|  |  |
| --- | --- |
| 成绩 |  |

装

订

线

网络空间安全与计算机学院实验报告

实验课程名称**：神经网络与深度学习实验**

实验项目名称： 前馈神经网络（3）--自动梯度计算

学生姓名： 刘新媛

专 业： 人工智能

学 号： 20221205037

实验地点： C1-320

实验时间： 2024.10.23第7-8节

指导教师： 魏勇刚

预习报告部分

1. **实验目的：**使用预定义算子重新实现前馈神经网络
2. **实验原理：**

虽然我们能够通过模块化的方式比较好地对神经网络进行组装，但是每个模块的梯度计算过程仍然十分繁琐且容易出错。在深度学习框架中，已经封装了**自动梯度计算**的功能，我们只需要聚焦模型架构，不再需要耗费精力进行计算梯度。

1. **实验内容：**

[点击查看作业链接](https://blog.csdn.net/qq_38975453/article/details/126772521)

1.利用预定义算子重新实现前馈神经网络

飞桨提供了paddle.nn.Layer类，来方便快速的实现自己的层和模型。模型和层都可以基于paddle.nn.Layer扩充实现，模型只是一种特殊的层。同样在PyTorch中，模型和层也可以通过继承**torch.nn.Module**类来实现。

（1）使用pytorch的预定义算子来重新实现二分类任务。

（2）完善Runner类

（3）模型训练

（4）性能评价

2. 增加一个3个神经元的隐藏层，再次实现二分类，并与1做对比。

3. 自定义隐藏层层数和每个隐藏层中的神经元个数，尝试找到最优超参数完成二分类。可以适当修改数据集，便于探索超参数。

1. **实验所用设备：**pycharm、pytorch
2. **实验步骤：**1.数据预处理2.模型构建3.模型训练4.模型评价5.模型预测

实验报告部分

1. **实验步骤：**

1.利用预定义算子重新实现前馈神经网络

飞桨提供了paddle.nn.Layer类，来方便快速的实现自己的层和模型。模型和层都可以基于paddle.nn.Layer扩充实现，模型只是一种特殊的层。同样在PyTorch中，模型和层也可以通过继承**torch.nn.Module**类来实现。

**（1）使用pytorch的预定义算子来重新实现二分类任务，网络框架如下：**

|  |
| --- |
| **两层感知机（MLP）模型 - Model\_MLP\_L2\_V2** |
| 输入：输入数据集，每个样本的特征数量为 input\_size  输出：模型的预测输出，每个样本的输出特征数量为 output\_size  1. 初始化模型参数：  创建一个输入层到隐藏层的全连接层 fc1，其权重和偏置分别为 weight\_fc1 和 bias\_fc1。  创建一个隐藏层到输出层的全连接层 fc2，其权重和偏置分别为 weight\_fc2 和 bias\_fc2。  初始化权重：weight\_fc1 和 weight\_fc2 从均值为0，标准差为1的正态分布中随机初始化。  初始化偏置：bias\_fc1 和 bias\_fc2 被初始化为0。  设置激活函数为 Sigmoid 函数。  2. 定义前向传播过程：  对于每个输入样本 inputs：  a. 计算第一个全连接层的输出 z1 = fc1(inputs)。  b. 应用 Sigmoid 激活函数得到 a1 = sigmoid(z1)。  c. 计算第二个全连接层的输出 z2 = fc2(a1)。  d. 应用 Sigmoid 激活函数得到最终输出 a2 = sigmoid(z2)。 |

**（2）完善Runner类**

RunnerV2\_2 的训练器类，管理网络模型的训练、评估、测试和保存加载过程。  基于上一节实现的 RunnerV2\_1类，本节的 RunnerV2\_2 类在训练过程中使用**自动梯度计算**；模型保存时，使用**state\_dict方法（以字典的形式保存）**获取模型参数；模型加载用**set\_state\_dict方法**加载模型参数。只有具有可学习参数的层（如卷积层、线性层等）才有state\_dict。

**（3）模型训练**

|  |
| --- |
| **模型训练过程：** |
| 输入：  X\_train, y\_train：训练数据集及其标签  X\_dev, y\_dev：验证数据集及其标签  epoch\_num：训练轮数  model\_saved\_dir：模型保存目录  input\_size, hidden\_size, output\_size：网络参数  learning\_rate：优化器的学习率  输出：  训练好的**模型参数**，保存在 model\_saved\_dir 指定的路径  1. 初始化模型参数：  创建一个MLP模型实例 model，输入层维度为 input\_size，隐藏层维度为 hidden\_size，输出层维度为 output\_size。   1. 设置损失函数：   使用二元交叉熵损失函数 loss\_fn = F.binary\_cross\_entropy 作为模型的损失函数。  3. 设置优化器：  创建一个随机梯度下降（SGD）优化器 optimizer  4. 设置评价指标：  使用accuracy 函数作为评价模型性能的指标 metric。  5. 实例化训练器：  创建一个 RunnerV2\_2 类的实例 runner，传入模型、优化器、评价指标和损失函数。  6. 训练模型：  使用 runner 实例的 train 方法开始训练模型，传入训练集和验证集的数据，设置训练轮数 epoch\_num、日志打印频率 log\_epochs 和模型保存路径 model\_saved\_dir。 |

**（4）性能评价**

使用测试数据对训练完成后的最优模型进行评价，观察模型在测试集上的准确率以及loss情况：

2. 增加一个3个神经元的隐藏层，再次实现二分类，并与1做对比。

3. 自定义隐藏层层数和每个隐藏层中的神经元个数，尝试找到最优超参数完成二分类。可以适当修改数据集，便于探索超参数。

**二、 实验数据及结果分析：**

[点击查看代码运行结果及分析](https://blog.csdn.net/qq_73704268/article/details/143191556?sharetype=blogdetail&sharerId=143191556&sharerefer=PC&sharesource=qq_73704268&spm=1011.2480.3001.8118)

