2024 Deep Learning HW1

1. Tensorflow2.0 exercise

(1) Implementing the softmax function

```python
def softmax(x):
    ##########
    '''实现softmax函数，只要求对最后一维归一化，
    不允许用tf自带的softmax函数'''
    ##########
    x = tf.convert_to_tensor(x)
    rmax = tf.reduce_max(x, axis=1, keepdims=True)
    x_exp = tf.exp(x - rmax)
    x_sum = tf.reduce_sum(x_exp, axis=-1, keepdims=True)
    prob_x = x_exp / x_sum
    return prob_x
```

output:

```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

(2) Implementing the sigmoid function

```python
def sigmoid(x):
    ##########
    '''实现sigmoid函数， 不允许用tf自带的sigmoid函数'''
    ##########
    x = tf.convert_to_tensor(x)
    e_x = tf.exp(-x)
    prob_x = 1/(1+e_x)
    return prob_x
```

output:

```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

(3) Implementing the softmax cross entropy loss function

```
def softmax_ce(x, label):
    ##########
    '''实现 softmax 交叉熵loss函数， 不允许用tf自带的softmax_cross_entropy函数'''
    ##########
    cross_entropy = -tf.reduce_sum(label * tf.math.log(x), axis=-1)
    loss = tf.reduce_mean(cross_entropy)
    return loss
```

output:

```
True
```

(4) Implementing the sigmoid cross entropy loss function

```
def sigmoid_ce(x, label):
    ##########
    '''实现 softmax 交叉熵loss函数， 不允许用tf自带的softmax_cross_entropy函数'''
    ##########
    cross_entropy = label * -tf.math.log(x + 1e-8) + (1 - label) * -tf.math
     .log(1 - x + 1e-8)
    loss = tf.reduce_mean(cross_entropy)
    return loss
```

output:

```
[1. 0. 1. 1. 0. 1. 1. 0. 0. 0.]

True
```

2. tutorial_minst_fnn-numpy-exercise

(1) Implementing Matmul class

```
class Matmul:
    def __init__(self):
        self.mem = {}

    def forward(self, x, W):
        h = np.matmul(x, W)
        self.mem={'x': x, 'W':W}
        return h

    def backward(self, grad_y):
        '''
        x: shape(N, d)
        w: shape(d, d')
        grad_y: shape(N, d')
        '''
        x = self.mem['x']
        W = self.mem['W']

        ####################
        '''计算矩阵乘法的对应的梯度'''
        ####################
        grad_x = np.matmul(grad_y, W.T)
        grad_W = np.matmul(x.T, grad_y)
        return grad_x, grad_W
```

(2) Implementing Relu class

```
class Relu:
    def __init__(self):
        self.mem = {}

    def forward(self, x):
        self.mem['x']=x
        return np.where(x > 0, x, np.zeros_like(x))

    def backward(self, grad_y):
        '''
        grad_y: same shape as x
        '''
        ####################
        '''计算relu 激活函数对应的梯度'''
        ####################
        grad_x = np.where(self.mem['x']>0, grad_y, np.zeros_like(grad_y))
        return grad_x
```

3. tutorial_minst_fnn-tf2.0-exercise

为了证明“理论和实验证明，一个两层的 ReLU 网络可以模拟任何函数”，本次实验我分别用两层 Relu 网络来拟合一些简单函数，和用来进行对 mnist 数据集进行预测，来查看两层 Relu 网络的模拟效果。

首先来查看两层 Relu 网络对 mnist 数据集的预测效果。

(1) 构建连接层

首先构建一个全连接层，即每个神经元都与上一层的所有神经元相连。全连接层将数据与权重矩阵进行线性变换，在后续步骤中应用激活函数，从而实现数据的非线性变换。

```
class FullConnectionLayer:
    def __init__(self):
        self.mem = {}
    def forward(self, X, W):
        """
        :param X: shape(m,d),前向传播输入矩阵
        :param W: shape(d,d'),前向传播权重矩阵
        :return: 前向传播输出矩阵
        """
        self.mem['X'] = X
        self.mem['W'] = W
        H = np.matmul(X, W)
        return H
    def backward(self, grad_H):
        """
        :param grad_H: shape(m,d'),Loss关于 H 的梯度
        :return: grad_X: shape(m,d),Loss关于 X 的梯度
                 grad_W: shape(d,d'),Loss关于 W 的梯度
        """
        X = self.mem['X']
        W = self.mem['W']
        grad_X = np.matmul(grad_H, W.T)
        grad_W = np.matmul(X.T, grad_H)
        return grad_X, grad_W
```

