Week4

CH13. 텐서플로에서 데이터 적재와 전처리하기

아주 큰 규모의 데이터셋 효율적으로 로드하고 전처리해보자

- TF 변환
- TF 데이터셋

13.1 데이터 API

데이터셋: 연속된 데이터 샘플

```
#각 원소가 아이템으로 표현되는 데이터셋 생성
>>> X = tf.range(10)
>>> dataset = tf.data.Dataset.from_tensor_slices(X)
>>> dataset
<TensorSliceDataset shapes: (), types: tf.int32>

dataset = tf.data.Dataset.range(10) #위 코드와 같은 의미

#데이터셋의 아이템 순회
>>> for item in dataset:
>>>     print(item)
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
...
tf.Tensor(9, shape=(), dtype=int64)
```

13.1.1 연쇄 변환

데이터셋이 준비되면,

변환 메서드를 호출 → 여러 종류의 변환 수행

```
>>> dataset = dataset.repeat(3).batch(7)
>>> for item in dataset:
>>>     print(item)

tf.Tensor([0 1 2 3 4 5 6], shape=(7,), dtype=int64)
```

```
tf.Tensor([7 8 9 0 1 2 3], shape=(7,), dtype=int64)
tf.Tensor([4 5 6 7 8 9 0], shape=(7,), dtype=int64)
tf.Tensor([1 2 3 4 5 6 7], shape=(7,), dtype=int64)
tf.Tensor([8 9], shape=(2,), dtype=int64) #배치 크기 2 #drop_reminder = True 설정하면, 해당 출력 제외
```

`map( )` 메서드: 각 아이템에 변환 적용

`apply( )` 메서드: 데이터셋 전체에 변환 적용

`take( )` 메서드: 몇 개의 아이템만 보고 싶을 때 사용

```
>>> dataset = dataset.map(lambda x: x * 2) # 아이템:[0,2,4,6,8,10...]

>>> dataset = dataset.unbatch() # 각 아이템 = 하나의 정수 텐서 (7개의 정수로 이루어진 배치 X)
>>> dataset = dataset.filter(lambda x: x < 10) # 아이템: 0 2 4 6 8 0 2 4 6

>>> for item in dataset.take(3):
>>>     print(item)

tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
```

13.1.2 데이터 셔플링

`shuffle( )` 메서드로 샘플 섞기

<여러 파일에서 한 줄씩 번갈아 읽기>

`list_files( )` 함수: 파일 경로를 섞은 데이터셋 반환

`interleave( )` 메서드: 여러개의 파일을 한 줄씩 번갈아 읽을 때 사용

`skip( )` 메서드: 줄을 건너뛸 때 사용

```
#데이터셋 적재 및 분할 + 스케일 조정

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()
X_train_full, X_test, y_train_full, y_test = train_test_split(
    housing.data, housing.target.reshape(-1, 1), random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_full, y_train_full, random_state=42)

scaler = StandardScaler()
scaler.fit(X_train)
X_mean = scaler.mean_
X_std = scaler.scale_

#파일 분할
>>> train_filepaths
['datasets/housing/my_train_00.csv',...,'datasets/housing/my_train_19.csv']

#파일 경로를 포함한 데이터셋 생성
filepath_dataset = tf.data.Dataset.list_files(train_filepaths, seed=42)

#한 번에 여러개의 파일 한 줄씩 번갈아 읽기
n_readers = 5
dataset = filepath_dataset.interleave(
    lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
    cycle_length=n_readers)

>>> for line in dataset.take(5):
>>>     print(line.numpy())

b'4.6477,38.0,5.03728813559322,0.911864406779661,745.0,2.5254237288135593,32.64,-117.07,1.504'
b'8.72,44.0,6.163179916317992,1.0460251046025104,668.0,2.794979079497908,34.2,-118.18,4.159'
b'3.8456,35.0,5.461346633416459,0.9576059850374065,1154.0,2.8778054862842892,37.96,-122.05,1.598'
b'3.3456,37.0,4.514084507042254,0.9084507042253521,458.0,3.2253521126760565,36.67,-121.7,2.526'
b'3.6875,44.0,4.524475524475524,0.993006993006993,457.0,3.195804195804196,34.04,-118.15,1.625'

#결과: 바이 스트링 -> 파싱 후 스케일 조정 필요

>>> record_defaults=[0, np.nan, tf.constant(np.nan, dtype=tf.float64), "Hello", tf.constant([])]
>>> parsed_fields = tf.io.decode_csv('1,2,3,4,5', record_defaults)
>>> parsed_fields
```

```
[<tf.Tensor: shape=(), dtype=int32, numpy=1>,
 <tf.Tensor: shape=(), dtype=float32, numpy=2.0>,
 <tf.Tensor: shape=(), dtype=float64, numpy=3.0>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'4'>, #여기
 <tf.Tensor: shape=(), dtype=float32, numpy=5.0>]
```