1. **实验结论：**
2. 实验通过增加隐藏层神经元数目和自定义隐藏层层数，探索了不同超参数对模型性能的影响。结果表明，合理的超参数设置可以显著提升模型的准确率。

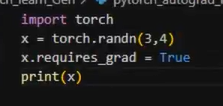
2.使用PyTorch框架中的自动梯度计算功能可以有效地简化神经网络训练过程中的梯度计算。

3.经过优化的模型在测试集上达到了较高的准确率，证明了模型的有效性和泛化能力。

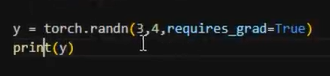
1. **总结及心得体会：**
2. 这次实验让我感受到pytorch 一个超大优势：可以**自动**实现反向梯度计算过程。这里我深入探讨了它的原理主要是由于**autograd**包为张量上的所有操作提供了自动[求导](https://so.csdn.net/so/search?q=%E6%B1%82%E5%AF%BC&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_42251120/article/details/_blank)机制。

但是张量本身不能自动求dao，若要想让其可以求导：

方法一：设置requires



方法二：



方法三：使用函数from torch.autograd import Variable 可以进行自动求导

Variable(torch.from\_numpy(x\_trian))

设置张量的属性 .requires\_grad 为 True，那么它将会追踪对于该张量的所有操作。当完成计算后可以通过调用 .backward()，来自动计算所有的梯度。这个张量的所有梯度将会自动累加到.grad属性。讲到这里我回顾了一下整个实验中好像我都没对参数设置过 .requires\_grad,经查阅资料，是因为Model\_MLP\_L2\_V2 继承自 nn.Module，PyTorch ，模型的参数（如 nn.Linear 层的权重和偏置）在模型创建时会自动设置为需要计算梯度的变量，即它们默认具requires\_grad=True 的属性！！！【涨芝士啦~】

2.实验中我对super()函数的用法也很陌生，之前好像从未见过， 搜了一下也不难，主要用于访问当前类的父类（也叫基类）的方法。在面向对象编程中，如果一个类继承了另一个类（即子类继承父类），子类可以通过 super() 调用父类的方法。model.parameters() 返回模型中所有需要更新的参数；调用 optimizer.step() 时，优化器会根据计算的 梯度信息来更新模型的参数。

3.实验中调用了优化器SGD，实现了随机梯度下降算法，之前也是手动实现算法的。主要作用是根据损失函数的梯度来更新模型的参数，使损失最小化。随机梯度下降的公式为：

表示模型的参数，β表示学习率， 是损失函数关于的梯度（在反向传播计算得到），可见和我们手动实现是算法功能是一样的，编写自定义梯度又容易出错，直接调用pytorch封装好的这些器件真的很方便。

4.这次实验掌握了使用 torch.nn.Module 来构建自己的神经网络，体会到了深度学习的含义，搭建大致过程如下：

|  |
| --- |
| **使用 torch.nn.Module 来构建神经网络：** |
| * **定义网络结构：**继承 torch.nn.Module 类，并定义自己的网络层作为类的属性。   class MyLayer(nn.Module):   * **初始化网络层：**在类的 \_\_init\_\_ 方法中初始化网络层。   super(MyLayer, self).\_\_init\_\_()  ................   * **前向传播：**定义 forward 方法来指定数据通过网络的方式。   def forward(self, x):   * **实例化和使用模型：**创建类的实例，并通过调用 forward 方法来使用模型。   model = MyModel()  output = model(input\_data) |

1. 上次实验的代码很清晰，所以这次很快就把代码改好了，大多数时间用在了调参上面，该说不说，超参数的调整真的还蛮麻烦的，为了防止盲目的调参，再跳参数之前我先学习了一部分调参的技巧（对应也写在了博客里），最终的学习score达到了0.99，与上个实验对比来说，还是比较满意的。每次写实验报告总是能学到很多知识点（在理论课中通常学不到的），海量的资料看起来有时候也会头晕，这里就要感谢学长的博客里推荐的链接（毕竟也是学长层层筛选出的优质博客，真的很有帮助）

|  |
| --- |
| **参考链接：** |
| [PyTorch基础知识——自动求导](https://blog.csdn.net/qq_42251120/article/details/126837608?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217A1C1D4-E4C2-4E4D-8C8F-C6CD00C7C394%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=17A1C1D4-E4C2-4E4D-8C8F-C6CD0)  [SGD原理 & pytorch代码解析](https://blog.csdn.net/Lizhi_Tech/article/details/131683183?ops_request_misc=&request_id=&biz_id=102&utm_term=SGD%E4%BC%98%E5%8C%96%E5%99%A8&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-131683183.142^v100^pc_search_result)  [深度学习笔记:详解优化器之随机梯度下降（SGD）](https://blog.csdn.net/weixin_53765658/article/details/136668114?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522D10EEC92-E061-45D0-8E02-0198096D3458%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=D10EEC92-E061-45D0-8E02-0)  [yTorch学习笔记：使用state\_dict来保存和加载模型](https://blog.csdn.net/DeliaPu/article/details/124666157?ops_request_misc=%257B%2522request%255Fid%2522%253A%252281648083-3524-406F-B4E5-01E0BC4FF0DC%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=81648083-3524-406F-B4E5-01E0BC4FF)  [pytorch 中的重要模块化接口nn.Module - 今夜无风 - 博客园](https://www.cnblogs.com/demo-deng/p/10621904.html)  [pytroch 掌握深度模型构建精髓 - 今夜无风 - 博客园](https://www.cnblogs.com/demo-deng/p/12354158.html) |