# MLP实验报告

## 董一21307130209

1. **MLP的构造**

**1.1全连接层FC**

全连接层，是MLP最重要的一环，它需要在训练时接受一个输入X，将其与自身的权重矩阵w相乘得到输出z，最后通过激活函数得到激活后的输出a。而在反向传播时，则需要接受回传的误差dA，向前一层传递新的误差，同时计算，用于更新w。于是我为FC写了一个专门的类来使用它：

class FC(Layer):  
 # 全连接层  
 def \_\_init\_\_(self, output\_size=10, activation="relu", input\_size=10)

它的初始化会添加如下成员：

* Activation：激活函数
* Derivative：激活函数的导数
* X：输出。用于计算梯度
* A：激活后的输出
* W：权重矩阵
* Z：未激活的输出
* dZ，dW，dA：Z，W，A关于loss的导数

它有forward方法，用于执行前项传播：

def forward(self, X):  
 self.X = X  
 self.Z = np.matmul(self.W, X)  
 self.A = self.activation(self.Z)  
 return self.A

backward方法，用于执行反向传播：

def backward(self, dA):  
 self.dZ = dA \* self.derivative(self.Z)  
 self.dW = np.matmul(self.dZ, self.X.T) / len(self.X[0])  
 return np.matmul(self.W.T, self.dZ)

update方法，用于更新w：

def update(self):  
 self.W -= self.optimizer.update(self.dW)

这里有个optimizer，是我后来为改进算法添加的，用于选择不同的优化器。

这样子，就几乎实现了FC层所需要的全部功能。

**1.2激活函数activation**

全连接层需要激活函数来初始化，对于每一个激活函数，除了知道其函数形式func，还需要知道其导函数derivative，这里我根据PPT以及网络得到了部分可行的激活函数：

* linear：线性激活函数，导函数为1
* ReLU：
* Sigmoid:
* Softmax:

它的导函数有点特别，因为它一般作为输出层使用，而且一般配合CrossEntropy损失函数使用，我会在介绍交叉熵时介绍softmax的求导方法

* Tanh：
* LeakyReLU:

我的程序允许自定义的激活函数，只需要有func与derivative方法即可。

**1.3损失函数loss\_func**

对于误差回传，首先需要的是求得，我的代码中同样将损失函数抽象出来做了封装。代码中提供三种损失函数：

* MSE：

MSE通常用于回归模型，最后一层一般选用linear激活函数。

* Logistic:

Logistics常用于多标签分类，一般和sigmoid激活函数一起使用

* CrossEntropy：

其中T为真实类别的标签。若y为独热编码，那么，这里就是为什么softmax不需要计算导数的原因，因为可以直接算出loss关于z的导数，而不是关于a的导数，而且形式非常简单，无需像PPT上面那么复杂。交叉熵损失函数的偏导代码如下：

def derivative(self, y, y\_hat):  
 tmp = y\_hat.copy()  
 tmp[y[0], np.arange(len(y[0]))] -= 1  
 return tmp

综上，所有FC层、损失函数与激活函数都得到了实现，现在需要一个将这些东西整合起来的东西。

**1.4模型model**

现在用一个model类整合所有FC层，自动划分好batch和epoch。我实现的model有如下方法：

Forward：将所有FC层依次执行前项传播

def forward(self, X):  
 for layer in self.layers:  
 X = layer.forward(X)  
 return X

backward：从后往前，所有FC层依次反向传播

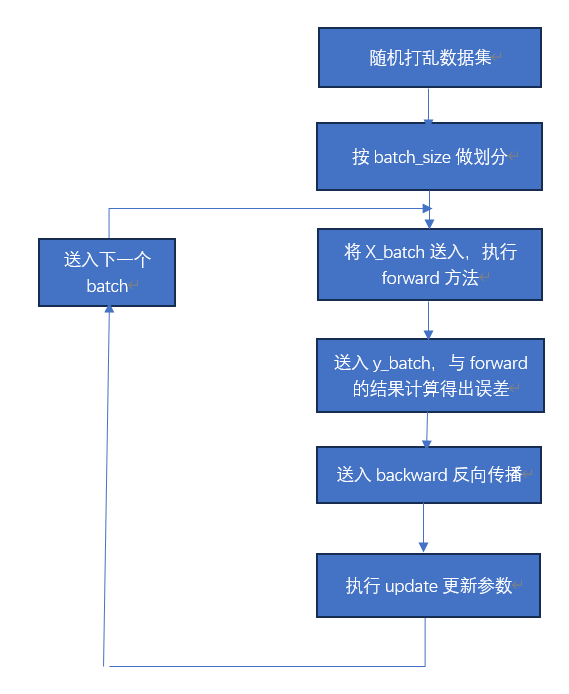
def backward(self, dA):  
 for layer in reversed(self.layers):  
 dA = layer.backward(dA)  
 return dA

update：所有FC依次更新参数

def update(self, alpha):  
 for layer in self.layers:  
 layer.update(alpha)

