机器学习第一次作业

岳东旭 2201212864*

2022 年 10 月 18 日

作业内容：手写体识别

1 数据处理

本部分介绍如何将拿到的 mnist 数据集进行预处理，进而得到可以被网络使用的数据集。

1.1 独热码

对于手写数字分类任务，神经网络的输出维度为 [bs, 10]，为了方便计算 Loss，需要把输入的标签编为 **one-hot vector**。实现方法如下：

```
def one_hot(label):
    temp = np.zeros((label.shape[0], 10))  # 创建全0array
    label = label.squeeze()  # 将dim=2压缩到dim=1
    temp[range(label.shape[0]), label] = 1  # numpy花式索引，将对应标签置为1
    return temp
```

1.2 归一化

mnist 数据集中的特征为手写数字图片，在进入神经网络前需要对输入特征进行归一化，防止因输入过大过小造成神经网络学习能力下降。实现方式如下：

```
def normalize(x):
    x = x / 255  # 仅对图片进行归一化操作，防止输入过大导致学习速率下降
    return x
```

*Machine learning

1.3 数据载入

调用以上数据处理方法，载入并处理训练和测试数据。

```
def load_data(path):
    data = pd.read_csv(path, delimiter=',').values  # 使用pandas的read_csv方法读入为dataframe再转为numpy array
    return normalize(data[:, 1:]), one_hot(data[:, :1])  # 将数据与标签分开

X_train, y_train = load_data('./archive/mnist_train.csv')  # 载入训练集数据，分为特征与标签
X_test, y_test = load_data('./archive/mnist_test.csv')  # 载入测试集数据，分为特征与标签
```

打印并观察训练集与测试集的数据维度：

- 训练集特征维度 (60000, 784), 标签维度 (60000, 10)
- 测试集特征维度 (10000, 784), 标签维度 (10000, 10)

1.4 shuffle

载入数据后，由于要应用 mini-batch SGD 算法，因此需要定义针对训练数据的数据打乱操作：

```
def shuffle(data, label):
    idx = list(range(data.shape[0]))
    np.random.shuffle(idx)
    data = data[idx]  # 花式索引打乱
    label = label[idx]
    return data, label
```

2 激活函数与 softmax

在此部分，我们定义激活函数和 softmax 运算，激活函数采用 Relu，考虑到反向传播时的微分操作，因此定义了 Relu 函数的微分运算。

2.1 激活函数

采用 Relu 激活函数，实现前向传播和反向传播，Relu 运算可以抽象为公式：

$$max(x, 0)$$

```
def ReLU(x):
    return np.maximum(0, x)  # Relu线性整流单元运算
def dReLU(x):
    return 1 * (x > 0)
```

2.2 softmax 运算

softmax 主要是为了在最后一层输出各个类别的可能概率，等价于如下公式:

$$Softmax\left(z_i\right) = \frac{e^{z_i}}{\sum_{c=1}^{C} e^{z_c}}$$

实现方法如下:

```
def softmax(z):
    z = z - np.max(z, axis=1).reshape(z.shape[0], 1)
    return np.exp(z) / np.sum(np.exp(z), axis=1).reshape(z.shape[0], 1)
```

