MNIST 손글씨
TF 튜토리얼

TF2 Quickstart Experts

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/advanced.ipynb?hl=ko

파일

- advanced.ipynb

데이터 로드와 전처리

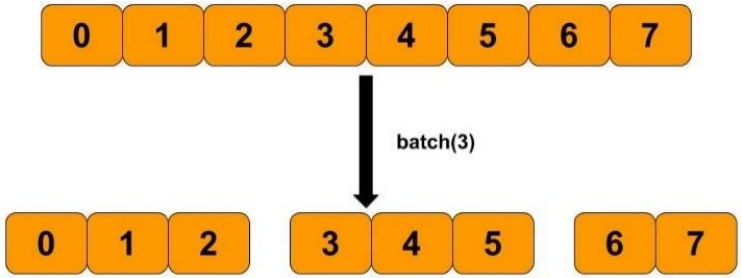
```
[1]  1 import tensorflow as tf
     2
     3 from tensorflow.keras.layers import Dense, Flatten, Conv2D
     4 from tensorflow.keras import Model
```

Load and prepare the MNIST dataset.

```
[3]  1 mnist = tf.keras.datasets.mnist
     2
     3 (x_train, y_train), (x_test, y_test) = mnist.load_data()
     4 x_train, x_test = x_train / 255.0, x_test / 255.0
     5
     6 # x_train : (NUM_SAMPLE, 28, 28) -> (NUM_SAMPLE, 28, 28 , 1)
     7 # ...은 해당 데이터 객체의 모든 axis를 표현
     8 # 위에서 255.0으로 나누어주게 되면 float64로 되므로 자료형을 float32로 해야 error가 없다.
     9 ## x_train[:,:,:, tf.newaxis]
    10 # Add a channels dimension
    11 x_train = x_train[..., tf.newaxis].astype("float32")
    12 x_test = x_test[..., tf.newaxis].astype("float32")
```

DataSet 저장

- **Dataset 사용**
  - 데이터를 섞고 iterator를 사용
  - 모델에 공급할 dataset으로부터 일정부분(batch)데이터를 가져옴
  - from_tensor_slices(tensors)
    - **텐서에서 일부 또는 전체를 반환**
  - shuffle(n): 섞을 버퍼 수
  - batch(n): 다음 작업에 할당할 수 지정
    - **for 문의 시퀀스에 사용**
    - **batch_size**
      - **훈련에서 가중치와 편향의 패러미터를 수정하는 데이터 단위 수**




```
# 원 자료를 섞어서 DataSet에 저장
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(10000).batch(32)

# 테스트 자료도 DataSet에 저장
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

tf.keras.Model 클래스를 상속

- _init_ 메서드에서 층을 만들어 클래스 객체의 속성으로 지정
- 정방향 패스는 call 메서드에 정의

```
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()
```

손실 함수와 최적화 방법

Choose an optimizer and loss function for training:

```
[6]  1 loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
     2
     3 optimizer = tf.keras.optimizers.Adam()
```

Select metrics to measure the loss and the accuracy of the model. These metrics accumulate the values over epochs and then print the overall result.

```
[7]  1 train_loss = tf.keras.metrics.Mean(name='train_loss')
     2 train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')
     3
     4 test_loss = tf.keras.metrics.Mean(name='test_loss')
     5 test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

tf.keras.metrics.SparseCategoricalAccuracy()

- 예측이 맞은 정확도(맞은 확률)를 반환

```
[11]  1 m = tf.keras.metrics.SparseCategoricalAccuracy()
      2 _ = m.update_state([[2], [1]], [[0.1, 0.9, 0.8], [0.05, 0.95, 0]])
      3 print(m.result().numpy())          # 틀림          # 맞음
      4
      5 _ = m.update_state([[2], [1]], [[0.1, 0.1, 0.8], [0.05, 0.95, 0]])
      6 print(m.result().numpy())          # 맞음          # 맞음
```

```
0.5
0.75
```

계산해 반영

```
1 m.reset_states() # 다시 시작
2 _ = m.update_state([[2], [1]], [[0.1, 0.9, 0.8], [0.05, 0.95, 0]],
3                    sample_weight=[0.7, 0.3])
4 m.result().numpy()                # 가중치
```

```
0.3
```

tf.keras.metrics.Mean()

- 평균 구하기

```
[12]  1 m.reset_states()
      2 _ = m.update_state([[2], [1]], [[0.1, 0.9, 0.8], [0.05, 0.95, 0]],
      3                    sample_weight=[0.7, 0.3])
      4 m.result().numpy()
```

```
0.3
```

```
[13]  1 m = tf.keras.metrics.Mean()
      2 _ = m.update_state([1, 3, 5, 7])
      3 print(m.result().numpy())
      4
      5 m.reset_states()
      6 _ = m.update_state([1, 3, 5, 7], sample_weight=[1, 1, 0, 0])
      7 print(m.result().numpy())
```

```
4.0
2.0
```

