IAI实验二报告：文本情感分类

张凯文 计12 2021010729 2023年5月7日

任务简介

本次实验任务是实现文本情感的二分类，具体而言，输入一个中文句子，输出一个情感标签，为 1 表示负面情感，为 0 表示正面情感。

模型结构与流程分析

本次实验我共实现了五大类模型结构：CNN、RNN、MLP、Self Attention 和基于 Bert 的模型。

CNN

对于输入的（已分词）的句子 s，首先经过裁剪或填充（padding）到固定长度 n，然后经过预训练的嵌入（Embedding）层，将每个词 s_i 映射为一个 d 维词向量 x_i，因此每个句子可表示为：

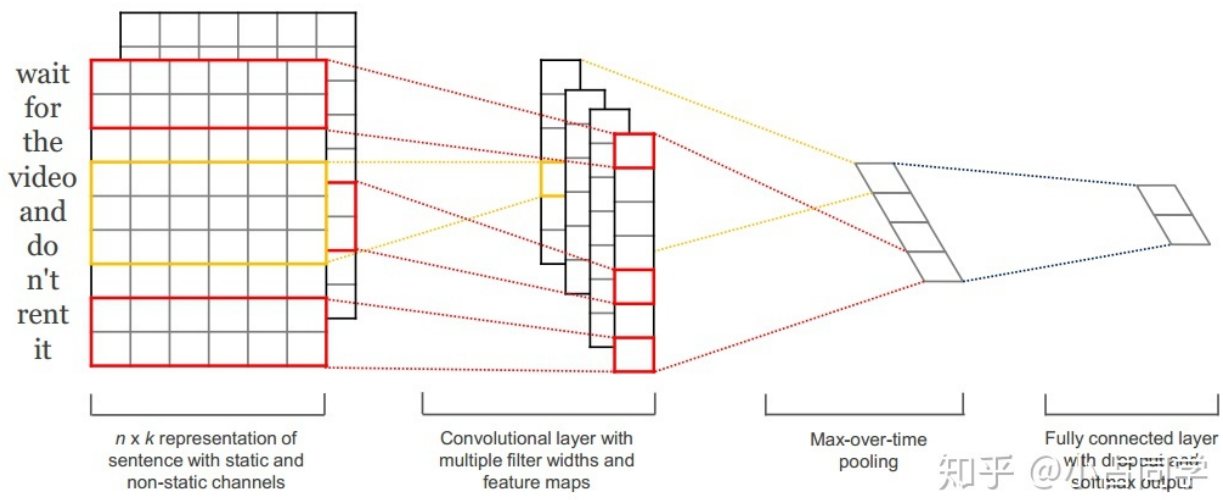
$$X = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^{n \times d}$$

从第 i 个词开始的 k 个词可以表示一个词窗口 $X_{i:i+k-1}$，通过一个 k 维卷积核 $\mathbf{W} \in \mathbb{R}^{k \times d}$、偏差（bias） $b \in \mathbb{R}$ 和非线性激活函数（如 RELU）f 的作用，可以提取一个特征：

$$c_i = f(\mathbf{W} * X_{i:i+k-1} + b)$$

其中 $*$ 为互相关算子，进行类似内积的操作。卷积核在 X 第一维度方向扫描一遍，即可提取到句子的一个特征图 $\mathbf{c} = (c_i, c_2, \cdots, c_{n-k+1})^T \in \mathbb{R}^{n-k+1}$，在经过一层最大池化操作即可提取出 X 一个特征 h。 经过 m 个不同维度的卷积核，依次提取句子在不同窗口粒度衡量下的特征，最后经过一个 Dropout 层和一个线性层变换，即可得到最终的二维输出 y。通过 Softmax 函数，可依此计算出属于两种情感类别的概率。

上述模型结构和流程主要参考了论文：Convolutional Neural Networks for Sentence Classification，其主要流程可参见下图：



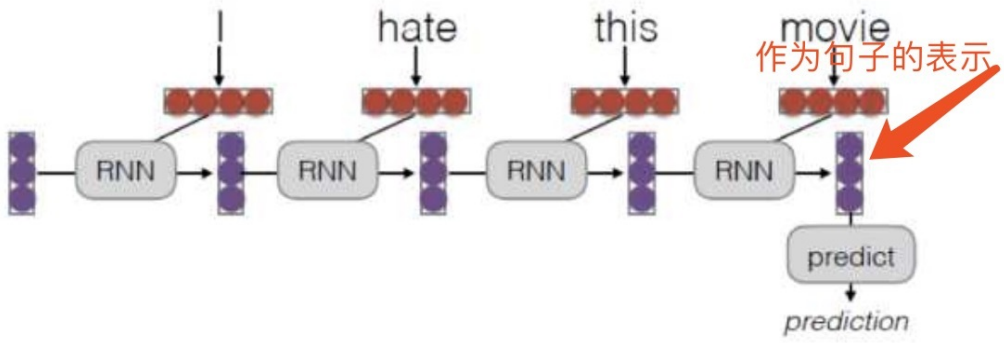
RNN

经过同样的嵌入层将输入的句子映射为 $n \times d$ 矩阵后，RNN 结构将词向量按顺序逐一输入网络，通过传递隐藏层特征向量 h_t 提取句子特征。具体而言，以原始 RNN 的第一层为例，输入第 i 个词向量时，由前一时刻的隐藏层特征向量 h_{i-1} 和 x_i，施加激活函数 f，可以计算得到下一时刻的隐藏层特征向量：

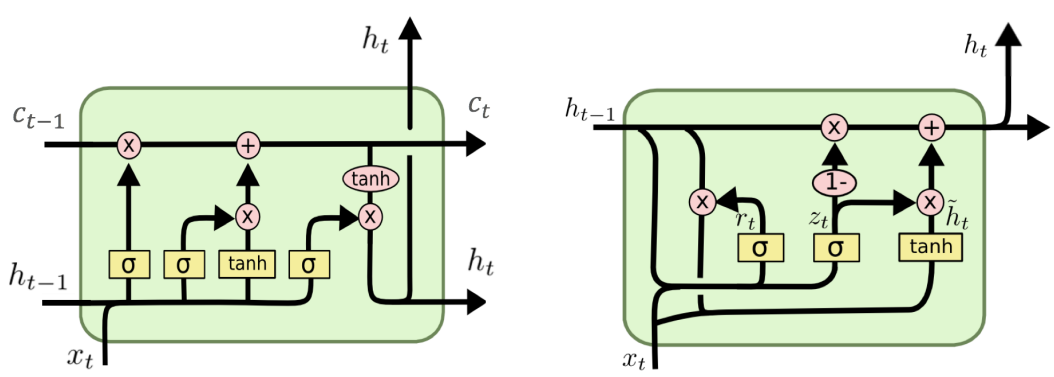
$$h_i = f(h_{i-1}\mathbf{W}^T + x_i\mathbf{U}^T + b)$$

取最后一个隐藏层特征向量 h_n 作为整个句子的特征，再经过一个线性层映射到二维向量，即得到输出 y。在具体实现中，RNN 的结构可以选取原始 RNN、LSTM 和 GRU，提取特征的方式可以设置单向或双向、一层或多层。特别地，对于双向结构，我将前向和反向过程得到的最终隐藏层向量拼接起来作为整个句子的特征向量输入最终线性层。

其大致流程可参见下图：



特别地，LSTM 和 GRU 是在原始 RNN 结构基础上为解决梯度爆炸等问题做出的改进，在宏观上的架构不变，但提取中间特征 h_t 的方式较为不同，具体可参见下图：



LSTM
(Long-Short Term Memory)

GRU
(Gated Recurrent Unit)

