通过 numpy 实现神经网络

Author：王豪(karlieswift)

$\delta^{(l)}$	第 L 层的误差项。反映了不同神经元对网络能力的贡献程度，从而比较好地解决了贡献度分配问题（Credit Assignment Problem，CAP）．
$z^{(l)}$	第 L 层的净输入值
a^{l}	第 L 层的经过非线性函数的值 $a^{l} = f_{l}^{'}(z^{(l)})$，也就是下一层的输入 $X^{(l+1)}$
$Loss(y,t)$	单个样本的损失函数。y,t 为列向量，分别代表预测值和真实/目标值
$\odot$	Hadamard 积，对应元素相乘
$\bullet$	np.dot(X,Y) 矩阵运算

$$\delta^{(l)} = \frac{\partial Loss(y,t)}{\partial z^{(l)}} = f_{l}^{'}(z^{(l)}) \odot ((W^{(l+1)})^{(T)} \bullet \delta^{(l+1)}) \quad (1)$$

因为 $f_{l}^{'}(z^{(l)})=f(a^{l}) \odot [1-f(a^{l})]=f(X^{(l+1)}) \odot [1-f(X^{(l+1)})]$ (2)

对于 sigmoid 函数，其偏导数可以用函数值直接表示，所以可以直接将导数 $f_{l}^{'}(z^{(l)})$ 改写成 $f(X^{(l+1)}) \odot [1-f(X^{(l+1)})]$，这样可以在反向传播的 for 循环里直接用下一层的数据，而不是用上一层的数据(因为反向传播的参数是从最后一次往前传，在求第 L 层的 $\delta^{(l)}$ 时，是通过上一层的参数得到的，当前层的净输入 $Z^{(l)}$ 是没有记录的，当然可以在前向传播的时候记录净输入值 $Z^{(l)}$，但在 for 循环的时候需要单独处理数组越界即-1 的位置)。

$$\delta^{(l)} = \frac{\partial Loss(y,t)}{\partial z^{(l)}} = f(X^{(l+1)}) \odot [1-f(X^{(l+1)})] \odot ((W^{(l+1)})^{(T)} \bullet \delta^{(l+1)}) \quad (3)$$

隐藏层的参数梯度:
参数 W 和 b 的梯度

$$\frac{\partial Loss(y,t)}{\partial W^{(l)}} = \delta^{(l)} \bullet (a^{(l-1)})^{T} = \delta^{(l)} \bullet (X^{(l)})^{T} \quad (4)$$

$$\frac{\partial Loss(y,t)}{\partial b^{(l)}} = \delta^{(l)} \quad (5)$$

输出(最后一层)的参数更新:

$Loss(y,t)=\frac{1}{2}(y-t)^2$ (6)

因为最后一层的时候为 $y=a^{end}=f(z^{end})$

所以 $\delta^{(end)}=\frac{\partial Loss(y,t)}{\partial z^{(end)}}=\frac{\partial(\frac{1}{2}(y-t)^2)}{\partial z^{(end)}}=\frac{\partial(\frac{1}{2}(f(z^{(end)})-t)^2)}{\partial z^{(end)}}=(f(z^{(end)})-t)(f^{'}(z^{(end)}))$

$=(y-t)(f^{'}(z^{(end)}))=(y-t)(a^{(end)})(1-a^{(end)})$ (7)

所以 $\delta^{(end)}=\frac{\partial Loss(y,t)}{\partial z^{(end)}}=(y-t)(f^{'}(z^{(end)}))=(y-t)(a^{(end)})(1-a^{(end)})$ (8)

综上所述最终会用到的公式:

求最终输出层的误差项:

$\delta^{(end)}=\frac{\partial Loss(y,t)}{\partial z^{(end)}}=(y-t)(f^{'}(z^{(end)}))=(y-t)(a^{(end)})(1-a^{(end)})$ (8)

求隐藏层的误差项:

$\delta^{(l)}=\frac{\partial Loss(y,t)}{\partial z^{(l)}}=f(X^{(l+1)})\odot[1-f(X^{(l+1)})]\odot((W^{(l+1)})^{(T)}\bullet\delta^{(l+1)})$ (3)

