> *范峻铭 22121286*
>
> *作业完整代码同步在我的githubhttps://github.com/HatcherRobotics/intelligent-information-processing*

1、 分析为什么平方损失函数不适用于分类问题。

最小化平方损失函数本质上等同于在误差服从高斯分布的假设下的极大似然估计，在分类问题下大部分时候误差并不服从高斯分布。更直观地说，平方损失函数是通过真实值与预测值间的距离反映优化的程度，而在分类问题中常用one-hot的形式进行编码，其预测值与真实值间的距离没有实际意义。

2、 假设有N个样本$x(1),x(2),\cdots,x(N)$服从正态分布$N(\mu,\sigma^2)$,其中μ未知。

(1)使用最大似然估计来求解最优参数μ^{ML}。

$$
\begin{aligned}
&\because x(1),x(2),\cdots,x(N)\sim N(\mu,\sigma^2)\\
&\therefore p(x_i|\mu,\sigma)=\frac{1}{\sqrt{2\pi}\sigma}\cdot exp\{-\frac{(x_i-\mu)^2}{2\sigma^2}\}\\
&\therefore p(\boldsymbol{x}|\mu,\sigma)=\prod_{i=1}^{N}p(x_i|\mu,\sigma)=(\frac{1}{\sqrt{2\pi}\sigma})^N\cdot\prod_{i=1}^{N}exp\{-\frac{(x_i-\mu)^2}{2\sigma^2}\}\\
&\therefore log[p(\boldsymbol{x}|\mu,\sigma)]=Nlog\frac{1}{\sqrt{2\pi}\sigma}+\sum_{i=1}^{N}-\frac{(x_i-\mu)^2}{2\sigma^2}\\
&\because \mu^{ML}=\min_{\mu}log[p(\boldsymbol{x}|\mu,\sigma)]\\
&s.t.\ \frac{\partial log[p(\boldsymbol{x}|\mu,\sigma)]}{\partial\mu}=0\\
&\therefore\sum_{i=1}^{N}[-\frac{2(x_i-\mu)}{2\sigma^2}\cdot(-1)]=0\\
&\therefore\sum_{i=1}^{N}(x_i-\mu)=0\\
&\therefore\sum_{i=1}^{N}x_i-N\mu=0\\
&\therefore\mu=\frac{1}{N}\sum_{i=1}^{N}x_i\\
&\therefore\mu^{ML}=\frac{1}{N}\sum_{i=1}^{N}x_i
\end{aligned}
$$

(2)若参数μ为随机变量，并服从正态分布$N(\mu_0,\sigma_0^2)$，使用最大后验估计来求解最优参数μ^{MAP}。

已知似然为

$$p(x|\mu,\sigma)=(\frac{1}{\sqrt{2\pi}\sigma})^N\cdot\prod_{i=1}^{N}exp-\frac{(x_i-\mu)^2}{2\sigma^2}$$

由μ为随机变量且服从参数为μ_0,σ_0的正态分布

先验即为

$$p(\mu)=\frac{1}{\sqrt{2\pi\sigma_0}}\cdot exp(-\frac{(\mu-\mu_0)^2}{2\sigma_0^2})$$

由贝叶斯公式知

$$
\begin{aligned}
&后验p(\mu|x,\sigma)\propto p(x|\mu,\sigma)\cdot p(\mu)\\
&\therefore log(p(\mu|x,\sigma))\propto log(p(x|\mu,\sigma))+log(p(\mu))\\
&\therefore log(p(\mu|x,\sigma))\propto Nlog(\frac{1}{\sqrt{2\pi}\sigma})+\sum_{i=1}^{N}-\frac{(x_i-\mu)^2}{2\sigma^2}+log\frac{1}{\sqrt{2\pi\sigma_0}}-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}\\
&令\frac{\partial log(p(\mu|x,\sigma))}{\partial\mu}=\sum_{i=1}^{N}-\frac{2(x_i-\mu)}{2\sigma^2}\cdot(-1)-\frac{2(\mu-\mu_0)}{2\sigma^2}=0\\
&\therefore\sum_{i=1}^{N}\frac{x_i-\mu}{\sigma^2}=\frac{\mu-\mu_0}{\sigma_0^2}\\
&\therefore\frac{\sum_{i=1}^{N}x_i-N\cdot\mu}{\sigma^2}=\frac{\mu-\mu_0}{\sigma_0^2}\\
&\therefore\sigma_0^2(\sum_{i=1}^{N}x_i-N\mu)=\sigma^2(\mu-\mu_0)
\end{aligned}
$$