(2) 实现激活函数、损失函数

接着分别实现 Relu 激活函数和损失函数。其中损失函数选择为交叉熵函数。

```
class Relu:
    def __init__(self):
        self.mem = {}

    def forward(self, x):
        self.mem['x']=x
        return np.where(x > 0, x, np.zeros_like(x))

    def backward(self, grad_y):
        grad_x = np.where(self.mem['x']>0, grad_y, np.zeros_like(grad_y))
        return grad_x
```

```
class CrossEntropy():
    def __init__(self):
        self.mem = {}
        self.epsilon = 1e-12  # 防止求导后分母为 0

    def forward(self, p, y):
        self.mem['p'] = p
        log_p = np.log(p + self.epsilon)
        return np.mean(np.sum(-y * log_p, axis=1))

    def backward(self, y):
        p = self.mem['p']
        return -y * (1 / (p + self.epsilon))
```

(3) 建立模型

本次实验只建立一个简单的二层神经网络，两层网络均采用全连接神经网络，前后两层神经网络均用 relu 函数进行激活，从而证明理论和实验证明，一个两层的 ReLU 网络可以模拟任何函数。

```
class myModel:
    def __init__(self):
        self.W1 = np.random.normal(size=[28*28+1, 100])
        self.W2 = np.random.normal(size=[100, 10])

        self.mul_h1 = FullConnectionLayer()
        self.mul_h2 = FullConnectionLayer()
        self.relu1 = Relu()
        self.relu2 = Relu()
        self.cross_en = CrossEntropy()

    def forward(self, x, label):
        x = x.reshape(-1, 28*28)
        bias = np.ones(shape=[x.shape[0], 1])
        x = np.concatenate([x, bias], axis=1)
        self.h1 = self.mul_h1.forward(x, self.W1)
        self.h1_relu = self.relu1.forward(self.h1)
        self.h2 = self.mul_h2.forward(self.h1_relu, self.W2)
        self.h2_relu = self.relu2.forward(self.h2)
        self.loss = self.cross_en.forward(self.h2_relu, label)

    def backward(self, label):
        self.loss_grad = self.cross_en.backward(label)
        self.h2_relu_grad = self.relu2.backward(self.loss_grad)
        self.h2_grad, self.W2_grad = self.mul_h2.backward(self.h2_relu_grad)
        self.h1_relu_grad = self.sigmoid.backward(self.h2_grad)
        self.h1_grad, self.W1_grad = self.mul_h1.backward(self.h1_relu_grad)
```

(4) 实际训练

本次实验采用数据集为 mnist 数据集，对其训练集进行 50 轮训练，接着对其测试集进行预测，最终预测正确率为 0.6483。

```
epoch 49 : loss 1.3081298160774804 ; accuracy 0.6387166666666667
test loss 1.2449487320274255 ; accuracy 0.6483
```

