|  |
| --- |
| 电子科技大学  **计算机专业类课程** |
| **实验报告** |
| **课程名称：人工智能综合实验II**  **学院专业：计算机科学与工程学院**  **学生姓名：崔若晨**  **学　　号：2021060902008**  **指导教师：石小爽** |
| **日　　期： 2023 年 9 月 5 日** |
|  |
| 电子科技大学计算机学院实验中心 |

**电 子 科 技 大 学**

**实 验 报 告**

**实验一**

# 一、实验室名称：

电子科技大学清水河校区主楼A2-413-1

# 二、实验项目名称：环境熟悉及深度学习初探

# 三、实验原理：

1.线性回归的基本理论，线性回归模型是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。

2．softmax回归的基本理论，softmax回归模型是线性回归模型在多分类问题上的推广，在多分类问题中，类标签y可以取两个以上的值。

3.多层感知器的基本理论，多层感知器模型是在线性回归或者softmax回归的基础上添加多层的操作，并有层与层之间有激活函数。

4. 对比分析在深度学习领域是使用不同参数、模型结构等选择出最优模型，达到分析比较的目的。可视化分析是人工操作将数据进行关联分析，并做出完整的分析图表。

# 四、实验目的：

1） 学习基本的Pytorch的基本数据操作

2） 深度学习模型搭建的基本流程

3） 利用Pytorch实现线性回归、Softmax及多层感知器回归模型

4) 掌握实验结果的分析能力，包括两个方面：对比分析与可视化分析

# 五、实验内容：

1. 实验数据

描述数据集的基本情况

给出训练集、验证集以及测试集的划分情况

2. 结果分析

给出不同参数下的模型正确率，选择出最优参数的模型。以表格的形式展示，并给出文字的实验分析，如（下面的至少选择4项以上进行展示）

1. 采用不同批训练的大小（batch\_size）
2. 迭代次数的不同（epoch）
3. 采用不同的优化算法
4. 采用避免过拟合的方式（dropout、正则项等）
5. 隐藏层的维度大小
6. 不同的学习率（learning\_rate）
7. 网络深度的大小

给出网络训练过程的变化图，以折线方式显示，并给出文字的实验分析，如

1. loss变化图
2. 准确率变化图

# 六、实验器材（设备、元器件）：

硬件平台：PC端

开发环境：操作系统Windows11

测试环境： Pytorch

# 七、实验步骤：

1. 问题描述

通过softmax回归模型和多层感知机模型实现对FashionMNIST数据集的分类

2. 算法的概要设计与分析

多层感知机（MLP）算法：将FashionMNIST数据集的图片以Tensor形式输入神经元。

线性回归算法：利用线性回归方程的最小平方函数来计算预测值与真实值的差（loss），其公式为：

Softmax算法设计：计算MLP输出层loss之后向量的每一个分量的占比，公式为，结果最大的分量对应的标签为网络预测结果。

交叉熵：在实际使用中大部分事物都不是均匀分布的，这时计算信息量就可以使用交叉熵，它是在非均匀分布下信息量的一种表述，其公式为

3. 核心算法的详细设计与实现

1. 核心算法设计：

多层感知机（MLP）算法：MLP的工作原理如图 1所示，通过公式和激活函数，计算出各个样本的估计值。其中为权重，表示神经元之间的连接强度，为偏置。常见的激活函数有Sigmoid、tanh、ReLU等，可将神经元的输出幅度限制在一定范围内。

