tutorial_minst_cnn-tf2.0-exercise-ch05

October 30, 2025

0.1

[5]:
```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets, Model

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
    # Add a channels dimension
    x = x[..., tf.newaxis].astype("float32")
    x_test = x_test[..., tf.newaxis].astype("float32")
    #   tf.data      batch
    train_ds = tf.data.Dataset.from_tensor_slices(
    (x, y)).shuffle(10000).batch(32)
    test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(1000)
    return train_ds, test_ds
```

0.2

[6]:
```
class myModel(Model):
    def __init__(self):
        super(myModel, self).__init__()
        ######################
        '''     '''
        # 1 patch 7x7, in size 1, out size 32      1 padding  same,    relu␣
 ↪(tf.keras.layers.Conv2D)
        self.conv1 = layers.Conv2D(filters=32, kernel_size=7, strides=1,␣
 ↪padding='same', activation='relu')
        # 1       2 ;         2; padding    same (layers.MaxPool2D)
        self.pool1 = layers.MaxPool2D(pool_size=2, strides=2, padding='same')
        # 2 patch 5x5, in size 32, out size 64      1 padding   same,    relu
```

```
        self.conv2 = layers.Conv2D(filters=64, kernel_size=5, strides=1,␣
 ↪padding='same', activation='relu')
        # 2       2 ;         2; padding    same
        self.pool2 = layers.MaxPool2D(pool_size=2, strides=2, padding='same')
        #
        self.flatten = layers.Flatten()
        #    1: output dim 1024,    relu  (tf.keras.layers.Dense)
        self.d1 = layers.Dense(1024, activation='relu')
        #    2: output dim 10
        self.d2 = layers.Dense(10)  #      logits

        ####################
    def call(self, x, training=False):
        ####################
        '''        logits'''
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = self.d1(x)
        x = self.d2(x)
        return x
        ####################
        return logits

model = myModel()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()
```

0.3 loss

[ ]:
```
#
train_loss = tf.keras.metrics.Mean(name='train_loss')

#              0,1,2...    one-hot
train_accuracy = tf.keras.metrics.
 ↪SparseCategoricalAccuracy(name='train_accuracy')

#
test_loss = tf.keras.metrics.Mean(name='test_loss')

#
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')


# @tf.function     Python      TensorFlow
```

```
@tf.function
def train_one_step(model, optimizer, x, y):
    #   GradientTape
    with tf.GradientTape() as tape:
        #    training=True       Dropout BatchNorm
        predictions = model(x, training=True)
        #    loss_object        SparseCategoricalCrossentropy
        loss = loss_object(y, predictions)

    #         loss
    gradients = tape.gradient(loss, model.trainable_variables)

    #
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    #
    train_loss(loss)          #    batch   loss
    train_accuracy(y, predictions)  #    batch


# @tf.function
@tf.function
def test_step(model, images, labels):
    #    training=False       Dropout   BatchNorm
    predictions = model(images, training=False)
    #    batch
    t_loss = loss_object(labels, predictions)

    #
    test_loss(t_loss)               #
    test_accuracy(labels, predictions)  #
```

0.4

[ ]:
```
#   MNIST   train_ds        test_ds
#     tf.data.Dataset        batch
train_ds, test_ds = mnist_dataset()

#     epochs
EPOCHS = 5

#       epoch
for epoch in range(EPOCHS):
    #   epoch
    #   loss   accuracy
    train_loss.reset_state()        #
    train_accuracy.reset_state()    #
```

```
    test_loss.reset_state()         #
    test_accuracy.reset_state()     #

    #       batch
    # images:    [batch_size, 28, 28, 1]
    # labels:    [batch_size]    0~9
    for images, labels in train_ds:
        #
        # -    +
        # -    +
        # -    batch   loss   accuracy
        train_one_step(model, optimizer, images, labels)

    #
    #
    for test_images, test_labels in test_ds:
        #
        # -    training=False
        # -
        # -
        test_step(model, test_images, test_labels)

    #   epoch
    # .result()    epoch   batch    loss /   accuracy
    print(
        f'Epoch {epoch + 1}, '                           #     1
        f'Loss: {train_loss.result():.4f}, '            #     4
        f'Accuracy: {train_accuracy.result() * 100:.2f}%, '  #
        f'Test Loss: {test_loss.result():.4f}, '        #
        f'Test Accuracy: {test_accuracy.result() * 100:.2f}%' #
    )
```

```
Epoch 1, Loss: 0.10241908580064774, Accuracy: 96.80332946777344, Test Loss:
0.05334934592247009, Test Accuracy: 98.15999603271484
Epoch 2, Loss: 0.03817015141248703, Accuracy: 98.83000183105469, Test Loss:
0.03605320304632187, Test Accuracy: 98.83999633789062
Epoch 3, Loss: 0.025063052773475647, Accuracy: 99.16999816894531, Test Loss:
0.0350714735686779, Test Accuracy: 98.90999603271484
Epoch 4, Loss: 0.019615016877651215, Accuracy: 99.38166809082031, Test Loss:
0.037959299981594086, Test Accuracy: 99.0
Epoch 5, Loss: 0.016512813046574593, Accuracy: 99.50666809082031, Test Loss:
0.04503753036260605, Test Accuracy: 98.68000030517578
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: