Python程序设计作业

人工智能2004班-邵宗贺-U2020-----

说明：

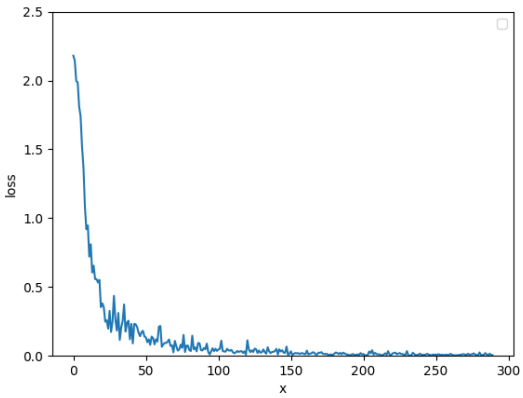
为提高训练精确度，防止数据较少出现过拟合，在实现卷积神经网络部分时，*mnist*手写数据集改用了*mnist.py*中的 *load_mnist*函数下载

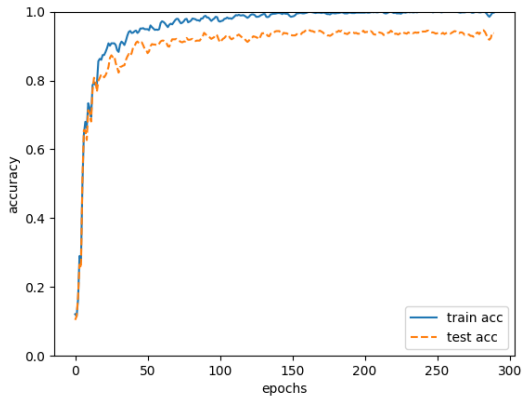
1. 神经网络实现及训练结果

1.1 简单全连接层神经网络

sklearn 下载的数据集数据只有1796个，选出1000个作为训练集，剩余作为测试集；
为防止训练时出现过拟合，采取减少循环的次数的方法，并仅仅搭建一层隐藏层作为全连接层神经网络，隐藏层的激活函数选用$Relu()$函数

测试结果：
训练集精确度：0.997，测试集精确度：0.948
loss = 0.0013720728614192318





1. 2 扩展全连接神经网络

扩展神经网络是指在简单神经网络加入几种学习的方法。

Dropout，在训练时，会随机选出一部分隐藏层的神经元进行删除，从而使得被删除的神经元不在再进行信号的传递

在进行初始权重的设定时，在选择激活函数时选用激活函数相匹配的初始值，与$Relu()$函数相适应的"He初始值"，简单来说就是一个标准差为$\sqrt{\frac{2}{n}}$的高斯分布，与$Sigmoid()$函数相适应的$Xavier$初始值，这样的话隐藏层的层数增多也不会影响数据的广度，下面的测试数据中也只搭建三层[200,64,64,]为例

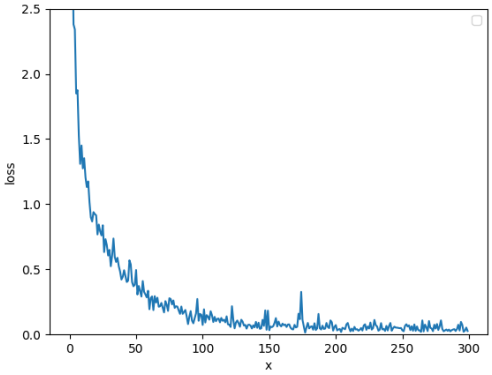
在进行扩展全连接神经网络的实现中发现利用sklearn下载的数据集用于训练的结果并不如简单全连接神经网络：

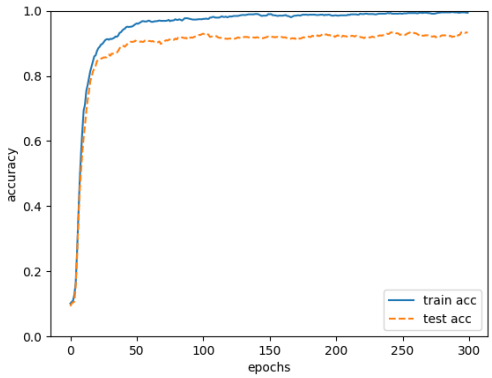
- 利用sklean下载的数据集的训练结果：

  这里也只是搭建一层隐藏层，激活函数选用$Relu$函数，隐藏层结点[32,]