以下三种模型为额外实现，作为上述两种模型的对比，作简要说明：

MLP

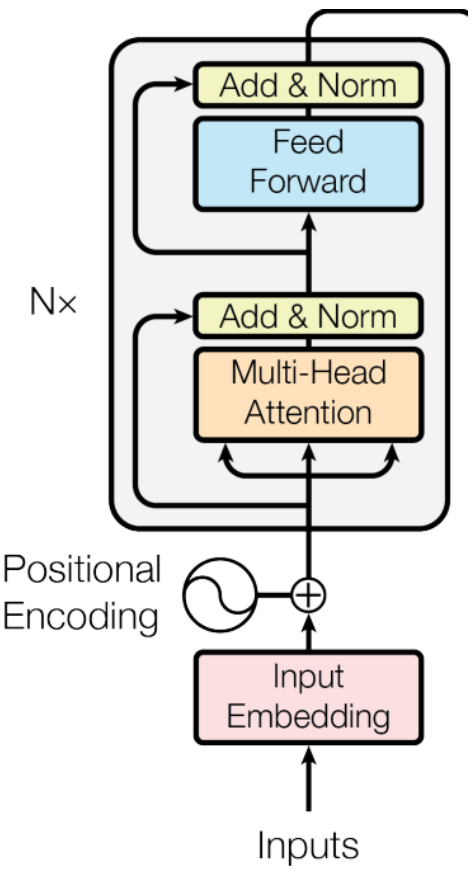
MLP 网络将词向量矩阵 X 直接通过一层线形层映射到特征向量：

$$h = f(X\mathbf{W}^T + b)$$

h 还可以进一步经过若干带残差连接的线性层提取特征，最终经过一个线性层映射到输出 y。

Self Attention

通过论文 Attention is all you need 中介绍的 Transformer Encoder 结构提取句子的特征向量 h，其结构可参见下图：



其中提取特征的核心部分 Multi-Head Attention 结构可参见下图：

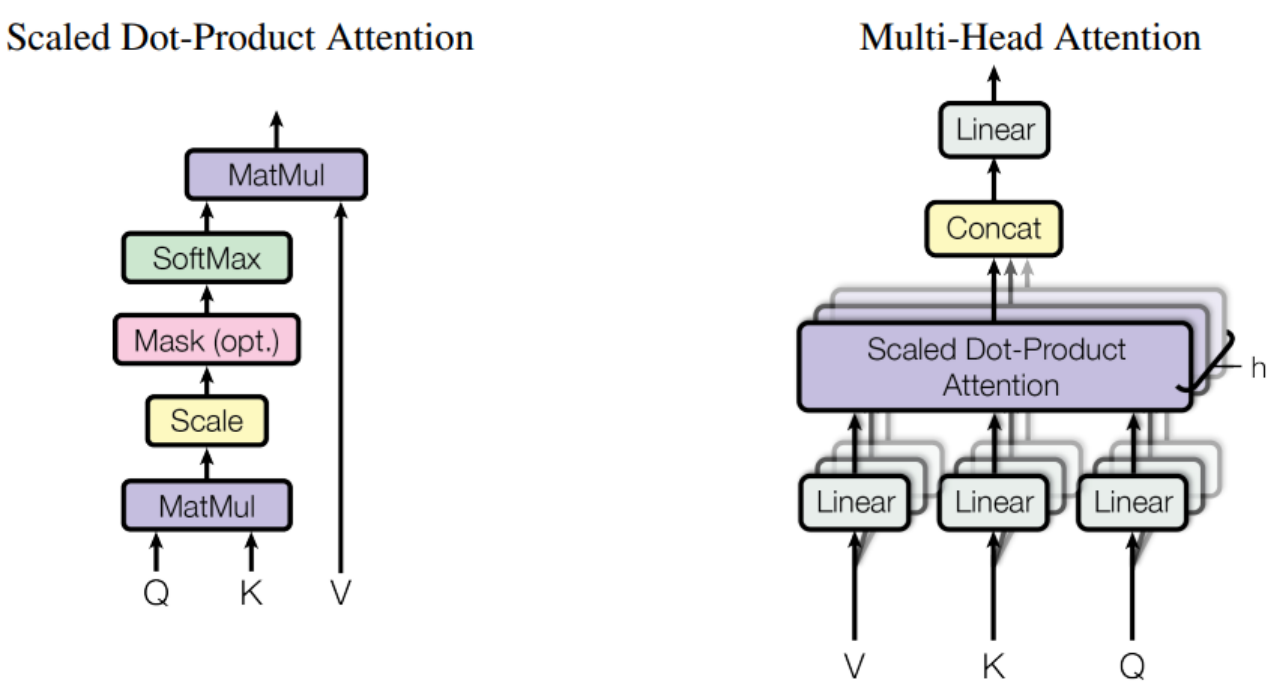


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

具体而言，对输入的词向量矩阵，首先与一个可学习的表征位置信息的嵌入层输出（Positional Encoding）相加，然后输入若干 Encoder Layer 组成的 Encoder 中。设某一层输入为 x，输入 MultiHead Attention 层：

$$\begin{aligned}\text{MultiHead}(Q,K,V) &= \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

在自注意力中，取 $Q=K=V=x$，每个 head 对应的权值矩阵类似 CNN 中的卷积核，从不同角度提取句子特征。与 CNN 不同的是，Attention 中不再存在窗口机制，可以更有效地进行长序列建模。对于 MultiHead Attention 层的输出，再经过残差连接、标准化和线性层即可得到该层提取的特征。

对于提取出的特征 h，可以再输入线性层、CNN 或再接一个单头注意力层等结构，映射到二维输出 y，在代码中均有相应实现。

Bert Based

使用预训练语言模型 bert-base-chinese · Hugging Face 作为嵌入层（冻结参数），再连接一个可训练的简单线性输出层，直接得到二维输出 y。利用预训练语言模型中包含的大量语言先验知识，能更好地理解句子语义，直接进行情感分类。

实验结果

> 实验环境：主要使用本机电脑自带的单张 NVIDIA GeForce MX450 显卡。在前期大量预实验调参阶段，辅助使用了实验室的 NVIDIA Tesla A100 40G。

> 所使用的 python 库置于根目录下 `requirements.txt` 中，可以通过 `pip` 一键安装。

在上交的 checkpoint 中，使用 `config/cnn_config.py` `config/rnn_config.py` `config/mlp_config.py` `config/attn_config.py` `config/bert_config.py` 中定制的配置，在测试集上的实验结果如下（可在主目录下在 `main.py` 中选择相应配置，运行 `python main.py -t` 进行测试复现）：

	CNN	RNN (双向LSTM)	MLP (三层)	Self Attention	Bert Based
准确率	82.38%	79.67%	76.42%	82.38%	85.91%
F-score	82.76%	80.11%	77.04%	83.79%	86.53%