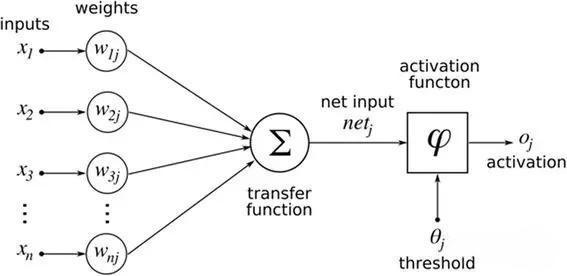


图 1 MLP原理图

最典型的MLP包括包括三层（如图 2）：输入层、隐藏层和输出层，MLP神经网络不同层之间是全连接的，即上一层的任何一个神经元与下一层的所有神经元都有连接。

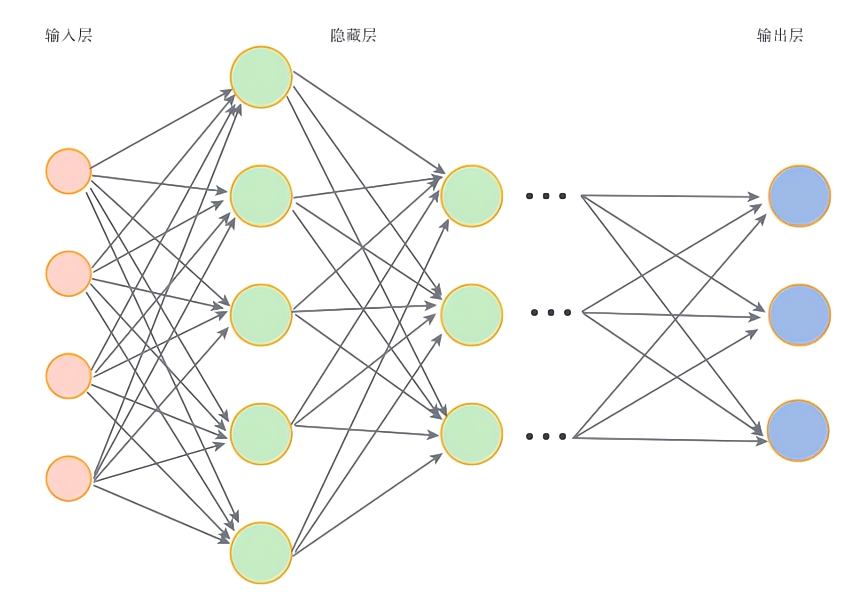


图 2 MLP结构图

MLP的训练分为正向传播和反向传播两种。正向传播FP主要是为了求损失。在这个过程中，我们根据输入的样本，给定的初始化权重值W和偏置项的值b，计算最终输出值以及输出值与实际值之间的损失值。如果损失值不在给定的范围内则进行反向传播的过程，否则停止和的更新。反向传播BP主要是通过回传误差来更新网络的权重值W和偏置b。将正向传播计算的误差通过隐层向输入层逐层反传，计算每一层W和b对loss的梯度，通过梯度下降法更新W和b使损失减少。然后再次进行正向传播，如此循环，直到满足要求。如图 3所示。

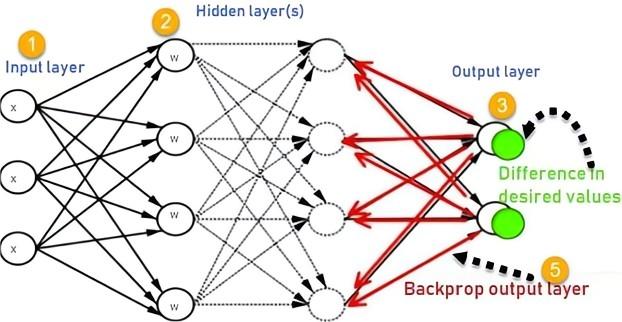


图 3 MLP的工作流程

1. 网络框架结构搭建介绍：

网络框架如图 4所示，使用单层感知机模型，输入层维度为784（），输出层维度为10。

# 

图 4网络框架

1. 核心算法流程：

|  |
| --- |
| **输入：**训练集FashionMNIST  感知机网络net  Adam优化器optim  CrossEntropy损失函数loss\_fun  归一化函数Softmax函数  正确率计算函数evaluate\_accuracy  **过程：**   1. 将数据集传入DataLoader并设置batchsize 2. **for** epoch 1 to 20 3. 从DataLoader中取出img,target 4. 将img传入网络net得到output 5. lossloss\_fun(output,target) 6. accuracyevaluate\_accuracy(output,target) 7. BP算法更新net的权重w和偏置b 8. **end for**   **输出：**loss,accuracy,折线图 |