모델을 tf.GradientTape로 학습

- **클래스 GradientTape**
  - 자동 미분(주어진 입력 변수에 대한 연산의 그래디언트(gradient)를 계산하는 것) 수행
  - 실행된 모든 연산을 테이프(tape)에 "기록"
    - **테이프에 "기록된" 연산의 그래디언트를 계산**
      - **후진 방식 자동 미분(reverse mode differentiation)을 사용**
  - 입력 W와 b에 대한 loss의 미분 값 자동 계산
    - **tape.gradient(loss, model.trainable_variables)**
  - 예측 값과 손실을 계산하여, 손실에 대한 [w, b]의 미분 값인 gradients를 최적화 과정에 적용
    - **optimizer.apply_gradients(zip(gradients, model.trainable_variables))**

훈련 데이터는 최적화 과정에 반영

```
@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).
    predictions = model(images, training=True)
    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)
```

테스트 함수

검증이나 테스트 데이터는
최적화 과정에 반영하지 않으며,
손실 값과 정확도 만을 측정함

```
@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).
  predictions = model(images, training=False)
  t_loss = loss_object(labels, predictions)

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

손실과 정확도를 계산에 반영

5회 학습

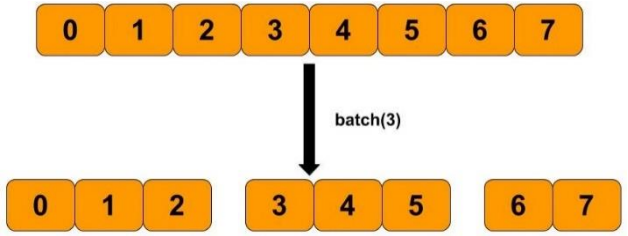
```
EPOCHS = 5

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)

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




```
Epoch 1, Loss: 0.1323878914117813, Accuracy: 95.98666381835938, Test Loss: 0.06412370502948761, Test Accuracy: 97.88999938964844
Epoch 2, Loss: 0.04152834042906761, Accuracy: 98.73832702636719, Test Loss: 0.044795189052820206, Test Accuracy: 98.58999633789062
Epoch 3, Loss: 0.021145612001419067, Accuracy: 99.32500457763672, Test Loss: 0.05164792388677597, Test Accuracy: 98.30999755859375
Epoch 4, Loss: 0.012047547847032547, Accuracy: 99.62999725341797, Test Loss: 0.06204463541507721, Test Accuracy: 98.30999755859375
Epoch 5, Loss: 0.008747215382754803, Accuracy: 99.71333312988281, Test Loss: 0.05799045041203499, Test Accuracy: 98.44999694824219
```

데이터셋 tf.DataSet

```
[17]  1 dataset = tf.data.Dataset.range(14)
      2 dataset = dataset.batch(3, drop_remainder=True)
      3 list(dataset.as_numpy_iterator())
```

```
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8]), array([ 9, 10, 11])]
```

```
[18]  1 dataset = tf.data.Dataset.range(14)
      2 dataset = dataset.shuffle(5).batch(3)
      3 list(dataset.as_numpy_iterator())
```

```
[array([4, 0, 3]),
 array([6, 5, 1]),
 array([10,  9,  2]),
 array([12,  8,  7]),
 array([11, 13])]
```

메소드 from_tensor_slices()

- **tf.data.Dataset.from_tensor_slices()**

```
[20]  1 dataset = tf.data.Dataset.range(14)
      2 dataset = dataset.shuffle(5).batch(4)
      3 for i in dataset:
      4     print(i)
```

```
tf.Tensor([0 5 2 1], shape=(4,), dtype=int64)
tf.Tensor([ 8  7  9 11], shape=(4,), dtype=int64)
tf.Tensor([ 6 10 13 12], shape=(4,), dtype=int64)
tf.Tensor([3 4], shape=(2,), dtype=int64)
```

```
[24]  1 train = tf.data.Dataset.from_tensor_slices(([1, 2, 3, 4, 5, 6], [11, 12, 13, 14, 15, 16])).shuffle(2).batch(4)
      2
      3 for x, y in train:
      4   print(x, y)
```

```
tf.Tensor([2 3 4 5], shape=(4,), dtype=int32) tf.Tensor([12 13 14 15], shape=(4,), dtype=int32)
tf.Tensor([1 6], shape=(2,), dtype=int32) tf.Tensor([11 16], shape=(2,), dtype=int32)
```

```
import tensorflow as tf

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

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 : (NUM_SAMPLE, 28, 28) -> (NUM_SAMPLE, 28, 28 , 1)
# ...은 해당 데이터 객체의 모든 axis를 표현
# 위에서 255.0으로 나누어주게 되면 float64로 되므로 자료형을 float32로 해야 error가 없다.
## x_train[:,:,:, tf.newaxis]
# Add a channels dimension
x_train = x_train[..., tf.newaxis].astype("float32")
x_test = x_test[..., tf.newaxis].astype("float32")

# 원 자료를 섞어서 DataSet에 저장
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(10000).batch(32)
# 테스트 자료도 DataSet에 저장
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)

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):
  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()

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

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')
```

```
@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).
    predictions = model(images, training=True)
    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)

@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).
  predictions = model(images, training=False)
  t_loss = loss_object(labels, predictions)

  test_loss(t_loss)
  test_accuracy(labels, predictions)

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)

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