

Project1

文件夹结构

—— 20302010040-于康

- └—— report.pdf 实验文档
- └—— task1.py Part1-1代码
- └—— task1.pkl Part1-1模型
- └—— task2.py Part1-2代码
- └—— task2.pkl Part1-2模型
- └—— task3.py Part2代码
- └—— task3.pth Part2模型
- └—— train 数据集

一、反向传播算法

（一）代码基本结构与实验过程（结构参数比较）

本部分有回归和分类两个任务，二者均实现了反向传播算法，具有相似的代码结构。

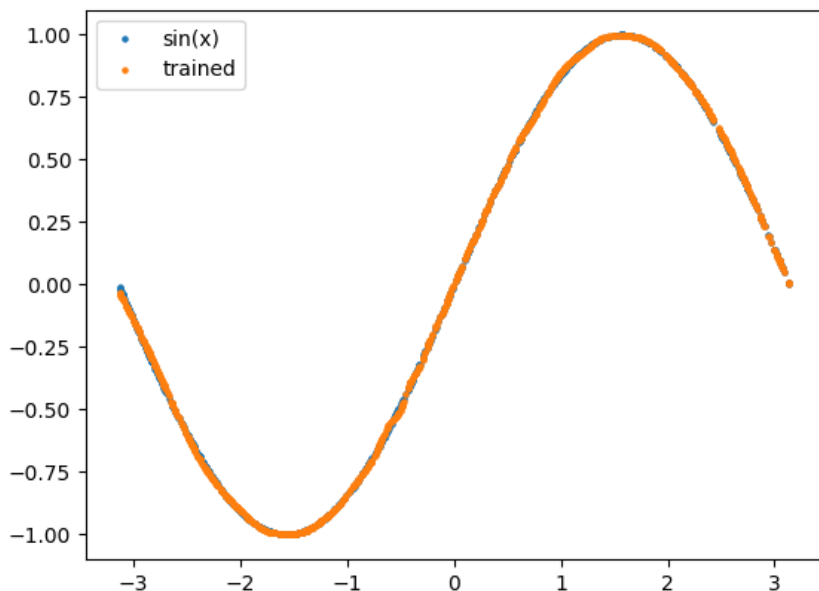
1.拟合函数 $y = \sin(x), x \in [-\pi, \pi]$

见文件`task1.py`

该文件实现了对于目标函数的拟合，通过测试计算了平均误差，满足了实验要求，并且利用`matplotlib.pyplot`分别画出 $\sin(x)$ 和拟合的图形。

```
yuki@MacBook pj1 % python3 task1.py
```

```
average error:0.004285910153285816
```

