tutorial_minst_fnn-numpy-exercise

October 24, 2025

0.1

[13]:
```python
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # or any {'0', '1', '2'}

# -   MNIST
# -      0~255      0~1
# -
def mnist_dataset():
    (x, y), (x_test, y_test) = datasets.mnist.load_data()
    #normalize
    x = x/255.0
    x_test = x_test/255.0

    return (x, y), (x_test, y_test)
```

0.2 Demo numpy based auto differentiation

[14]:
```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']

        ####################
        '''       '''
        ####################
        # y = x @ W; grad_y: shape (N, d')
        # grad_x = grad_y @ W^T, shape (N, d)
        grad_x = np.matmul(grad_y, W.T)
        # grad_W = x^T @ grad_y, shape (d, d')
        grad_W = np.matmul(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
        '''
        ####################
        ''' relu      '''
        ####################
        # ReLU' = 1 if x > 0 else 0
        mask = (self.mem['x'] > 0).astype(grad_y.dtype)
        grad_x = grad_y * mask
        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
```

0.3 Gradient check

[15]:
```
import tensorflow as tf

x = np.random.normal(size=[5, 6])
W = np.random.normal(size=[6, 4])
aa = Matmul()
out = aa.forward(x, W) # shape(5, 4)
grad = aa.backward(np.ones_like(out))
print (grad)

with tf.GradientTape() as tape:
    x, W = tf.constant(x), tf.constant(W)
    tape.watch(x)
    y = tf.matmul(x, W)
    loss = tf.reduce_sum(y)
    grads = tape.gradient(loss, x)
    print (grads)

import tensorflow as tf

x = np.random.normal(size=[5, 6])
aa = Relu()
out = aa.forward(x) # shape(5, 4)
grad = aa.backward(np.ones_like(out))
print (grad)

with tf.GradientTape() as tape:
    x= tf.constant(x)
    tape.watch(x)
    y = tf.nn.relu(x)
    loss = tf.reduce_sum(y)
    grads = tape.gradient(loss, x)
    print (grads)

import tensorflow as tf
x = np.random.normal(size=[5, 6], scale=5.0, loc=1)
label = np.zeros_like(x)
label[0, 1]=1.
label[1, 0]=1
label[1, 1]=1
label[2, 3]=1
label[3, 5]=1
label[4, 0]=1
```

```python
print(label)
aa = Softmax()
out = aa.forward(x) # shape(5, 6)
grad = aa.backward(label)
print (grad)

with tf.GradientTape() as tape:
    x= tf.constant(x)
    tape.watch(x)
    y = tf.nn.softmax(x)
    loss = tf.reduce_sum(y*label)
    grads = tape.gradient(loss, x)
    print (grads)

import tensorflow as tf

x = np.random.normal(size=[5, 6])
aa = Log()
out = aa.forward(x) # shape(5, 4)
grad = aa.backward(label)
print (grad)

with tf.GradientTape() as tape:
    x= tf.constant(x)
    tape.watch(x)
    y = tf.math.log(x)
    loss = tf.reduce_sum(y*label)
    grads = tape.gradient(loss, x)
    print (grads)
```

```
(array([[-0.77029111, -0.01587331, -3.9170444 ,  0.03231269, -0.54126406,
        -2.44267438],
       [-0.77029111, -0.01587331, -3.9170444 ,  0.03231269, -0.54126406,
        -2.44267438],
       [-0.77029111, -0.01587331, -3.9170444 ,  0.03231269, -0.54126406,
        -2.44267438],
       [-0.77029111, -0.01587331, -3.9170444 ,  0.03231269, -0.54126406,
        -2.44267438],
       [-0.77029111, -0.01587331, -3.9170444 ,  0.03231269, -0.54126406,
        -2.44267438]]), array([[ 0.9279897 ,  0.9279897 ,  0.9279897 ,
