浙江水利水电学院

实验报告

（ 2019-2020 学年 2 学期）

|  |  |
| --- | --- |
| 课 程 名 称： | 神经网络与深度学习 |
| 班 级： | 软件工程S19-2 |
| 学 号： | 2019b31064-068-073 |
| 姓 名： | 陈威-陈榆-徐文昊 |
| 实验室（中心）名 称： | 软件工程实验室 |
| 教学单位： | 信息工程与艺术设计学院 |

2020年 4 月 18 日

实 验 名 称：使用神经网络识别手写体数字 指导教师：张鹏亮

实 验 日 期：2020.4.7/9 地 点：现南411

同组学生姓名：2019b31064\_陈威、2019b31068\_陈榆、2019b31073\_徐文昊

|  |
| --- |
| 实验内容及要求：  **1.*实验内容：***  本次实验为使用卷积神经网络识别手写体数字。  本次任务分为两阶段。  第一阶段为任务一~任务四，必做任务。主要为定义卷积神经网络模型，加载数据，定义训练过程。  第二阶段为任务五~任务六，扩展阶段。主要为模型训练、实验数据记录和分析。  详细内容参见《实验一指导书》  **2.*实验要求：***  完成各阶段任务，完成各阶试验结果及其分析。  详细内容参见《实验一指导书》 |
| 主要仪器名称及型号：   1. 计算机：Windows，i5 3Ghz,4GB内存以上，或更高的配置 2. 开发工具：Python /Numpy/PyTorch 3. 开发环境：PyCharm/jupyter notebook |
| 实验过程：（可附页）  **实验项目工程结构：**    图 1-1 项目工程结构  其中各模块的说明如下：   |  |  | | --- | --- | | resources/ans/exp\_data | 文件夹，保存实验结果数据 | | resources/dataset | 文件夹，数据集 | | analysis.py | 实验结果数据分析 | | common.py | 任务1、2的实现 | | mnist.py | 实验相关的函数，给其他模块调用 | | module1.py – module6.py | 6种模型的训练 | | MyModule.py | 模型的定义 | | painter.py | 封装的plot绘图函数 |   **任务一代码设计：**   1. 载入数据集和测试集     图2-1 加载数据集流程图   1. 将标签向量转换为one-hot编码   设计：传入n\*1的标签向量y。预设一个n\*10的张量t。遍历每一个y[i]，将t中y[i]对应位置置为1。代码由徐文昊编写。流程如图2-2。    图2-2 转换one-hot编码流程图  **任务二代码设计：**  参考了电子教材3.5节展示FashionMNIST数据集的方式展示。因任务需要写出注解，所以下面由陈威给出具体实现的代码截图。见图2-3。    图2-3 展示数据集图片代码  **任务三代码设计：**  对于6种模型，将其抽象为两个类。分别表示单隐藏层模型与双隐藏层模型。通过传入的参数来实例化。具体实现由徐文昊编写代码。详情见类图2-4。    图2-4模型类图  其中，类Net2Layers为单隐藏层模型。参数说明如下*。*   |  |  |  | | --- | --- | --- | | hidden\_size | int | 隐藏层神经元个数 | | activation | str | 激活函数类型。  “sigmoid” or “relu”。 | | out\_activate | bool | 输出层是否使用激活函数。  True or False。 |   类Net3Layers为双隐藏层模型。参数说明如下*：*   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | hidden\_size | int[] | 数组，隐藏层神经元个数 |  |  |  | | activation | str | 激活函数类型。  “sigmoid” or “relu”。 |  |  |  | | out\_activate | bool | 输出层是否使用激活函数。  True or False。 |  |  |  |   **任务四代码设计：**   1. 根据不同的激活函数初始化权重及偏置   在设计神经网络模型类时，我们就考虑到了这个问题。我们将初始化权重及偏置逻辑封装在了模型的类中，通过传入不同的激活函数来初始化，隐藏了具体的实现细节，方便不同的组员调用。流程如图2-5。    图2-5 权重及偏置初始化流程图   1. 针对损失函数MSE转换标签形式   详见任务一代码设计第二小节。   1. 设计实验探寻最佳的超参数组合   定义不同的学习率、epoch、激活函数、输出层是否使用激活函数四个类型的参数数组，通过foreach循环穷举，定义模型并训练，保存实验数据以供分析。  