MNIST Classification with MLP

于雨琛　经 02 计 08　2020011846

2023.03.15

I 核心代码

I.1 SGD

```
class SGD():
    ...
    layer.diff_W = layer.grad_W + self.weightDecay * layer.W # weightdecay
    layer.diff_W = -self.learningRate * layer.diff_W # 学习率
    layer.diff_b = layer.grad_b + self.weightDecay * layer.b
    layer.diff_b = -self.learningRate * layer.diff_b
    ...
```

I.2 EuclideanLoss

```
def forward(self, logit, gt):
    self.loss = np.sum((logit - gt)**2) / (2 * logit.shape[0]) # 直接计算 MSE
    self.acc = np.mean(np.argmax(logit, axis=1) == np.argmax(gt, axis=1)) # 准确率
    self.input = logit # 保存到类中
    self.label = gt # 保存到类中
    return self.loss
def backward(self):
    return (self.input - self.label) / self.input.shape[0] # 导数
```

I.3 CrossEntropyLoss

```
def forward(self, logit, gt):
    self.label = gt # 保存到类中
    self.output = np.exp(logit) / np.sum(np.exp(logit), axis=1, keepdims=True) # sigmoid
    self.loss = -np.sum(gt * np.log(self.output + EPS)) / gt.shape[0] # 计算 loss
```

```
    predictions = np.argmax(self.output, axis=1) # 预测
    self.acc = np.mean(predictions == np.argmax(self.label, axis=1)) # 得到预测结果
    return self.loss
backward(self):
    return (self.output - self.label) / self.label.shape[0] # 导数
```

I.4 FullyConnectedLayer

```
def forward(self, logit, gt):
    self.input = Input
    output = Input @ self.W + self.b # 点乘
    self.output = output # 记录
    return output
def backward(self, delta):
    self.grad_W = np.dot(self.input.T, delta) # 求梯度
    self.grad_b = np.sum(delta, axis=0)
    grad_input = np.dot(delta, self.W.T)
    return grad_input
```

I.5 ReluLayer

```
def forward(self, Input):
    self.input = Input
    return np.maximum(0, Input) # 取大于等于 0
def backward(self, delta):
    return delta * (self.input >= 0) # 导数
```

I.6 SigmoidLayer

```
def forward(self, Input):
    output = 1 / (1 + np.exp(-Input)) # sigmoid
    self.output = output
    return output
def backward(self, delta):
    return delta * self.output * (1 - self.output) # 导数
```

II 训练与测试

II.1 训练超参数：

batch_size = 100 max_epoch = 5 init_std = 0.01 learning_rate_SGD = 0.1
weight_decay = 0.001 disp_freq = 50

II.2 训练准确率：

(1) Euclidean Loss and Sigmoid Activatio: 训练集 0.8931，验证集 0.9044，测试集 0.9035。

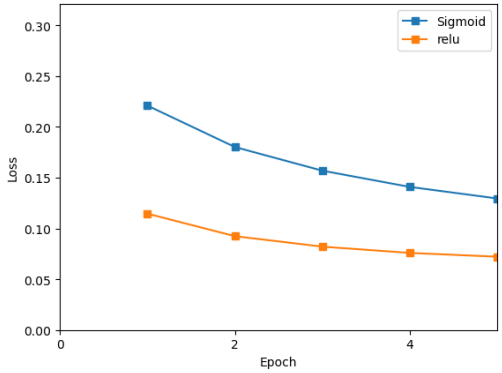
(2) Euclidean Loss and ReLU Activation: 训练集 0.9410，验证集 0.9460，测试集 0.9441。

(3) Softmax Cross-Entropy Loss and Sigmoid Activation: 训练集 0.9096，验证集 0.9176，测试集 0.9158。

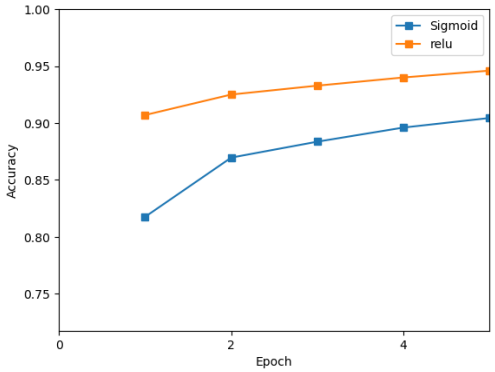
(4) Softmax Cross-Entropy Loss and ReLU Activation: 训练集 0.9609，验证集 0.9646，测试集 0.9608。