训练集精确度：0.994,测试集精确度：0.933

loss=0.02074662244801393



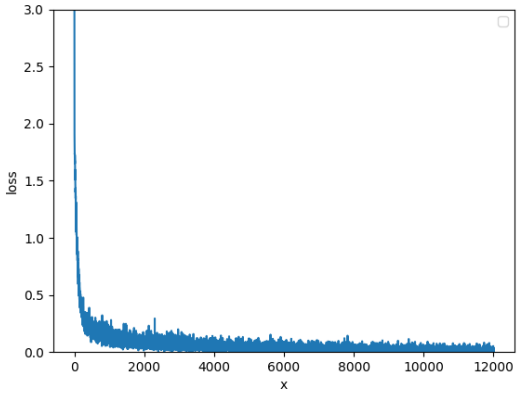


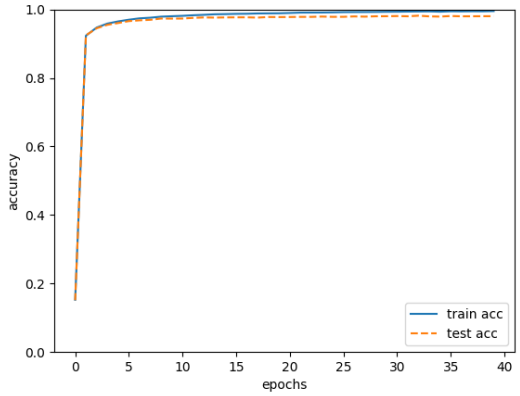
- 利用mnist下载的数据集的训练结果：

  不断改变相关参数，得到一个较优的结果：循环次数设为12000次，每次训练取出的元素个数（batch_size）设为100，在更新参数的函数中使用了$Adam()$,并将学习率(learning_rate）设为0.001

  训练集精确度：0.995,测试集精确度：0.981

  loss = 0.0018219183037276974





1.3 卷积神经网络

对于简单神经网络而言，会失去数据本身的形状，因次在实现时设定卷积层（conv）以及池化层（pool）（注：这里只实现并利用了Max池化）

卷积神经网络，简单CNN的实现，使用的层具体为：

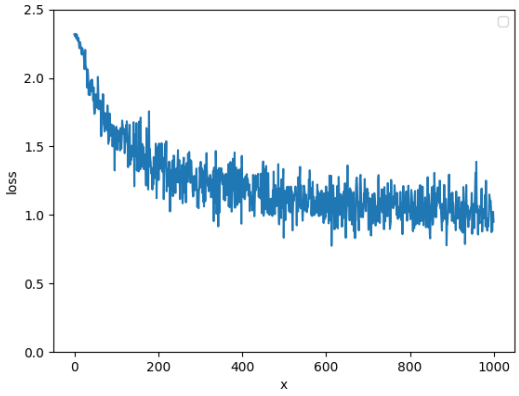
$$conv-relu-conv-relu-pool-conv-relu-conv-relu-pool-\\conv-relu-conv-relu-pool-affine-relu-dropout-affine-dropout-softmax$$

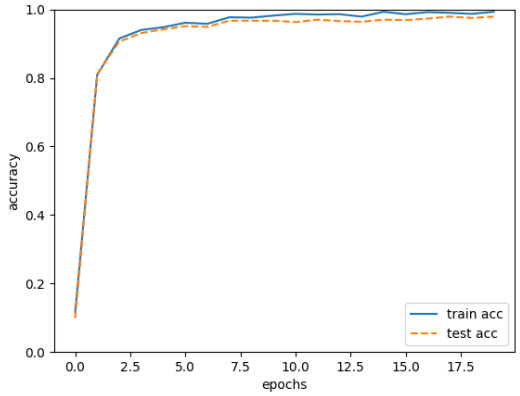
在实现时：在更新参数的函数中使用了$Adam()$,并将学习率(learning_rate）设为0.001

- 处理花费时间较长的情况下减少数据

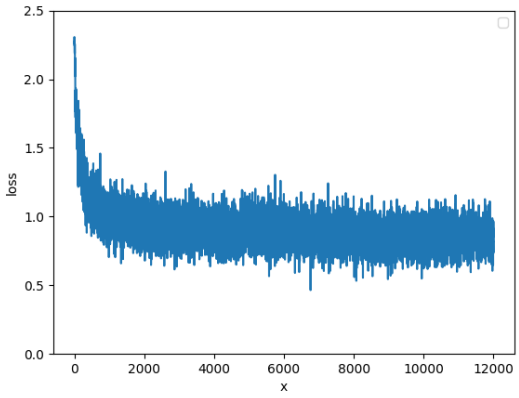
  这里减少数据，训练集取训练数据集中的前5000个数据，测试集取测试数据集中的前1000个数据