任务流程详见图2-6。    图2-6 探索超参数流程图  **任务五代码设计：**  结合任务五与任务六。我组认为如果不统一记录数据的格式以及方式，会对后续的分析产生不利影响，很多封装的分析函数等将不能通用，也无法对数据进行很好的横向与纵向比较。所以经过我们一致讨论，将记录实验数据的代码封装在训练方法中，结果记录到一个字典返回，通过key-value键值对的形式存储。方便后续取得分析以及数据持久化。在后续训练不同模型时调用。下面列出需要记录的数据key。   |  |  | | --- | --- | | learning\_rate | 学习率 | | num\_epochs | 迭代周期 | | batch\_size | batch大小 | | loss\_x | 训练集当前迭代次数 | | loss\_y | 训练集对应loss值 | | loss\_y\_test | 测试集loss值 | | accuracy\_train | 训练集精度 | | accuracy\_test | 测试集精度 |   代码由徐文昊实现，关键代码见图2-7。    图2-7 记录数据代码实现 |
| 实验数据记录或图片：（可附页）  **任务一：**  对于子任务1，采用pytorch的API，并未遇到特别的问题。  对于子任务2，通过断点调试，分析data\_loader中的成员变量，发现有data.dataset.data与data.dataset.targets两个成员变量分别保存数据集及对应标签，取得之后转换one-hot即可。由徐文昊给出了代码实现，由于任务很基础，并未遇到问题。结果见图3-1-1。Debug过程发现的成员变量见图3-1-2。    图3-1-1 任务一结果图    图3-1-2 data\_loader中存储的训练集和测试集形式图  **任务二：**  参考了电子教材3.5节展示FashionMNIST数据集的方式展示。具体的实现及注解已由陈威在任务二代码设计中给出。下面直接给出结果，见图3-2。    图3-2 任务二结果图  **任务三：**  具体的设计已在任务三代码设计中给出。在抽象成类的过程中，我们遇到的问题有：  （1）如何自定义一个模型类。  （2）模型类中如何实现自定义的隐藏层。  （3）如何在自定义的模型类中初始化权重和偏置。  解决方法：   1. 通过查阅pytorch官方文档，发现可以继承torch.nn.Module类。在\_\_init\_\_构造函数中申明各个层的定义，在forward中实现层之间的连接关系，实际上就是前向传播的过程。 2. 继续查阅官方文档，发现可以通过torch.nn.Sequential类来实现。它是一个连续的容器，torch.nn.Module将按照它们在构造函数中传递的顺序添加。同时，激活函数也可以通过该类定义。 3. 查看torch.nn.Module的源代码之后，发现参数是全部保存在私有成员变量\_parameters中的。结合老师给的代码，可以通过遍历字典named\_parameters方式，根据不同的激活函数，利用nn.init模块完成初始化权重和偏置。   部分实现：    图3-3 模型的代码实现图  **任务四：**  本次训练完全应用了pytorch的自动求导及反向传播特性。具体的设计以及代码已由前文给出。遇到的问题有：   1. 如何设置权重衰减。   解决方式：  （1）我组采用Adam优化器。查阅文档得知，Adam优化器的实例化方法中，有个参数是weight\_decay，即权重衰减的值。我们采用了推荐的1e-8。  **任务五：**  本小组将数据的记录统一封装成了字典，具体格式已由前文给出。具体的代码实现由徐文昊编写。下面每个组员根据自己负责的模型给出部分实验数据的展示。  2019b31073\_徐文昊：  1.数据的代码的调试和解决：  在代码实现方面，因良好的编码水平，并未遇到需要调试解决的问题。  在实验需要记录的数据方面，由陈威提示，根据任务四寻找最佳的超参数epoch,需要记录测试集的loss，此key在结果字典中并未记录。通过补充训练方法在结果字典中添加了此key及对应的数据。  下面给出训练的60种参数组合之一的实验数据结果。  2.表格数据记录：   |  |  |  | | --- | --- | --- | | **key** | **value** | **备注** | | learning\_rate | 0.001 | 学习率 | | num\_epochs | 100 | epoch | | batch\_size | 5000 | batch\_size | | loss\_x | [0,1,2,…]，由下文可视化给出 | 迭代次数 | | loss\_y | […]，由下文可视化给出 | 对应loss | | loss\_y\_test | […] | 测试集对应loss | | accuracy\_train | 97.65241666666667 | 训练集精度 | | accuracy\_test | 97.9 | 测试集精度 | | duration | 598.3123314380646 | 训练时长(秒) |   3.