参数比较

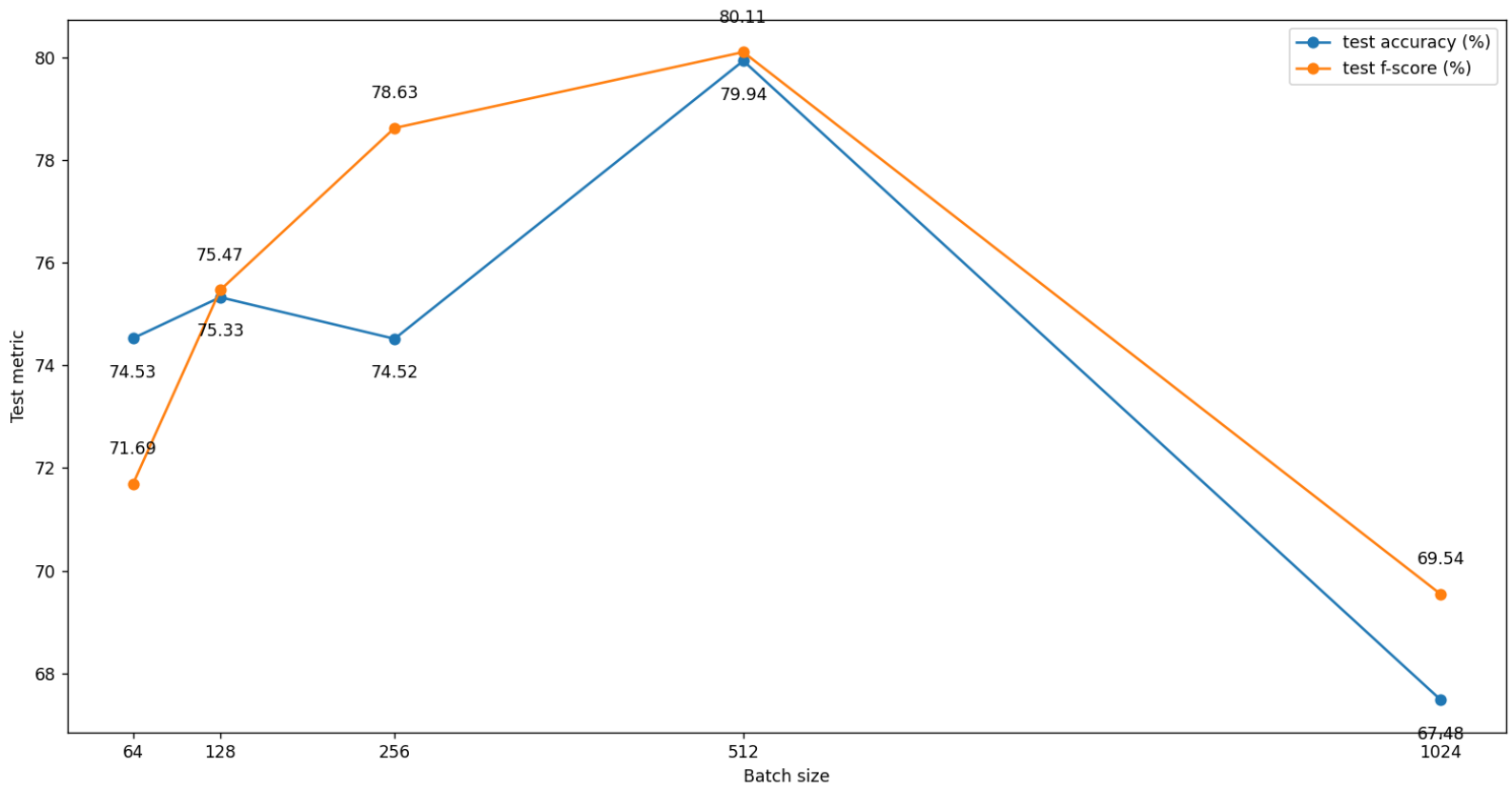
我主要在 TextCNN 上进行了大量调参实验，下面展示实验结果并分析原因。

基础配置：

batch size	512
learning rate	0.0005
sentence length	60
hidden size	64
dropout rate	0.5
kernel sizes	[3, 4, 5]
epoch	10 (不使用早停)
optimizer	Adam
scheduler	StepLR (step_size=5)

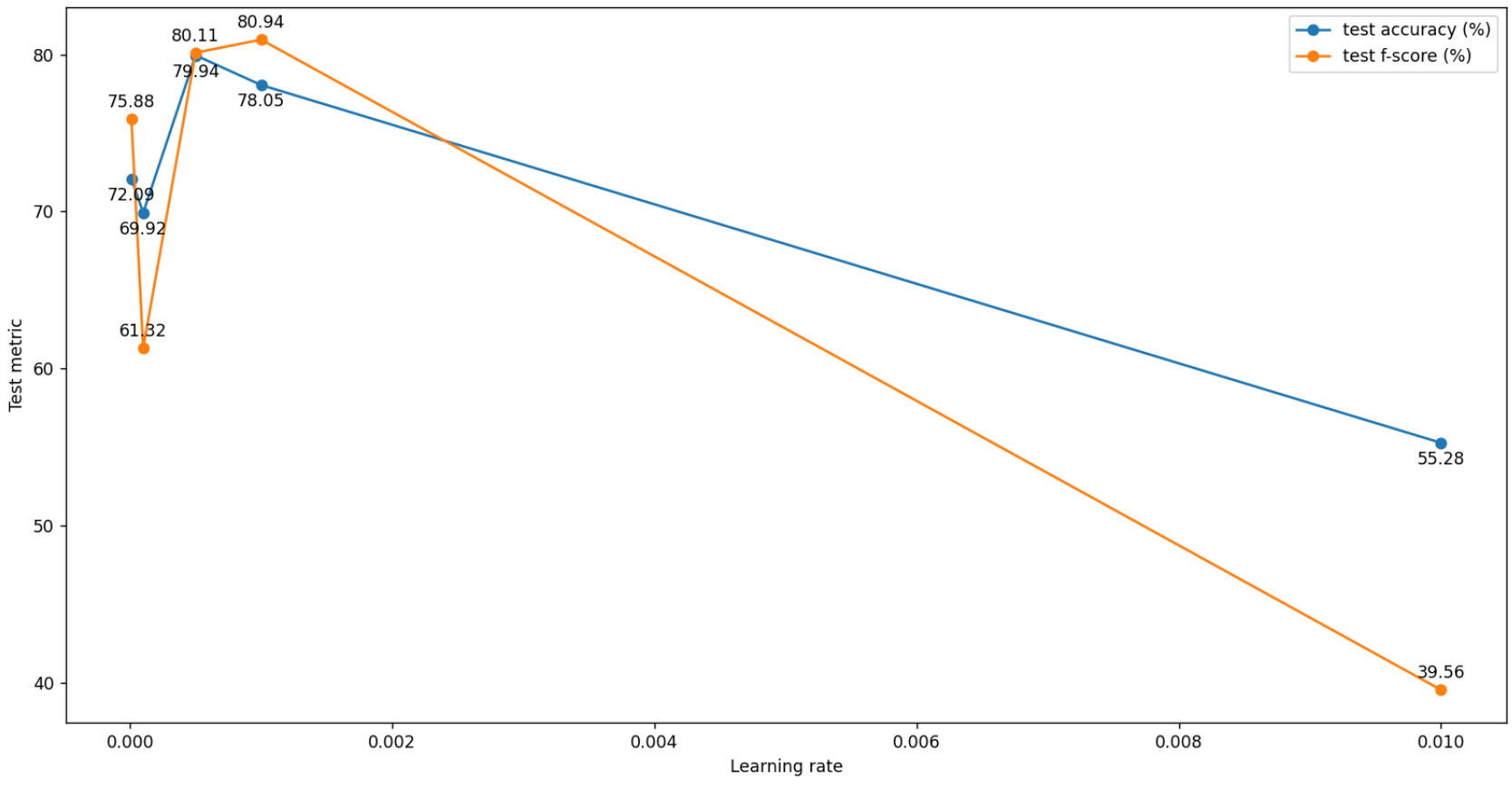
这一配置是在经过前期大量预实验后总结出来的相对较好的初始配置。下列实验是在上述配置基础上修改特定某一项进行的，并以在测试集上的准确率和 f-score 作为对比评价指标。

Batch size



实验发现，batch size 的合适选取大小与显卡显存有关，一般而言设置在 128-512 较好，不宜过大或过小。batch size 过小，训练时间较长，同时随机性增强，梯度震荡严重，不利于收敛；batch size过大，不同 batch 的梯度方向没有任何变化，容易陷入局部极小值。

