AI-A-2023 Fall PJ1-Part1 report

柳世纯 20307130111

1. 实验原理

作为大四老狗并且在NLP lab打工了一年多, 做到本次实验还是感慨：合抱之木, 生于毫末. 第一节内容主要基于我对于ML以及DL的理解, 并且, 向Hinton致敬.

1.1 机器学习

滥觞于祖师爷图灵, AI衍生出三大历史路径：符号、连接、行为. 经历了70多年沉浮, 连接主义终于在历史长河中得以沉积, 并于2022年底爆发出新一轮的浪潮.

机器学习在做什么？机器学习的本质是一个优化问题. 优化问题的本质, 是在约束下拟合到训练数据的先验分布. 数据具备特征, 而端到端的训练免去了繁冗的特征工程, 由此发展了深度学习. 何凯明的残差神经网络加深了模型, 而Attention机制拓宽了模型；又深又宽, 此之谓"大模型".

1.2 神经网络

任何函数都可以被泰勒展开为多项式（皮亚诺余项收敛）, 多项式分解为线性+非线性部分. 神经网络就是对多项式的逼近, 它由一系列神经元相互连接而成, 每一个神经元处理最基础的标量算术并使用激活函数引入非线性映射.

神经元相互排列, 构成了输入层、隐藏层、输出层；反映拟合效果的指标为损失函数, 由损失函数对中间神经元的参数进行优化的算法为梯度下降算法. 损失函数的选择取决于任务的选择, 本质是对于数据分布的描述. 回归问题中可以使用mse, 分类问题多用cross entropy；

1.3 梯度下降

梯度下降两大步骤：前向传播和反向传播. 前向传播就是上一层的输入流向这一层, 这一层的输出流向下一层, 直至得到最终的结果并计算损失函数. 反向传播就是根据损失函数对各层神经元参数进行调整. 这个过程是从后往前依次求梯度, 求出梯度值并结合学习率得到参数更新的大小.

2. 实验过程

2.1 拟合sin曲线

具体代码详见`pj1-part1.ipynb`文件.

数据生成

题目要求的函数

$$y = \sin(x), x \in [-\pi, \pi] \tag{1}$$

解决方案：x均匀分布于$[-\pi, \pi]$, y在$\sin(x)$真实值基础上增加高斯分布的噪声变量得到.

```python
def sin_data_generate(filename: str):
    x_min, x_max = -1 * np.pi, np.pi
    n_samples = 5000
    x = np.random.uniform(x_min, x_max, size=n_samples)
    y_true = np.sin(x)
    # add noise
    noise_scale = 0.05
    y_sample = y_true + np.random.normal(scale=noise_scale, size=n_samples)
    ...
```

神经元

定义基础的功能：前向传递数据、反向传递梯度.

```
class Layer:
    def __init__(self):
        pass

    def forward(self, x):
        raise NotImplementedError

    def backward(self, grad):
        raise NotImplementedError
```

全连接层`Linear`: 参数随机初始化, 累计微分为0; 前向计算为线性矩阵乘法运算; 反向计算从上一层拿到梯度继续处理, 分别处理对于weight和bias的偏导数.

```
class Linear(Layer):
    def __init__(self, in_dim: int, out_dim: int, last: bool = False):
        super().__init__()
        self._type = "layer"
        ...
        self.dw = np.zeros_like(self.weight)
        self.db = np.zeros_like(self.bias)

    def forward(self, x: ndarray) -> ndarray:
        self.x = x
        x1 = x.reshape(x.shape[0], -1)
        val_out = np.matmul(x1, self.weight) + self.bias
        return val_out

    def backward(self, dL_dz):
        # dL_dz为后向传播的输入梯度(1, 1),a是经过激活函数的输出,z是经过线性计算的输出
        x = self.x
        dz_dw = x.reshape(x.shape[0], -1)

        dL_dw = np.matmul(dz_dw.T, dL_dz)
        dL_db = np.sum(dL_dz, axis=0, keepdims=True)

        dL_da = np.matmul(dL_dz, np.transpose(self.weight))
        dL_da = dL_da.reshape(x.shape)
        self.dw += dL_dw
        self.db += dL_db
        return dL_da
```

