エキスパートのための Tensorflow 2.0 入門

MokkeMeguru

<2019-08-24 土>

Contents

1 TensorFlow ライブラリのインポート

```
from __future__ import division, absolute_import
from __future__ import print_function, unicode_literals
from functools import reduce
from tqdm import tqdm

import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
```

2 MNIST データセットの前処理

2.1 データセットの読み込み

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]
```

2.2 データセットのシャッフル・バッチ化

```
train_ds = tf.data.Dataset.from_tensor_slices(
    (x_train, y_train)).shuffle(10000).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
print("dataset", train_ds)
```

dataset <BatchDataset shapes: ((None, 28, 28, 1), (None,)), types: (tf.float64, tf.uint8)>

3 Keras モデルの作成

```
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(32, 3, activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10, activation='softmax')

    def call(self, x):
        return reduce(lambda x, f: f(x),
                      [x, self.conv1, self.flatten, self.d1, self.d2])


model = MyModel()
```

4 最適関数と損失関数の作成

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()
```

5　訓練時のログの作成

```
train_loss = tf.keras.metrics.Mean(name='train_loss')
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')
```

6　モデルの訓練

6.1　訓練時の 1 ステップを定義

```
@tf.function
def train_step(image, label):
  with tf.GradientTape() as tape:
    predictions = model(image)
    loss = loss_object(label, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(label, predictions)
```

6.2　テスト時の 1 ステップを定義

```
@tf.function
def test_step(image, label):
    predictions = model(image)
    t_loss = loss_object(label, predictions)

    test_loss(t_loss)
    test_accuracy(label, predictions)
```

6.3　モデルを訓練する

```
EPOCHS = 5

for epoch in range(EPOCHS):
    for image, label in train_ds:
        train_step(image, label)
```

```
6
7       for test_image, test_label in test_ds:
8           test_step(test_image, test_label)
9
10      template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test
        ↪ Accuracy: {}'
11      print(
12          template.format(epoch + 1, train_loss.result(),
13                          train_accuracy.result() * 100, test_loss.result(),
14                          test_accuracy.result() * 100))
```

Epoch 1, Loss: 0.1349046528339386, Accuracy: 95.96833801269531, Test Loss: 0.07161739468574524, Test Accuracy: 97.69999694824219

Epoch 2, Loss: 0.08821088820695877, Accuracy: 97.36000061035156, Test Loss: 0.06225990131497383, Test Accuracy: 98.0

Epoch 3, Loss: 0.06569895893335342, Accuracy: 98.02166748046875, Test Loss: 0.05950223654508591, Test Accuracy: 98.15666961669922

Epoch 4, Loss: 0.05297775939106941, Accuracy: 98.38541412353516, Test Loss: 0.061324529349803925, Test Accuracy: 98.16999816894531

Epoch 5, Loss: 0.04419006407260895, Accuracy: 98.64633178710938, Test Loss: 0.06336682289838791, Test Accuracy: 98.18800354003906