数据可视化：    训练集上的损失函数变化趋势。x轴表示每个epoch内的迭代次数，y轴表示对应的loss值，因篇幅所限，此图为了仅仅展示了部分loss值(20个epoch)。见loss是以震荡的趋势逐渐减小。  2019b31064\_陈威：  1.数据的代码的调试和解决：  统一抽象成函数封装，调用即可。  2.表格数据记录：   |  |  |  | | --- | --- | --- | | **key** | **value** | **备注** | | learning\_rate | 0.001 | 学习率 | | num\_epochs | 100 | epoch | | batch\_size | 5000 | batch\_size | | loss\_x | [0,1,2,…1100] | 迭代次数 | | loss\_y | [0.3054652214050293…],具体参数根据下图 | 对应loss | | loss\_y\_test | […] | 测试集对应loss | | accuracy\_train | 96.92618333333333 | 训练集精度 | | accuracy\_test | 97.85 | 测试集精度 | | duration | 1114.10209441185 | 训练时长(秒) |   3.数据可视化：    x轴表示迭代次数，y轴表示对应的loss值。  根据图所示，随着迭代次数的增加，loss逐渐变小。在迭代次数较少向逐渐递增的趋势下，可以看到loss下降的趋势比较明显，随着迭代次数的增大，loss的趋势相对较缓。  2019b31068\_陈榆：  1.数据的代码的调试和解决：  调用写好的函数。  2.表格数据记录：   |  |  |  | | --- | --- | --- | | **key** | **value** | **备注** | | learning\_rate | 0.001 | 学习率 | | num\_epochs | 100 | epoch | | batch\_size | 5000 | batch\_size | | loss\_x | […] | 迭代次数 | | loss\_y | […] | 对应loss | | loss\_y\_test | […] | 测试集对应loss | | accuracy\_train | 98.63368333333334 | 训练集精度 | | accuracy\_test | 98.08 | 测试集精度 | | duration | 860.1268711090088 | 训练时长(秒) |   3.数据可视化： |
| 实验结论：  2019b31073\_徐文昊（负责解释任务六的子任务5~6）：  算法：通过穷举不同的超参数组合来进行训练，寻找最优的超参数。  优点：易于实现，易于理解，方便组员协作。能够在不太大的超参数组合范围内找到较为合适的组合。  缺点：训练时间长，训练次数多，需要算力高。对预设的超参数要求高，不能有太不合理的预设超参数在里面。  任务三：本次我负责模型2（2-layer NN, 1000隐藏层神经元，28\*28-1000-10）以及模型5（3-layer NN, 500+300隐藏层神经元，28\*28-500-300-10，权重衰减）。  特别的，对于模型5的权重衰减的实现已在上文说明。  任务四：  对于学习率和epoch，分析实验数据。首先看loss的趋势。训练集loss不断下降，测试集loss也不断下降，说明网络仍在学习；训练集loss不断下降，测试集趋于不变，说明网络可能过拟合; 训练集loss趋于不变，测试集loss趋于不变，说明学习遇到瓶颈，可能需要减小学习率或批量数目。  下面分别给出模型2和模型5表现较好的loss图像极其对应参数，分别为图4-1，图4-2。    图4-1 模型2的loss分析图    图4-2 模型5的loss分析图  两个模型的batch size都是5000，epoch都是200。其中模型2的学习率为0. 0001，模型5的学习率为0.001。  再看横向对比相同batch size与epoch的模型2，不同学习率的模型。其他条件相等，学习率为0.001时，模型过拟合了。如图4-3。  图4-3 学习率横向对比图  对应模型5，横向对比的参数组合表现也不是那么好。  可见，在所尝试的超参数组合中，对于模型2,1e-4的学习率，200的epoch，5000的batch size是比较好的一对组合。  对于模型5,1e-3的学习率则比较合适。  任务五：  本次实验，对于模型2，我总共尝试了36种超参数组合，总共花费训练时间60625秒，折合17小时。得到如下图4-4所示数据。    图4-4 模型2所得数据图  其中文件名格式为【学习率\_epoch\_激活函数\_输出层是否使用.txt】。  在测试集表现最好的模型的参数组合及精度如下表。   |  |  | | --- | --- | | 参数 | 值 | | 学习率 | 0. 0001 | | epoch | 200 | | batch size | 5000 | | 激活函数 | relu | | 输出层是否使用激活函数 | 是 | | 训练集精度 | 94.98806666666667 | | 测试集精度 | 98.08 | | 训练时长（秒） | 3641.3022739887238 |   对于模型5，由于时间关系，尝试了24种组合。总花费时间17415秒，折合5小时。如图4-5。    图4-5 模型5所得数据图  在测试集表现最好的模型的参数组合及精度如下表。   |  |  | | --- | --- | | 参数 | 值 | | 学习率 | 0.001 | | epoch | 200 | | batch size | 5000 | | 激活函数 | sigmoid | | 输出层是否使用激活函数 | 否 | | 训练集精度 | 97.3599 | | 测试集精度 | 98.12 | | 训练时长（秒） | 1015.8488721847534 |   任务六：   1. 对于使用均方误差损失函数的模型，输出层是否一定要使用激活函数？   说明：不一定。本小组对于每一个超参数组合，都训练并测试了是否使用激活函数两种情况。对于使用MSE损失函数的模型1到模型4，统计了4个模型输出层使用激活函数对测试集精度的影响图。见图4-6。    图4-6 输出层是否使用激活函数对测试集精度的影响图  可知，对于不同的超参数及模型，输出层是否使用激活函数的效果是不一样的。要具体问题具体分析。   1. 根据2层和3层网络的训练数据，分析模型的宽度和深度对最终结果的影响。   说明：针对每一组超参数，绘制出测试集精度图。    图4-7 6种不同参数组合精度图  图中同一列的模型参数相同。  通过比较同一宽度模型（如模型1、2、6或3、4、5），我们可以得出：有时候时候，模型的宽度加宽，精度会提高一些。但是在一些超参数组合下，并不是模型越宽效果越好。如参数组合7，反而倒过来了。  通过比较不同深度模型，我们发现不是隐藏层层数越多模型的表现越好的。  原因可能是：模型越复杂，往往包含冗余的附加条件越多，即那些越可能造成过拟合的因素越多，越容易过拟合。   1. 根据sigmoid和relu激活函数在同一结构模型中的表现，分析它们对最终结果的影响。   说明：统计同一组超参数使用不同激活函数的在测试集上的表现。得到图4-8。    图4-8 激活函数对测试集精度的影响  可见，在MNIST数据集中，给定的6种模型，激活函数使用relu，测试集精度更高的概率大一些。   1. 对比你们的模型的训练结果和已知该模型的错误率，如有较大差距，请分析问题在哪里。   说明：错误率对比如下表。   |  |  |  | | --- | --- | --- | | 模型 | 已知错误率 | 训练结果 | | 模型1(300) | 4.7% | 2.23% | | 模型2(1000) | 4.5% | 1.92% | | 模型3(300+100) | 3.05% | 2.15% | | 模型4(500+150) | 2.95% | 1.87% | | 模型5(500+300) | 1.53% | 1.90% | | 模型6(800) | 1.6% | 1.76% |   可见除了模型5与模型6，其他四个模型精度都要好过已知。对于模型5、6，可能是超参数不够合理，如学习率太大或太小，epoch太小等情况。查阅资料发现，MNIST数据集上，很多模型的精度已经能做到99%了。   1. 对于这些模型，你该如何进行选择，只是根据实验中在测试集上的精度或错误率吗？是否可以采用K折交叉验证？   说明：仅仅根据测试集上的精度或错误率选择是很片面的。应该综合训练时长、模型精度、算法复杂度、落地的可能性大小、是否方便排错等等因素综合选择。同时，测试集精度并不能完全表示一个模型的性能。在训练数据不多的情况下，单一的把数据都用来训练容易导致过拟合。此时可以考虑使用K折交叉验证来评估，降低模型方差。   1. 改进的方法。   说明：可以通过在一定范围内生成随机学习率，epoch等来进行随机训练。也可以通过分析当前训练中的数据，动态调整下一次训练超参数的方式（如保存待训练的数据到数据库或通过接口动态传输）进行剪枝。或通过搜索生成超参数的树，根据树的中序序列来进行训练。针对每一组超参数，可以多训练几次，取最好的一次的权重和偏置。  2019b31064\_陈威：任务三：神经网络模型的选择 本次实验我所负责的模型是模型1（2-layer NN, 300隐藏层神经元，28\*28-300-10）和模型3（3-layer NN, 300+100隐藏层神经元，28\*28-300-100-10）。任务四：模型训练过程设计  模型1的趋势图    模型3的趋势图  如图所示，可以看到随着迭代次数的增加，loss不断下降。