

LOSS

一、内容

在本部分将实现的Loss，CategoricalCrossentropy类（继承了Loss类）。本部分只实现forward method，反向传播将在后续加入。

二、代码

一、Loss父类

1. 实现

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

    # 统一通过调用calculate方法计算损失
    def calculate(self, prediction, y):
        # 对于不同的损失函数，通过继承Loss父类，并实现不同的forward方法。
        data_loss = np.mean( self.forward(prediction, y) )
        # 注意，这里计算得到的loss不作为类属性储存，而是直接通过return返回
        return data_loss
```

二、CategoricalCrossentropy类

1. 公式

$$L_i = - \sum_j y_{i,j} \log(\hat{y}_{i,j})$$

当预测属于A、B、C三个类的概率分别是0.7, 0.1、0.2, 其实类别为A, 测 L_i 计算如下。其中i表示对第i个sample计算得到的loss

$$L_i = - \sum_j y_{i,j} \log(\hat{y}_{i,j}) = -(1 \cdot \log(0.7) + 0 \cdot \log(0.1) + 0 \cdot \log(0.2)) = -(-0.35667494393873245 + 0 + 0) = 0.35667494393873245$$

2. 实现

```
class Loss_CategoricalCrossentropy(Loss):
    def __init__(self):
        pass

    def forward(self, y_pred, y_true):
        # 多少个样本
        n_sample = len(y_true)

        # 为了防止log(0), 所以以1e-7为左边界
        # 另一个问题是将置信度向1移动, 即使是非常小的值,
        # 为了防止偏移, 右边界为1 - 1e-7
        y_pred = np.clip(y_pred, 1e-7, 1 - 1e-7)

        loss = - np.log(y_pred)
        if len(y_true.shape) == 2: # 标签是onehot的编码
            loss = np.sum(loss * y_true, axis=1)
        elif len(y_true.shape) == 1: # 只有一个类别标签
            # 注意loss = loss[:, y_ture]是不一样的, 这样会返回一个矩阵
            loss = loss[range(n_sample), y_true]

        return loss
```

3. 实例

```
# 生成数据
X, y = spiral_data(samples=100, classes=3)
# 构建一个含三个神经元的Dense层实例
dense1 = Layer_Dense(2, 3)
# 构建ReLU激活函数
activation1 = Activation_ReLu()
# 构建一个含4个神经元的Dense层实例
dense2 = Layer_Dense(3, 4)
# 构建Softmax激活函数
activation2 = Activation_Softmax()
# 构建损失函数
loss = Loss_CategoricalCrossentropy()

# 前向传播
```

```

dense1.forward(X)
activation1.forward(dense1.output)
dense2.forward(activation1.output)
activation2.forward(dense2.output)
dataloss = loss.calculate(activation2.output, y)

# 输出结果
print('loss =', dataloss)

# 计算正确率
soft_output = activation2.output
# 返回最大confidence的类别作为预测类别
prediction = np.argmax(soft_output, axis=1)
# 如果y是onehot编码
if len(y.shape) == 2:
    # 将其变为只有一个标签类别
    y = np.argmax(y, axis=1)

accuracy = np.mean(prediction == y)
print("accuracy =", accuracy)

```

```

loss = 1.3862825751306984
accuracy = 0.36666666666666666

```

三、Binary Cross-Entropy

1. 公式

$$\begin{aligned}
 L_{i,j} &= (y_{i,j})(-\log(\hat{y}_{i,j})) + (1 - y_{i,j})(-\log(1 - \hat{y}_{i,j})) = \\
 &= -y_{i,j} \cdot \log(\hat{y}_{i,j}) - (1 - y_{i,j}) \cdot \log(1 - \hat{y}_{i,j})
 \end{aligned}$$

由于一个模型可以包含多个二进制输出，而且每个输出都不像交叉熵损失那样输出每个类别的confidence，所以在单个输出上计算的损失将是一个损失向量，其中每个输出都包含一个值。与CategoricalCrossentropy最大的不同是：

- CategoricalCrossentropy中的每个类别是互斥的，
- Binary Cross-Entropy中二进制输出是互斥的，但多个二进制之间不互斥，
- 例如：男女互斥，高矮互斥，但男女与高矮之间不互斥。

2. 实现

```

class Loss_BinaryCrossentropy(Loss):
    def __init__(self):
        pass

    def forward(self, y_pred, y_true):
        # 多少个样本
        n_sample = len(y_true)
        # 这里要特别注意，书上都没有写明
        # 当只有一对二进制类别时，y_pred大小为(n_sample, 1), y_ture大小为
(n_sample,)
        # (n_sample,)和(n_sample, 1)一样都可以广播，只是(n_sample,)不能转置
        # 所以下面的loss大小会变成(n_sample, n_sample)
        # 当有二对二进制类别时，y_pred大小为(n_sample, 2), y_ture大小为
(n_sample, 2)
        if len(y_true.shape) == 1: # y_true是个行向量
            y_true = y_true.reshape(-1, 1)
        # 为了防止log(0)，所以以1e-7为左边界
        # 另一个问题是将置信度向1移动，即使是非常小的值，
        # 为了防止偏移，右边界为1 - 1e-7
        y_pred = np.clip(y_pred, 1e-7, 1 - 1e-7)
        loss = - np.log(y_pred) * y_true - np.log(1 - y_pred) * (1 - y_true)
        # 这里的求平均和父类中的calculate求平均的维度不同
        # 这里是对多对的二进制求平均
        # calculate中的求平均是对每个样本可平均
        loss = np.mean(loss, axis=-1)
        return loss

```