0.9279897 ],
       [-0.28202534, -0.28202534, -0.28202534, -0.28202534],
       [ 3.12880362,  3.12880362,  3.12880362,  3.12880362],
       [-0.58476253, -0.58476253, -0.58476253, -0.58476253],
       [ 1.19042537,  1.19042537,  1.19042537,  1.19042537],
       [ 0.11341984,  0.11341984,  0.11341984,  0.11341984]]))
tf.Tensor(
[[-0.77029111 -0.01587331 -3.9170444   0.03231269 -0.54126406 -2.44267438]
```

```
 [-0.77029111 -0.01587331 -3.9170444   0.03231269 -0.54126406 -2.44267438]
 [-0.77029111 -0.01587331 -3.9170444   0.03231269 -0.54126406 -2.44267438]
 [-0.77029111 -0.01587331 -3.9170444   0.03231269 -0.54126406 -2.44267438]
 [-0.77029111 -0.01587331 -3.9170444   0.03231269 -0.54126406 -2.44267438]],
shape=(5, 6), dtype=float64)
[[1. 0. 1. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0.]
 [0. 1. 1. 0. 0. 1.]
 [0. 0. 1. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0.]]
tf.Tensor(
[[1. 0. 1. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0.]
 [0. 1. 1. 0. 0. 1.]
 [0. 0. 1. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0.]], shape=(5, 6), dtype=float64)
[[0. 1. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0.]]
[[-2.79198754e-09  2.69340155e-03 -1.32531574e-11 -6.05432825e-04
  -1.19227441e-05 -2.07604317e-03]
 [ 2.34876061e-05  1.14954767e-07 -3.79652822e-10 -6.24252518e-13
  -1.45271233e-05 -9.07505730e-06]
 [-2.71887218e-10 -2.45755348e-09 -1.32220956e-08  1.13626016e-05
  -3.70924522e-06 -7.63740483e-06]
 [-1.51405943e-12 -1.86060929e-09 -2.24999728e-13 -2.47439333e-12
  -7.93492806e-16  1.86482353e-09]
 [ 1.51858765e-01 -1.73699796e-03 -5.94448556e-02 -3.04049986e-05
  -9.95271788e-06 -9.06365535e-02]]
tf.Tensor(
[[-2.79198754e-09  2.69340155e-03 -1.32531574e-11 -6.05432825e-04
  -1.19227441e-05 -2.07604317e-03]
 [ 2.34876061e-05  1.14954767e-07 -3.79652822e-10 -6.24252518e-13
  -1.45271233e-05 -9.07505730e-06]
 [-2.71887218e-10 -2.45755348e-09 -1.32220956e-08  1.13626016e-05
  -3.70924522e-06 -7.63740483e-06]
 [-1.51405943e-12 -1.86060929e-09 -2.24999728e-13 -2.47439333e-12
  -7.93492806e-16  1.86482353e-09]
 [ 1.51858765e-01 -1.73699796e-03 -5.94448556e-02 -3.04049986e-05
  -9.95271788e-06 -9.06365535e-02]], shape=(5, 6), dtype=float64)
[[-0.         -1.15431613 -0.          0.         -0.         -0.        ]
 [-1.67809415 -0.78904586  0.         -0.          0.         -0.        ]
 [ 0.         -0.         -0.         -3.15355667  0.          0.        ]
 [ 0.          0.          0.          0.         -0.         -4.60987734]
 [ 1.7624878  -0.         -0.         -0.          0.         -0.        ]]
tf.Tensor(
[[-0.         -1.15431613 -0.          0.         -0.         -0.        ]
 [-1.67809415 -0.78904586  0.         -0.          0.         -0.        ]
 [ 0.         -0.         -0.         -3.15355667  0.          0.        ]
 [ 0.          0.          0.          0.         -0.         -4.60987734]
 [ 1.7624878  -0.         -0.         -0.          0.         -0.        ]],
shape=(5, 6), dtype=float64)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_37932\777580174.py:98:
RuntimeWarning: invalid value encountered in log
  out = np.log(x+self.epsilon)
```