求梯度:

$\frac{\partial Loss(y,t)}{\partial W^{(l)}}=\delta^{(l)}\bullet(a^{(l-1)})^T=\delta^{(l)}\bullet(X^{(l)})^T$ (4)

$\frac{\partial Loss(y,t)}{\partial b^{(l)}}=\delta^{(l)}$ (5)

```
"""
 Env: /anaconda3/python3.7
 Time: 2021/8/10 14:46
 Author: karlieswfit
 File: NN.py
 Describe: 通过 numpy 实现一个简单的神经网络 并通过 sklearn 鸢尾花数据集进行分类预测

"""

import numpy as np

class Activation_Function:
    def forward(self, X):
        return 1 / (1 + np.exp(-X))
    def backward(self, X):
        return X * (1 - X)
```

```python
# 定义一个单层隐藏的神经网络层
class SimpleNeuralNetwork:
    def __init__(self, input_size, output_size, activationFuction):
        self.input_size = input_size
        self.output_size = output_size
        self.activationFuction = activationFuction
        self.W = np.random.uniform(low=-0.5, high=0.5, size=(output_size, input_size))
        self.b = np.zeros(shape=(output_size, 1))

    def forward(self, X):
        self.input = X
        Z = np.dot(self.W, X) + self.b
        self.output = self.activationFuction.forward(Z)
        return self.output

    def backward(self, delta):
        # 计算上一层的 delta
        self.delta = self.activationFuction.backward(self.input) * np.dot(self.W.T, delta)
        self.W_grad = np.dot(delta, self.input.T)
        self.b_grad = delta

class NeuralNetworks:
    def __init__(self, layers):
        self.layers = []
        for i in range(len(layers) - 1):
            self.layers.append(SimpleNeuralNetwork(layers[i], layers[i + 1],
Activation_Function()))

    def train(self, input, target, lr):
        self.predict(input)
        self.caculated_gradient(target=target)
        self.update_W_b(lr=lr)

    def predict(self, input):
        for i in range(len(self.layers)):
            output = self.layers[i].forward(input)
            input = output
        return output

    def caculated_gradient(self, target):
        delta = (self.layers[-1].output - target) * self.layers[-
1].activationFuction.backward(self.layers[-1].output)
        for layer in self.layers[::-1]:
```

```
            layer.backward(delta)
            delta = layer.delta

    def update_W_b(self, lr):
        for layer in self.layers:
            layer.W -= layer.W_grad * lr
            layer.b -= layer.b_grad * lr

    def loss(self,y_pre,targer):
        return ((y_pre-targer)**2).sum()/2

def one_hot(target,n):
    list=[]
    for i in target:
        inner_list = []
        for j in range(n):
            if i==j:
                inner_list.append(1)
            else:inner_list.append(0)
        list.append(inner_list)
    return np.array(list).reshape(len(target),-1)


def train():
    from sklearn.datasets import load_iris
    epoch=400
    data = load_iris().data
    targets = load_iris().target
    targets=one_hot(target=targets,n=3)
    model=NeuralNetworks([4,6,3])
    for index in range(epoch):
        loss=0
        for (input,target) in zip(data,targets):
            model.train(input.reshape(-1,1),target.reshape(-1,1),lr=0.05)
            loss+=model.loss(model.predict(input.reshape(-1,1)),target.reshape(-1,1))
        if index%10==0:
            print('第{index}次迭代的 loss:{loss}'.format(index=index,loss=loss))

    return model


def test(model):
    from sklearn.datasets import load_iris
    data = load_iris().data
```

```python
    targets = load_iris().target
    sum=0
    for i in range(len(data)):
        y_pre=model.predict(data[i].reshape(-1,1)).argmax()
        sum+=(y_pre==targets[i])

    return sum/(len(data))


if __name__ == '__main__':
    model=train()
    print(test(model))


"""
结果：
第 360 次迭代的 loss:0.4684289119392209
第 370 次迭代的 loss:0.47037902629258793
第 380 次迭代的 loss:0.4748604631962376
第 390 次迭代的 loss:0.4822365144828569
0.96

"""
```