tutorial_minst_fnn-tf2.0-exercise

October 24, 2025

0.1

[3]:
```
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'}

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)
```

[4]:
```
print(list(zip([1, 2, 3, 4], ['a', 'b', 'c', 'd'])))
```

```
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]
```

0.2

[5]:
```
class myModel:
    def __init__(self):
        ####################
        '''     '''
        ####################
        #         (784 -> 256)
        self.W1 = tf.Variable(tf.random.normal([784, 256], stddev=0.1))
        self.b1 = tf.Variable(tf.zeros([256]))

        #         (256 -> 10)
        self.W2 = tf.Variable(tf.random.normal([256, 10], stddev=0.1))
        self.b2 = tf.Variable(tf.zeros([10]))

    def __call__(self, x):
        ####################
```

```
        '''        logits'''
        ####################
        #       (batch_size, 28, 28) -> (batch_size, 784)
        x = tf.reshape(x, [-1, 784])

        #      + ReLU
        h1 = tf.nn.relu(tf.matmul(x, self.W1) + self.b1)

        #      logits
        logits = tf.matmul(h1, self.W2) + self.b2

        return logits

model = myModel()

optimizer = optimizers.Adam()
```

0.3 loss

[6]:
```
@tf.function
def compute_loss(logits, labels):
    return tf.reduce_mean(
        tf.nn.sparse_softmax_cross_entropy_with_logits(
            logits=logits, labels=labels))

@tf.function
def compute_accuracy(logits, labels):
    predictions = tf.argmax(logits, axis=1)
    return tf.reduce_mean(tf.cast(tf.equal(predictions, labels), tf.float32))

@tf.function
def train_one_step(model, optimizer, x, y):
    with tf.GradientTape() as tape:
        logits = model(x)
        loss = compute_loss(logits, y)

    # compute gradient
    trainable_vars = [model.W1, model.W2, model.b1, model.b2]
    grads = tape.gradient(loss, trainable_vars)
    for g, v in zip(grads, trainable_vars):
        v.assign_sub(0.01*g)

    accuracy = compute_accuracy(logits, y)

    # loss and accuracy is scalar tensor
    return loss, accuracy
```

```
@tf.function
def test(model, x, y):
    logits = model(x)
    loss = compute_loss(logits, y)
    accuracy = compute_accuracy(logits, y)
    return loss, accuracy
```

0.4

[7]:
```
train_data, test_data = mnist_dataset()
for epoch in range(50):
    loss, accuracy = train_one_step(model, optimizer,
                                    tf.constant(train_data[0], dtype=tf.
 ↪float32),
                                    tf.constant(train_data[1], dtype=tf.int64))
    print('epoch', epoch, ': loss', loss.numpy(), '; accuracy', accuracy.
 ↪numpy())
loss, accuracy = test(model,
                      tf.constant(test_data[0], dtype=tf.float32),
                      tf.constant(test_data[1], dtype=tf.int64))

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434              2s
0us/step
epoch 0 : loss 3.1276379 ; accuracy 0.0888
epoch 1 : loss 3.014446 ; accuracy 0.089383334
epoch 2 : loss 2.9246955 ; accuracy 0.08865
epoch 3 : loss 2.8498628 ; accuracy 0.08918333
epoch 4 : loss 2.7853532 ; accuracy 0.0904
epoch 5 : loss 2.7284527 ; accuracy 0.09138333
epoch 6 : loss 2.677417 ; accuracy 0.09328333
epoch 7 : loss 2.6310568 ; accuracy 0.0951
epoch 8 : loss 2.5885172 ; accuracy 0.09826667
epoch 9 : loss 2.5491605 ; accuracy 0.1018
epoch 10 : loss 2.5125062 ; accuracy 0.10598333
epoch 11 : loss 2.478175 ; accuracy 0.11025
epoch 12 : loss 2.445873 ; accuracy 0.11438333
epoch 13 : loss 2.4153671 ; accuracy 0.119166665
epoch 14 : loss 2.3864667 ; accuracy 0.1243
epoch 15 : loss 2.3590162 ; accuracy 0.13026667
epoch 16 : loss 2.3328855 ; accuracy 0.13656667
epoch 17 : loss 2.3079612 ; accuracy 0.14325
epoch 18 : loss 2.2841454 ; accuracy 0.15095
epoch 19 : loss 2.2613494 ; accuracy 0.15991667
epoch 20 : loss 2.2394917 ; accuracy 0.16958334
```

```
epoch 21 : loss 2.218495 ; accuracy 0.17923333
epoch 22 : loss 2.19829 ; accuracy 0.191
epoch 23 : loss 2.178809 ; accuracy 0.20251666
epoch 24 : loss 2.159989 ; accuracy 0.21461667
epoch 25 : loss 2.1417706 ; accuracy 0.2268
epoch 26 : loss 2.124102 ; accuracy 0.23951666
epoch 27 : loss 2.1069314 ; accuracy 0.25136667
epoch 28 : loss 2.0902154 ; accuracy 0.26315
epoch 29 : loss 2.0739107 ; accuracy 0.27455
epoch 30 : loss 2.0579805 ; accuracy 0.2853
epoch 31 : loss 2.0423932 ; accuracy 0.29588333
epoch 32 : loss 2.0271213 ; accuracy 0.30628332
epoch 33 : loss 2.0121408 ; accuracy 0.31613332
epoch 34 : loss 1.9974307 ; accuracy 0.32538334
epoch 35 : loss 1.9829726 ; accuracy 0.33448333
epoch 36 : loss 1.9687496 ; accuracy 0.34331667
epoch 37 : loss 1.954749 ; accuracy 0.35218334
epoch 38 : loss 1.9409591 ; accuracy 0.36078334
epoch 39 : loss 1.9273704 ; accuracy 0.36836666
epoch 40 : loss 1.9139758 ; accuracy 0.37705
epoch 41 : loss 1.9007679 ; accuracy 0.38435
epoch 42 : loss 1.8877404 ; accuracy 0.39195
epoch 43 : loss 1.8748868 ; accuracy 0.39963335
epoch 44 : loss 1.8622018 ; accuracy 0.40635
epoch 45 : loss 1.8496826 ; accuracy 0.41313332
epoch 46 : loss 1.837325 ; accuracy 0.42013332
epoch 47 : loss 1.8251245 ; accuracy 0.42658332
epoch 48 : loss 1.8130786 ; accuracy 0.43286666
epoch 49 : loss 1.8011855 ; accuracy 0.43971667
test loss 1.7851627 ; accuracy 0.4466
```