II.3 训练结果

Euclidean Loss

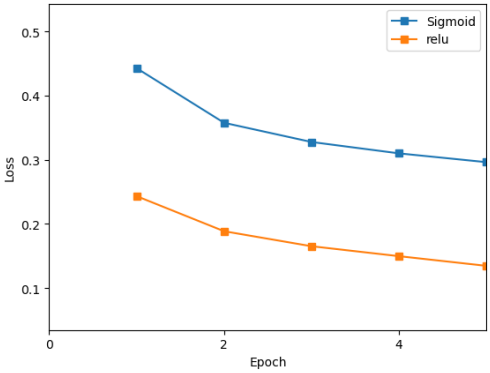


loss

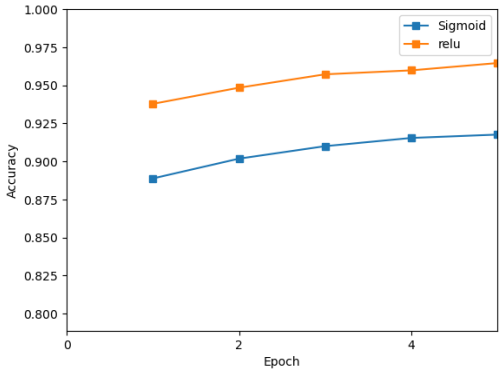


accuracy

Cross Entropy Loss



loss



accuracy

III 结果解释

III.1 ReLU vs. Sigmoid

可以看到，不论是在 Euclidean Loss 还是 Cross-Entropy Loss 中，使用 ReLU 激活函数的效果都好于 Sigmoid（准确率更高，损失更少）。即从本次训练与测试角度，ReLU 激活函数优于 Sigmoid 激活函数。这是由于以下几点：

第一，相比于 Sigmoid，ReLU 的运算更简单，没有幂运算，提高了运算速度；

第二，Sigmoid 反向传播的过程中，饱和区域非常平缓，接近于 0，容易出现梯度消失的问题，减缓收敛速度。Relu 的梯度大多数情况下是常数，不会出现梯度消失的问题，有助于损失持续降低。

ReLU 会使一部分神经元的输出为 0，这减少了隐层内部的相互依存关系，缓解了过拟合问题的发生，也起到了类似于 Dropout 的效果。

III.2 Euclidean vs. Cross- Entropy

可以看到，不论是在 ReLU Activation 还是 Sigmoid Activation 中，使用 Cross-Entropy 损失函数的效果都好于 Euclidean（准确率更高，损失更少）。即从本次训练与测试角度，采用交叉熵损失函数效果更好。这是由于以下几点：

第一，损失函数角度。在分类问题中，采用 onehot 编码，在这种情况下应该只关注分类正确与否，例如（0.8, 0.1, 0.1）与（0.8, 0.15, 0.5）应该是相同的。可以发现使用交叉熵函数时这两者确实相同，但使用均方误差则会有区别，这是不合理的。

第二，交叉熵函数默认数据分布满足多项式分布，均方误差函数默认数据分布满足高斯分布，在分类问题中多项式分布是更合理的假设。

IV 双隐层

IV.1 模型

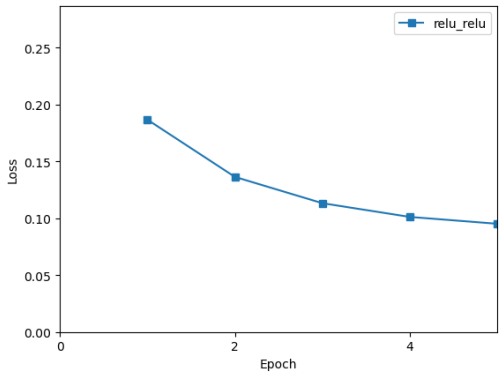
```
myMLP = Network()
myMLP.add(FCLayer(784, 256))
myMLP.add(ReLULayer())
myMLP.add(FCLayer(256, 128))
myMLP.add(ReLULayer())
myMLP.add(FCLayer(128, 10))
```

采用双隐层都是 ReLULayer，损失函数为交叉熵函数。训练超参数同上。

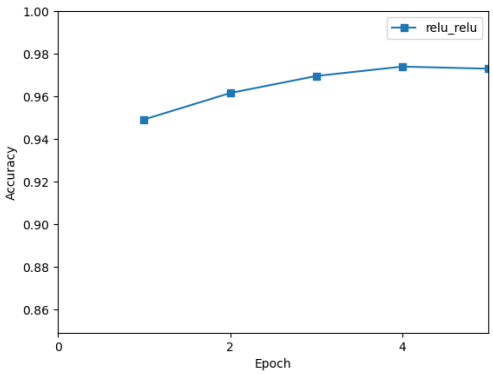
IV.2 训练准确率

训练集 0.9762，验证集 0.9730，测试集 0.9699。

IV.3 训练结果



loss



accuracy

IV.4 与单隐层比较

可以看到，上面双隐层的准确率达到了 0.97，比单隐层的表现要出色。我认为主要原因是双隐层的训练次数更多，隐藏层数多也导致了参数更多，从训练中的输出能够看到 loss 仍然在持续下降，准确率也在上升，说明还有进一步上升空间，如果训练轮数足够多可能单隐层和双隐层的区别会逐渐减小至无。

V 调参

V.1 增加 epoch

将训练轮数增加到 20 轮，其余不变，采用 ReLu+Cross-Entropy 的单隐层模型。

训练集 0.9823，验证集 0.9766，测试集 0.9755。

可以看到，随着训练轮数增加，训练准确率也有进一步增加。说明 5 轮还没有达到过拟合的地步，可以继续进步。

V.2 减小 batch_size

将 batch_size 变成 50，其余不变，采用 ReLu+Cross-Entropy 的单隐层模型。

训练集 0.9706，验证集 0.9714，测试集 0.9694。

可以看到，与之前相比有所提升但变化不大，这可能是因为训练轮数增加。

V.3 减小 weightdecay

将 weightdecay 变成 0，其余不变，采用 ReLu+Cross-Entropy 的单隐层模型。

训练集 0.9625，验证集 0.9658，测试集 0.9632。

可以看到，与之前相比，准确率有所提升，说明在这个问题中还没有达到过拟合阶段。