$\therefore \mu^{MAP} = \frac{\sigma_0^2 \sum_{i=1}^{N} x_i + \sigma^2 \mu_0}{N\sigma_0^2 + \sigma^2}$

3、 在习题2-6中，证明当$N \to \infty$ 时，最大后验估计趋向于最大似然估计。

由最大似然估计得到的μ值为

$\mu^{ML} = \frac{1}{N} \sum_{i=1}^{N} x_i$

当$N \to \infty$时即

$\mu^{ML} = \lim_{N \to +\infty} \frac{1}{N} \sum_{i=1}^{N} x_i$

对最大后验估计，当$N \to \infty$时

$$\mu^{MAP} = \lim_{N \to +\infty} \frac{\lim_{N \to +\infty} \sigma_0^2 \sum_{i=1}^{N} x_i + \sigma^2 \mu_0}{N\sigma_0^2 + \sigma^2} = \frac{\lim_{N \to \infty} \sigma_0^2 \frac{1}{N} \cdot \sum_{i=1}^{N} x_i + \lim_{N \to +\infty} \frac{\sigma^2 \mu_0}{N}}{\sigma_0 + \lim_{N \to +\infty} \frac{\sigma^2}{N}} = \frac{\sigma_0^2}{\sigma_0^2} \lim_{N \to +\infty} \frac{1}{N} \sum_{i=1}^{N} x_i = \lim_{N \to +\infty} \frac{1}{N} \sum_{i=1}^{N} x_i = \mu^{ML}$$

$\therefore$ 当$N \to \infty$ 时，最大后验估计趋向于最大似然估计

4、 在 Softmax 回归的风险函数（公式 (3.39)）中，如果去掉正则化项会有什么影响?

Softmax 由指数函数组成，计算结果会非常大，可能会产生溢出；会导致参数矩阵W中，对应每个类别的矩阵向量都非常大；使用部分训练集进行训练时，容易出现过拟合的现象。

5、利用数学仿真验证习题4的结论

利用MNIST数据集完成Softmax回归即多分类任务。

```
mnist_train = torchvision.datasets.MNIST(root='~/test/Datasets/MNIST',train=True, download=True, transform=transforms.ToTensor())
mnist_test = torchvision.datasets.MNIST(root='~/test/Datasets/MNIST',train=False, download=True, transform=transforms.ToTensor())
batch_size = 256
train_iter = torch.utils.data.DataLoader(mnist_train, batch_size=batch_size,shuffle=True, num_workers=2)
test_iter = torch.utils.data.DataLoader(mnist_test, batch_size=batch_size,shuffle=False, num_workers=2)
```

建立网络模型，根据数据集的输入输出选择一层全连接层，经过激活函数后再加上Softmax。全连接层的输入维度为784(28x28),输出维度为10。激活函数选择Sigmoid函数。Softmax将神经网络的输出限制在(0,1)之间，即将输出变成概率分布的形式。

```
class My_SoftmaxNet(nn.Module):
    def __init__(self,num_inputs,num_outputs):
        super().__init__()
        self.num_inputs = num_inputs
        self.num_outputs = num_outputs
        self.w = nn.Parameter(torch.normal(0, 0.01, size=(self.num_inputs, self.num_outputs), requires_grad=True))
        self.b = nn.Parameter(torch.zeros(self.num_outputs, requires_grad=True))
        self.sigmoid = nn.Sigmoid()
        for param in self.parameters():
            if param.dim() > 1:
                nn.init.xavier_uniform_(param)

    def softmax(self,x):
        X_exp = torch.exp(x)
        partition = X_exp.sum(1, keepdim=True)
        return X_exp / partition  # 这里应用了广播机制

    def forward(self,x):
        initial_output = self.sigmoid(torch.mm(x.view(-1,self.num_inputs),self.w)+self.b)
        softmax_output = self.softmax(initial_output)
        return softmax_output
```

