文本情感分类实验

梁业升 2019010547（计 03）

2022 年 5 月 12 日

本实验实现及原理参考了 `https://github.com/bentrevett/pytorch-sentiment-analysis`。

实验环境

系统：`macOS 12.3.1`
环境：`Python 3.9.10`

1 模型概述

1.1 RNN

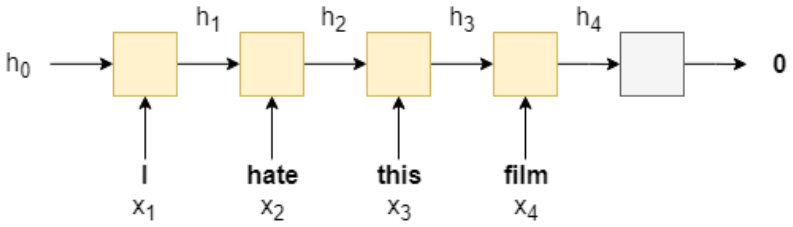


图 1: RNN

RNN 以 1 个词 x 和一个隐藏状态 h_0 作为输入，并产生下一个隐藏状态 h。我们对词序列 $X = \{x_1, \ldots, x_T\}$ 循环使用 RNN，将当前词 x_t 以及前一个隐藏状态 h_{t-1} 作为输入，产生隐藏状态 h_t，即

