13.1.3 ~ 13.1.6
데이터 전처리 ~ 데이터셋 사용하기

2021-04-11 백관구

13.1.3. **데이터 전처리**
~ 13.1.4. **데이터 적재와 전처리를 합치기**

- 앞에서는 대규모 데이터를 다루는 데에 유용하게 사용할 수 있는 방법들(적재, 셔플)을 소개함
- 13.1.3 ~ 13.1.4 에서는 앞에서 소개한 데이터 적재에 전처리(정규화, 표준화) 기법을 적용하는 방법을 소개할 예정

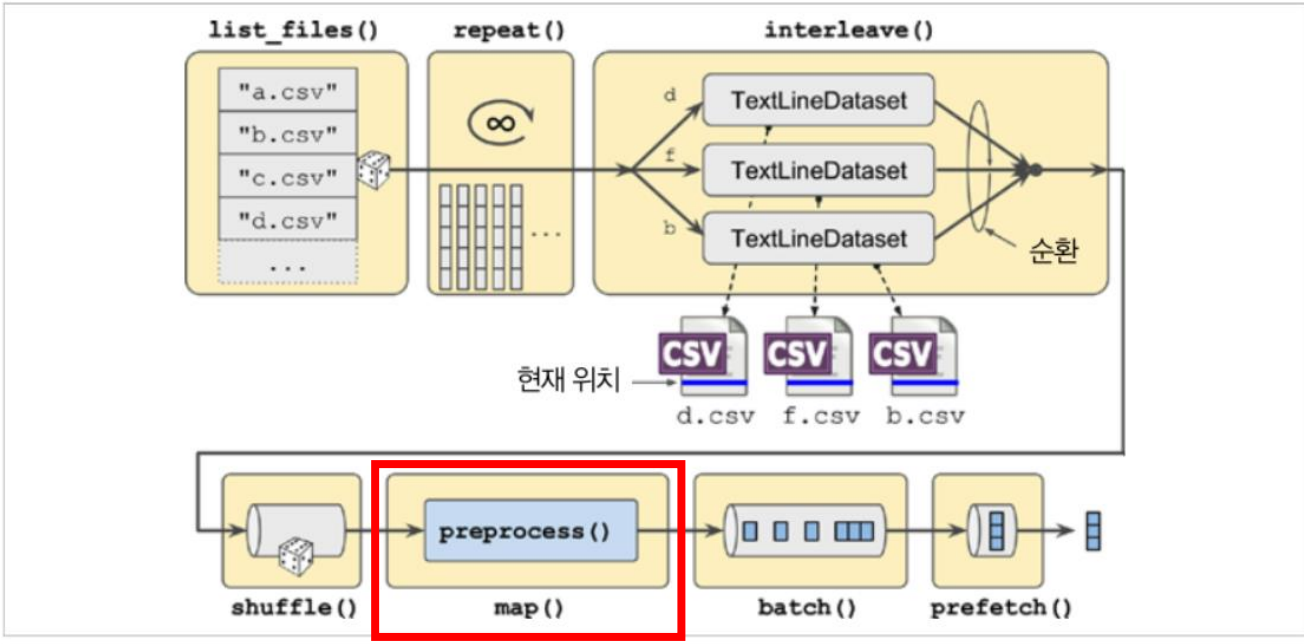


그림 13-2 여러 CSV 파일에서 데이터를 적재하고 전처리하기

13.1.3. **데이터 전처리**

- preprocess 라는 표준화 함수를 생성
- 전제 조건: 훈련 세트에 있는 각 특성의 통계치(평균, 표준편차)를 알고 있어야 함

```
X_mean, X_std = [...]   # 훈련 세트에 있는 각 특성의 평균과 표준편차
n_inputs = 8

def preprocess(line):
  defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
  fields = tf.io.decode_csv(line, record_defaults=defs)
  x = tf.stack(fields[:-1])
  y = tf.stack(fields[-1:])
  return (x - X_mean) / X_std, y
```