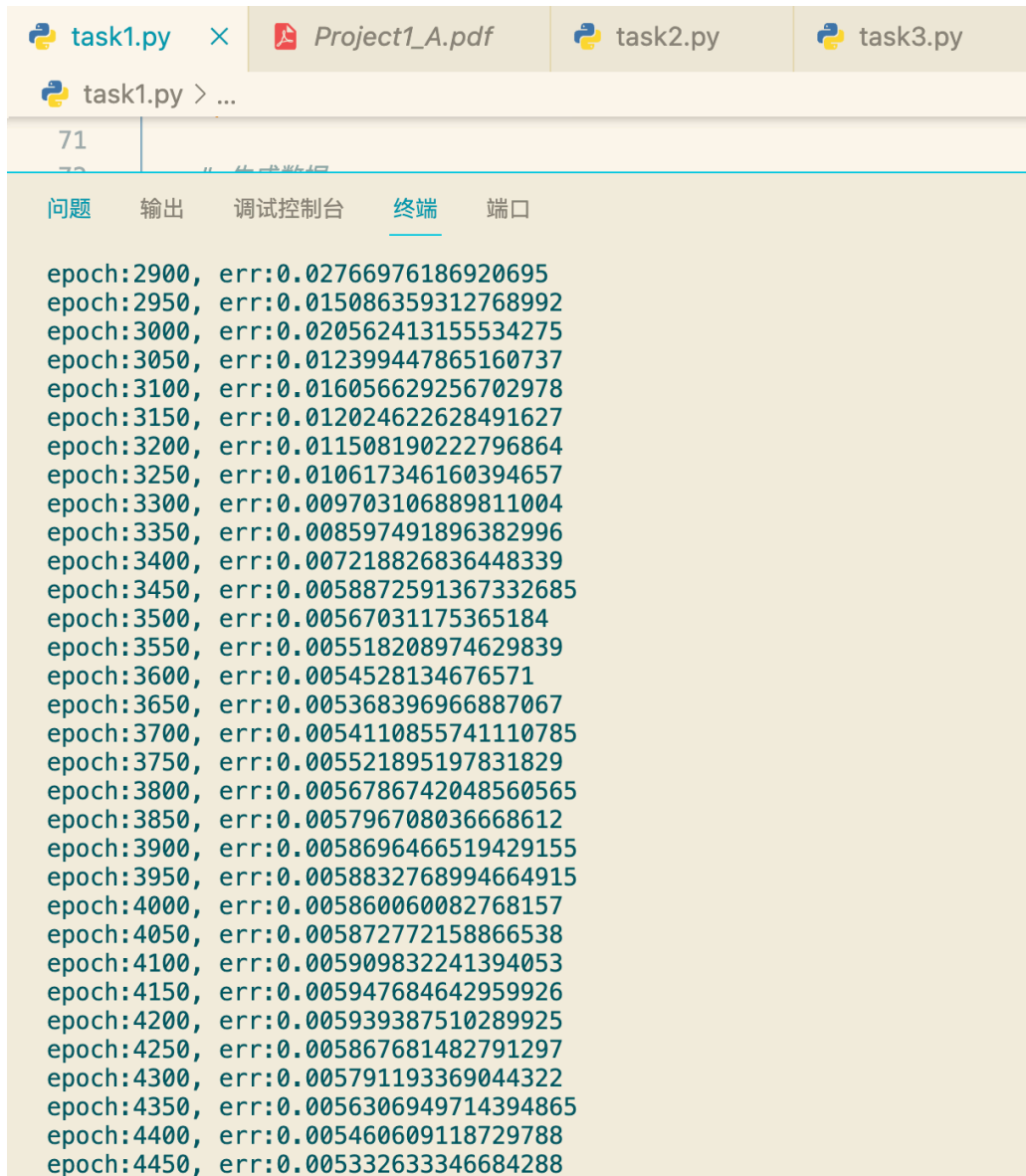


该文件的主要内容为类NeuralNetwork，该类利用反向传播算法实现，主要函数如下描述：

- `__init__`: 初始化，设置层数、神经元个数、学习率等，随机初始化权重和偏置
- `tanh`和`tanh_derivative`: 激活函数及其导函数
- `forward`: 前向传播函数，输入单个x值，得到其预测值
- `backward`: 反向传播函数，输入正确值，逐层调整每层的权重与偏置
- `train`: 训练函数，每个epoch都调用`forward`和`backward`函数，借助训练集来调整权重和偏置；同时，每50个epoch结束后，借助测试集来对当前状态的正确率进行测试并输出
- `predict`: 预测函数，对于输入的测试集，对于model当前状态，返回其正确率
- `save`: 保存当前model状态
- `load`: 将状态文件加载到model

在main函数中，首先对网络结构参数进行定义（可伸缩易调整的网络结构）；之后在 $-\pi$ 到 π 等间距生成数量为1000的训练集（X_train和Y_train），在 $-\pi$ 到 π 随机生成数量为1000的测试集（X_test和Y_test）；利用上述类构建model并训练，将结果保存在`task1.pkl`；之后输出model对于测试集的平均误差，并且画出 $\sin(x)$ 和拟合图形。

该回归任务较为简单，我选取的隐藏层为[50, 50]，学习率0.01，epoch大小为5000，从数据中可以看出，该参数配置下，在第3000多个epoch时，平均误差已经可以满足任务要求。



```
epoch:2900, err:0.02766976186920695
epoch:2950, err:0.015086359312768992
epoch:3000, err:0.020562413155534275
epoch:3050, err:0.012399447865160737
epoch:3100, err:0.016056629256702978
epoch:3150, err:0.012024622628491627
epoch:3200, err:0.011508190222796864
epoch:3250, err:0.010617346160394657
epoch:3300, err:0.009703106889811004
epoch:3350, err:0.008597491896382996
epoch:3400, err:0.007218826836448339
epoch:3450, err:0.0058872591367332685
epoch:3500, err:0.00567031175365184
epoch:3550, err:0.005518208974629839
epoch:3600, err:0.0054528134676571
epoch:3650, err:0.005368396966887067
epoch:3700, err:0.0054110855741110785
epoch:3750, err:0.005521895197831829
epoch:3800, err:0.0056786742048560565
epoch:3850, err:0.005796708036668612
epoch:3900, err:0.0058696466519429155
epoch:3950, err:0.0058832768994664915
epoch:4000, err:0.005860060082768157
epoch:4050, err:0.005872772158866538
epoch:4100, err:0.005909832241394053
epoch:4150, err:0.005947684642959926
epoch:4200, err:0.005939387510289925
epoch:4250, err:0.005867681482791297
epoch:4300, err:0.005791193369044322
epoch:4350, err:0.0056306949714394865
epoch:4400, err:0.005460609118729788
epoch:4450, err:0.005332633346684288
```