$$h_t = \mathrm{RNN}(x_t, h_{t-1}) \tag{1}$$

最后一个隐藏状态 h_t 经过一个全连接层后即可得到预测值。

RNN 有多个变种，如双向 RNN：

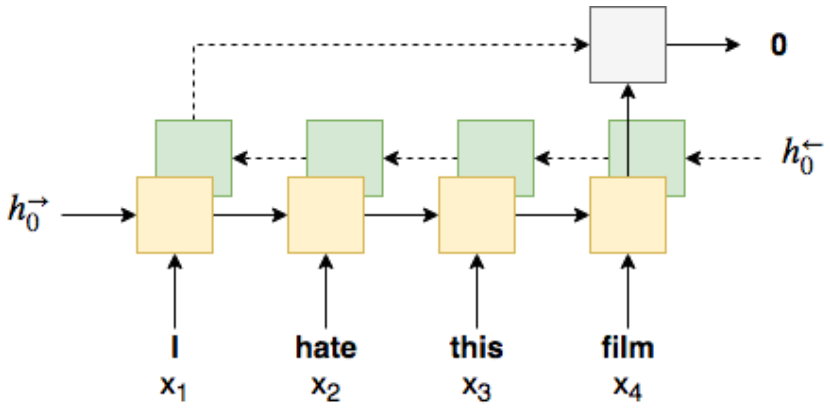


图 2: 双向 RNN

多层 RNN：

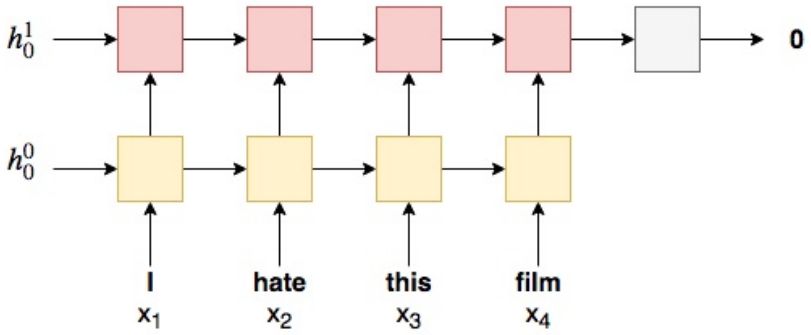


图 3: 多层 RNN

标准的 RNN 有梯度消失的问题。我们使用如下图所示的 LSTM 解决此问题：

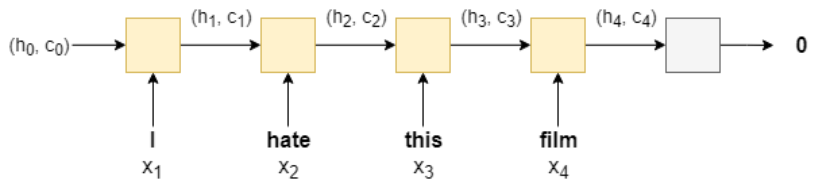


图 4: LSTM

与标准 RNN 相比，LSTM 多了一个额外的状态 c：

$$(h_t, c_t) = \text{LSTM}(x_t, h_{t-1}, c_{t-1}) \tag{2}$$

LSTM 使用多个门控制信息流入和流出 c，以达到“记忆”的功能。

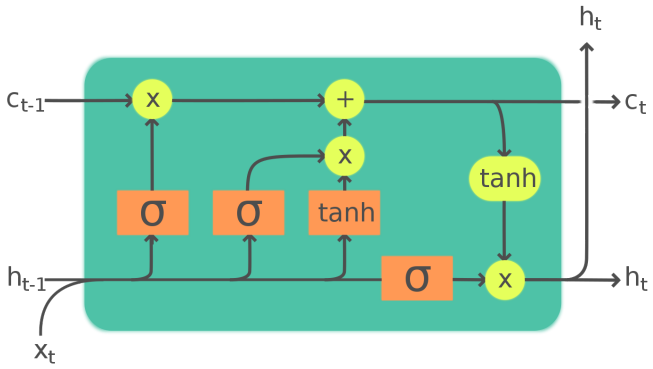


图 5: LSTM cell

1.2 CNN

将词向量纵向排列，得到一个二维的输入。

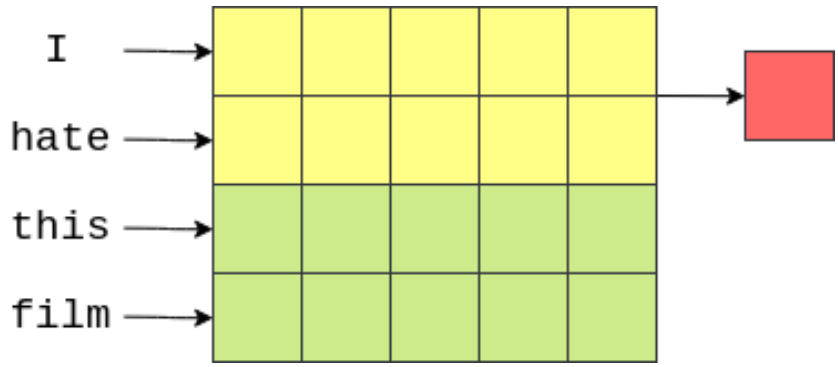


图 6: 卷积层

输入经过卷积层后得到一个一维或多维的向量。我们对其进行最大池化：

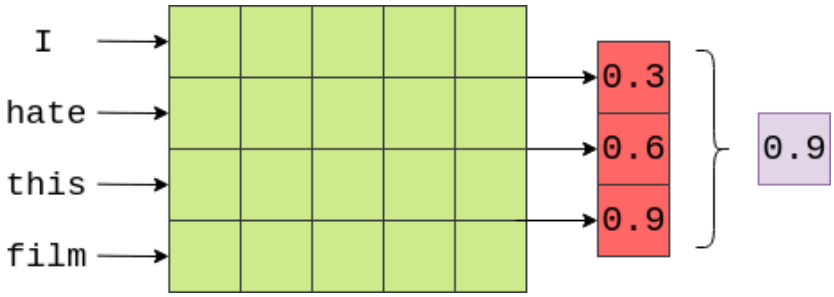


图 7: 最大池化

为了提取不同长度的词的组合的特征（如“非常好”和“非常”“好”），我们使用多个大小（如 1，2，3）的卷积核，并将进行最大池化的结果拼接在一起，通过全连接层产生输出。