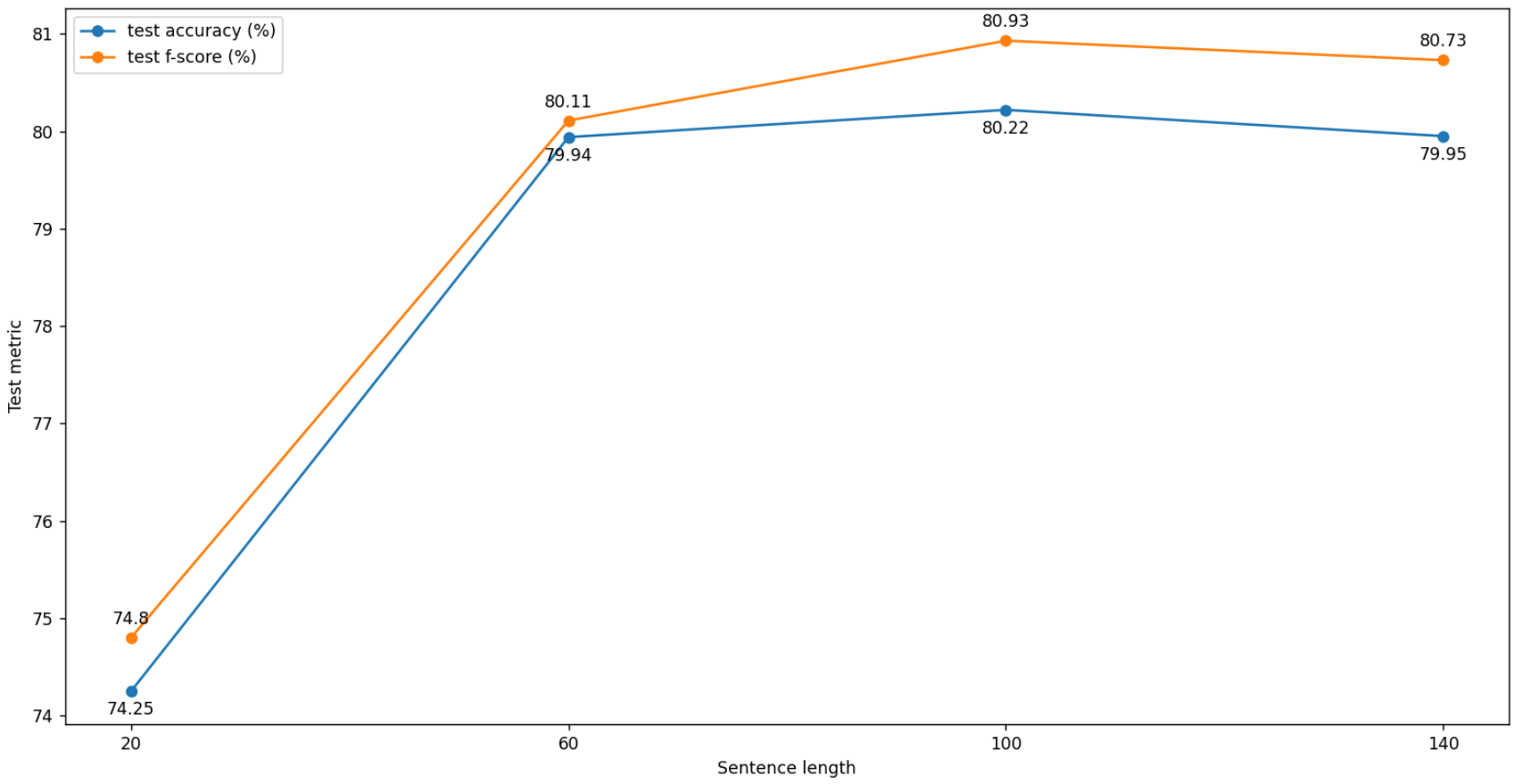
Learning rate



> x 轴取值为：[1e-5, 1e-4, 5e-4, 1e-3, 1e-2]

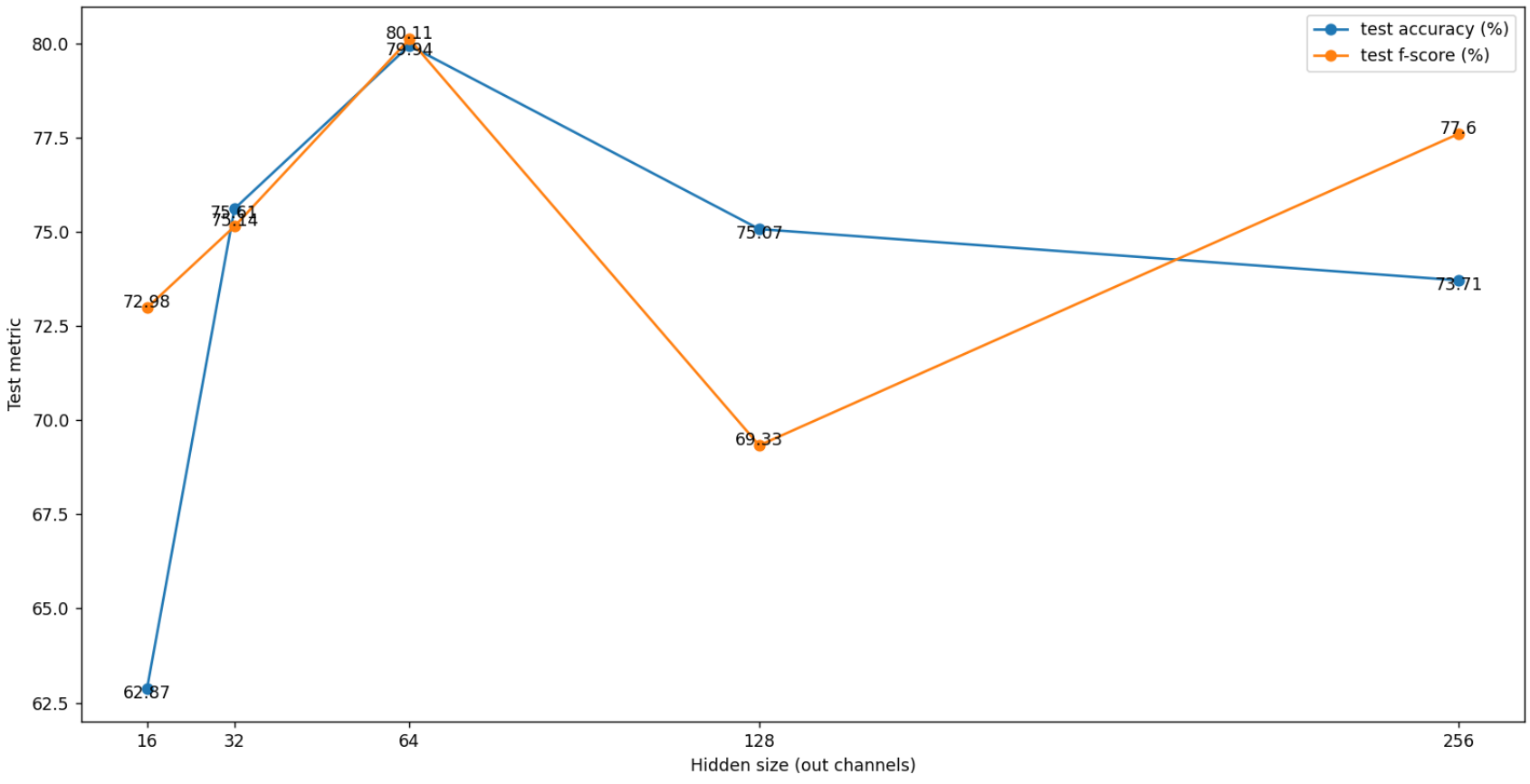
learning rate 也不宜过大或过小，一般取在 1e-3 附近比较合适。learning rate 过大，可能导致待学习的参数在最小值附近波动，无法收敛到最小值；learning rate 过小，可能导致待学习的参数收敛缓慢，在有限的 epoch 和 iterations 内无法收敛到最小值，也更容易进入一些局部极值（如本实验中 1e-4 的情况，很有可能就是在这个初始点附近进入了一个局部极值没有出来）。