13.1.3. 데이터 전처리

- preprocess 라는 표준화 함수를 생성
- 전제 조건: 훈련 세트에 있는 각 특성의 통계치(평균, 표준편차)를 알고 있어야 함

```
X_mean, X_std = [...]   # 훈련 세트에 있는 각 특성의 평균과 표준편차
n_inputs = 8
```

X_mean = [20, 21, ..., 4]
X_std = [10, 11, ..., 0.5]

```
def preprocess(line):
  defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
  fields = tf.io.decode_csv(line, record_defaults=defs)
  x = tf.stack(fields[:-1])
  y = tf.stack(fields[-1:])
  return (x - X_mean) / X_std, y
```

13.1.3. 데이터 전처리

- preprocess 라는 표준화 함수를 생성
- 전제 조건: 훈련 세트에 있는 각 특성의 통계치(평균, 표준편차)를 알고 있어야 함

```
X_mean, X_std = [...]   # 훈련 세트에 있는 각 특성의 평균과 표준편차
n_inputs = 8
```

```
preprocess(b'4.2083,44.0,5.3232,0.9171,846.0,2.3370,37.47,-122.2,2.782')
```

```
def preprocess(line):
  defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
  fields = tf.io.decode_csv(line, record_defaults=defs)
  x = tf.stack(fields[:-1])
  y = tf.stack(fields[-1:])
  return (x - X_mean) / X_std, y
```

13.1.3. 데이터 전처리

- preprocess 라는
- 전제 조건: 훈련 서
  차)를 알고 있어야

```
defs = [0.] * 8 + [tf.constant([], dtype = tf.float32)]
defs
```

↑기본값 없음(만약 누락된 값이 입력되면 에러 발생)

[2] ✓ 0.1s

```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
<tf.Tensor: shape=(0,), dtype=float32, numpy=array([], dtype=float32)>]
```

```
X_mean, X_std = [...]
n_inputs = 8

def preprocess(line):
    defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
    fields = tf.io.decode_csv(line, record_defaults=defs)
    x = tf.stack(fields[:-1])
    y = tf.stack(fields[-1:])
    return (x - X_mean) / X_std, y
```

13.1.3. 데이터 전처리

- preprocess 라는 표준화 함수를 생성
- 전제 조건: 훈련 세트에 있는 각 특성의 통계치(평균, 표준편차)를 알고 있어야 함

```
X_mean, X_std = [...]   # 훈련 세트에 있는 각 특성의 평균과 표준편차
n_inputs = 8
```

x y

```
preprocess(b'4.2083,44.0,5.3232,0.9171,846.0,2.3370,37.47,-122.2,2.782')
```

```
def preprocess(line):
  defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
  fields = tf.io.decode_csv(line, record_defaults=defs)
  x = tf.stack(fields[:-1])
  y = tf.stack(fields[-1:])
  return (x - X_mean) / X_std, y
```

↑표준화

13.1.3. 데이터 전처리

```
X_mean, X_std = [...]   # 훈련 세트에 있는 각 특성의 평균과 표준편차
n_inputs = 8

def preprocess(line):
  defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
  fields = tf.io.decode_csv(line, record_defaults=defs)
  x = tf.stack(fields[:-1])
  y = tf.stack(fields[-1:])
  return (x - X_mean) / X_std, y
```

```
>>> preprocess(b'4.2083,44.0,5.3232,0.9171,846.0,2.3370,37.47,-122.2,2.782')
(<tf.Tensor: id=6227, shape=(8,), dtype=float32, numpy=
 array([ 0.16579159,  1.216324  , -0.05204564, -0.39215982, -0.5277444 ,
        -0.2633488 ,  0.8543046 , -1.3072058 ], dtype=float32)>,
 <tf.Tensor: [...], numpy=array([2.782], dtype=float32)>)
```

13.1.4. 데이터 적재와 전처리를 합치기

```
def csv_reader_dataset(filepaths, repeat=1, n_readers=5,
                       n_read_threads=None, shuffle_buffer_size=10000,
                       n_parse_threads=5, batch_size=32):
    dataset = tf.data.Dataset.list_files(filepaths).repeat(repeat)
    dataset = dataset.interleave(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
        cycle_length=n_readers, num_parallel_calls=n_read_threads)
    dataset = dataset.shuffle(shuffle_buffer_size)
    dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
    return dataset.batch(batch_size).prefetch(1)
```