1.3 模型优化

为缓解过拟合的问题，我们在模型的全连接层前加上 Dropout 层，即随机将一些神经元输出置 0。

2 实验结果

我们用 3 个标准来评价模型的效果：测试集上的准确率（Accuracy）、测试集上的 F1-Score，以及训练一个周期需要的时间。

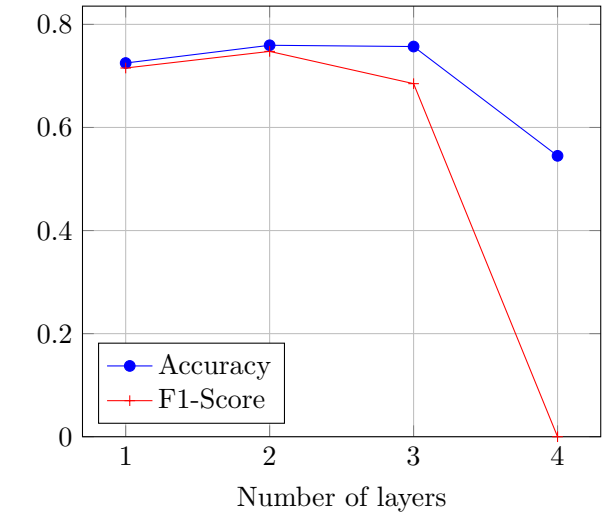
训练在 Google Colab 上进行。由于以下针对不同参数的实验并不在同一时间进行，而平台在不同时间的可用资源情况不同，因此不同实验对应的训练用时不具有可比性。

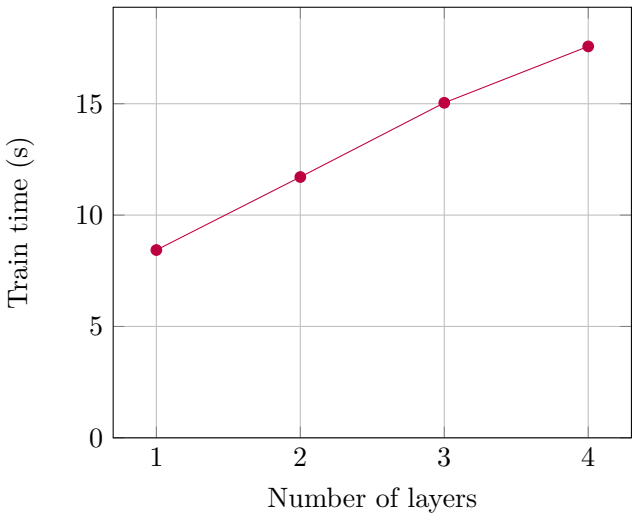
2.1 LSTM

我们考察 3 个超参数对模型效果的影响：层数 `layer_num`、Dropout 层的丢弃概率 `dropout_rate` 和一次处理的输入数量 `batch_size`。

2.1.1 LSTM 层数

固定 `dropout_rate = 0`，`batch_size = 64`。实验结果如下：

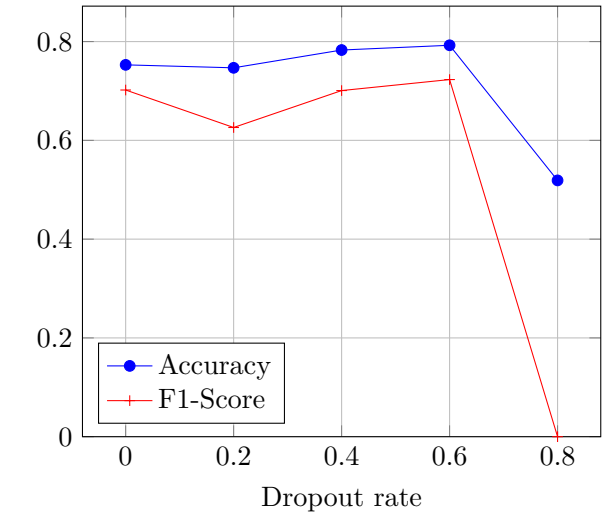


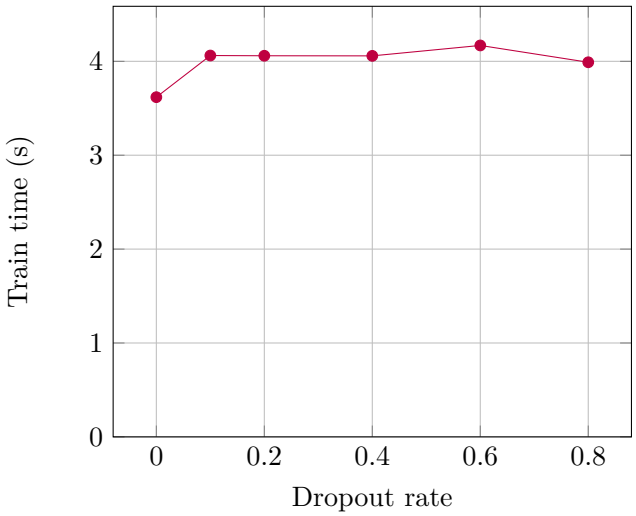


可见，双层 LSTM 比单层 LSTM 在效果上有略微的提高，但极其有限；当层数继续增加时，效果反而会有明显的下降；在层数为 4 时模型几乎完全无法训练，推测是出现了梯度消失的问题。同时，训练时间相对层数大约线性增长。因此，增加层数并不能起到明显的作用，反而会增加训练时间。