1. 训练执行流程：

依据单一变量原则和控制变量法。每次训练在最初模型上做一个参数的改动。每次训练都计算训练集上的损失，训练集上的正确率，测试集上的正确率并做出相应的折线图。

1. 核心代码展示：

|  |
| --- |
| 核心代码 |
| from re import L  num\_inputs = 784  num\_hidden\_layer = 256  num\_outputs = 10  *#定义网络框架*  class **LinearNet**(nn.Module):  def \_\_init\_\_(self, num\_inputs, num\_outputs, num\_hidden\_layer=None):  super(LinearNet, self).\_\_init\_\_()  self.net = nn.Sequential(  nn.Linear(num\_inputs, num\_hidden\_layer),  nn.Dropout(0.5),  nn.ReLU(inplace=True),  nn.Linear(num\_hidden\_layer,num\_outputs)  )  self.init\_weights()  def init\_weights(self):  nn.init.kaiming\_normal\_(  self.linear.weight, mode='fan\_in', nonlinearity='leaky\_relu') *# 对权重采用凯明初始化方法*  self.linear.bias.data.fill\_(0)  def forward(self, x): *# x shape: (batch, 1, 28, 28)*  x = x.view(-1, 784)  output = self.net(x)  return output  *#定义device*  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  *#输出初始化的参数*  linear\_net = LinearNet(num\_inputs, num\_outputs, num\_hidden\_layer)  linear\_net=linear\_net.to(device)  *#定义损失函数SoftMax和交叉熵函数*  loss = nn.MSELoss() *#nn.MSELoss是交叉熵函数和softmax函数的结合*  loss=loss.to(device)  *# 定义优化器*  import torch.optim  optimizer = torch.optim.Adam(linear\_net.parameters())  *# 定义评价函数*  def evaluate\_accuracy(data\_iter, net, device=None):  if device **is** None **and** isinstance(net, torch.nn.Module):  *# 如果没指定device就使用net的device*  device = list(net.parameters())[0].device  acc\_sum, n = 0.0, 0  with torch.no\_grad():  for X, y **in** data\_iter:  if isinstance(net, torch.nn.Module):  net.eval() *# 评估模式, 这会关闭dropout*  acc\_sum += (net(X.to(device)).argmax(dim=1) == y.to(device)).float().sum().cpu().item()  net.train() *# 改回训练模式*  else: *# 自定义的模型, 3.13节之后不会用到, 不考虑GPU*  if('is\_training' **in** net.\_\_code\_\_.co\_varnames): *# 如果有is\_training这个参数*  *# 将is\_training设置成False*  acc\_sum += (net(X, is\_training=False).argmax(dim=1) == y).float().sum().item()  else:  acc\_sum += (net(X).argmax(dim=1) == y).float().sum().item()  n += y.shape[0]  return acc\_sum / n  *# 模型训练过程*  num\_epochs = 20 *#训练轮数*  losses=[] *#收集损失*  test\_accuracy = [] *#收集测试集正确率*  train\_accuracy = [] *#收集训练集正确率*  for epoch **in** range(num\_epochs):  train\_l\_sum, train\_acc, test\_acc,n = 0.0, 0.0, 0.0,0  for X, y **in** train\_iter:  X=X.to(device)  y=y.to(device) *#将训练数据和标签都放在device上面*  output = linear\_net(X)  temp = torch.zeros(len(y),10)  temp =temp.to(device)  for i,label **in** enumerate(y):  temp[i,label]=1  y = temp *#对标签进行重构，满足[0,...,1,...0]的形式*  l = loss(output, y)  optimizer.zero\_grad() *#清除梯度*  l.backward()  optimizer.step() *#更新网络参数*  train\_l\_sum += l  losses.append(l.item())  n += 1  train\_acc = evaluate\_accuracy(train\_iter, linear\_net,"cuda")  test\_acc = evaluate\_accuracy(test\_iter, linear\_net,"cuda")  train\_accuracy.append(train\_acc)  test\_accuracy.append(test\_acc)  print('epoch **%d**, loss **%.4f**, train acc **%.3f**, test acc **%.3f**'  % (epoch + 1, train\_l\_sum / n, train\_acc , test\_acc))  *# 训练过程可视化*  plt.plot(losses)  plt.title("training losses")  plt.savefig("./loss.png") *#保存loss图片*  plt.show()  plt.subplot(121)*#创建子图1*  plt.plot(train\_accuracy)  plt.title("training accuracy")  plt.xlabel("epoch")  plt.subplot(122) *# 创建子图2*  plt.plot(test\_accuracy)  plt.title("test accuracy")  plt.xlabel("epoch")  ax=plt.gca()  ax.yaxis.tick\_right()  plt.savefig("./accuracy.png") |