激活函数: `Sigmoid`, `ReLU`;

```
def sigmoid(x: ndarray):
    x_ravel = x.ravel()
```

```
    length = len(x_ravel)
    y = []
    for index in range(length):
        if x_ravel[index] >= 0:
            y.append(1.0 / (1 + np.exp(-x_ravel[index])))
        else:
            y.append(np.exp(x_ravel[index]) / (np.exp(x_ravel[index]) + 1))
    return np.array(y).reshape(x.shape)


class Sigmoid(Layer):
    def __init__(self):
        super().__init__()
        self._type = "active"
        pass

    def forward(self, z):
        self.z = z
        a = sigmoid(z)
        return a

    def backward(self, grad_output):
        z = self.z
        a = sigmoid(z)
        return grad_output * a * (1 - a)


class ReLu(Layer):
    def __init__(self):
        super().__init__()
        self._type = "active"
        pass

    def forward(self, z):
        self.z = z
        return np.maximum(z, 0)

    def backward(self, grad_output):
        z = self.z
        z[z <= 0] = 0
        z[z > 0] = 1
        return grad_output * z

```

损失函数

本次采用的是`mse_loss`

```python
def mse_loss(predict: ndarray, label: ndarray):
    loss = np.square(predict - label).mean() / 2
    grad = (predict - label).reshape(1, 1)
    return loss, grad
```

网络

定义前向, 后向计算过程, 参数更新方式, 以及保存和加载参数的方法.

此次项目中用到的第一个网络结构为简单的仅仅含输入输出头的网络.

```python
nn = NeuralNetwork()
nn.setup_layer(Linear(1, 100))
nn.setup_layer(Sigmoid())
nn.setup_layer(Linear(100, 1, last=True))
```

训练

随机梯度下降. 每次训练一个epoch之前将样本顺序打乱(但是对齐样本和标签), 在每一个样本上对全量参数进行更新.

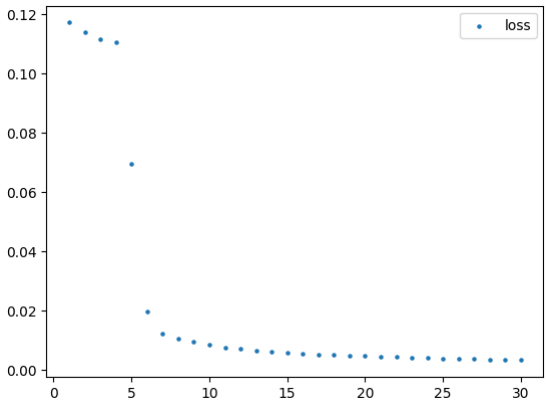
小跑30个epoch

```python
epochs = 30
lr = 1e-2
train(nn, x, y, mse_loss, epochs, lr)
```

训练过程:

```
1 epoch, average error:0.11725076805849771
2 epoch, average error:0.1141583505983662
3 epoch, average error:0.11174014443251183
4 epoch, average error:0.1105807682556639
5 epoch, average error:0.06959304635296822
6 epoch, average error:0.019749290547838923
7 epoch, average error:0.012359905048463155
8 epoch, average error:0.010530872979259922
9 epoch, average error:0.009478219856890291
10 epoch, average error:0.008353225042481168
11 epoch, average error:0.007575515449875874
12 epoch, average error:0.007090855145576874
13 epoch, average error:0.006575175334260657
14 epoch, average error:0.006054411507754137
15 epoch, average error:0.005720220264111501
16 epoch, average error:0.005428445508395655
17 epoch, average error:0.005212938453419483
18 epoch, average error:0.004929726866383251
19 epoch, average error:0.004796954593078246
20 epoch, average error:0.004594328070648486
21 epoch, average error:0.004408527679259166
22 epoch, average error:0.004287458469188808
23 epoch, average error:0.004126119009473579
24 epoch, average error:0.003958200172972088
25 epoch, average error:0.0038351438897617
...
28 epoch, average error:0.0034481883877143745
29 epoch, average error:0.003310875939892047
30 epoch, average error:0.003204069997669693
-----save parameters-----
```

在train set的loss变化为



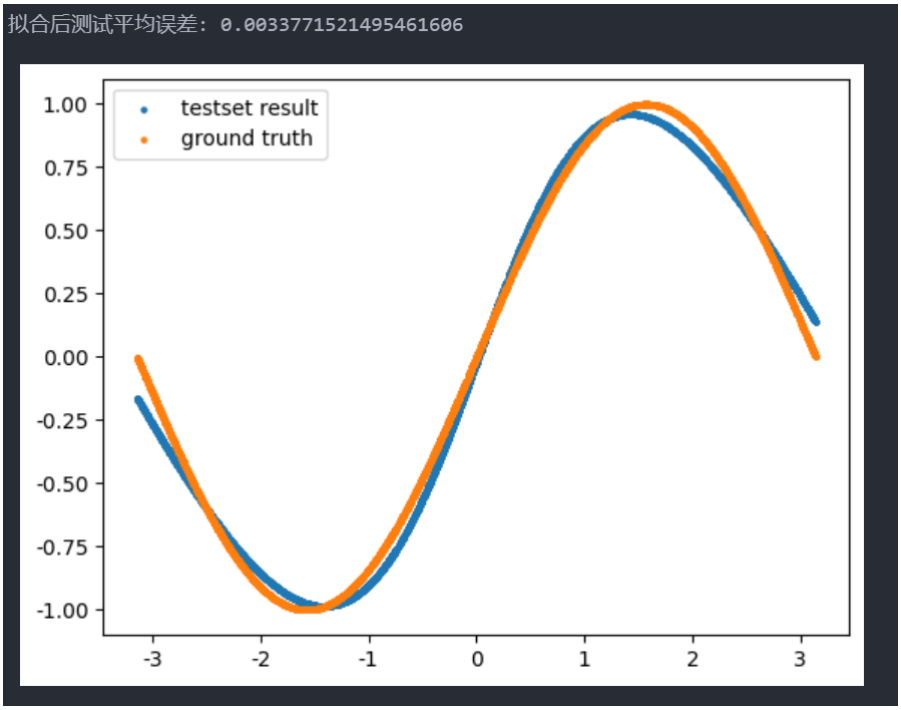
可以看到后面loss已经进入到一个相对平稳的阶段.

