4. 훈련 루프 상세

주요 내용

신경망 모델의 `fit()` 메서드를 실행할 때 텐서플로우 내부에서 훈련 루프가 진행되는 과정을 상세히 살펴보기

모델 지정

- 세 개의 `Dense` 층으로 구성된 순차 모델을 MNIST 데이터셋을 이용하여 훈련

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

model = keras.Sequential([layers.Dense(256, activation="relu"),
                          layers.Dense(512, activation="relu"),
                          layers.Dense(10, activation="softmax")])
```

옵티마이저, 손실 함수, 평가지표 지정

- 일반적인 방법: 모델의 `compile()` 메서드 호출
- 여기서는 `compile()` 메서드를 실행하는 대신 컴파일 과정에 요구되는 API 직접 선언

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

옵티마이저 API 선언

- 아래코드: 모델 컴파일에 사용된 문자열 `'rmsprop'`에 해당

```
optimizer = keras.optimizers.RMSprop(learning_rate=1e-3)
```

손실 함수 API 선언

- `SparseCategoricalCrossentropy` 클래스: 0, 1, 2 등 정수 형식의 타깃(레이블)을 예측하는 다중 클래스 분류 모델을 훈련시키는 경우 사용
- 아래코드: 모델 컴파일에 사용된 문자열 `'sparse_categorical_crossentropy'`에 해당

```
loss_fn = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

평가지표 API 선언

- `SparseCategoricalAccuracy` 클래스: 0, 1, 2 등 정수 형식의 타깃(레이블)을 예측하는 다중 클래스 분류 모델을 훈련시키는 경우 사용
- 아래코드: 모델 컴파일에 사용된 문자열 `'accuracy'`에 해당

```
train_acc_metric = keras.metrics.SparseCategoricalAccuracy()
```