Sentence Length



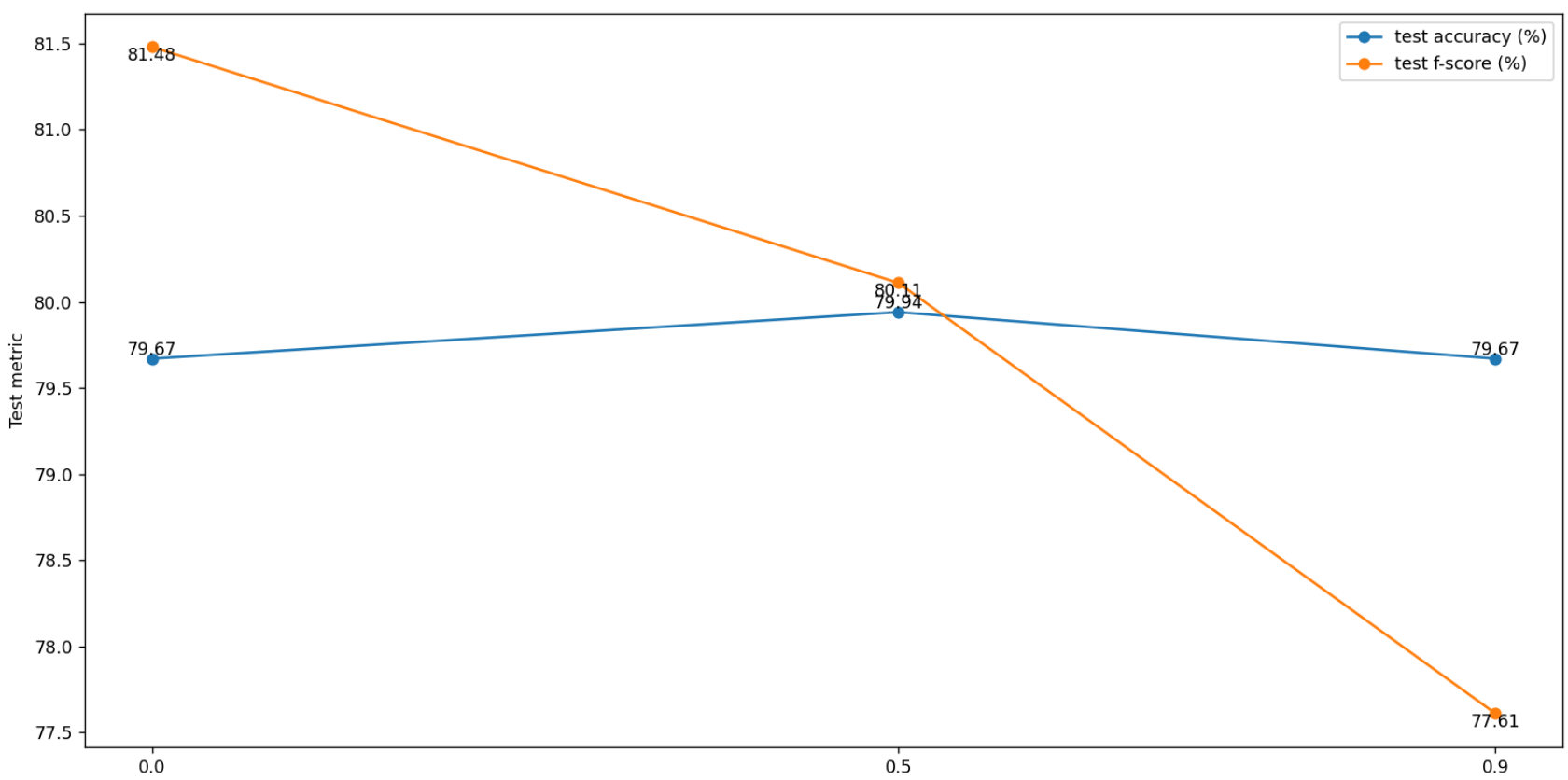
由于所有方法中都首先通过裁剪或填充对输入句长做了统一，这一句长参数也对语义提取产生影响。大致浏览发现，数据集中绝大多数句子都在 40-60 词，最长大约不超过 150 词。从测试结果也可以看出，sentence length 取在 60 以上结果就基本不变，取得过小自然会影响语义提取（但 CNN 可能能较为有效地识别空串填充，因此实验中取得较大时效果并未出现明显下滑）。

Hidden size



hidden size 涉及表征句子特征的参数量，参数量过小不能有效表达句子特征，参数量过大容易在训练集上过拟合。

Dropout rate



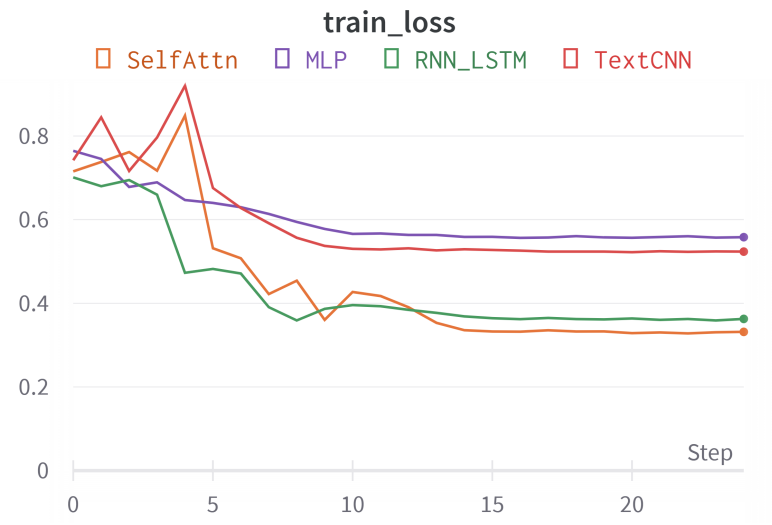
在参数量选取合适的情况下，模型对 dropout rate 相对而言没有那么敏感。