3 神经网络的实现

本节阐述神经网络的架构、参数设置以及前向传播、反向传播的原理及实现

3.1 网络架构及初始化

可以在此设定训练及网络的超参数,如学习率,batch-size, learning-rate, 训练轮数，以及是否加载预训练的网络参数。

网络架构采用两层 MLP，一共 4 组可学习参数

```
class NeuralNetwork:
    def __init__(self, x_train=X_train, y_train=y_train, x_test=X_test,
     y_test=y_test, batch=64, lr=5e-1, epochs=120, pre_train=False):
        self.input = x_train
        self.target = y_train
        self.x_test = x_test
        self.y_test = y_test
        self.batch = batch  # 一个batch大小
        self.epochs = epochs  # 训练轮数
        self.lr = lr  # 学习率

        self.x = []  # 用以保存样本特征
```

```
        self.y = []  # 保存样本标签
        self.loss = []  # 保存每个epoch的loss值
        self.train_acc = []  # 保存训练准确率
        self.test_acc = []  # 保存测试准确率
        if pre_train:  # 如果之前保存了模型参数，则不需要再训练了
            self.load_parameters()
        else:
            self.init_weights()

    def init_weights(self):
        self.W1 = np.random.randn(self.input.shape[1], 256)  # [784, 256]
        self.W2 = np.random.randn(self.W1.shape[1], self.target.shape[1])  # [256, 10]

        self.b1 = np.random.randn(self.W1.shape[1], )  # [256, ]
        self.b2 = np.random.randn(self.W2.shape[1], )  # [10, ]
```

如果不加载预训练模型，则将网络参数随机初始化。

3.2 前向传播

即神经网络从前往后，从输入到输出的计算过程。参考公式如下：

$$z^{(1)} = x * W^{(1)} + b^{(1)}$$

$$a^{(1)} = Relu(z^{(1)})$$

$$z^{(2)} = a^{(1)} * W^{(2)} + b^{(2)}$$

$$a^{(2)} = Softmax(z^{(2)})$$

```
    def feedforward(self):
        self.z1 = self.x.dot(self.W1) + self.b1  # [bs, 256] + b1:[256] 广播机制 = [bs, 256]
        self.a1 = ReLU(self.z1)  # a1:[bs, 256]

        self.z2 = self.a1.dot(self.W2) + self.b2  # w2:[256, 10]  z2:[bs, 10]
        self.a2 = softmax(self.z2)  # [bs, 10]

        self.error = self.a2 - self.y  # [bs, 10]
```

3.3 反向传播

为了更新优化网络参数，需要对每个可优化参数利用链式法则进行微分，再利用梯度下降更新，参考公式：

$$dW^{(2)} = \frac{\partial L}{\partial a^{(2)}} * \frac{\partial a^{(2)}}{\partial W^{(2)}}$$

$$db^{(2)} = \frac{\partial L}{\partial a^{(2)}} * \frac{\partial a^{(2)}}{\partial b^{(2)}}$$

$$dW^{(1)} = \frac{\partial L}{\partial a^{(2)}} * \frac{\partial a^{(2)}}{\partial a^{(1)}} * \frac{\partial a^{(1)}}{\partial z^{(1)}} * \frac{\partial z^{(1)}}{\partial w^{(1)}}$$

$$db^{(1)} = \frac{\partial L}{\partial a^{(2)}} * \frac{\partial a^{(2)}}{\partial a^{(1)}} * \frac{\partial a^{(1)}}{\partial z^{(1)}} * \frac{\partial z^{(1)}}{\partial b^{(1)}}$$

代码实现：

```
def backprop(self):
    dcost = (1 / self.batch) * self.error  # [bs, 10]

    DW2 = np.dot(dcost.T, self.a1).T # 链式法则 dcost:[bs, 10], a1:[bs, 256], dw2:[256, 10]
    DW1 = np.dot((np.dot((dcost), self.W2.T) * dReLU(self.z1)).T, self.x).T  # dw1:[784, 256]

    db2 = np.sum(dcost, axis=0) #db2:[10, ]
    db1 = np.sum(np.dot((dcost), self.W2.T) * dReLU(self.z1), axis=0)  # db1:[256, ]

    self.W2 = self.W2 - self.lr * DW2 # 根据梯度更新, minibatch-SGD
    self.W1 = self.W1 - self.lr * DW1

    self.b2 = self.b2 - self.lr * db2
    self.b1 = self.b1 - self.lr * db1
```