# 八、实验数据及结果分析：

1. **实验数据：**

Fashion-MNIST是一个替代[MNIST手写数字集](http://yann.lecun.com/exdb/mnist/)的图像[数据集](https://so.csdn.net/so/search?q=%E6%95%B0%E6%8D%AE%E9%9B%86&spm=1001.2101.3001.7020)。它是由Zalando（一家德国的时尚科技公司）旗下的[研究部门](https://research.zalando.com/)提供。其涵盖了来自10种类别的共7万个不同商品的正面图片。Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。60000/10000的训练测试数据划分（即训练集有60000张图片，测试集有10000张图片），图片尺寸为2828。图 5展示了数据集对应10个类别可视化的结果

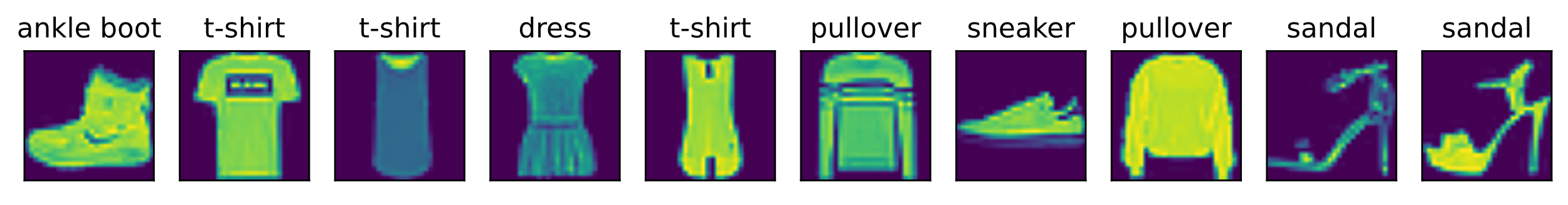


图 5数据集

1. **结果分析：**
2. **线性回归模型和Softmax模型的对比**

由图 6可知，Softmax的表现要优于线性回归模型（MSEloss），这大概是因为Softmax与交叉熵配合使用，使用指数形式的[Softmax](https://so.csdn.net/so/search?q=Softmax&spm=1001.2101.3001.7020" \t "_blank)函数能够将差距大的数值距离拉的更大，而交叉熵能更好的体现分类，对误分类能有很好的区分。而MSE，对梯度的更新较慢。尤其输出值预测的接近0或1时，梯度几乎为0。

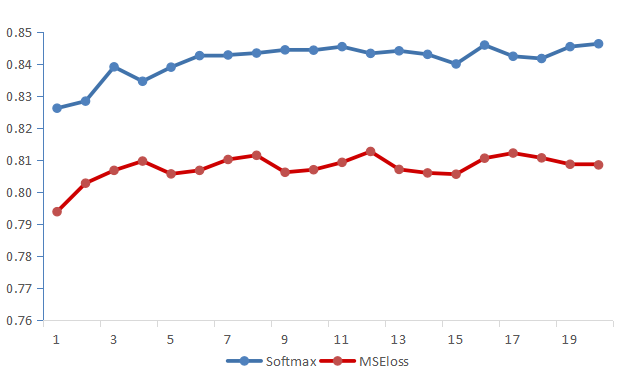


图 6 线性回归与Softmax回归的正确率

1. **不同的迭代次数**  
    探究神经网络在不同迭代次数下的训练结果。图 7是训练50个epoch的traing loss。每一个batchsize=32记录一次loss，可以看到loss从训练开始迅速下降，之后一直在0~0.5之间振荡。网络模型为单层感知机，使用Softmax回归，优化器使用Adam优化器。