- map 메서드: 각 아이템에 preprocess 변환을 적용
- num_parallel_calls: 병렬로 작업을 수행 ➔ 더 빠른 작업 (n_parse_threads 수만큼 자원을 활용)

13.1.3. 데이터 전처리

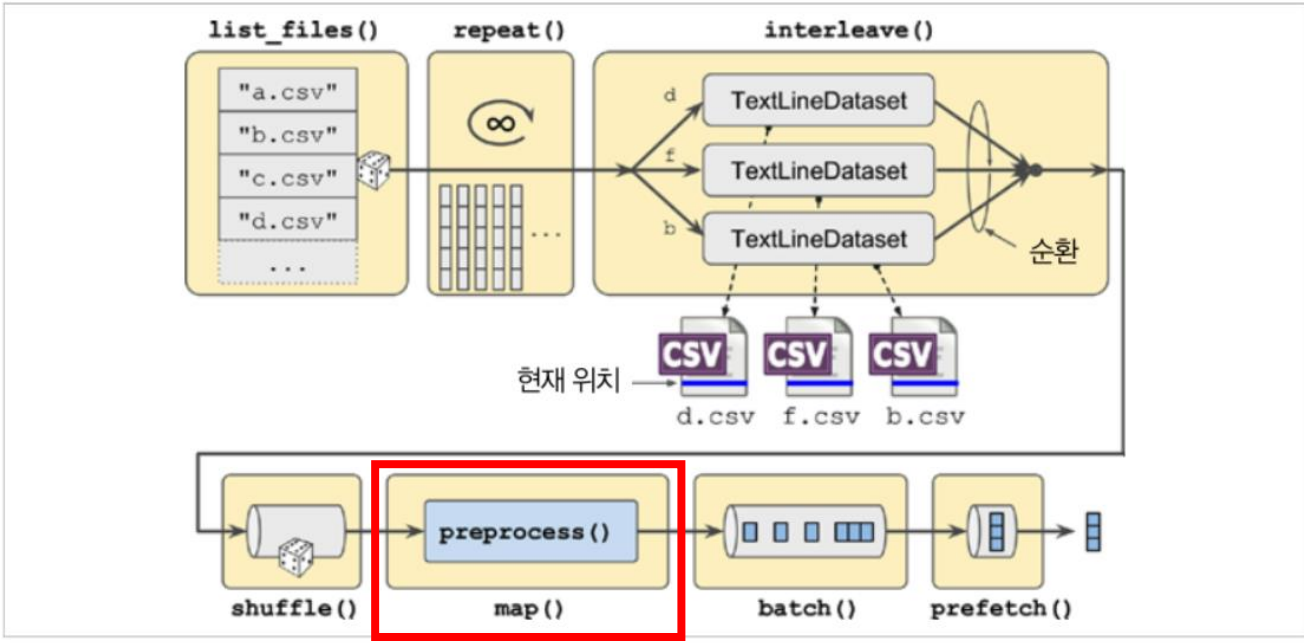


그림 13-2 여러 CSV 파일에서 데이터를 적재하고 전처리하기

13.1.5. **프리페치** (prefetch; **선인출**)