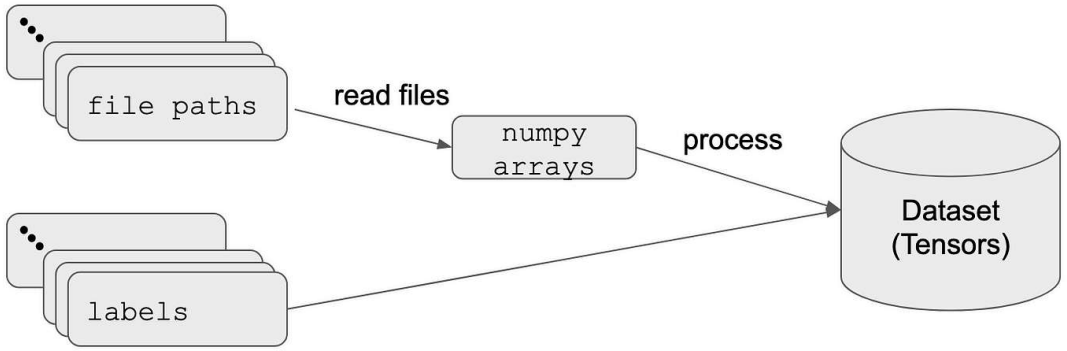
데이터셋 준비

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = np.reshape(x_train, (-1, 784))
x_test = np.reshape(x_test, (-1, 784))
```

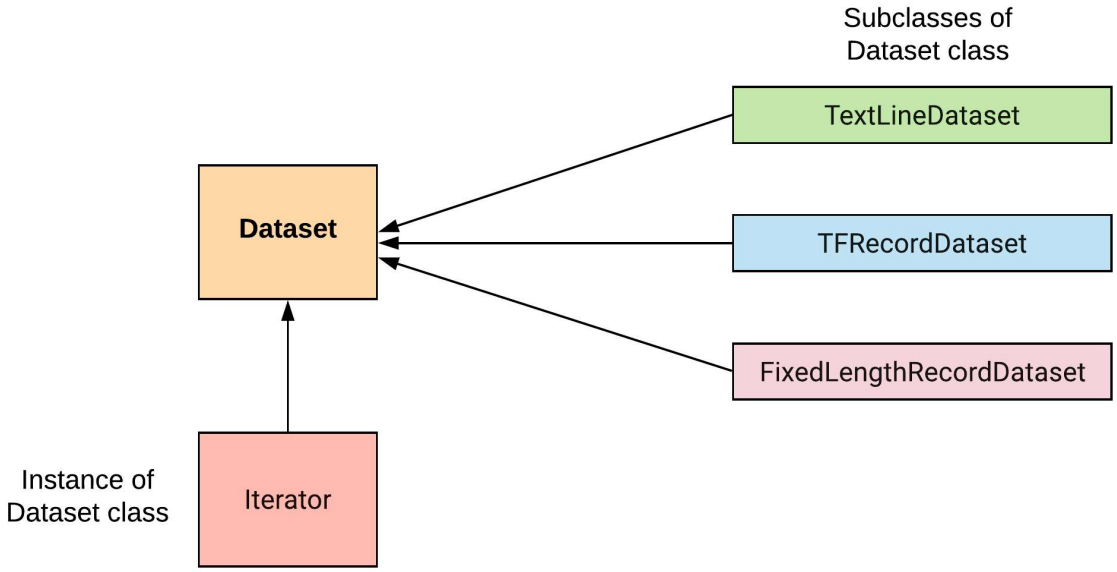
배치 묶음 `Dataset` 객체 지정

- 텐서플로우의 `tf.data.Dataset` 자료형 활용
- `tf.data.Dataset`: 머신러닝 모델 훈련에 사용되는 대용량 데이터의 효율적인 처리 지원



`tf.data.Dataset`의 자식 클래스

- `tf.data.Dataset`의 다양한 API(메서드)를 이용하면 적정한 배치를 모델 훈련에 제공하는 이터러블 객체를 생성
- 아래 이미지: 대용량 데이터를 다룰 때 유용한 `tf.data.Dataset`의 자식 클래스들을 보여줌



이터러블 객체

- 이터러블iterable 객체: `for` 반복문에 활용될 수 있는 값
- 참고: 이터러블, 이터레이터, 제너레이터

MNIST 훈련과 테스트용 `Dataset` 객체 지정

```
batch_size = 128

# 훈련용 Dataset 객체
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(batch_size)
```

훈련 루프

```
epochs = 10

for epoch in range(epochs):
    for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch_train, training=True)
            loss_value = loss_fn(y_batch_train, logits)

        grads = tape.gradient(loss_value, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
        train_acc_metric.update_state(y_batch_train, logits)

        if step % 100 == 0 and step > 0:
            print(f"  - {step}번째 스텝 손실값: {loss_value:.4f}")

    train_acc = train_acc_metric.result()
    print(f"  - 에포크 훈련 후 모델 정확도: {train_acc:.4f}")

    train_acc_metric.reset_state()
```

`@tf.function` 데코레이터

- 텐서플로우의 텐서를 다루는 함수에 `@tf.function` 데코레이터를 추가
- 훈련 스텝 과정에 `@tf.function` 데코레이터를 적용하면 모델 훈련 속도가 7-8배 빨라짐

```
@tf.function
def train_step(x, y):
    with tf.GradientTape() as tape:
        logits = model(x, training=True)
        loss_value = loss_fn(y, logits)
    grads = tape.gradient(loss_value, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
    train_acc_metric.update_state(y, logits)
    return loss_value
```

보다 빠른 훈련 루프

```
epochs = 10
for epoch in range(epochs):
    for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
        loss_value = train_step(x_batch_train, y_batch_train)

        if step % 100 == 0 and step > 0:
            print(f"  - {step}번째 스텝 손실값: {loss_value:.4f}")

    train_acc = train_acc_metric.result()
    print(f"  - 에포크 훈련 후 모델 정확도: {train_acc:.4f}")

    train_acc_metric.reset_state()
```

주의사항

- `@tf.function` 데코레이터를 추가한다 해서 모델 훈련 속도가 항상 빨라지는 것은 아님
- 보다 자세한 내용은 Better performance with tf.function 참고.