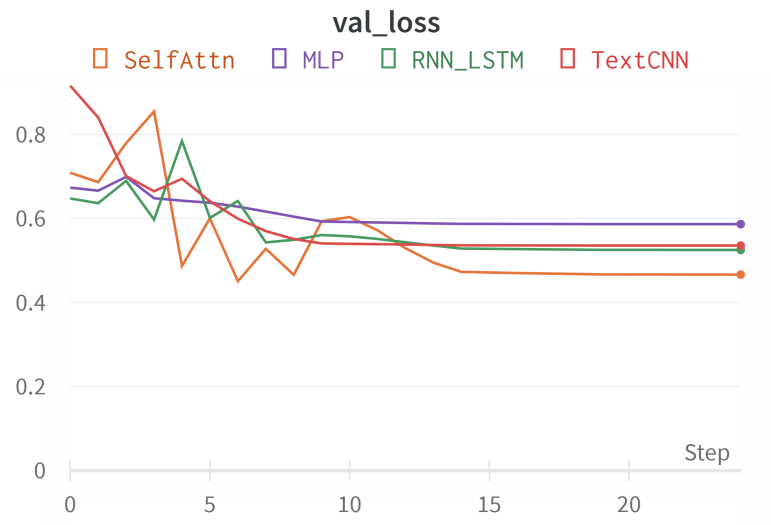
模型比较

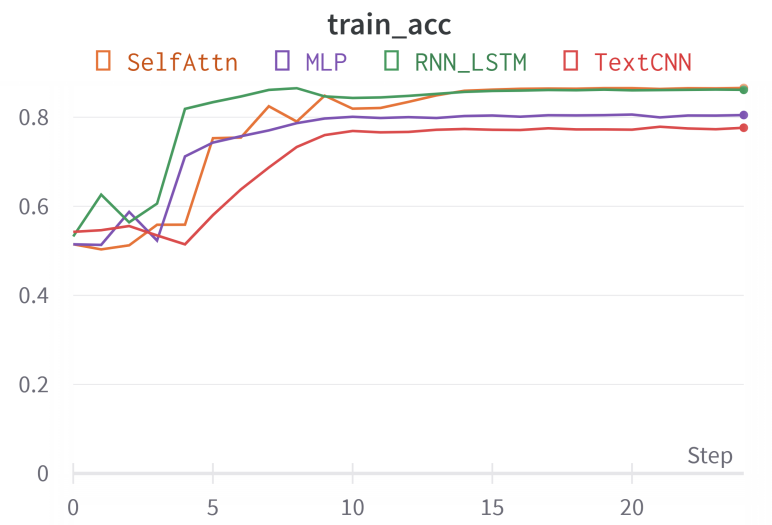
由 **实验结果** 部分给出的数据可以看出：

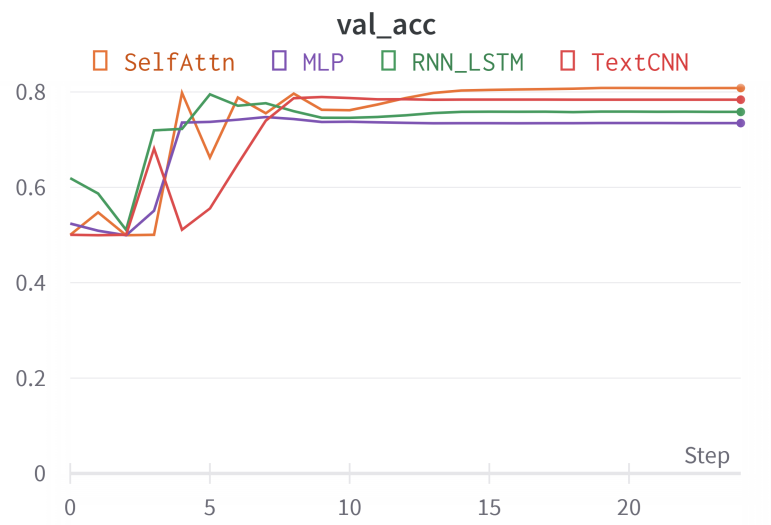
1. 基于 Bert 预训练模型的情感分类在准确率和 F-score 上均取得了最优效果，并且其下游模型结构简单，超参数种类少，对超参数相对不敏感，易于训练和调试，是更加现代高效的文本分类方法。
2. 效果其次的是采用自注意力机制提取句子特征的模型，相比 CNN，它可以更有效地对长序列建模而非仅在小窗口内提取特征，更适宜 NLP 任务。值得注意的是，Self Attention 结构对 sentence length 这一超参数更为敏感，将其设置为 100 的效果显著好于设置为 60 的效果。
3. 再次为文本分类问题特定设计的 TextCNN，最后为 RNN 和 MLP。这三类基础模型的优缺点比较详见 **问题思考** 部分第 4 问。

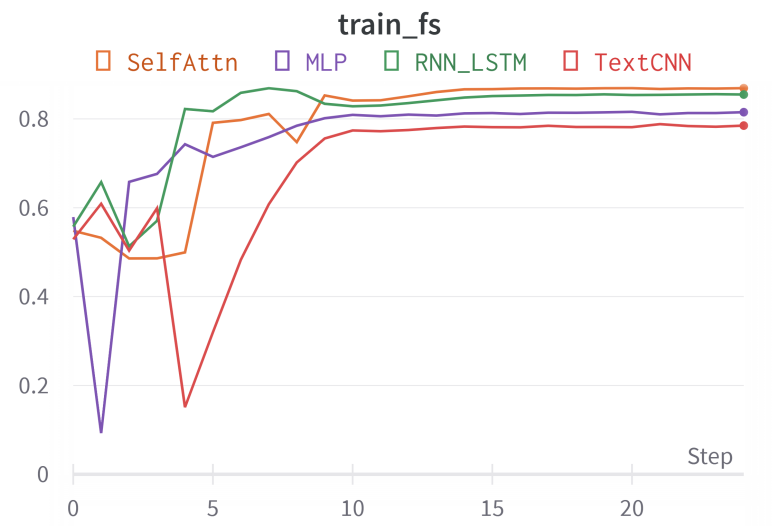
各个模型在训练过程中，训练时和验证时每个 epoch 的 loss、accuracy 和 f-score 变化如下图所示：

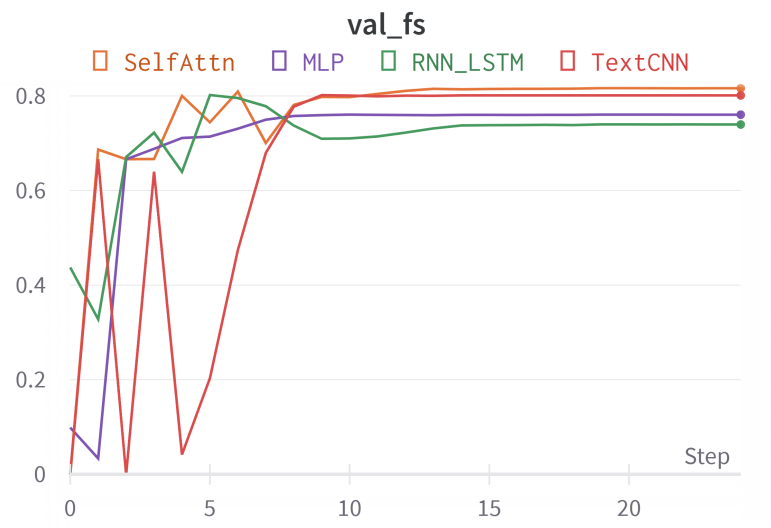












这里各模型共用超参数配置相同。由于 Bert Based Model 使用的嵌入层不同，结构相对特殊，故不放在一起比较。

可以看出，TextCNN 和 MLP 相较 RNN 和 Self Attention 在训练集上收敛较快；RNN 略有过拟合，在训练集上表现出色但在验证集上表现一般；Self Attention 语言建模能力更强，在训练集和验证集上表现均很出色。

问题思考

一、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。

当在验证集上的 loss 不再下降或评价指标（如本题中的 accuracy、f-score 等）的时候，就可以及时停止。在我的实现中，我在训练配置中设置了 `epoch` 和 `early_stop` 选项，可以设置最大 epoch 数以及选择是否采用早停。若 `early_stop` 设为 `False`， `epoch` 即为固定值；若 `early_stop` 设为 `True`，则需要进一步设置 `early_stop_steps`，此时程序会在验证集的平均 f-score 达到最大值是记录模型参数，当连续 `early_stop_steps` 个 epoch 验证集 f-score 都未超过历史最大值时停止训练，并使用历史记录的模型参数作为最终结果。

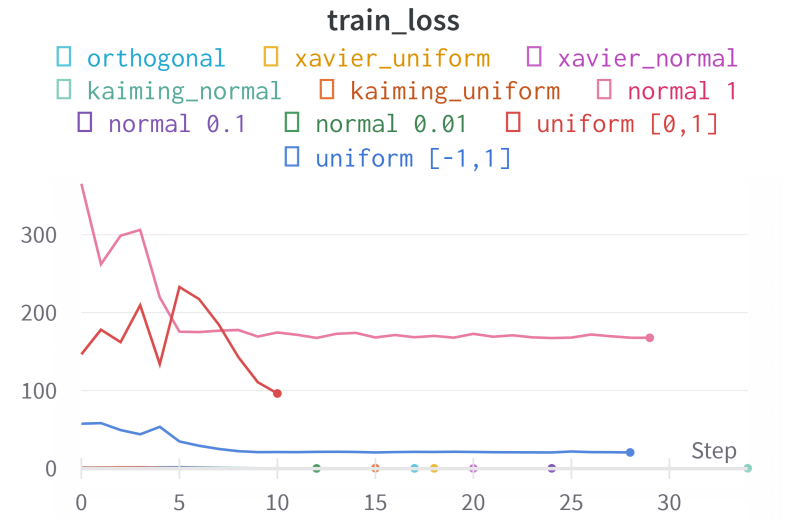
固定迭代次数的方法实现起来较为方便，但需要根据不同模型结构事先进行大量预实验选择较为合适的迭代次数，且迭代次数过多时容易造成过拟合；通过验证集调整的方法能根据模型在验证集上的实际表现自动早停，避免在训练集上过拟合，但由于仅以验证集的表现作为评价指标，也可能出现错误早停的情况。

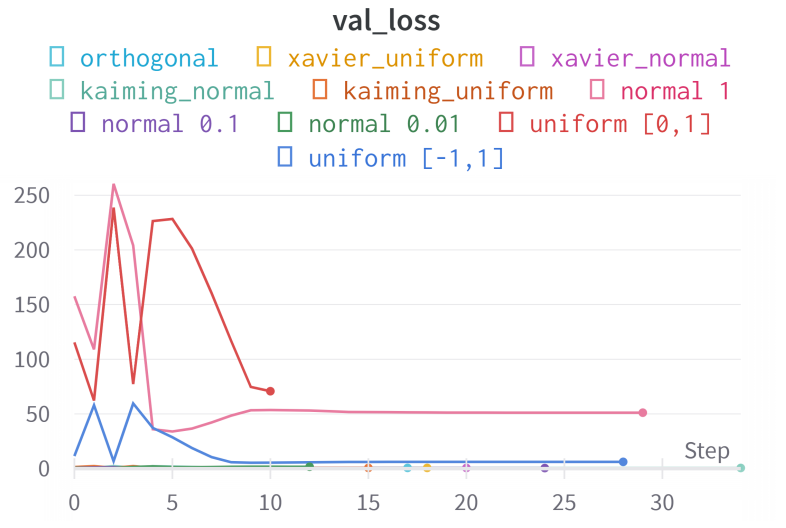
二、实验参数的初始化是怎么做的？不同的方法适合哪些地方？（现有的初始化方法为零均值初始化，高斯分布初始化，正交初始化等）