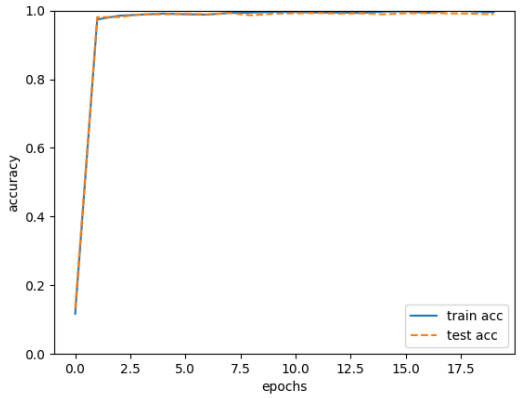
测试集精确度：0.974





- 使用全部数据进行CNN的训练
  测试集精确度：0.9927





2. 结果分析

这里的结果分析以sklearn 下载的数据集在全连接神经网络上的训练为据

2.1 总体结果

总体效果见上。

2.2 学习率的调节

最开始，我按照教材的前部分使用Sigmoid函数继续训练，学习率定为0.1之后，精度一直很小，几乎是没有任何训练效果，在和同学们交流后，将学习率改成0.01发现立刻模型的accuracy 上去了，并且很轻松就上了0.93、0.94。很容易明白这个原理，最初学习率为0.1 的时候因为模型学习率太大导致步长太长始终无法达到最小值。$sigmoid()$函数刚开始需要将学习率调的高一点，但太高就一直无法下降了，所以我就将学习率降到0.01，由此最终结果很容易就上0.94。

在后面选用其他激活函数时，对学习率的调节的重要性也是更不言而喻的。

2.3 激活函数的选取

激活函数	Acc
relu()	0.937
Sigmoid()	0.948
tanh()	0.902
cos()	0.221

在实现过程中，也了解到$cos()$激活函数，但在具体训练时发现，无论如何调整参数，$cos()$训练出的结果总是很差，在提交的代码中将用到$cos()$的部分剔除。

这里仅仅使用简单的训练，可以看到$Sigmoid()$训练效果是最好的，而$relu()$与$tanh()$则要稍微差一点。

3. 具体代码定义及逻辑

说明：函数内部的具体实现逻辑在代码文件的注释中进行说明

- active.py

详细定义了激活函数层的正向传播与反向传播函数

```
class Relu(object):
    """Relu 激活函数
    """
class Sigmoid(object):
    """Sigmoid 激活函数
    """
class tanh(object):
    """"tanh激活函数
    """
```

- function.py

```
def sigmoid(x):
    """
    Sigmoid激活函数的正向传播计算函数
    """
def sigmoid_grad(x):
    """
    Sigmoid激活函数的反向传播计算函数
    """
def softmax(x):
    """
    交叉损失熵的计算函数
    """
def cross_entropy_error(y, t):
    """
    交叉损失熵的计算函数
    """
def numerical_gradient(f, x):
    """
    数值微分的计算
```

```
    """
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """
    将输入数据展开成适合滤波器（权重），用于实现卷积层的正向传播
    """
def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
    """
    im2col的逆处理，用于实现反向传播
    """
```

- layers.py

```
class Affine(object):
    """
    在神经网络连接时定义连接层的函数
    """
class SoftmaxWithLoss(object):
    """
    交叉损失熵层的实现
    由Softmax函数与loss函数构成，在这里的实现采用了批的处理
    """

# 在以下的计算梯度类中，都定义了update函数用于梯度的变化更新
class SGD(object):
    """
    SGD:随机梯度下降法
    """
class AdaGrad(object):
    """
    AdaGrad：学习率衰减的思想
    可以通过RMSProp方法改善无止境学习后更新量变为0的问题
    """
class Adam(object):
    """
    Adam：融合 Momentum 与 AdaGrad
    """

class Linear(object):
    """
    全连接层的实现类
    """
    def __init__(self, input_size, hidden_size_list, output_size,
weight_init_std=0.01):
        """
        类定义
        """
    def loss(self, x, t):
        """
        损失值计算
        """
    def accuracy(self, x, t):
        """
        精确度计算
        """
    def numerical_gradient(self, x, t):
        """
        数值微分法计算梯度
```

```
        """
    def gradient(self, x, t):
        """
        误差反向传播法计算梯度
        """
class Nesterov(object):
class RMSprop(object):
    """
    Nesterov与RMSprop是在实现卷积层时用到的新的梯度计算类，也包含了update函数
    """
class Convolution(object):
    """
    卷积层的实现
    """
class Pooling(object):
    """
    池化层的实现
    """
```

注意，在卷积层的实现中，前向传播中输出大小为(H, W),滤波器大小为(FH, FW),则输出大小为$OH = \frac{H+2P-FH}{S} + 1$，$OW = \frac{W+2P-FW}{S} - 1$

- main_skl.py

  功能为利用sklearn库实现全连接层以及扩展全连接层函数，并将训练结果展示为图形

- Multilayer.py

```
class BatchNormalization(object):
    """
    BatchNormalization：对数据分布进行正规化的层
    """

    def __init__(self, gamma, beta, momentum=0.9, running_mean=None,
running_var=None):
    def forward(self, x, train_flg=True):

    def __forward(self, x, train_flg):

    def backward(self, dout):

    def __backward(self, dout):

class Dropout(object):
    """
    Dropout
    """

class MultiLayerNetExtend(object):
    """扩展版的全连接的多层神经网络
    具有Weiht Decay、Dropout、Batch Normalization的功能
    """

    def __init__(self, input_size, hidden_size_list, output_size,
                 activation='relu', weight_init_std='relu',
weight_decay_lambda=0,
                 use_dropout=False, dropout_ration=0.5, use_batchnorm=False):
        """
```

```
        :param input_size: 输入大小（MNIST的情况下为784）
        :param hidden_size_list: 隐藏层的神经元数量的列表（e.g. [100, 100, 100]）
        :param output_size: 输出大小（MNIST的情况下为10）
        :param activation: 'relu' or 'sigmoid'
        :param weight_init_std: 指定权重的标准差（e.g. 0.01）
                                指定'relu'或'he'的情况下设定“He的初始值”
                                指定'sigmoid'或'xavier'的情况下设定“Xavier的初始值”
        :param weight_decay_lambda: Weight Decay（L2范数）的强度
        :param use_dropout: 是否使用Dropout
        :param dropout_ration: Dropout的比例
        :param use_batchnorm: 是否使用Batch Normalization
        """
```

- mnist.py

  本文件是教材所附源代码，用来下载和储存mnist数据集

- main_mni.py
  存放利用mnist下载数据集并分别利用简单神经网络的训练以及扩展神经网络的训练的功能
- cnn.py

```
class DeepConvNet:
    """
    网络结构如下所示
        conv - relu - conv- relu - pool -
        conv - relu - conv- relu - pool -
        conv - relu - conv- relu - pool -
        affine - relu - dropout - affine - dropout - softmax
    """
    def predict(self, x, train_flg=False):
    def loss(self, x, t):
    def accuracy(self, x, t, batch_size=100):
    def gradient(self, x, t):
    # 以上与前面的神经网络全连接层是类似的原理
    def save_params(self, file_name='./pkl/params.pkl'):
        """
        作用是把数据利用Python自带的pickle存入params.pkl文件
        """
    def load_params(self, file_name="./pkl/params.pkl"):
        """
        读取pkl文件的功能函数
        """

class Trainer:
    """
    进行神经网络的训练的类
    """

    def __init__(self, network, x_train, t_train, x_test, t_test,
                 epochs=20, mini_batch_size=100,
                 optimizer='SGD', optimizer_param={'learning_rate': 0.01},
                 evaluate_sample_num_per_epoch=None, verbose=True):

    def train_step(self):

    def train(self):
```

- train_cnn.py

  存放利用mnist下载数据集并分别利用cnn.py中实现的卷积神经网络进行的训练函数

4. 心得体会

实际上之前我已经自己学习实现过神经网络了，但当时对于原理的具体实现并不会得很透彻，这一遍的学习中，我又从头开始详细地学习了每一部分，从多层感知机开始学习，到梯度下降、反向传播，再到学习矩阵求导确实学到了更多细节的知识。

最初学习时只是简单实现了全连接层和激活函数封装成类之后写了个简单的前馈网络，在学到卷积后自己实现了卷积层、池化层, 其中卷积层$im2col()$和池化层$col2im()$ 这两个函数的实现将整个图片的卷积操作转化为一个二维的大矩阵乘法，我感觉这两个函数非常精妙，实现起来也非常精巧。再到后来在卢仁智老师分享的资料中也自己学习并利用TensorFlow进行了CNN的训练。