ACTIVATION

一、内容

在本部分将实现常用的Activation Function，例如：ReLu（Rectified Linear units Function）、Softmax、Sigmoid。本部分只实现forward method，反向传播将在后续加入

二、代码

一、Sigmoid

1. 公式

其中$z_{i,j}$表示这个激活函数的输入，$\sigma_{i,j}$表示单个输出值。索引i表示当前样本，索引j表示当前样本中的当前输出。$\sigma_{i,j}$可理解成对第j对类别，例如猫狗分类中狗类别的confidence(置信度)。当然，一个模型可能要对多对类别分类，例如：高矮、胖瘦等。Sigmoid用于二分类

$$\sigma_{i,j} = \frac{1}{1 + e^{-z_{i,j}}}$$

2. 实现

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

      def forward(self, input):
            # input的大小是nx1，n是Activation输入的sample数量，每个sample只有一个维度。
            # 所以前一个hidden layer必须是Layer_Dense(n, 1)
            self.output = 1 / ( 1 + np.exp(-input) )
```

3. 实例

```
# 生成数据
X, y = spiral_data(samples=100, classes=2)
# 构建一个含三个神经元的Dense层实例
dense = Layer_Dense(2,1)
# 构建Softmax激活函数
activation1 = Activation_Softmax()

# 前向传播
dense.forward(X)
activation1.forward(dense.output)
# 输出结果
print(activation1.output[:5])
```

```
[[0.5       ]
 [0.50001083]
 [0.50002728]
 [0.50004487]
 [0.50005302]]
```

输出是nx1大小，表示二分类别的confidence(置信度)

二、ReLu

1. 公式

$$y = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

2. 实现

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

    def forward(self,input):
        self.output = np.maximum(0,input)
```

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
dense = Layer_Dense(2,3)
# 构建Softmax激活函数
activation1 = Activation_ReLu()

# 前向传播
dense.forward(X)
activation1.forward(dense.output)
# 输出结果
print(activation1.output[:5])
```

```
[[0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.13083886e-04 1.02476287e-04 0.00000000e+00]
 [1.34458443e-04 9.74786515e-05 0.00000000e+00]
 [3.10216678e-04 2.70888477e-04 0.00000000e+00]
 [5.09942272e-04 5.18245929e-04 3.90370390e-04]]
```

三、Softmax

Softmax函数是一种将j个实数向量转换为j个可能结果的概率分布的函数。索引i表示当前样本，索引j表示当前样本中的当前输出，$S_{i,j}$表示j个可能结果的概率。

1. 公式

$$S_{i,j} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}$$

2. 实现

softmax函数对输入值非常敏感，而且很容易产生极端的概率分布。这可能会导致模型过度自信地预测某个类别，而忽略了其他类别的可能性。为了避免这种情况，我们可以在进行指数运算之前，从输入值中减去最大值。这样做不会改变softmax函数的结果，因为分子和分母都会被同一个常数除以。但是，这样做可以使输入值更小，从而避免指数运算产生过大的数字。

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

    def forward(self,input):
        # 要有keepdims=True参数设置
        # 如没有设置，则np.max(input, axis=1)后的列向量会变成行向量，
        # 而行向量长度不与input的每一行长度相同，
        # 则无法广播
        # 进行指数运算之前，从输入值中减去最大值，使输入值更小，从而避免指数运算产生过大的数字
        self.output = np.exp(input - np.max(input, axis=1, keepdims=True))
        self.output = self.output / np.sum(self.output, axis=1, keepdims=True)
```

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
dense = Layer_Dense(2,3)
# 构建Softmax激活函数
activation1 = Activation_Softmax()

# 前向传播
dense.forward(X)
activation1.forward(dense.output)
# 输出结果
print(activation1.output[:5])
```

```
[[0.33333333 0.33333333 0.33333333]
 [0.33334212 0.33331066 0.33334722]
 [0.33335077 0.33331317 0.33333605]
 [0.3333612  0.33328698 0.33335181]
 [0.33336365 0.33331746 0.33331889]]
```