train：模型训练。

在train执行的过程中，每一个执行如下步骤：



根据这个过程即可完成训练。

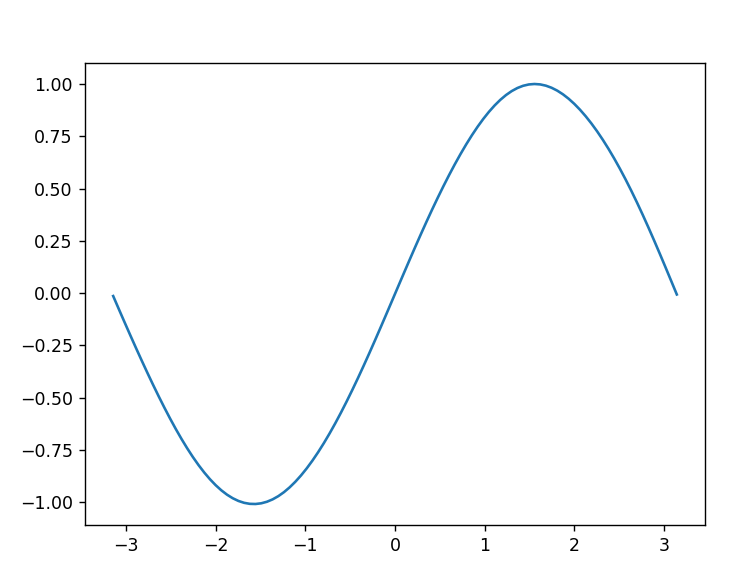
1. **实验过程  
   2.1 sin函数的拟合。**

首先要使用np.random.rand生成1000个数据，用np.sin生成标签值。我在这里使用了不同的网络，其实大部分网络都能较好的拟合出sin的形状，比如

mymodel = Model(  
 [FC(64, "sigmoid", 1),  
 FC(16, "sigmoid", 64),  
 FC(1, "linear", 16)]  
)

训练的batch为32，epoch为500。由于是数值回归，所以输出层采用linear激活函数，损失函数使用MSE。

得到的图片为：

****

在上的L1误差为0.003左右：



在加深或减少一层网络后得到的L1误差仍在0.002~0.004之间，说明在500轮内该网络已经能较好地拟合sin函数。

**2.2 对手写数据的分类**

经查看，手写数据共10类，每张图片64×64像素，可以用flatten函数将其展开成4096×1的向量。在经过一系列优化后，采用了一个比较简单的网络：

mymodel = Model(  
 [FC(256, "LeakyReLU", 4096),  
 FC(48, "LeakyReLU", 256),  
 FC(10, "softmax", 48)]  
)

训练100轮后，准确率在训练集上有0.96，验证集上有0.83，在kaggle上有0.89.期间还使用了更深更宽的网络，但是效果几乎差不多。

1. **优化**
   1. **优化器optimizer**

由于保证计算速度，我最开始使用带batch的SGD算法，但是在实验中发现，SGD在训练过程中波动较大，不能完成很好的收敛效果。所以参照网上的介绍，改用Adam优化来下降梯度。

Adam优化使用动量算法，使用历史梯度的加权均值与平方均值来修正梯度。具体方法如下：

1. 计算梯度
2. 计算历史梯度动量
3. 计算梯度平方指数移动平均
4. 纠正，，
5. 更新参数

代码见optimizer.py的Adam类update方法。在使用Adam优化后，模型能更快的收敛，同时准确率从0.75左右提升到0.85,我还对sin回归也使用了Adam优化，得到的图像更加对称，而且原点出十分光滑（原本的图像原点处是一个拐点的形状）

* 1. **LeakyReLU函数**

在执行图像分类任务时，我发现使用sigmoid和tanh激活函数时，能在一个hidden层结构的时候达到0.85左右的准确率，但是一旦增加到两个hidden层，那么网络将丧失分类能力，即容易掉入0.1准确率的局部解。我在尝试查看回传的梯度时，发现前面的梯度往往十分的小（1e-4以下），替换成ReLU之后，梯度又十分稀疏，于是我采用LeakyReLU激活函数。这时网络才显现出良好的分类性能。同时可以接受更深的网络，这时在训练集上能达到0.96的准确率。

* 1. **正则化**

我在加深网络层数后，往往在训练集上能达到0.96左右的准确率，但在验证集上却只有0.77左右的准确率，这说明模型出现了过拟合。我采用的是L2正则化，即损失函数为

很明显，求导只需要在原有梯度上加上即可。加上正则化后，在验证集上的准确率达到了0.84。

* 1. **降维**

有名的mnist数据集使用28×28的分辨率即可完成手写数字分类，在查看部分图像后，我发现完全可以利用降维来缩小数据大小。我使用了PCA保留0.95的降维，降维后数据维度从4096降到1084，我最终使用了1000维的PCA降维。这个结果在验证集上能达到0.86的准确率，在训练集上能达到0.98的准确率，但是在测试集上仍然只有0.90左右的准确率。虽说并没有显著提升效果，但是这种数据预处理将训练速度加快了近3倍。

不过由于这个需要使用sklearn的库，并且不是很好嵌入到load\_parameter方法里面，所以在提交的的模型中没有使用降维。

1. **代码使用**

FC为全连接层，可以修改节点个数和激活函数，根据调整FC的个数就能调整隐藏层个数。

mymodel = Model(  
 [FC(64, "sigmoid", 1),  
 FC(1, "linear", 64)]  
)  
X = np.random.rand(1000, 1) \* 2 \* pi - 1 \* pi  
y = np.sin(np.sum(X, axis=1))  
loss\_func = Func.MSE()  
mymodel.train(X, y, 32, 500, 0.001, loss\_func, Adam)

train方法接受9个参数，分别是训练集特征X，训练集标签y，batch数，epoch数，学习率alpha，损失函数loss\_func，优化器（Adam），验证集特征valid\_X，验证集标签valid\_y（没有默认在训练集上面验证）。

我的save和load\_parameter保存的是pkl文件，这个方便一点。具体使用如下：

mymodel = Model([]) #一定要放一个空列表进来，不然报错  
mymodel.load\_parameter("model.pkl")

由于分类器输出的是10维向量，所以可以用：

np.argmax(mymodel.predict(inputs), axis=1)来显示具体类别。