测试

测试代码

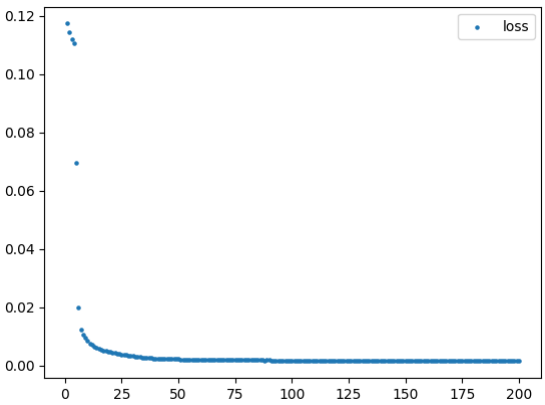
```
nn.load_parameter1()

x_test, y_test = read_file("test.csv")
pre = nn.predict(x_test)
y_sin = np.sin(x_test)
avg_error = np.square(pre - y_sin).mean()
print("拟合后测试平均误差:", avg_error)

plt.figure()
plt.scatter(x_test, pre, s=5, label="testset result")
plt.scatter(x_test, y_sin, s=5, label="ground truth")
plt.legend()
plt.show()
```

```
拟合后测试平均误差: 0.0033771521495461606
```

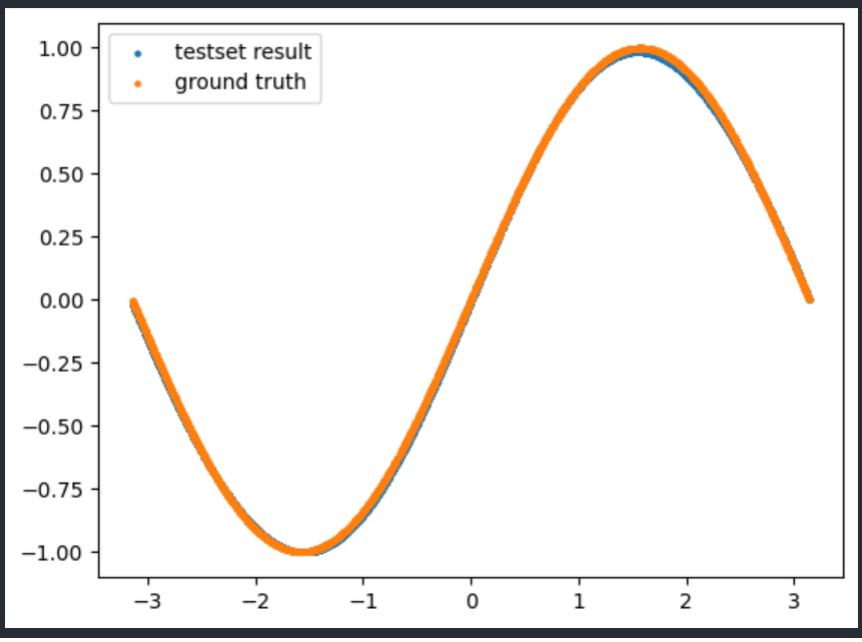


至此, 第一个任务完成课程要求.