4 训练及测试

本节介绍训练神经网络和测试验证的方法。

4.1 训练

在每个 epoch 中，首先对训练数据进行 shuffle 操作，然后再每个 batch 上进行前向传播和反向传播，计算损失值，统计当前的识别准确率，并在每个 epoch 结束保存模型的参数。

```
    def train(self, is_shuffle=True, loss_fn='MSEloss'):
        for epoch in range(self.epochs):
            l = 0
            acc = 0
            best_acc = 0
            if shuffle:  # 默认打乱数据
                self.input, self.target = shuffle(self.input, self.
    target)
            val_acc = []
            for batch in range(self.input.shape[0] // self.batch - 1):
                start = batch * self.batch  # 开始的位置
                end = (batch + 1) * self.batch  # 结束位置
                self.x = self.input[start:end]  # 索引样本特征
                self.y = self.target[start:end]  # 索引样本标签
                self.feedforward()  # 执行前向传播
                self.backprop()  # 执行反相传播
                if loss_fn == 'MSEloss':
                    l += np.mean(self.error ** 2)  # 计算均方误差
                acc_now = np.count_nonzero(np.argmax(self.a2, axis=1)
    == np.argmax(self.y, axis=1)) / self.batch  # 计算准确率
                acc += acc_now
                if batch%100 == 0:
                    val_acc.append(self.test())
            print(f"epoch:{epoch}, loss:{l}, train_accuracy:{acc_now},
    val_accuracy:{np.mean(val_acc)}")
            self.loss.append(l / (self.input.shape[0] // self.batch))
            self.train_acc.append(acc * 100 / (self.input.shape[0] //
    self.batch))
            if np.mean(val_acc) > best_acc:
                best_acc = np.mean(val_acc)
                self.save_parameters()  # 保存模型参数，方便下次加载
        return self.loss, self.train_acc
```

在实际中，训练了 200 个 epoch，画出了训练过程中 loss 随训练时间的变化，以及训练集准确率的变化，如图 1, 2所示，可以看出 loss 在最初的阶段随训练轮数增加迅速减小，而后趋于平缓，识别准确率的变化则相反。