1 Final Gradient Check

[16]:
```python
import tensorflow as tf

label = np.zeros_like(x)
label[0, 1]=1.
label[1, 0]=1
label[2, 3]=1
label[3, 5]=1
label[4, 0]=1

x = np.random.normal(size=[5, 6])
W1 = np.random.normal(size=[6, 5])
W2 = np.random.normal(size=[5, 6])

mul_h1 = Matmul()
mul_h2 = Matmul()
relu = Relu()
softmax = Softmax()
log = Log()

h1 = mul_h1.forward(x, W1) # shape(5, 4)
h1_relu = relu.forward(h1)
h2 = mul_h2.forward(h1_relu, W2)
h2_soft = softmax.forward(h2)
h2_log = log.forward(h2_soft)


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

print(h2_log_grad)
print('--'*20)
```

```
# print(W2_grad)

with tf.GradientTape() as tape:
    x, W1, W2, label = tf.constant(x), tf.constant(W1), tf.constant(W2), tf.
 ↪constant(label)
    tape.watch(W1)
    tape.watch(W2)
    h1 = tf.matmul(x, W1)
    h1_relu = tf.nn.relu(h1)
    h2 = tf.matmul(h1_relu, W2)
    prob = tf.nn.softmax(h2)
    log_prob = tf.math.log(prob)
    loss = tf.reduce_sum(label * log_prob)
    grads = tape.gradient(loss, [prob])
    print (grads[0].numpy())
```

```
[[  0.         243.13113111   0.           0.           0.
    0.        ]
 [146.48378867   0.           0.           0.           0.
    0.        ]
 [  0.           0.           0.          33.06505154   0.
    0.        ]
 [  0.           0.           0.           0.           0.
    3.04999949]
 [210.29977784   0.           0.           0.           0.
    0.        ]]
----------------------------------------
[[  0.         243.13113116   0.           0.           0.
    0.        ]
 [146.48378869   0.           0.           0.           0.
    0.        ]
 [  0.           0.           0.          33.06505154   0.
    0.        ]
 [  0.           0.           0.           0.           0.
    3.04999949]
 [210.29977789   0.           0.           0.           0.
    0.        ]]
```

1.1

[17]:
```
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()
```

1.2 loss

[18]:
```
def compute_loss(log_prob, labels):
     return np.mean(np.sum(-log_prob*labels, axis=1))


def compute_accuracy(log_prob, labels):
    predictions = np.argmax(log_prob, axis=1)
    truth = np.argmax(labels, axis=1)
    return np.mean(predictions==truth)

def train_one_step(model, x, y):
    model.forward(x)
    model.backward(y)
    model.W1 -= 1e-5* model.W1_grad
    model.W2 -= 1e-5* model.W2_grad
    loss = compute_loss(model.h2_log, y)
    accuracy = compute_accuracy(model.h2_log, y)
    return loss, accuracy

def test(model, x, y):
    model.forward(x)
    loss = compute_loss(model.h2_log, y)
    accuracy = compute_accuracy(model.h2_log, y)
    return loss, accuracy
```