(5) 多层神经网络尝试

根据的多层神经网络(深度学习)的思想，将神经网络由两层拓展到四层，添加两层全连接层，由 relu 函数进行激活，接着查看训练效果。模型搭建如下：

```
class myModel2:
    def __init__(self):
        self.W1 = np.random.normal(size=[28*28+1, 100])
        self.W2 = np.random.normal(size=[100, 100])
        self.W3 = np.random.normal(size=[100, 100])
        self.W4 = np.random.normal(size=[100, 10])

        self.mul_h1 = FullConnectionLayer()
        self.mul_h2 = FullConnectionLayer()
        self.mul_h3 = FullConnectionLayer()
        self.mul_h4 = FullConnectionLayer()
        self.relu1 = Relu()
        self.relu2 = Relu()
        self.relu3 = Relu()
        self.relu4 = Relu()
        self.cross_en = CrossEntropy()
        self.learning_rate = 1e-5
```

```python
def forward(self, x, label):
    x = x.reshape(-1, 28*28)
    bias = np.ones(shape=[x.shape[0], 1])
    x = np.concatenate([x, bias], axis=1)
    self.h1 = self.mul_h1.forward(x, self.W1)
    self.h1_relu = self.relu1.forward(self.h1)
    self.h2 = self.mul_h2.forward(self.h1_relu, self.W2)
    self.h2_relu = self.relu2.forward(self.h2)
    self.h3 = self.mul_h3.forward(self.h2_relu, self.W3)
    self.h3_relu = self.relu3.forward(self.h3)
    self.h4 = self.mul_h4.forward(self.h3_relu, self.W4)
    self.h4_relu = self.relu4.forward(self.h4)
    self.loss = self.cross_en.forward(self.h4_relu, label)

def backward(self, label):
    self.loss_grad = self.cross_en.backward(label)

    self.h4_relu_grad = self.relu4.backward(self.loss_grad)
    self.h4_grad, self.W4_grad = self.mul_h4.backward(self.h4_relu_grad)
    self.h3_relu_grad = self.relu3.backward(self.h4_grad)
    self.h3_grad, self.W3_grad = self.mul_h3.backward(self.h3_relu_grad)
    self.h2_relu_grad = self.relu2.backward(self.h3_grad)
    self.h2_grad, self.W2_grad = self.mul_h2.backward(self.h2_relu_grad)
    self.h1_relu_grad = self.relu1.backward(self.h2_grad)
    self.h1_grad, self.W1_grad = self.mul_h1.backward(self.h1_relu_grad)
```

训练结果如下所示：

```
epoch 47 : loss -10.134587340536498 ; accuracy 0.09835
epoch 48 : loss -10.155736751390043 ; accuracy 0.0984
epoch 49 : loss -10.176633751107556 ; accuracy 0.09843333333333333
test loss -10.214302931024111 ; accuracy 0.0986
```

可以看到结果非常之差，远不如两层神经网络的效果。事实证明，神经网络存在梯度消失和梯度爆炸等问题，在深层网络能够收敛的前提下，随着网络深度的增加，正确率开始饱和甚至下降，称之为网络的退化(degradation)问题。随着网络的加深，出现了训练集准确率下降的现象，文献(Hek, Identity mappings in Deep Residual Networks[M])可以确定这不是由于过拟合造成的（过拟合的情况训练集应该准确率很高）

接着我们来看一下两层 Relu 网络对于一些普通简单函数的拟合效果。

（1）模型的搭建

此处选择用线性连接层表示 layers，此时 input、hidden、output 维度均可自定义。

```python
class myModel3(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(myModel3, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()
        self.learning_rate = 1e-2

    def forward(self, x):
        h1 = self.layer1(x)
        h1_relu = self.relu(h1)
        h2 = self.layer2(h1_relu)
        return h2
```

（2）模型训练

此处选择三个函数 fun1-3 来进行拟合实验，分别用 fun1、2、3 的 y 作为预期值，从而算出预测值的 loss 值，调整参数。

```
def train(fun, model, epochs=5000):
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=model.learning_rate)

    x_train = torch.linspace(-10, 10, 100).view(-1,1)
    y_train = fun(x_train)

    for epoch in range(epochs):
        optimizer.zero_grad()
        outputs = model(x_train)
        loss = criterion(outputs, y_train)
        loss.backward()
        optimizer.step()

    return x_train, y_train, model
```

```
def fun1(x):
    return np.sin(x)

def fun2(x):
    return np.sin(x) * np.sin(x) + x**3 * np.cos(x) + 4*x +2

def fun3(x):
    return x**4 - x**3 + x**2 - x + 1
```

```
funs = [fun1, fun2, fun3]
results = []

for fun in funs:
    model = myModel3(1, 100, 1)
    x_train, y_train, trained_network = train(fun, model)
    y_pred = trained_network(x_train)
    results.append((x_train, y_train, y_pred))
```

（3）结果可视化

可以看到两层 Relu 网络拟合效果很好，但有时也并不完全重合，比如第二个函数，函数较为复杂，拟合效果有待提升。