后面跑了200个epoch, 得到了更高精度的结果.



```
拟合后测试平均误差: 0.00011006798489977597
```



2.2 手写汉字分类

数据处理

由于我们的网络的限制, 需要先将图片转化为向量形式, 标签转化为one hot向量.

```python
def image2vec(path: str):
    x_data = []
    y_data = []
    files = os.listdir(path)
    for i in range(1, len(files) + 1):
        sub_files = os.listdir(path + "/{0}".format(i))
        for j in range(1, len(sub_files) + 1):
            dirpath = path + "/{0}/{1}.bmp".format(i, j)
            img_cv = cv2.imread(dirpath).reshape(1, -1)
            img_cv = np.squeeze(img_cv).tolist()
            x_data.append(img_cv)
            y = [0] * 12
            y[i - 1] = 1
            y_data.append(y)
    return x_data, y_data
```

得到原始数据集之后, 使用 `StratifiedShuffleSplit` 进行分层采样, 通过 `8:2` 划分为数据集与测试集.

```
split = StratifiedShuffleSplit(
    n_splits=1, test_size=0.2, train_size=0.8, random_state=904
)

for train_index, test_index in split.split(x, y):
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

网络结构

和上一个模型一样, 较为简单

```
nn = NeuralNetwork()
nn.setup_layer(Linear(2352, 250))
nn.setup_layer(Sigmoid())
nn.setup_layer(Linear(250, 12, last=True))
```

损失函数

采用经过softmax的交叉熵损失函数.

```
def softmax(Z):
    A = np.exp(Z)
    return A / np.sum(A, axis=1, keepdims=True)

def softmax_cross_entropy_loss(y: ndarray, label: ndarray):
    predict = softmax(y)
    loss = -np.sum(label * np.log(predict), axis=1)
    grad = (predict - label) / y.shape[0]
    return np.mean(loss), grad
```

训练

由于数据集中含有较大噪声, 因此采用betch方式来使训练更加稳定.

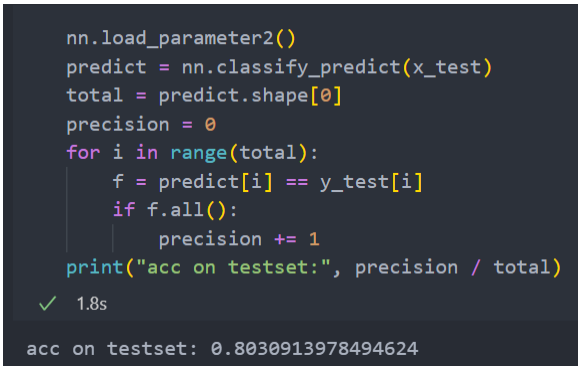
```
for i in range(0, len(XX), batch_size):
    x = np.array(XX[i : i + batch_size])
    label = np.array(YY[i : i + batch_size])
    label = label.reshape(label.shape[0], -1)
    predict = nn.forward(x)

    loss, grad = loss_function(predict, label)

    epoch_loss += loss
    nn.backward(grad)
    nn.update_parameters(learning_rate)
```

超参数为

```
epochs = 100
learning_rate = 1e-4
batch_size = 64
```

测试

在原始数据集中切了`20%`作为测试集, 结果为



准确率达到`80.31%`

2.3 调参

人人都躲不过沦落为调参侠的一天, 还好这个实验较小, 跑一次也不久, 可以快速得到不同超参下的数据.

成熟一点的方法有贝叶斯搜索之流, 但杀鸡焉用牛刀? 我直接人肉网格搜索.

拟合sin曲线

神经元数目

在200个epochs, 使用sigmoid激活函数, learning rate = 0.01时, 当神经元个数从100增加到250时, 拟合后测试平均误差不断增大. 说明随着网络参数增多, 可能了出现过拟合的情况.

学习率

我是想达到要求就好, 并且想迅速解决任务不要跑过多的epoch(毕竟是在CPU上面运行), 就只在0.1和1e-2上面进行测试. 测下来发现0.01比较好.

手写汉字分类

神经元数目

在100个epochs, batch_size = 64, learning_rate = 2e-4下, 神经元个数从80逐渐增加到250, 模型预测准确率逐渐增大, 说明随着模型参数增加, 分类能力增强; 当神经元个数从250增加到300时, 准确率反而降低, 说明模型出现过拟合. 对于这个数据集, 不适宜使用过于复杂的网络结构.

学习率

当learning rate = [1e-3, 1e-2]时, 网络经过训练后结果测试acc很低, 而误差减小不明显; 原因是学习率过大, 导致结果在极小值附近震荡, 无法达到极小值点;

当学习率为1e-5时损失函数下降的比较慢, 需要更多个epochs才能收敛到极小值;

根据以上测试, 学习率在1e-4数量级是比较合适的.

batch size

尝试了32, 64, 192的数值, 取64的batch size可以达到最好的结果. 一定大小的batch size是在权衡了总运行epoch和测试acc的结果.

3. 实验总结

前几个月看到Andrej Karpathy用C打造了Baby-Llama, 我佩服得五体投地. 没想到也轮到我"手撕"神经网络, 算是拙劣的模仿, 以及回到20年前的机器学习时代.