损失函数选择交叉熵损失函数，优化器选择小批量随机梯度下降，学习率设为0.3。

L2正则化可以由全值衰减(weight decay)表示，对超参数wd选择不同的值进行实验分析，分别取1e-3,1e-2,1e-1和0。

当wd取1e-3时，训练的损失与准确率如下：

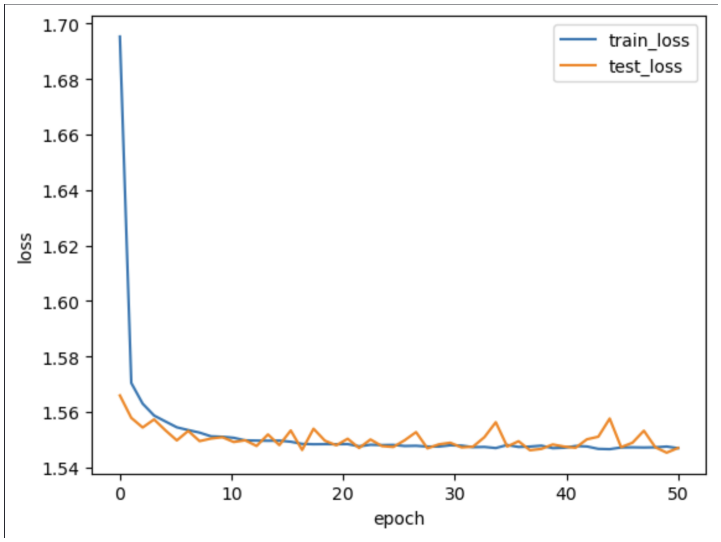

图1-1 wd=1e-3 loss

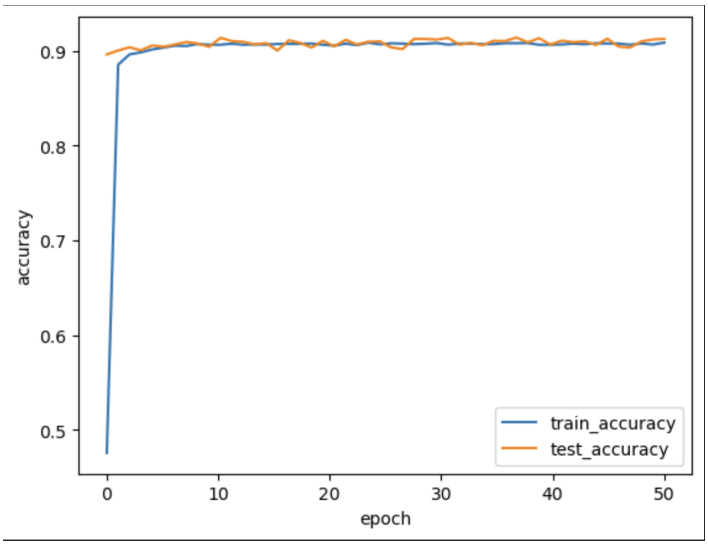

图1-2 wd=1e-3 accuracy

当wd取1e-2时，训练的损失与准确率如下：

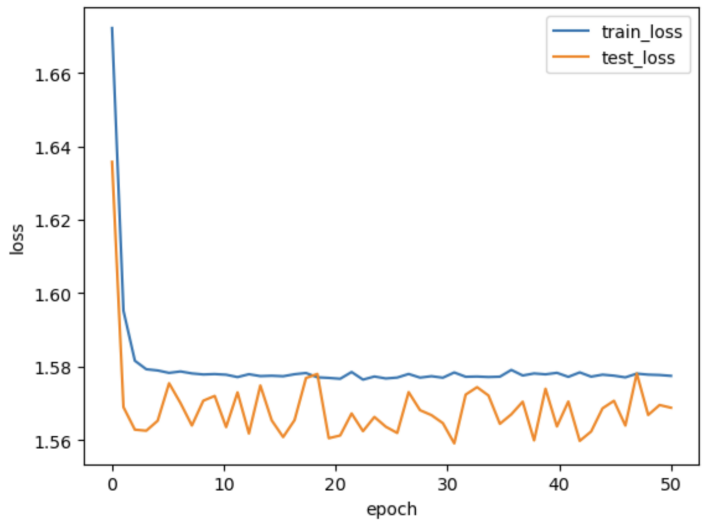

图2-1 wd=1e-2 loss

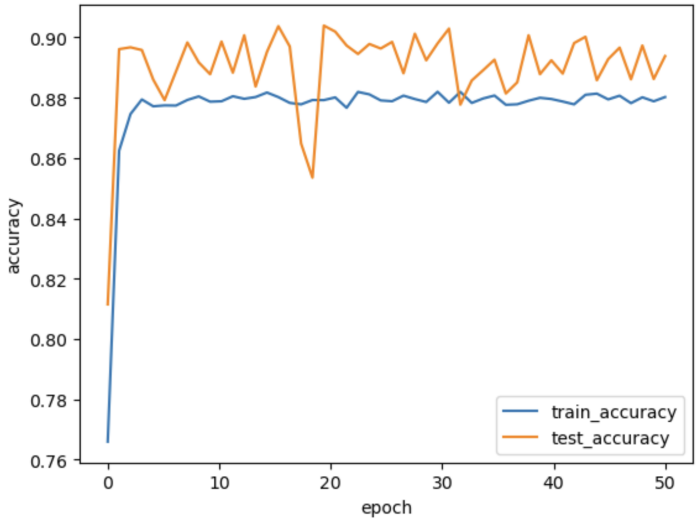

图2-2 wd=1e-2 accuracy

当wd取1e-1时，训练的损失与准确率如下：

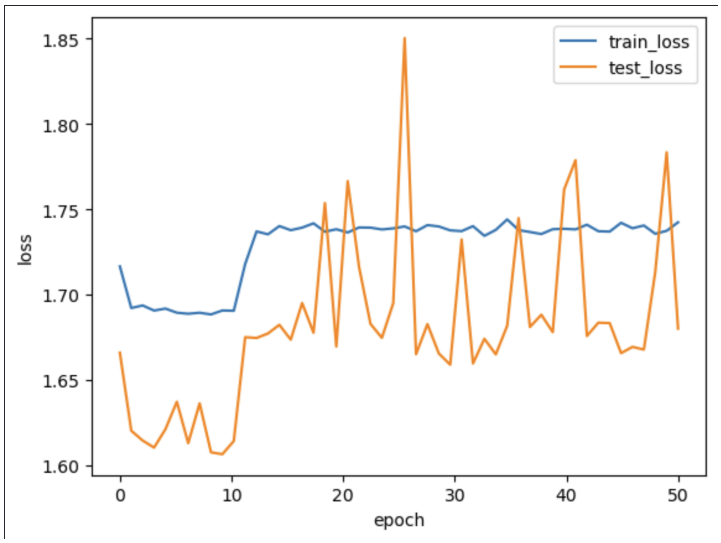

图3-1 wd=1e-1 loss

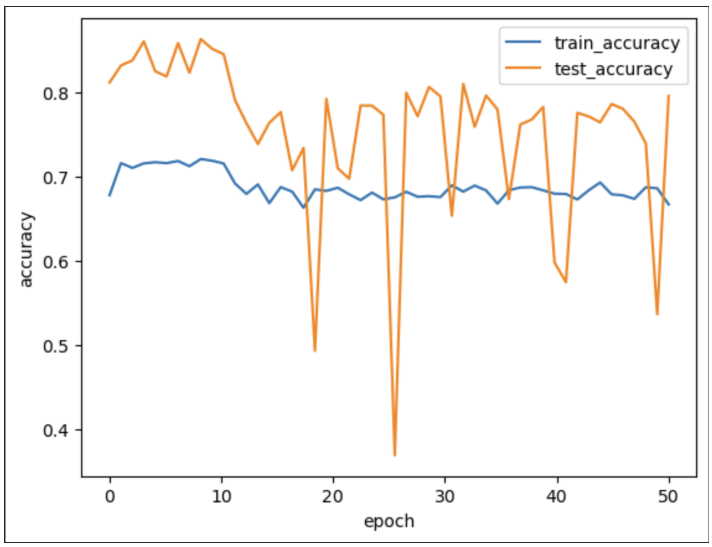

图3-2 wd=1e-1 accuracy

当wd取0时，即不进行正则化操作，训练的损失与准确率如下：

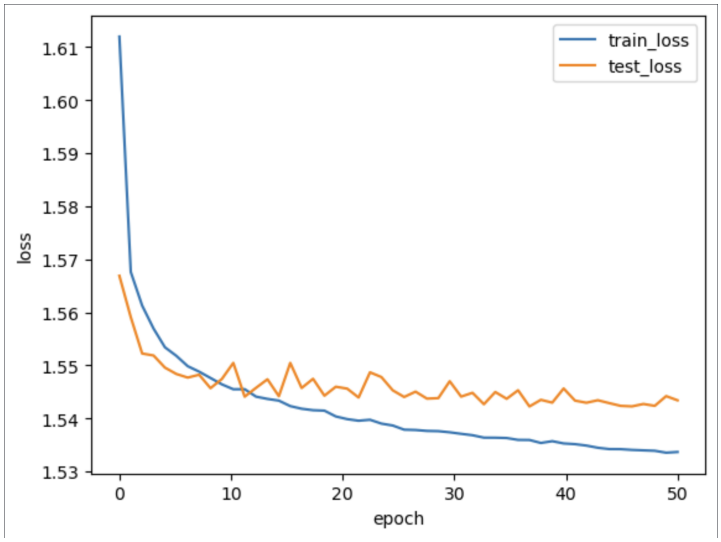

图4-1 wd=0 loss

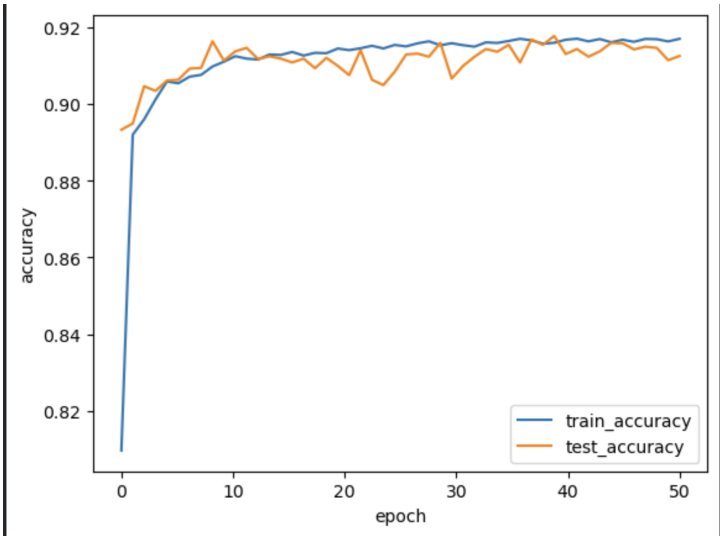

图4-2 wd=0 accuracy

通过调整权重衰减的超参数可以得到以下结论：

1.当参数值过大(>0.1)时，会影响正常的梯度下降，对损失函数的伤害过大，最终导致模型无法收敛。

2.不使用正则化，由于MNIST数据集本身数据量较大，较难发生过拟合的情况，虽然模型收敛达到了较好的准确率，但最终的训练结果中测试集的准确率低于训练集。