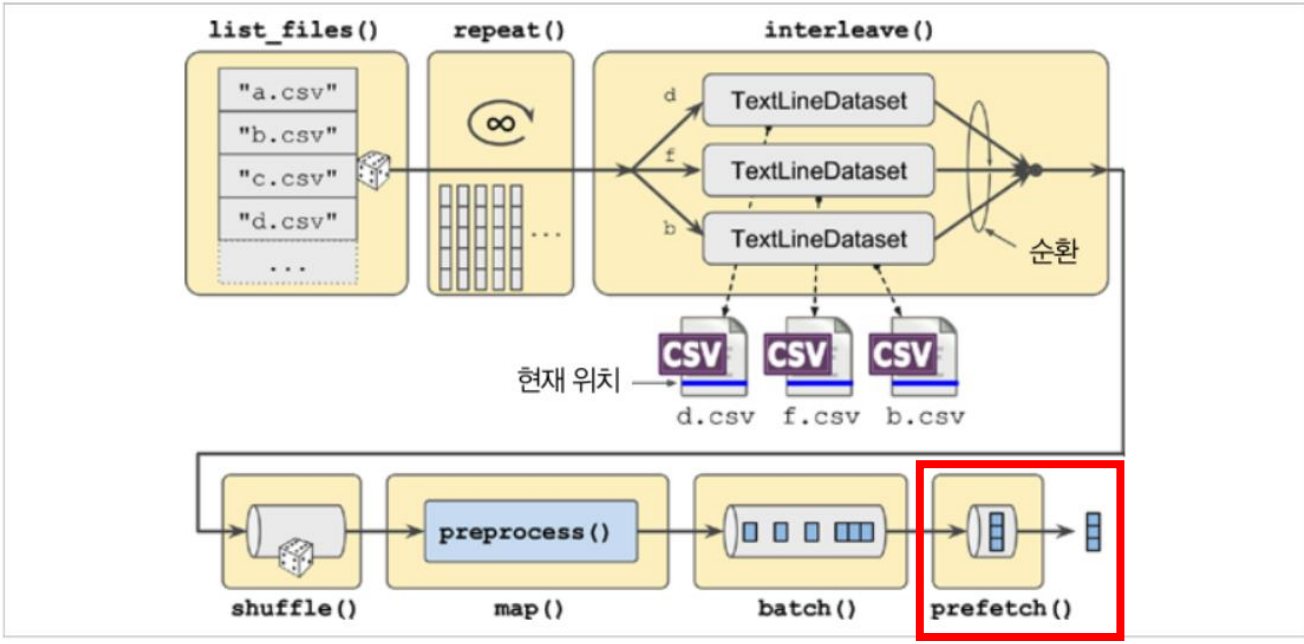


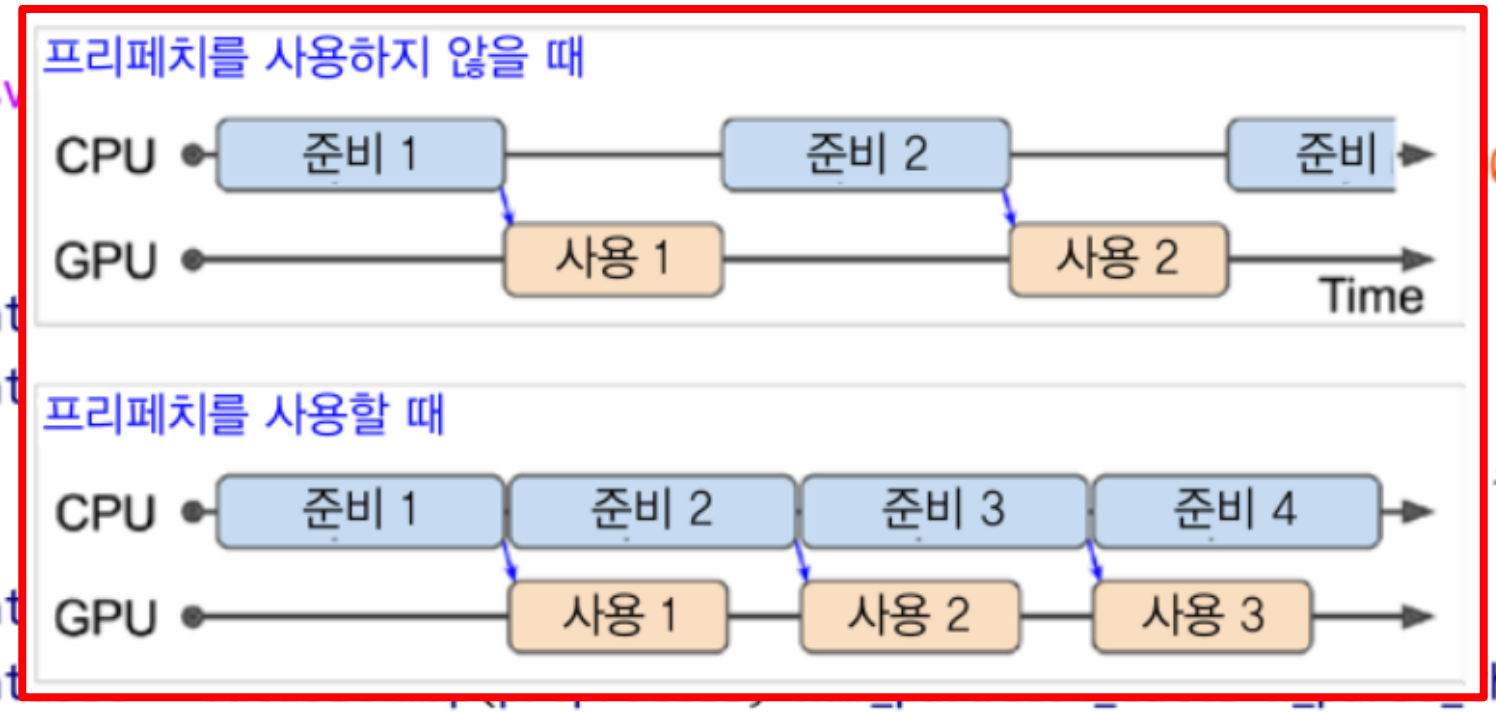
그림 13-2 여러 CSV 파일에서 데이터를 적재하고 전처리하기

13.1.5. **프리페치** (prefetch)

```
def csv_reader_dataset(filepaths, repeat=1, n_readers=5,
                       n_read_threads=None, shuffle_buffer_size=10000,
                       n_parse_threads=5, batch_size=32):
    dataset = tf.data.Dataset.list_files(filepaths).repeat(repeat)
    dataset = dataset.interleave(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
        cycle_length=n_readers, num_parallel_calls=n_read_threads)
    dataset = dataset.shuffle(shuffle_buffer_size)
    dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
    return dataset.batch(batch_size).prefetch(1)
```

- prefetch(1): 훈련 알고리즘이 한 배치로 작업하는 동안, 동시에 다음 훈련시킬 1개의 배치를 미리 준비함

13.1.5. **프리페치** (prefetch)




```
def csv
                                                                    000,
    dat
    dat
                                                                    ,
    dat
    dat                                                             hreads)
