**: 通过随机旋转、平移、缩放、翻转等数据变换方法生成新的数据样本.
- **Early Stopping**: 将训练数据集分为训练集和验证集, 观察 **Loss** 和 **Accuracy** 的变化及时停止训练.
- **Regularization**: 对模型的复杂度进行限制来减少过拟合的风险.
- **Model Ensemble**: 将多个不同的模型进行集成, 提高模型的泛化能力.

### 5.4 试分析 CNN、RNN、MLP 三者的优缺点.

CNN、RNN 和 MLP 是常见的神经网络模型, 各自有其独特的优缺点.

- **CNN**
  - 优点是可以并行计算, 提高训练速度, 对句子中的局部特征提取效果好.
  - 缺点是只能处理固定长度的向量, 同时难以处理长距离依赖关系, 不适合需要长期记忆的序列任务.
- **RNN**
  - 优点是可以处理序列数据, 能对上下文信息进行建模.
  - 缺点是训练时间较长, 且容易出现梯度消失或梯度爆炸的问题.
- **MLP**
  - 优点是结构简单, 易于实现和理解.
  - 缺点是只能处理固定长度的向量, 无法处理变长的序列数据.

## 6 实验心得体会

通过本次实验, 我深刻地认识到了神经网络模型的构建和训练过程中各个环节的重要性. 在**数据加载**的过程中, 我学会了如何读取和预处理不同类型的数据; 在**模型构建**的过程中, 我理解了卷积神经网络和循环神经网络的结构和原理, 并掌握了如何添加不同类型的层和激活函数, 以及如何设置权重初始化和正则化等技巧; 在**设置训练方式和损失函数**的过程时, 我学会了如何设置不同的优化器和

学习率调整策略, 以及如何选择适当的损失函数来衡量模型的性能; 在**数据处理**的过程中, 我还使用了 Python 自带的 `logging` 库记录结果, 并通过 `Matplotlib` 库进行实验数据可视化.

通过不断地**调整参数和修改模型结构**, 我得到了不同的训练结果, 也学会了依据不同指标评估和比较模型的性能. 通过这次实验, 我不仅加深了对**神经网络与深度学习**中的重要问题的理解, 还提高了自己的**实践能力和代码能力**, 对未来的学习研究和应用打下了坚实的基础.