13.1.3 데이터 전처리

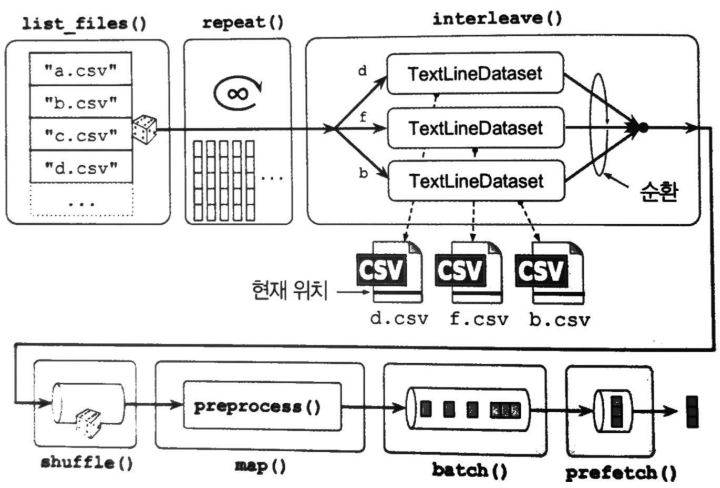
```
#전처리

n_inputs = 8 # X_train.shape[-1]

@tf.function
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: shape=(8,), dtype=float32, numpy=
 array([ 0.16579157,  1.216324  , -0.05204565, -0.39215982, -0.5277444 ,
        -0.2633488 ,  0.8543046 , -1.3072058 ], dtype=float32)>,
 <tf.Tensor: shape=(1,), dtype=float32, numpy=array([2.782], dtype=float32)>)
```

13.1.4 데이터 적재와 전처리를 합치기

다음 코드는 헬퍼 함수로

데이터셋을 효율적으로 적재, 전처리, 셔플링, 반복 배치를 적용한 데이터셋을 만들어 반환해준다.

```
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)
    dataset = dataset.batch(batch_size)
    return dataset.prefetch(1)
```