2.1.2 Dropout 概率

固定 `layer_num = 2`，`batch_size = 64`。实验结果如下：

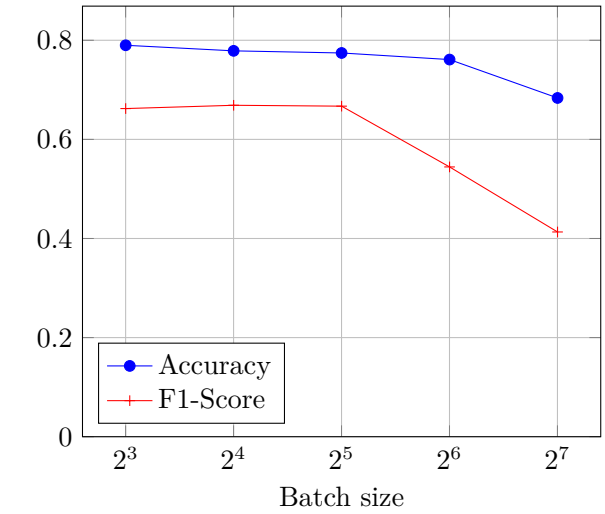


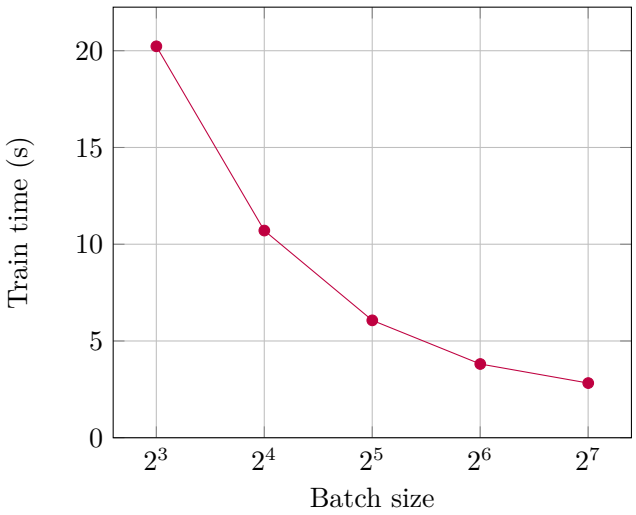


相比不使用 Dropout 层，当随机丢弃的概率为 0.2 时，模型效果略有下降；概率为 0.4、0.6 时，模型效果有所上升；概率为 0.8 时，模型几乎无法训练。实验结果表明，经过对参数进行一定比例的丢弃，可以在一定程度上缓解过拟合的问题。

2.1.3 Batch size

固定 `layer_num = 2`，`dropout_rate = 0`。实验结果如下：





模型效果总体上与单批处理的数量呈负相关，而训练时间随单批处理的数量呈负相关。这是容易理解的，因为更粗粒度的训练所保留的训练集的细节更少，因此准确率更低；同时因为其前向和反向传播的次数更少，因此用时更少。