1.3

[22]:
```
#     npz     MNIST
mnist_data = np.load('d:\\code\\python\\deeplearning\\homework4\\mnist.npz')
train_data = (mnist_data['x_train'] / 255.0, mnist_data['y_train'])
test_data = (mnist_data['x_test'] / 255.0, mnist_data['y_test'])
train_label = np.zeros(shape=[train_data[0].shape[0], 10])
test_label = np.zeros(shape=[test_data[0].shape[0], 10])
train_label[np.arange(train_data[0].shape[0]), np.array(train_data[1])] = 1.
test_label[np.arange(test_data[0].shape[0]), np.array(test_data[1])] = 1.

for epoch in range(50):
    loss, accuracy = train_one_step(model, train_data[0], train_label)
    print('epoch', epoch, ': loss', loss, '; accuracy', accuracy)
loss, accuracy = test(model, test_data[0], test_label)

print('test loss', loss, '; accuracy', accuracy)
```

```
epoch 0 : loss 23.947478099091448 ; accuracy 0.0892
epoch 1 : loss 22.611990702846374 ; accuracy 0.13101666666666667
epoch 2 : loss 21.33722764655784 ; accuracy 0.1745
epoch 3 : loss 20.145158667901303 ; accuracy 0.21501666666666666
epoch 4 : loss 19.208372648311634 ; accuracy 0.2524
epoch 5 : loss 18.518030434514717 ; accuracy 0.27541666666666664
epoch 6 : loss 17.69135279383483 ; accuracy 0.2980333333333333
epoch 7 : loss 16.224156558597404 ; accuracy 0.35196666666666665
epoch 8 : loss 15.429518432896138 ; accuracy 0.3852333333333333
epoch 9 : loss 15.04154725814835 ; accuracy 0.3995666666666667
epoch 10 : loss 14.728959564328473 ; accuracy 0.4103333333333333
epoch 11 : loss 14.503962699684362 ; accuracy 0.42188333333333333
epoch 12 : loss 14.127649850663667 ; accuracy 0.4303
epoch 13 : loss 13.763182261290053 ; accuracy 0.4464
epoch 14 : loss 13.3231684373166 ; accuracy 0.4541333333333333
epoch 15 : loss 13.060826622270946 ; accuracy 0.4699
epoch 16 : loss 12.575917907059017 ; accuracy 0.4755333333333333
epoch 17 : loss 12.454326642921629 ; accuracy 0.49385
epoch 18 : loss 11.577407511778953 ; accuracy 0.5093666666666666
epoch 19 : loss 11.262506630772348 ; accuracy 0.5307166666666666
epoch 20 : loss 10.71433019086592 ; accuracy 0.5373833333333333
epoch 21 : loss 10.463936022181976 ; accuracy 0.5573333333333333
epoch 22 : loss 9.682991834739212 ; accuracy 0.5732333333333334
epoch 23 : loss 9.452765895974714 ; accuracy 0.5919833333333333
epoch 24 : loss 8.952730893950935 ; accuracy 0.6019
epoch 25 : loss 8.938052164026645 ; accuracy 0.6138666666666667
epoch 26 : loss 8.211352313927723 ; accuracy 0.63495
epoch 27 : loss 8.123777002730957 ; accuracy 0.64175
epoch 28 : loss 7.832957996829675 ; accuracy 0.6520166666666667
epoch 29 : loss 7.800013726358961 ; accuracy 0.6556333333333333
epoch 30 : loss 7.497256859834995 ; accuracy 0.6670166666666667
epoch 31 : loss 7.451954581240074 ; accuracy 0.6699
epoch 32 : loss 7.211749199694508 ; accuracy 0.6796666666666666
epoch 33 : loss 7.216994641184621 ; accuracy 0.6794833333333333
epoch 34 : loss 7.118040797365639 ; accuracy 0.6841833333333334
epoch 35 : loss 7.19319287976786 ; accuracy 0.6811666666666667
epoch 36 : loss 7.188057037907005 ; accuracy 0.68385
epoch 37 : loss 7.04267748195216 ; accuracy 0.6875
epoch 38 : loss 7.0197544130100225 ; accuracy 0.6915333333333333
epoch 39 : loss 6.734995516959494 ; accuracy 0.70015
epoch 40 : loss 6.7211047808151285 ; accuracy 0.7042
epoch 41 : loss 6.467938898192994 ; accuracy 0.7112333333333334
epoch 42 : loss 6.43614138053969 ; accuracy 0.71545
epoch 43 : loss 6.2742479308495955 ; accuracy 0.7201333333333333
epoch 44 : loss 6.236603372497509 ; accuracy 0.72375
epoch 45 : loss 6.1203342294964616 ; accuracy 0.7278333333333333
epoch 46 : loss 6.08467224368504 ; accuracy 0.7294
epoch 47 : loss 6.0145485118163355 ; accuracy 0.73225
epoch 48 : loss 5.9835262880142 ; accuracy 0.73355
epoch 49 : loss 5.919216784282603 ; accuracy 0.7367833333333333
test loss 5.618739981156455 ; accuracy 0.7517
```