    return dataset.batch(batch_size).prefetch(1)
```

- prefetch(1): 훈련 알고리즘이 한 배치로 작업하는 동안, 동시에 다음 훈련시킬 1개의 배치를 미리 준비함

13.1.5. 프


```
def csv_reader_datase

    dataset = tf.data
    dataset = dataset
        lambda filepa
        cycle_length=n_readers, num_parallel_calls=n_read_threads)
    dataset = dataset.shuffle(shuffle_buffer_size)
    dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
    return dataset.batch(batch_size).prefetch(1)
```

- 데이터 적재와 전처리에 num_parallel_calls를 사용해 병렬 처리와 동시에, 프리페치를 적용하면 훈련 속도 면에서 더욱 향상됨

13.1.6. tf.keras와 데이터셋 사용하기

- 13.1.5까지 구성한 csv_reader_dataset 함수를 실제로 사용하는 방법 ➔ 간단함!

```
# 훈련, 검증, 테스트 데이터셋 설정
train_set = csv_reader_dataset(train_filepaths)
valid_set = csv_reader_dataset(valid_filepaths)
test_set = csv_reader_dataset(test_filepaths)

# 케라스 모델 학습
model.fit(train_set, epochs=10, validation_data=valid_set)

# 케라스 모델 테스트 예측
model.evaluate(test_set)
new_set = test_set.take(3).map(lambda X, y: X)    # 새로운 샘플이 3개 있다고 가정합니다.
model.predict(new_set)    # 새로운 샘플이 들어 있는 데이터셋
```