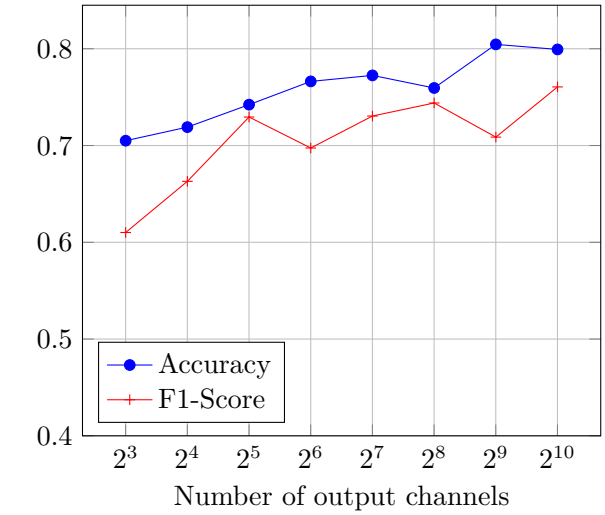
由实验结果不难得到，在实验的条件下，以 32 为单批处理的数量较为合适。

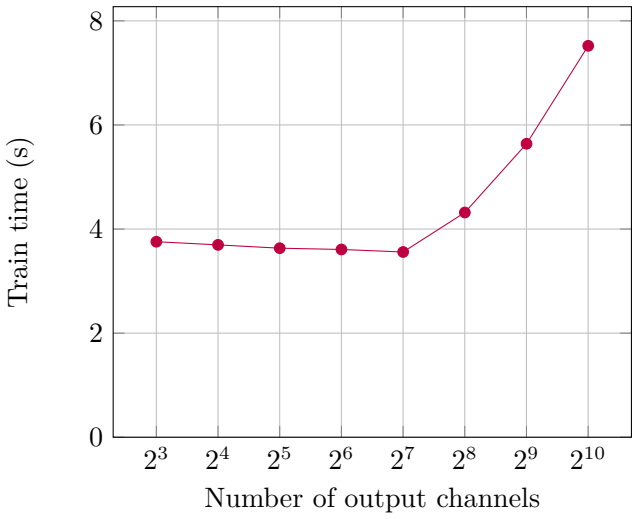
2.2 CNN

我们考察 3 个超参数对模型效果的影响：卷积层输出通道数 `channel_num`、Dropout 层的丢弃概率 `dropout_rate` 和一次处理的输入数量 `batch_size`。