其他不同结构参数比较：

hidden layers	epoch_size	learning_rate	average error
[64, 32]	5000	0.01	0.012017223582916431
[32, 16]	5000	0.01	0.027362683570559706
[16, 8]	5000	0.01	0.007233931759085376

2.对12个手写汉字进行分类

见文件`task2.py`

该文件实现了对12个手写汉字进行分类，将给定的数据按照9:1的比例分成了训练集和测试集，在选定参数下，保存了模型的最优状态，对于训练集的识别率为86.5591%。

```
问题 输出 调试控制台 终端 端口
yuki@MacBook pj1 % python3 task2.py
acc:0.8655913978494624
```

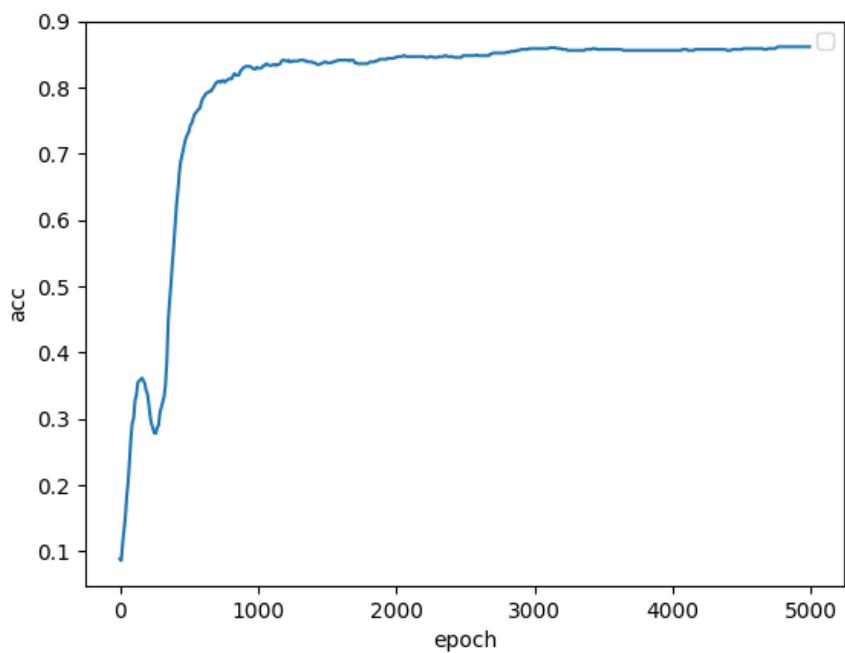
该文件由类NeuralNetwork，数据集导入函数load_dataset，主函数main组成，其中类NeuralNetwork与task1.py中的类似，不同之处在于：

- 激活函数使用sigmoid
- 每一次参数的更新所需要损失函数并不是由一个数据获得的，而是由batch_size大小的一组数据进行调整
- 训练过程中，画出预测准确率与epoch数的关系图，并将最大预测准确率的状态进行保存

对于数据集导入函数load_dataset，借助cv工具将bmp图片转换为灰度信息的矩阵，并拉平为一维向量进行归一化，标签集则为长度为12的一维向量，对应类别的值为1，其余为0。

在main函数中，首先对网络结构参数进行定义（可伸缩易调整的网络结构）；之后导入数据集并按比例分为训练集和测试集；利用上述类构建model并训练，画出预测准确率与epoch数的关系图，并将最大预测准确率的状态进行保存。

相比task1的回归任务，该任务我指定的网络参数较复杂，隐藏层为[128, 64]，学习率0.05，epoch大小为5000，batch大小为100。epoch-acc图示如下：



其他不同结构参数比较：

hidden layers	epoch_size	learning_rate	average error
[256, 128]	5000	0.05	0.8615591397849462
[128, 128]	5000	0.05	0.8629032258064516
[128, 64]	5000	0.05	0.8655913978494624
[64, 32]	5000	0.05	0.864247311827957

(二) 对反向传播算法的理解

反向传播算法通过计算网络预测与实际目标之间的误差，然后根据这个误差来更新神经网络中的权重，以逐渐优化网络的性能。实际训练中，由**前向传播**来逐层计算得到预测结果，之后由损失函数评估差值，计算梯度，**反向传播**逐层更新梯度和偏置，以达到使损失函数收敛到最小值。

反向传播算法中，网络结构中的各种参数的配置影响训练的结果。隐藏层的结构显示出网络的复杂性，相比之下，第二个分类任务需要更复杂的网络结构，需要的结点数目更多；另外，在分类任务中，起初使用了将学习率设置为0.005，epoch大小设置为1000，训练完发现准确率不到一半，调整网络结构发现也未起到效果，后面将学习率改为0.05，发现训练完后准确率达到80%以上，才发现是因为**学习率过低导致收敛速度慢**。

二、卷积神经网络

(一) 代码基本结构与实验过程（设计实验改进网络）

该文件用PyTorch实现了卷积神经网络（CNN）模型，用于对12个手写汉字进行分类。由继承于 `torch.nn.module` 的类 `My_CNN`，训练函数 `train`，预测函数 `predict`，数据导入函数 `load_dataset` 以及主函数 `main` 组成。

类 `My_CNN` 继承自 `torch.nn.module`，需要重写函数 `__init__` 和 `forward`:

- 第一个卷积层输入通道为1，输出通道为16，卷积核大小为5，步长为1，填充为2，实现 $(1, 28, 28) \rightarrow (16, 28, 28)$
- ReLu层
- 进行池化操作（2x2），实现 $(16, 28, 28) \rightarrow (16, 14, 14)$
- 第二个卷积层输入通道为16，输出通道为32，卷积核大小为5，步长为1，填充为2，实现 $(16, 14, 14) \rightarrow (32, 14, 14)$
- ReLu层
- 进行池化操作（2x2），实现 $(32, 14, 14) \rightarrow (32, 7, 7)$
- 全连接层，实现 $32 * 7 * 7 \rightarrow 12$

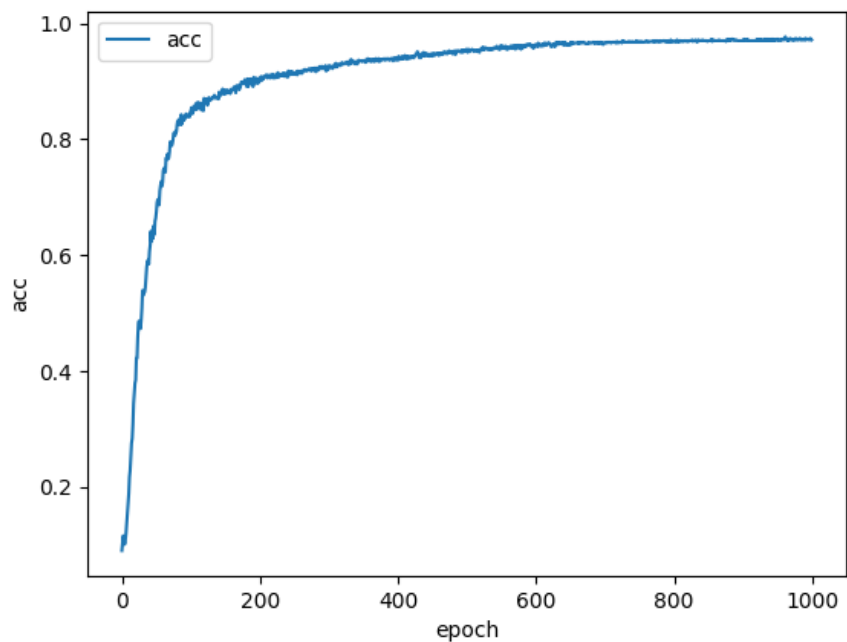
训练函数train整体思路与part1分类任务类似，由batch_size大小的一组数据对权重和偏置进行调整，利用训练集进行测试，画出预测准确率与epoch数的关系图，并将最大预测准确率的状态进行保存；不同之处在于训练过程利用了pytorch提供的损失函数与优化器。预测函数predict利用测试集计算预测结果准确率。导入函数与之前类似，但不需要拉平操作，而是将每个数据以(1, 28, 28)形式作为输入。主函数main导入了数据集，将其分为训练集和测试集，并封装到批处理的加载器中，之后创建了CNN模型，定义了SGD优化器和交叉熵损失函数，进行训练。

该分类程序对于测试集的准确率达到97.715%，相比part1中的网络有了显著提高。

问题 输出 调试控制台 终端 端口

```
● yuki@MacBook pj1 % python3 task3.py
acc:0.9771505376344086
```

训练过程中的epoch-acc图示如下：



（二）对网络设计的理解

卷积层利用卷积核在输入数据上滑动并执行卷积操作，用于捕捉输入数据的局部特征；池化层用于减小特征图的空间维度，同时保留了最显著的特征；全连接层将特征图展平成一个向量，并通过权重矩阵与每个神经元连接，生成最终的分类结果。

为了提高模型性能，可以尝试增加卷积层的数量以增加模型的深度；设置合理的卷积核大小，较小的卷积核通常用于捕获输入数据的细节和局部特征，而较大的卷积核用于检测更大的特征或全局特征；卷积层中的填充可以使边界信息利用充分，弥补边界信息缺失的问题。