COMBINE THE LOSS AND ACTIVATION FUNCTION

一、内容

在之前内容中已经实现了了Categorical Cross-Entropy函数和Softmax激活函数，但是还可以进一步来加速计算。这部分是因为两个函数的导数结合起来使整个代码实现更简单、更快。除此之外，Binary Cross-Entropy loss和Sigmoid也能结合。

二、Categorical Cross-Entropy loss and Softmax activation

公式

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

> 在Backpropagation的Softmax部分讲到了$\frac{\partial S_{i,j}}{\partial z_{i,k}}$的计算，且$\hat{y}_{i,j} = S_{i,j}$，所以有：

image-20230809092620943

$$\frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = \begin{cases} \hat{y}_{i,j} \cdot (1 - \hat{y}_{i,k}) & j = k \\ -\hat{y}_{i,j} \cdot \hat{y}_{i,k} & j \neq k \end{cases}$$

> 在Backpropagation的Categorical Cross-Entropy loss部分讲到了：

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \frac{\partial}{\partial \hat{y}_{i,j}}[-\sum_j y_{i,j} log(\hat{y}_{i,j})] = -\sum_j y_{i,j} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(\hat{y}_{i,j}) =$$

$$= -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j} = -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}}$$

> 综上有：

$$\frac{\partial L_i}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \hat{y}_{i,k} \cdot (1 - \hat{y}_{i,k}) - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} (-\hat{y}_{i,j}\hat{y}_{i,k})$$

$$= -y_{i,k} \cdot (1 - \hat{y}_{i,k}) + \sum_{j \neq k} y_{i,j}\hat{y}_{i,k} = -y_{i,k} + y_{i,k}\hat{y}_{i,k} + \sum_{j \neq k} y_{i,j}\hat{y}_{i,k} =$$

$$= -y_{i,k} + \sum_j y_{i,j}\hat{y}_{i,k} = -y_{i,k} + \hat{y}_{i,k} = \hat{y}_{i,k} - y_{i,k}$$

> **注意：这里的**z**是**Softmax**的**input，L**是**Categorical Cross-Entropy**的**output

实现

```
class Activation_Softmax_Loss_CategoricalCrossentropy():
    def __init__(self):
        self.activation = Activation_Softmax()
        self.loss = Loss_CategoricalCrossentropy()

    # 注意：Activation_Softmax_Loss_CategoricalCrossentropy类中是调用forward计算loss
    # 因为它没有继承Loss类
    def forward(self, input, y_true):
        self.activation.forward(input)
        # 该类的output属性应该是Activation_Softmax()的输出
        self.output = self.activation.output
        # 该类返回的是loss
        return self.loss.calculate(self.output, y_true)

    # 其实y_pred一定等于self.output，但为了与之前代码一致
    def backward(self, y_pred, y_true):
        # 样本个数
        n_sample = len(y_true)
        if len(y_true.shape) == 2: # onehot编码
            # 直接套公式
            self.dinput = y_pred - y_true
        elif len(y_true.shape) == 1: # 只有一个类别
            self.dinput = y_pred.copy()
            # 需将每一行中y_true类别（索引）中的-1，其它-0（不操作）
            self.dinput[range(n_sample), y_true] -= 1
        # 每个样本除以n_sample，因为在优化的过程中要对样本求和
        self.dinput = self.dinput / n_sample
```

实例

```
##############################################
softmax_outputs = np.array([[0.7, 0.1, 0.2],[0.1, 0.5, 0.4],[0.02, 0.9, 0.08]])
class_targets = np.array([0, 1, 1])
softmax_loss = Activation_Softmax_Loss_CategoricalCrossentropy()
softmax_loss.backward(softmax_outputs, class_targets)
dvalues1 = softmax_loss.dinput

activation = Activation_Softmax()
activation.output = softmax_outputs
loss = Loss_CategoricalCrossentropy()
loss.backward(softmax_outputs, class_targets)
activation.backward(loss.dinput)
```

```
dvalues2 = activation.dinput

print('Gradients: combined loss and activation:')
print(dvalues1)
print('Gradients: separate loss and activation:')
print(dvalues2)
########################################################
```

```
Gradients: combined loss and activation:
[[-0.1         0.03333333  0.06666667]
 [ 0.03333333 -0.16666667  0.13333333]
 [ 0.00666667 -0.03333333  0.02666667]]
Gradients: separate loss and activation:
[[-0.1         0.03333333  0.06666667]
 [ 0.03333333 -0.16666667  0.13333333]
 [ 0.00666667 -0.03333333  0.02666667]]
```

> 将Activation和loss分开，或都合并都实现了相同的结果。

```
def f1():
    softmax_loss = Activation_Softmax_Loss_CategoricalCrossentropy()
    softmax_loss.backward(softmax_outputs, class_targets)
    dvalues1 = softmax_loss.dinput
def f2():
    activation = Activation_Softmax()
    activation.output = softmax_outputs
    loss = Loss_CategoricalCrossentropy()
    loss.backward(softmax_outputs, class_targets)
    activation.backward(loss.dinput)
    dvalues2 = activation.dinput

t1 = timeit(lambda: f1(), number=10000)
t2 = timeit(lambda: f2(), number=10000)
print(t2/t1)
```

```
3.6972609490061443
```

> 可以看到，当两种实现方法重复10000次以后，所用时间接近4倍。

三、Sigmoid and Binary Cross-Entropy Loss

> **这部分内容在书中并没有， 是我自己根据理解后，推导公式和编程实现的，并不代表完全正确。将在更深入学习后勘误。**

公式

> 参照Sigmoid和Binary Cross-Entropy的求代公式有（第个样本下标i，省去）：

$$\frac{\partial L}{\partial \hat{y}_j} = -\frac{1}{J}\left(\frac{\partial y_j}{\partial \hat{y}_j} - \frac{1-\partial y_j}{1-\partial \hat{y}_j}\right)$$

$$\frac{\partial \sigma_j}{\partial z_j} = \sigma_j(1-\sigma_j)$$

因为，Sigmoid的输出σ就是Binary Cross-Entropy的输入$\hat{y}$，写成矩阵形式，$\frac{\partial L}{\partial z}$和$\frac{\partial L}{\partial \hat{y}}$是行向量，$\frac{\partial \sigma}{\partial z}$是对角方阵。

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \sigma}{\partial z}$$

对每个标量进行计算有：

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \sigma_j}{\partial z_j} = \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_j} = -\frac{1}{J}(\frac{\partial y_j}{\partial \hat{y}_j} - \frac{1 - \partial y_j}{1 - \partial \hat{y}_j})\hat{y}_j(1 - \hat{y}_j) = \frac{\hat{y}_j - y_j}{J}$$