从图 3可以看出验证集的准确率并没有随训练轮数的增加而下降。

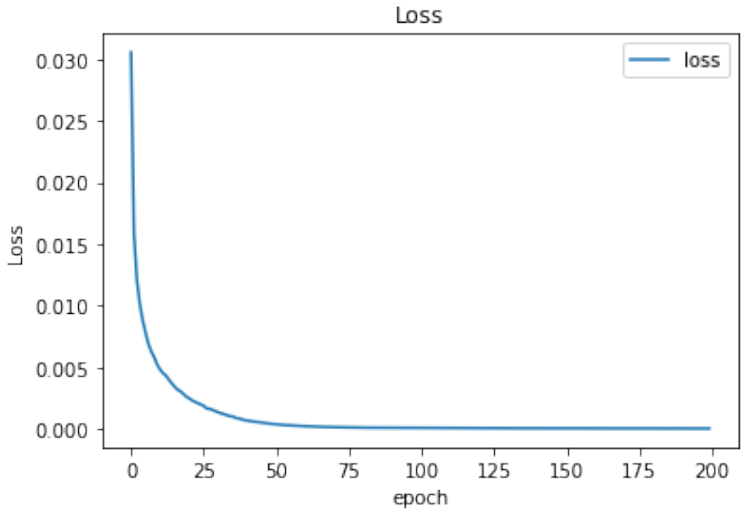

图 1: Loss via epoch

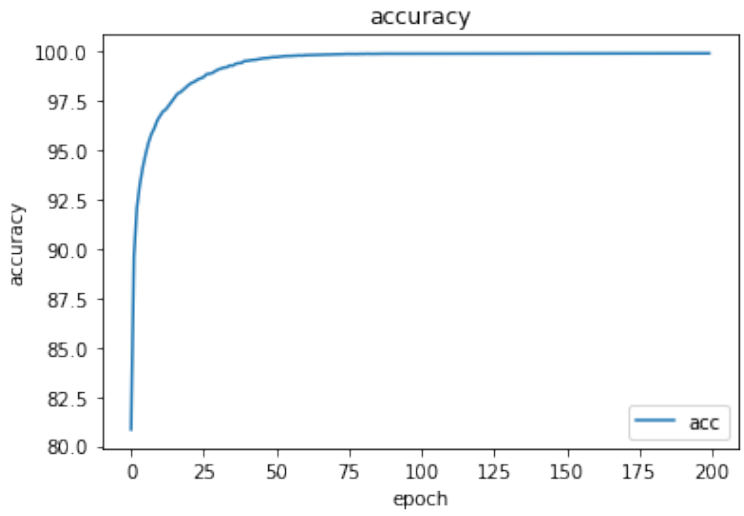

图 2: Accuracy via epoch

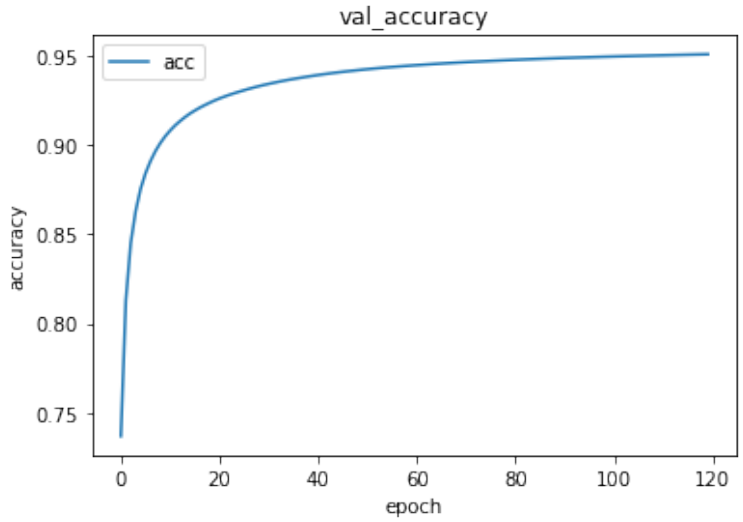

图 3: val Accuracy via epoch

4.2 测试

在测试集上，仅需要网络进行前向传播。

```
    def test(self, if_test=False):  # 模型测试方法
        acc = 0
        for batch in range(self.x_test.shape[0] // self.batch - 1):
            start = batch * self.batch
            end = (batch + 1) * self.batch
            self.x = self.x_test[start:end]
            self.y = self.y_test[start:end]
            self.feedforward()  # 测试步骤仅计算前向传播
            acc_now = np.count_nonzero(np.argmax(self.a2, axis=1) == np
    .argmax(self.y, axis=1)) / self.batch  # 计算准确率

            self.test_acc.append(acc_now)
        acc = np.mean(self.test_acc)  # 计算平均准确率并打印
        if if_test:
                print(f"accuracy:{acc}")
        return self.test_acc
```

在测试集上，经过对每个 batch 识别准确率的平均，得到最终的准确率为 **95.07**

5 总结

通过本次作业，深入了神经网络的底层架构，真正从 0 实现了神经网络，并在 mnist 数据集上取得了较好的表现。然而，受限于时间等原因，本项目还存在部分需要完善的地方，如没有进行网格化搜索调参，以及可以考虑采用更好的优化策略。

参考

1. 损失函数：https://zhuanlan.zhihu.com/p/35709485
2. softmax: https://zhuanlan.zhihu.com/p/25723112
3. numpy:
   - https://blog.csdn.net/u010089444/article/details/52738479
   - https://blog.csdn.net/hustqb/article/details/78090365
4. pyplot: https://wizardforcel.gitbooks.io/matplotlib-intro-tut/content/matplotlib/2.html
5. 激活函数求导：https://www.cnblogs.com/hutao722/p/9732223.html