2.2.1 输出通道数

固定 `dropout_rate = 0`，`batch_size = 64`。实验结果如下：

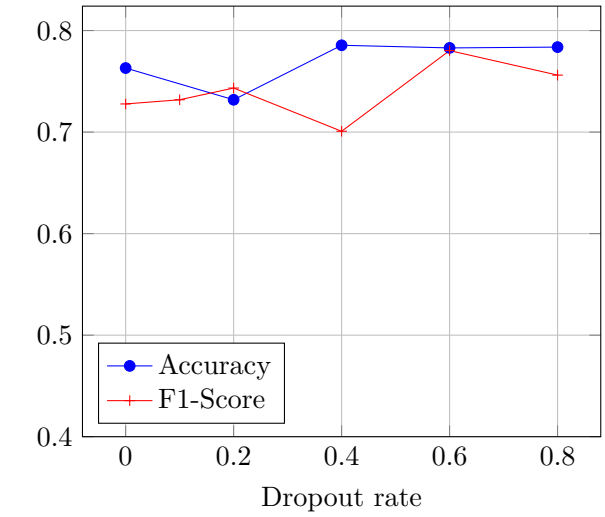


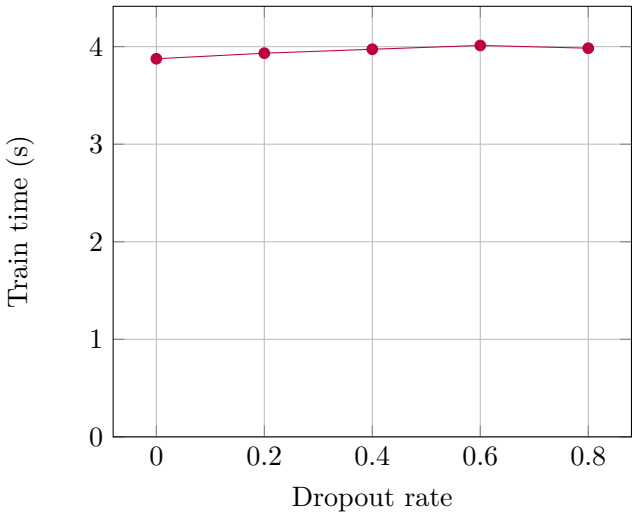


总体而言，在一定范围内，输出通道数越多，模型的效果越好。当然，更多的通道数带来的后果是更长的训练时间。

2.2.2 Dropout 概率

固定 `channel_num = 100`，`batch_size = 64`。实验结果如下：

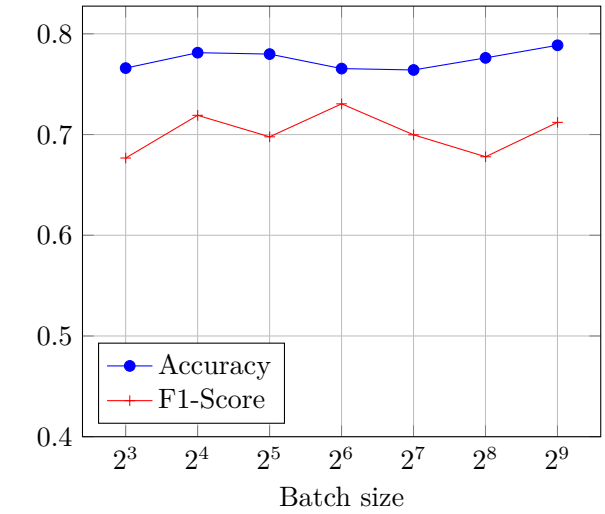


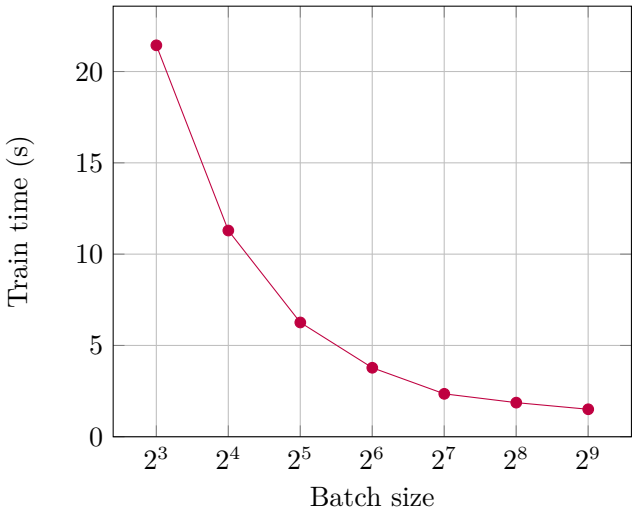


从结果来看，模型的效果和 dropout 的概率并无明显的关联。

2.2.3 Batch size

固定 `layer_num = 2`，`dropout_rate = 0`。实验结果如下：





从结果来看，模型的效果和分批处理的大小并无明显的关联。目前未想到较好的解释。

3 问题思考

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。

我的实现方式是：某个 epoch 验证集准确率相比之后连续 N 个 epoch 的准确率高时，停止训练。

固定迭代次数：便于准确找到最优训练模型，但所需的 epoch 数量不定，往往需要预设一个比较大的训练次数；然而对于收敛较早的模型较为浪费时间。

通过验证集调整：所需时间根据模型收敛速度动态调整，节省时间，但较难找到准确的衡量收敛状态的标准。

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值初始化，高斯分布初始化，正交初始化等）

实验时，我并不了解各种初始化方法，因此全部实验都使用 PyTorch 的默认初始化。

经查阅资料，各种初始化方法的特点如下：

- 零值初始化：由于初始化的值全都相同，每个神经元学到的东西也相同，将导致对称性问题
- 高斯分布初始化：有相同的偏差，权重有正有负
- 正交初始化：主要用于 RNN 网络，解决梯度消失、梯度爆炸问题

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。

在我的实现中使用了 Dropout 层，通过随机地抑制一些神经元减小神经元之间的耦合，以避免过拟合的问题。

试分析 CNN，RNN，全连接神经网络（MLP）三者的优缺点

- CNN
  - 优点：能提取局部信息，训练速度快
  - 缺点：输入大小固定
- RNN
  - 优点：有记忆效果，有时序关系，输入大小可变
  - 缺点：存在梯度消失和梯度爆炸问题
- RNN

- 优点：掌握全局信息
- 缺点：参数规模过大

4 心得体会

之前完全没有任何神经网络相关的编程经验，本次实验几乎从 0 开始学习了深度学习框架的使用方法，也巩固了课程的知识。

不过，课程给本实验提供的支持过少（没有必要的文档引导，甚至没有提供读取数据集的方法），导致体验并不太好。希望之后能够改善一下文档。