如果训练集loss不断下降，测试集loss趋于平缓，则说明可能出现过拟合的情况。相反有可能出现欠拟合。  任务五：模型训练和数据记录 由于本人电脑配置的原因，电脑上没有GPU，采用的是CPU跑数据，所以所花费的时间较长。 对于模型1，因为选择的组合较多加上电脑配置较差，所以跑了很长的时间。模型1所花时间为20.5小时。  下图为所跑的数据文件：    经过数据的筛选及比较，挑选出了最佳状态的参数：  learning\_rate（学习率）:0.0001 num\_epochs（epoch）:300 batch\_size:5000 激活函数:relu 输出层使用激活函数 accuracy\_train（训练集精度）: 95.53987222222223 accuracy\_test（测试集精度）:98.25 duration（所需时间）: 11760.47096657753  对于模型3，由于之前跑的模型数据所花费时间较长，经过商讨我们减少了相应的参数。模型3所花时间为11.5个小时。下图为所跑的数据文件：  经过数据的筛选及比较，挑选出了最佳状态的参数： learning\_rate（学习率）:0.001 num\_epochs（epoch）:100 batch\_size:5000 激活函数:relu 输出层不使用激活函数 accuracy\_train（训练集精度）:97.54493333333333 accuracy\_test（测试集精度）:97.92 duration（所需时间）:3680.986303806305任务六：分析（1）对于使用均方误差损失函数的模型，输出层是否一定要使用激活函数，请用实验数据说明你的结论。  答：不一定。    （2）根据2层和3层网络的训练数据，分析模型的宽度（即单个隐藏层神经元的个数）和深度（隐藏层的个数）对最终结果的影响，并说明可能的原因。 答：对于模型的宽度，随着模型的宽度增加，精度会提高。模型深度的增加，模型表现的并不是越来越好。随着宽度和深度的增加，会导致训练的时间大大增加。  可能原因：模型越复杂，会遇到过拟合的情况。  （3）根据sigmoid和relu激活函数在同一结构模型中的表现，分析它们对最终结果的影响。  答：根据模型所跑出的数据可以得出，在同一结构模型中，使用激活函数relu，在测试集精度更高，所花费时间相比之下较少。下图为其中一组数据：      （4）对比你们的模型的训练结果和已知该模型的错误率，如有较大差距，请分析问题在哪里。  答：可能由于学习率没有选择正确，遇到了过拟合或欠拟合的情况。  2019b31068\_陈榆：  本次实验我所负责的模型为模型4（3-layer NN, 500+150隐藏层神经元，28\*28-500-150-10）和模型6（2-layer NN, 800隐藏层神经元，28\*28-800-10）。  模型4、模型6的数据文件图：      任务六：  （1） 对于使用均方误差损失函数的模型，输出层是否一定要使用激活函数，请用实验数据说明你的结论。  答：不可以。激活函数的主要作用是提供网络的非线性建模能力。如果没有激活函数，那么该网络仅能够表达线性映射，此时即便有再多的隐藏层，其整个网络跟单层神经网络也是等价的。因此也可以认为，只有加入了激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。    （2） 根据2层和3层网络的训练数据，分析模型的宽度（即单个隐藏层神经元的个数）和深度（隐藏层的个数）对最终结果的影响，并说明可能的原因。  答：宽度让每一层学习到更加丰富的特征，更深的模型，有更好的非线性表达能力，可以学习更加复杂的变换，从而可以拟合更加复杂的特征输入。  （3） 根据sigmoid和relu激活函数在同一结构模型中的表现，分析它们对最终结果的影响。  答：采用Sigmoid等函数，计算激活函数时包含指数运算，计算量大,反向传播求导误差梯度时，求导.涉及除法，计算量相对较大，而采用ReLU函数时，整个过程的计算量要节省很多。  对于深层网络, Sigmoid函数,反向传播时,很容易出现梯度消失的情况(在Sigmoid接近饱和区时，变换太缓慢，导数趋于0,这种情况会造成信息丢失，从而无法完成深层网络的训练)  ReLU会导致一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存的关系，缓解了过拟合的发生。  （4） 对比你们的模型的训练结果和已知该模型的错误率，如有较大差距，请分析问题在哪里。  答：学习率和epoch。 |
| 教师评语：（成绩部分学生请勿填写和更改）  总体情况：  完成情况：优  实验报告：优  拓展能力：选择一项。  组员1： 2019b31064 陈威 成绩：选择一项。  组员2： 2019b31068 陈榆 成绩：选择一项。  组员3： 2019b31073 徐文昊 成绩：选择一项。  批阅教师：张鹏亮  2020年4月25日 |