2151131-朱沙桐-课后作业2-1

2151131 朱沙桐

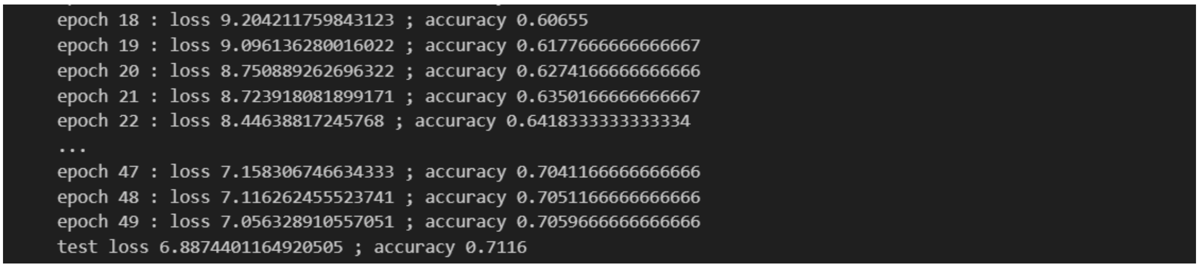
利用numpy搭建全连接神经网络

求导

```python
import numpy as np

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.dot(grad_y, W.T)
        grad_W = np.dot(x.T, grad_y)

        return grad_x, grad_W


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_y = np.where(self.mem['x'] > 0, grad_y, np.zeros_like(grad_y))
        grad_x = grad_y

        return grad_x


class Softmax:
    '''
    softmax over last dimention
    '''
    def __init__(self):
        self.epsilon = 1e-12
        self.mem = {}

    def forward(self, x):
        '''
        x: shape(N, c)
        '''
        x_exp = np.exp(x)
        partition = np.sum(x_exp, axis=1, keepdims=True)
        out = x_exp/(partition+self.epsilon)

        self.mem['out'] = out
        self.mem['x_exp'] = x_exp
        return out

    def backward(self, grad_y):
        '''
        grad_y: same shape as x
        '''
        s = self.mem['out']
        sisj = np.matmul(np.expand_dims(s,axis=2), np.expand_dims(s, axis=1)) # (N, c, c)
        g_y_exp = np.expand_dims(grad_y, axis=1)
        tmp = np.matmul(g_y_exp, sisj) #(N, 1, c)
        tmp = np.squeeze(tmp, axis=1)
        tmp = -tmp+grad_y*s
        return tmp

class Log:
    '''
    softmax over last dimention
    '''
    def __init__(self):
        self.epsilon = 1e-12
        self.mem = {}

    def forward(self, x):
        '''
        x: shape(N, c)
        '''
        out = np.log(x+self.epsilon)

        self.mem['x'] = x
        return out

    def backward(self, grad_y):
        '''
        grad_y: same shape as x
        '''
        x = self.mem['x']

        return 1./(x+1e-12) * grad_y

```

model

```
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 = Matmul()
        self.mul_h2 = Matmul()
        self.relu = Relu()
        self.softmax = Softmax()
        self.log = Log()


    def forward(self, x):
        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) # shape(5, 4)
        self.h1_relu = self.relu.forward(self.h1)
        self.h2 = self.mul_h2.forward(self.h1_relu, self.W2)
        self.h2_soft = self.softmax.forward(self.h2)
        self.h2_log = self.log.forward(self.h2_soft)

    def backward(self, label):
        self.h2_log_grad = self.log.backward(-label)
        self.h2_soft_grad = self.softmax.backward(self.h2_log_grad)
        self.h2_grad, self.W2_grad = self.mul_h2.backward(self.h2_soft_grad)
        self.h1_relu_grad = self.relu.backward(self.h2_grad)
        self.h1_grad, self.W1_grad = self.mul_h1.backward(self.h1_relu_grad)

model = myModel()
```

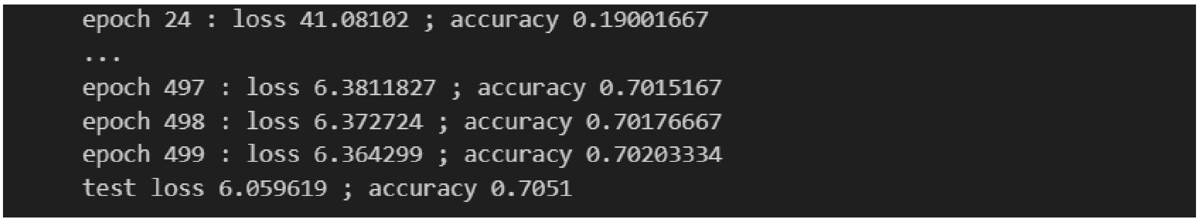


```
epoch 18 : loss 9.204211759843123 ; accuracy 0.60655
epoch 19 : loss 9.096136280016022 ; accuracy 0.6177666666666667
epoch 20 : loss 8.750889262696322 ; accuracy 0.6274166666666666
epoch 21 : loss 8.723918081899171 ; accuracy 0.6350166666666667
epoch 22 : loss 8.44638817245768 ; accuracy 0.6418333333333334
...
epoch 47 : loss 7.158306746634333 ; accuracy 0.7041166666666666
epoch 48 : loss 7.116262455523741 ; accuracy 0.7051166666666666
epoch 49 : loss 7.056328910557051 ; accuracy 0.7059666666666666
test loss 6.8874401164920505 ; accuracy 0.7116
```

利用pytorch搭建全连接神经网络

model

```python
class myModel:
    def __init__(self):
        ####################
        '''声明模型对应的参数'''
        ####################
        self.W1 = tf.Variable(tf.random.normal(
            [784, 98]), trainable=True, dtype=tf.float32)
        self.b1 = tf.Variable(tf.zeros([98]), trainable=True, dtype=tf.float32)
        self.W2 = tf.Variable(tf.random.normal(
            [98, 10]), trainable=True, dtype=tf.float32)
        self.b2 = tf.Variable(tf.zeros([10]), trainable=True, dtype=tf.float32)

    def __call__(self, x):
        ####################
        '''实现模型函数体，返回未归一化的logits'''
        ####################
        x = tf.reshape(x, [-1, 784])
        h1 = tf.nn.relu(tf.matmul(x, self.W1) + self.b1)
        logits = tf.matmul(h1, self.W2) + self.b2
        return logits

model = myModel()

optimizer = optimizers.Adam()
```



```
epoch 24 : loss 41.08102 ; accuracy 0.19001667
...
epoch 497 : loss 6.3811827 ; accuracy 0.7015167
epoch 498 : loss 6.372724 ; accuracy 0.70176667
epoch 499 : loss 6.364299 ; accuracy 0.70203334
test loss 6.059619 ; accuracy 0.7051
```