COSC 4337

MNIST_using gradient_totrain

[3]:
```
import tensorflow as tf
tf.enable_eager_execution()

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

[4]:
```
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

# Add a channels dimension
x_train = x_train[..., tf.newaxis].astype("float32")
x_test = x_test[..., tf.newaxis].astype("float32")
```

[5]:
```
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)
```

[6]:
```
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)

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)

# Create an instance of the model
model = MyModel()
```

[7]:
```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

optimizer = tf.keras.optimizers.Adam()
```

[8]:
```
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')
```

[9]:
```
@tf.function
def train_step(images, labels):
    with tf.GradientTape() as tape:
    # training=True is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).training=True
        predictions = model(images)
        loss = loss_object(labels, predictions)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

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

[10]:
```
@tf.function
def test_step(images, labels):
  # training=False is only needed if there are layers with different
  # behavior during training versus inference (e.g. Dropout)., training=False
    predictions = model(images)
    t_loss = loss_object(labels, predictions)

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

[11]:
```
EPOCHS = 5

for epoch in range(EPOCHS):
  # Reset the metrics at the start of the next epoch
        train_loss.reset_states()
        train_accuracy.reset_states()
        test_loss.reset_states()
        test_accuracy.reset_states()

        for images, labels in train_ds:
            train_step(images, labels)
        for test_images, test_labels in test_ds:
            test_step(test_images, test_labels)
```

```python
        print(
        f'Epoch {epoch + 1}, '
        f'Loss: {train_loss.result()}, '
        f'Accuracy: {train_accuracy.result() * 100}, '
        f'Test Loss: {test_loss.result()}, '
        f'Test Accuracy: {test_accuracy.result() * 100}'
        )
```

```
Epoch 1, Loss: 0.13104157149791718, Accuracy: 96.08000183105469, Test Loss:
0.07665331661701202, Test Accuracy: 97.44999694824219
Epoch 2, Loss: 0.041010525077581406, Accuracy: 98.75333404541016, Test Loss:
0.05792670324444771, Test Accuracy: 98.18999481201172
Epoch 3, Loss: 0.01980713941156864, Accuracy: 99.41500091552734, Test Loss:
0.06558661162853241, Test Accuracy: 98.11000061035156
Epoch 4, Loss: 0.01392700057476759, Accuracy: 99.53166961669922, Test Loss:
0.058207638561725616, Test Accuracy: 98.3699951171875
Epoch 5, Loss: 0.009554926306009293, Accuracy: 99.67333221435547, Test Loss:
0.06787833571434021, Test Accuracy: 98.29000091552734
```