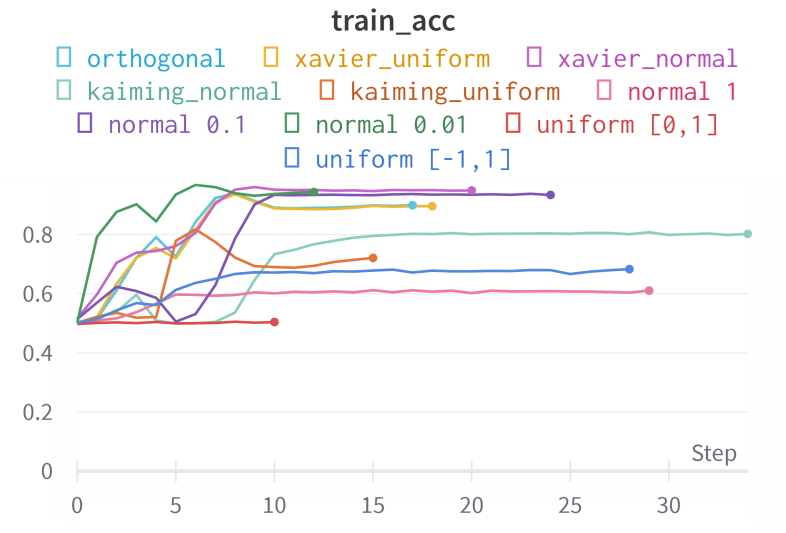
在 PyTorch 中，对线性层、卷积层等参数一般默认使用 `kaiming_uniform` 进行初始化。在我的实现中，可以在 `base_config` 中配置 `init_config` 选择 PyTorch 提供的各类初始化方法。各类常用初始化的大致做法和使用场景如下：

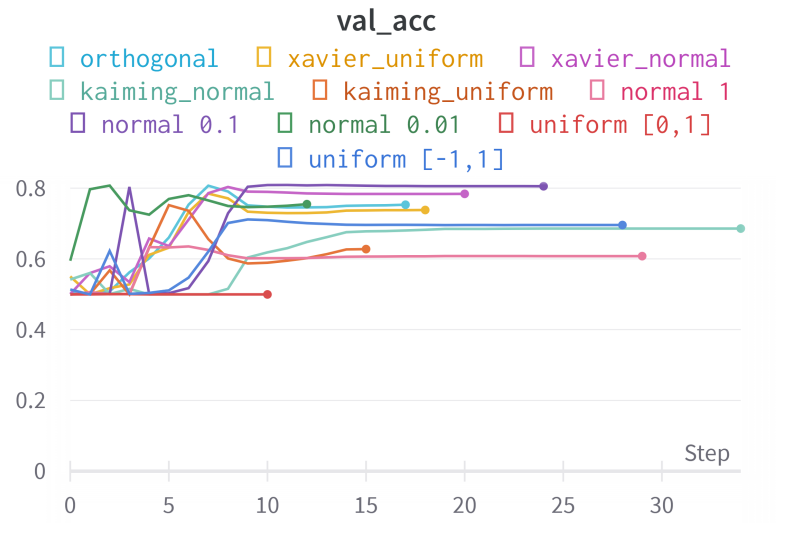
- **均匀分布初始化**：在一个给定的范围内，随机地从均匀分布中初始化参数值。这种方法可以保证参数的方差相同，但是可能会导致参数过大或过小，影响激活函数的输出。
- **正态分布初始化**：在一个给定的均值和标准差下，随机地从正态分布中初始化参数值。这种方法可以保证参数的期望为零，但是可能会导致参数出现异常值，影响激活函数的输出。
- K**aiming初始化**：是由何恺明等人在2015年提出的一种针对 ReLU 等非线性激活函数的初始化方法。它的基本思想是保持每一层的输入和输出的方差相同，避免梯度消失或爆炸。它的公式如下：
  - 如果权重矩阵 W 服从均匀分布，则 $W \sim U\left(-\sqrt{\frac{6}{n_l}}, \sqrt{\frac{6}{n_l}}\right)$ ，其中 n_l 是输入单元数。
  - 如果权重矩阵 W 服从正态分布，则 $W \sim N\left(0, \sqrt{\frac{2}{n_l}}\right)$ ，其中 n_l 是输入单元数。
- **Xavier初始化**：也叫 Glorot 初始化，是由 Glorot 和 Bengio 在 2010 年提出的一种针对 sigmoid 等对称性激活函数的初始化方法。它的基本思想是保持每一层的输入和输出的方差相同，避免梯度消失或爆炸。它的公式如下：
  - 如果权重矩阵 W 服从均匀分布，则 $W \sim U\left(-\sqrt{\frac{6}{n_l+n_{l+1}}}, \sqrt{\frac{6}{n_l+n_{l+1}}}\right)$ ，其中 n_l 和 n_{l+1} 分别是输入单元数和输出单元数。
  - 如果权重矩阵 W 服从正态分布，则 $W \sim N\left(0, \sqrt{\frac{2}{n_l+n_{l+1}}}\right)$ ，其中 n_l 和 n_{l+1} 分别是输入单元数和输出单元数。
- **正交初始化**：使用正交矩阵作为参数值。这种方法可以保证每一层的输入和输出之间没有线性相关性，避免梯度消失或爆炸。这种方法适用于使用 tanh 等饱和性激活函数的网络。

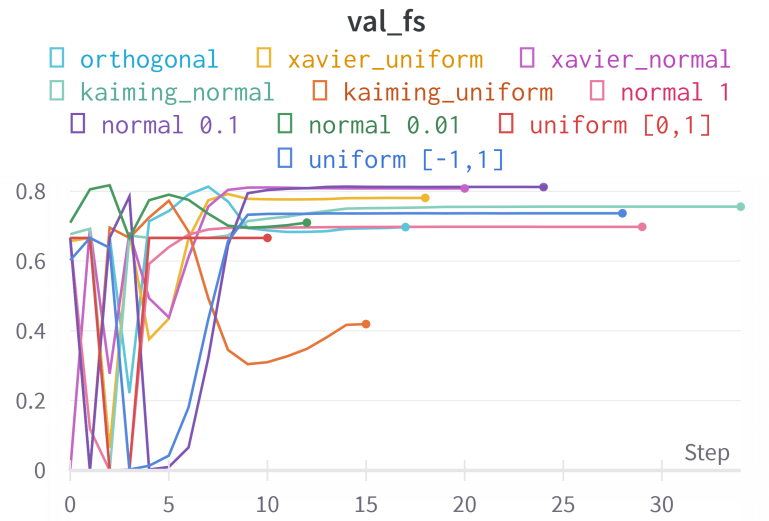
我通过实验实际检验了各类初始化方法的效果（其他配置均相同，使用前面提到的早停技术），训练过程记录如下：



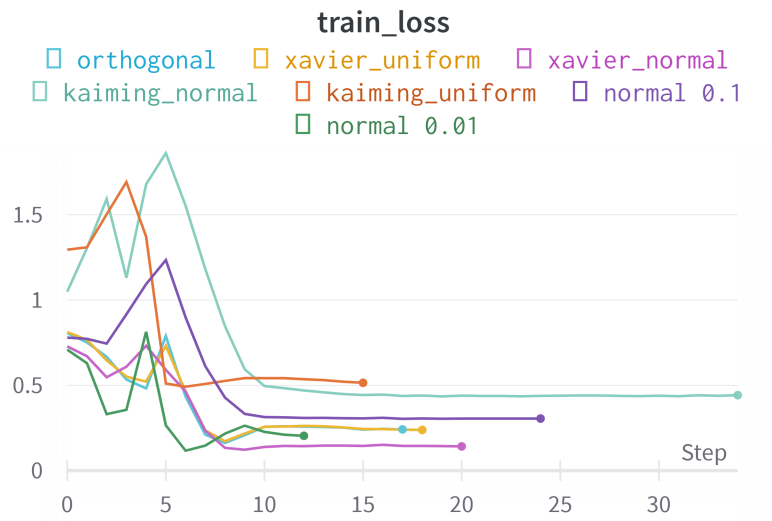


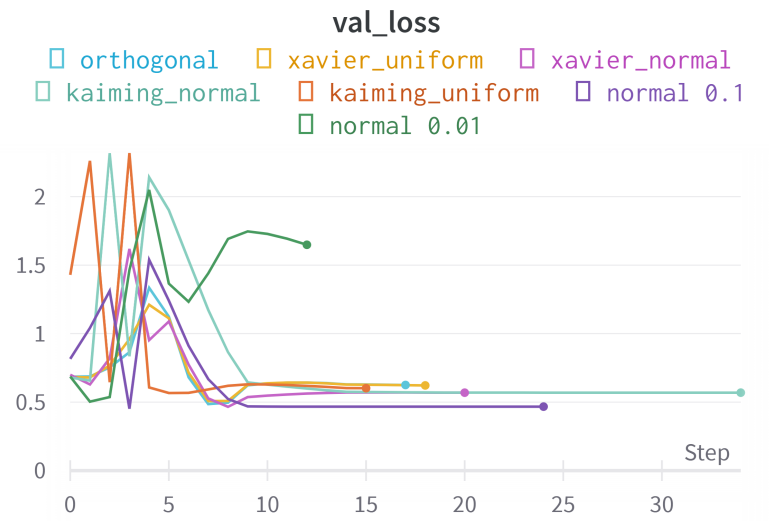






细节部分：





可以看出，初始化值不宜过大；在本实验中，使用各类正态初始化方法效果较好，收敛较快且验证集准确率较高，如标准差 0.01 的正态初始化。

三、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？

一些常见的防止过拟合的方法包括：

- **缩小神经网络的规模**：模型中的可学习的参数数量（由层数和每层节点数决定）越多，模型的记忆能力越强，也越容易学习到训练数据中的噪声或无关信息。因此，通过减少层数、每层节点数或 hidden size，可以限制模型的复杂性，防止过拟合。
- **权重正则化**：通过在损失函数中增加一个限制权重大小的成本项，可以使得模型中的权重值更小，更接近于零，从而减少模型的复杂性。常见的权重正则化技术有 L1 正则化和 L2 正则化。
- **Dropout**：在训练过程中，按照一定的概率随机地将一些神经元或连接断开，从而减少模型的复杂度和参数数量，防止模型过度依赖于某些特定的神经元或特征。
- **Early Stopping**：通过在每个 epoch 后检查验证损失，并在验证损失不再下降时停止训练，可以防止模型过度拟合训练数据。同时，还可以将权重重置为验证损失最小时的值，以保留最佳模型。

四、试分析CNN，RNN，全连接神经网络（MLP）三者的优缺点。

- **CNN**：卷积神经网络，主要由卷积层、池化层和全连接层组成。CNN主要用于处理图像数据，也可以处理文本、语音等数据。CNN具有以下优点：
  - **参数共享**：卷积层中的每个卷积核只对应一组参数，这些参数在整个输入特征图上共享，因此大大减少了参数量和计算量。
  - **局部感受野**：卷积层中的每个神经元只与上一层的一个局部区域相连，这样可以提取出输入数据中的局部特征，并保留空间信息。
  - **池化操作**：池化层可以对输入特征图进行降采样，从而进一步减少参数量和计算量，同时增加模型的鲁棒性和不变性。

    - **多层抽象**：CNN通过多个卷积层和池化层的叠加，可以逐层提取出输入数据中的低级到高级的抽象特征，并在最后的全连接层进行分类或回归。
    - **泛化能力强**：由于CNN具有较少的参数和较强的不变性，因此它的泛化能力比MLP强，更适合处理复杂的图像数据。

    但是，CNN也有以下缺点：

    - **对位置敏感**：CNN虽然具有平移不变性，但是对旋转、缩放等变换敏感，如果输入数据中存在这些变换，可能会影响模型的识别能力。为了解决这个问题，需要对输入数据进行数据增强，或者使用更复杂的网络结构。
    - **对长期依赖不敏感**：CNN虽然可以处理序列数据，如文本或语音，但是它对序列中的长期依赖关系不敏感，因为它的感受野是有限的，无法捕捉到距离较远的信息。为了解决这个问题，需要使用更深的网络结构或者结合RNN。
- **RNN**：循环神经网络，主要由循环层和全连接层组成。RNN主要用于处理序列数据，如文本、语音、视频等。RNN具有以下优点：
    - **动态长度**：RNN可以处理任意长度的序列数据，因为它在每个时间步都会接收一个输入，并产生一个输出和一个隐藏状态。隐藏状态可以保存上一个时间步的信息，并传递给下一个时间步。
    - **参数共享**：RNN在不同的时间步中使用相同的参数，因此可以减少参数量和计算量，并增加模型的泛化能力。
    - **记忆能力**：RNN通过隐藏状态可以记忆序列中的历史信息，并利用这些信息进行预测或生成。这样可以捕捉到序列中的依赖关系和上下文信息。
    - **灵活性**：RNN可以处理不同类型和结构的序列数据，如一对一、一对多、多对一、多对多等。例如，RNN可以用于文本分类（多对一）、机器翻译（多对多）、语音识别（多对一）、图像描述（一对多）等任务。

    但是，（原始）RNN 也有以下缺点：

    - **梯度消失或爆炸**：由于RNN在反向传播时需要沿着时间步展开计算梯度，因此当序列过长时，梯度可能会随着时间步的增加而指数级衰减或增长，导致梯度消失或爆炸的问题。这会影响模型的训练和收敛。
    - **无法捕捉长期依赖**：由于梯度消失的问题，RNN在处理较长的序列时，无法有效地将较早的信息传递到较后的时间步，因此无法捕捉到序列中的长期依赖关系。为了解决这个问题，需要使用更复杂的循环单元，如LSTM或GRU。
    - **计算效率低**：由于RNN在每个时间步都依赖于上一个时间步的输出和隐藏状态，因此无法并行化计算，导致计算效率低。为了解决这个问题，需要使用更高效的网络结构，如CNN或Transformer。
- **MLP**：多层感知机，也称为全连接神经网络或人工神经网络，是最基本的神经网络结构，由一个或多个全连接层组成。MLP 具有以下优点：
    - **模型先验较少，适用范围广**：MLP 可以处理任意形式的输入数据，只要将其展平为一维向量。
    - **具有较强拟合能力**：根据通用近似定理，MLP 具有拟合任意函数的能力，拟合能力非常强大。

    但是，MLP也有以下缺点：

    - **参数量过大**：由于每个神经元都与上一层和下一层的所有神经元相连，因此参数量随着层数和节点数呈指数增长，导致计算量大、训练慢、存储空间大等问题。
    - **无法利用空间或时间信息**：由于MLP将输入数据展平为一维向量，因此无法保留数据中的空间或时间信息，这对于图像、语音、文本等数据来说是不利的，因为这些数据中存在着重要的局部特征或序列关系[1]。
    - **容易过拟合**：由于MLP具有较强的拟合能力，如果模型过于复杂或数据过于稀疏，容易导致过拟合现象，即在训练集上表现很好，但在测试集上表现很差。为了防止过拟合，需要使用正则化、dropout等技术[1]。

心得体会

本次实验是我第一次独立完成一个深度学习项目，从数据处理、定义模型结构到训练、调参测试、撰写报告，这一过程使我受到了完整的项目训练，不仅增进了我对各类基本模型结构的理解，而且全方位提升了我应用 PyTorch 的工程能力，在完成之后颇有收获感和成就感。再次感谢马老师和助教！

附注：代码结构说明

代码和模型 checkpoint 已上传至 清华云盘 。

核心训练、测试代码在根目录下的 `main.py`；实验配置在 `config` 目录下， `config/config.py` 中可以方便配置各类实验参数，模型类型、batch size、learning rate、训练/测试模式等几类重要配置亦可通过命令行参数配置；数据处理和模型结构定义放在 `src` 目录下；实验结果部分相应的模型 checkpoint 放在 `ckpt` 目录下，其对应的配置文件都放在 `config` 目录下，文件中已配置模型加载相应 ckpt。