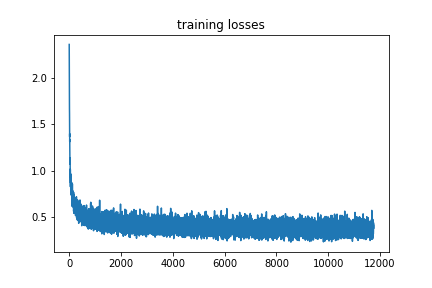


图 7 Epoch从1到50的training loss

图 8为该网路在训练集和测试集上的正确率。可以发现，训练集上的正确率整体高于测试集。训练集的最高正确率为87.4%，测试集的最高正确率为85.1%。

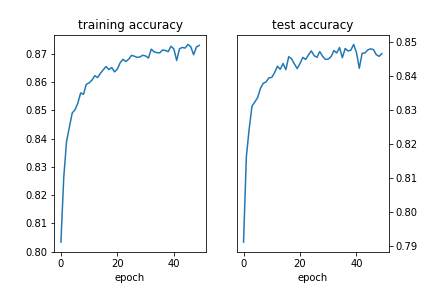


图 8 Epoch从1到50的训练集正确率和测试集正确率

分析正确率的折线图，未出现明显的过拟合，这大概是因为训练epoch并不算大，而且网路由单层感知机构成，网略深度较浅，结构较简单的缘故。

1. **采用不同批训练的大小（batch\_size）**

图 9为不同batchsize下在训练集和测试集上的正确率。选择了32,64,256的batchsize,网络为单层感知机，使用Softmax回归，优化器使用Adam。可以发现32和64的batchsize效果相当，略优于256的batchsize。因而256对于该神经网络是一个较大的batchsize，跑完一次epoch(全数据集)所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加，从而对参数的修正也就显得更加缓慢，batchsize增大到一定程度，其确定的下降方向已经基本不再变化（会影响随机性的引入）。

|  |  |
| --- | --- |
|  |  |

图 9 不同batchsize下的正确率

1. **采用不同的优化算法**

图 10为使用不同优化器，网络在测试集上的正确率。选择了Adam,SGD,Adagrad三种优化器。网络结构同上，batchsize取32。结果发现，Adam和Adagrad的效果相似，优于SGD。Adam和Adagrad都是在SGD模型上进行了优化。Adam在SGD上增加了动量算法，既能适应稀疏梯度，又能缓解梯度震荡的问题。Adagrad为不同的变量提供不同的学习率。它增加了罕见但信息丰富的特征的影响。

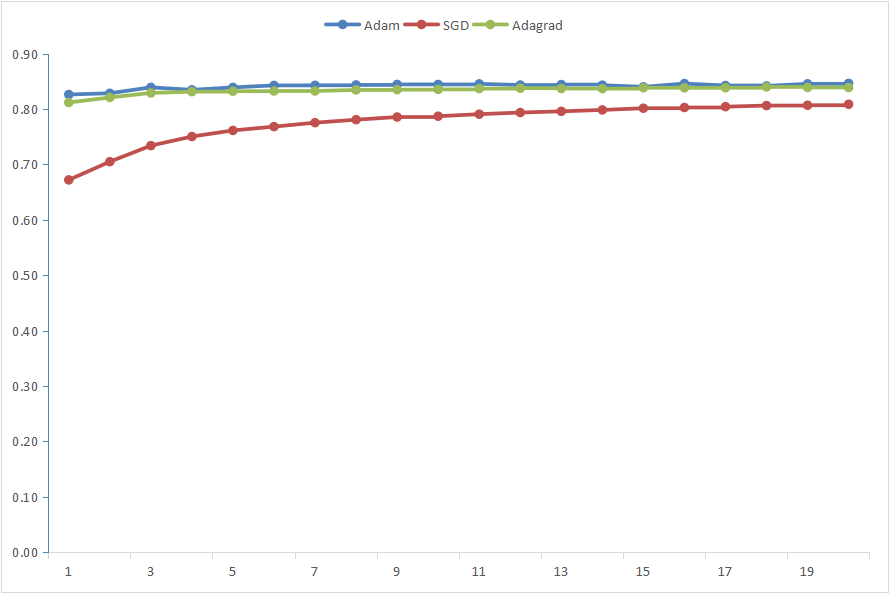


图 10 使用不同优化器的正确率

1. **采用避免过拟合的方式（dropout、正则项等）**

图 11为正则化方法对网络的影响。正则化方法有dropout和L2正则化。L2正则化的效果欠佳，这是因为网络层次较浅，由单层感知机构成，L2正则化在较深的网络有很好的表现。Dropout能以一定概率（此处p=0.5）去除一些神经元，提高网络的泛化性，在本网络中有一定体现。

|  |  |
| --- | --- |
|  |  |

图 11使用不同正则化方法的正确率

1. **不同的学习率（learning\_rate）**

图 12探讨不同学习率对Adam优化器的影响。Adam的默认学习率为1e-3,此处选择了3e-4和1e-2两个学习率来相互比较。1e-3和3e-4的效果相近，而1e-2的效果差于这两个学习率。学习率过大，会导致待优化的参数在最小值附近波动，不收敛；学习率过小，会导致待优化的参数收敛缓慢。学习率为1e-2时可以看到十分明显的波动

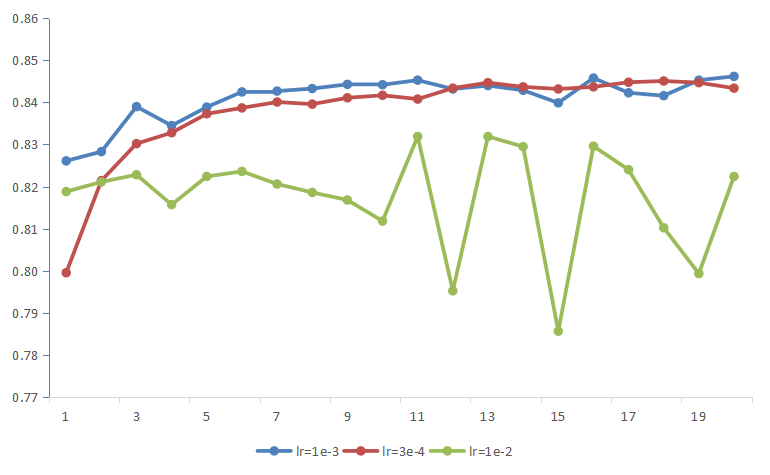


图 12 不同学习率下的正确率

1. **网络深度的大小**

图 13为多层感知机模型探究隐藏层维度的影响。结果显示隐藏层的效果欠佳，原因可能是因为多层感知机模型没有对图片特征进行提取，所以加入了影藏层也不能提供更多的信息。还有一个原因是出于对照和单一变量的要求，没有加入激活函数。

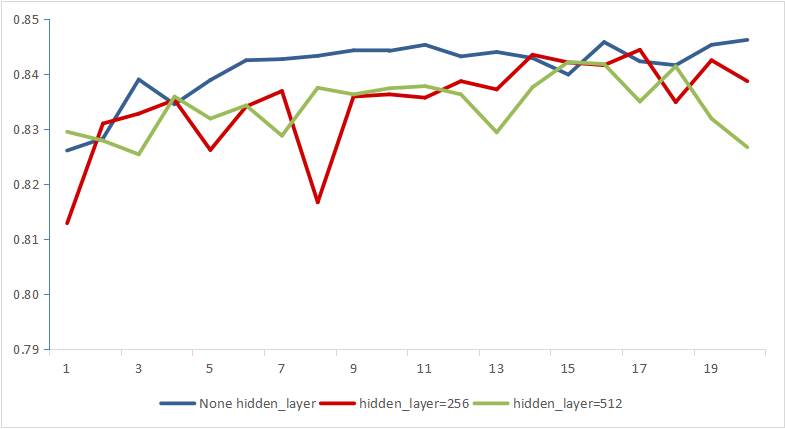


图 13 不同隐藏层维度下的正确率

1. **使用激活函数**

图 14探究激活函数对神经网络训练结果的影响。网络均采用了2层隐藏层，一层维度为512，另一层的维度为256，batchsize为32，训练epoch为20。可以很明显地看到，使用激活函数比不使用效果要好。激活函数为神经网络加入非线性因素的，提高神经网络对模型的表达能力，解决线性模型所不能解决的问题。同时ReLU函数还能解决了梯度消失问题，且计算速度和收敛速度快。

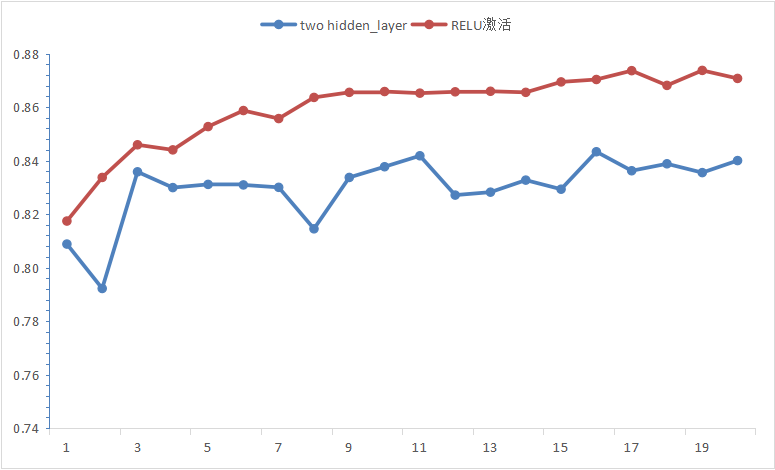


图 14 是否使用激活函数

将不同参数下模型的最高正确率整理为

表 1，考虑单一变量原则和控制变量法，每次实验模型只会改变一个参数，默认参数为:

。

表 1不同参数下模型的最高正确率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变化参数 | Batchsize=64 | Batchsize=32 | Batchsize=256 | Epoch=50 | Epoch=20 |
| 最高正确率 | 84.53% | 84.32% | 84.01% | **85.12%** | 84.32% |

续表1：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变化参数 | Adam | SGD | Adagrad | Hidden layer | dropout |
| 最高正确率 | **84.32%** | 79.35% | 83.75% | 83.87% | 83.33% |

续表2：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变化参数 | L2正则化 | lr=1e-3 | lr=3e-4 | ReLU | MSEloss |
| 最高正确率 | 82.11% | 84.32% | 84.34% | **87.57%** | 81.26% |

综合各个因素，最优模型如表 2。

表 2 最优模型参数

|  |  |
| --- | --- |
| 模型参数 | 参数取值 |
| batchsize | 32 |
| 优化器 | Adam |
| 学习率 | 1e-3 |
| 输入层维度 | 784 |
| 隐藏层维度 | 256 |
| 输出层维度 | 10 |
| 激活函数 | ReLU |
| 正则化方法 | droput(0.5) |
| 损失函数 | CrossEntropy |
| 最高正确率 | 89.53% |

# 总结及心得体会：

通过本次实验，我掌握了softmax函数，多层感知机，交叉熵，激活函数的基本原理和作用。 1) softmax用于多分类过程中，它将多个神经元的输出，映射到（0,1）区间内，可以看成概率来理解，从而来进行多分类。softmax通过指数进行加权，能够更好的放大不同类别之间的概率差异，从而提升网络的鲁棒性。 2) 多层感知机（MLP）模型是一个传统的神经网络，它能够解决单层感知机无法解决的非线性问题（单层感知机只能对线性数据进行分类）多层感知机最初是为了做分类的，但是它也可以处理回归问题，只要在最后将softmax等分类器改成sigmoid回归就行。它的训练方式也是“BP反向传导”。 3) 交叉熵损失函数是按照概率分布q的最优编码对真实分布为p的信息进行编码的长度。在给定q的情况下，如果 p和q越接近，交叉熵越小；如果 p 和 q越远，交叉熵就越大。因而，交叉熵常被用作分类任务的损失函数。 4) 激活函数为神经网络加入非线性因素的，单纯的感知机模型只能学到线性特征，而为感知机模型引入激活函数后能够更好地让网络学习非线性特征。激活函数往往需要连续并可导（允许少数点上不可导）的非线性函数。可导的激活函数可以直接利用数值优化的方法来学习网络参数。激活函数及其导函数要尽可能的简单，有利于提高网络计算效率。激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性。常见的激活函数有Sigmoid，tanh,ReLU函数等。其中是ReLU函数是目前最常用的激活函数，它具有单侧抑制、宽兴奋边界的生物学合理性，可缓解梯度消失问题。 5) 根据万能近似定理，对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的神经网络，只要其隐藏层神经元的数量足够多，它就可以以任意精度来近似任何一个定义在实数空间中的有界闭集函数。随着深度的增加，网络的表示能力呈指数增加，因而更深层的网络具有更好的泛化能力。但是，随着网络层数的增加，网络可能出现梯度消失等问题，使得整个网络很难训练。 6) 因为神经网络的拟合能力强，反而容易在训练集上产生过拟合。因此，在训练深层神经网络时，需要通过一定的正则化方法来改进网络的泛化能力。常见的方法有小批量梯度下降法（mini-batch），它既利于寻找全局最优解，梯度方差小，又训练速度快。L1和L2正则化方法，通过约束参数减小过拟合现象。还有dropout方法，通过随机丢弃一些网络结点，从而简化了网络，防止了过拟合现象。

通过本次实验，我对python基本语法更加熟悉，对pytorch框架更加熟悉。通过较熟练地使用pytorch建立基本的神经网络框架，创建损失函数，优化器等，能够完整地写出训练和测试过程，并将训练和测试放在cuda上。

# 十、对本实验过程及方法、手段的改进建议及展望：

**报告评分：**

**指导教师签字：**