3.当正则化的取值较合理时，如0.001,0.01时，模型训练时具备了良好的对抗过拟合的能力，最终的训练结果中测试集的精度高于训练集。

4.如图5所示，不使用权重衰减时，训练出的参数w的值会比较大，加入权值衰减后权重变得非常小。

```
Parameter containing:
tensor([[ 0.0233,  0.0620,  0.0852,  ..., -0.0761, -0.0468, -0.0421],
        [ 0.0456, -0.0509, -0.0645,  ..., -0.0798, -0.0010,  0.0363],
        [ 0.0369, -0.0183, -0.0393,  ..., -0.0144,  0.0724, -0.0670],
        ...,
        [ 0.0787,  0.0157, -0.0122,  ..., -0.0494,  0.0248, -0.0831],
        [-0.0428, -0.0755, -0.0499,  ...,  0.0125, -0.0112, -0.0628],
        [ 0.0237, -0.0478,  0.0007,  ...,  0.0470,  0.0029, -0.0176]],
       device='cuda:0', requires_grad=True)
Parameter containing:
tensor([-12.1397,  -3.0252,  -5.5441, -13.8110,  -2.8872,   5.0133, -10.9611,
          1.3444, -39.3126, -16.8312], device='cuda:0', requires_grad=True)
```

图5-1 wd=0 权重值

```
Parameter containing:
tensor([[ 1.5463e-17,  4.0000e-17,  2.9497e-19,  ..., -1.1876e-18,
          3.7682e-17,  1.3209e-17],
        [-3.6438e-17, -2.4142e-19, -3.5111e-17,  ..., -2.7933e-17,
         -3.1354e-18,  3.4505e-17],
        [ 2.0326e-18, -3.8919e-17,  2.8653e-17,  ...,  2.6196e-17,
         -2.1676e-17,  1.5811e-17],
        ...,
        [-5.6963e-18, -1.6546e-17, -1.2005e-17,  ..., -3.7526e-17,
         -3.3033e-17,  2.4133e-17],
        [ 6.0814e-18,  2.0321e-17,  1.3412e-17,  ...,  1.8462e-17,
          1.9203e-17,  1.7182e-17],
        [-4.0216e-17, -1.4743e-17, -9.9098e-18,  ..., -3.1342e-17,
          3.6880e-17,  2.2221e-17]], device='cuda:0', requires_grad=True)
Parameter containing:
tensor([ -6.5197,  -1.6623,  -4.4290,  -7.1528,  -1.5139,   3.4251,  -4.4208,
          0.1535, -21.8557,  -7.2762], device='cuda:0', requires_grad=True)
```

图5-2 wd=0.01 权重值