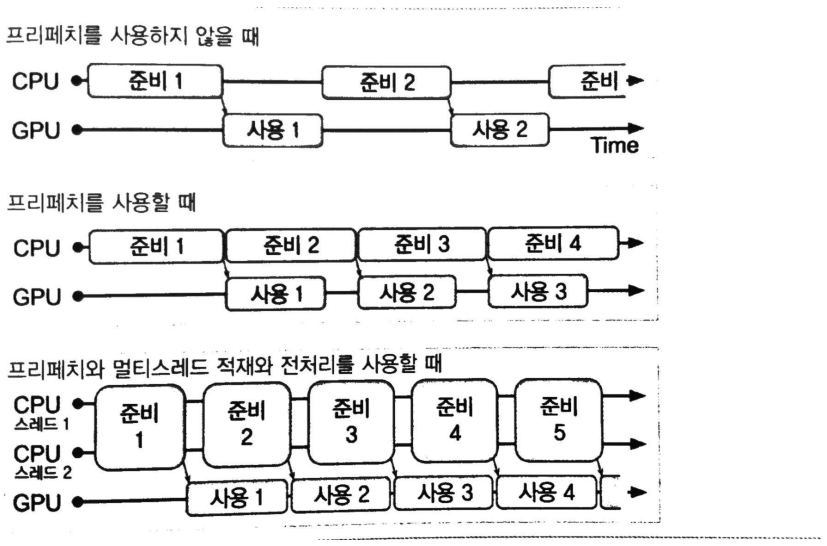


13.1.5 프리페치

`prefetch( )`

한 배치가 미리 준비되도록 함

→ 훈련 알고리즘이 한 배치로 작업하는 동안, 데이터셋이 동시에 다음 배치를 준비



13.1.6 tf.keras와 데이터셋 사용하기

```
train_set = csv_reader_dataset(train_filepaths, repeat=None)
valid_set = csv_reader_dataset(valid_filepaths)
test_set = csv_reader_dataset(test_filepaths)

keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)

model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1),
])

model.compile(loss="mse", optimizer=keras.optimizers.SGD(learning_rate=1e-3))

batch_size = 32
model.fit(train_set, steps_per_epoch=len(X_train) // batch_size, epochs=10,
          validation_data=valid_set)

model.evaluate(test_set, steps=len(X_test) // batch_size)

new_set = test_set.map(lambda X, y: X)
X_new = X_test
model.predict(new_set, steps=len(X_new) // batch_size)
```

```
#자신만의 훈련 반복 생성
optimizer = keras.optimizers.Nadam(learning_rate=0.01)
loss_fn = keras.losses.mean_squared_error

n_epochs = 5
batch_size = 32
n_steps_per_epoch = len(X_train) // batch_size
total_steps = n_epochs * n_steps_per_epoch
global_step = 0
for X_batch, y_batch in train_set.take(total_steps):
    global_step += 1
    print("\rGlobal step {}/{}".format(global_step, total_steps), end="")
    with tf.GradientTape() as tape:
        y_pred = model(X_batch)
        main_loss = tf.reduce_mean(loss_fn(y_batch, y_pred))
        loss = tf.add_n([main_loss] + model.losses)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

```
#전체 훈련 반복 생성
optimizer = keras.optimizers.Nadam(learning_rate=0.01)
loss_fn = keras.losses.mean_squared_error

@tf.function
def train(model, n_epochs, batch_size=32,
          n_readers=5, n_read_threads=5, shuffle_buffer_size=10000, n_parse_threads=5):
    train_set = csv_reader_dataset(train_filepaths, repeat=n_epochs, n_readers=n_readers,
                       n_read_threads=n_read_threads, shuffle_buffer_size=shuffle_buffer_size,
                       n_parse_threads=n_parse_threads, batch_size=batch_size)
    n_steps_per_epoch = len(X_train) // batch_size
    total_steps = n_epochs * n_steps_per_epoch
    global_step = 0
    for X_batch, y_batch in train_set.take(total_steps):
        global_step += 1
        if tf.equal(global_step % 100, 0):
```

```
            tf.print("\rGlobal step", global_step, "/", total_steps)
        with tf.GradientTape() as tape:
            y_pred = model(X_batch)
            main_loss = tf.reduce_mean(loss_fn(y_batch, y_pred))
            loss = tf.add_n([main_loss] + model.losses)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

train(model, 5)
```

CSV 파일은 널리 통용되지만 효율적 X

대규모의 복잡한(이미지, 오디오) 데이토 구조 지원 X → TFRecord !

13.2 TFRecord 포맷

TFRecord: 크기가 다른 연속된 이진 레코드를 저장하는 단순한 이진 포맷

```
#TFRecord 만들기

with tf.io.TFRecordWriter("my_data.tfrecord") as f:
    f.write(b"This is the first record")
    f.write(b"And this is the second record")

#TFRecord 읽기

>>> filepaths = ["my_data.tfrecord"]
>>> dataset = tf.data.TFRecordDataset(filepaths)
>>> for item in dataset:
>>>     print(item)

tf.Tensor(b'This is the first record', shape=(), dtype=string)
tf.Tensor(b'And this is the second record', shape=(), dtype=string)
```

13.2.1 압축된 TFRecord 파일

네트워크를 통해 읽어야 하는 경우 등 → 압축할 필요 有

```
#압축하기

options = tf.io.TFRecordOptions(compression_type="GZIP")
with tf.io.TFRecordWriter("my_compressed.tfrecord", options) as f:
    f.write(b"This is the first record")
    f.write(b"And this is the second record")

#압축 형식 지정하여 파일 읽기

>>> dataset = tf.data.TFRecordDataset(["my_compressed.tfrecord"],
                                      compression_type="GZIP")
>>> for item in dataset:
>>>     print(item)

tf.Tensor(b'This is the first record', shape=(), dtype=string)
tf.Tensor(b'And this is the second record', shape=(), dtype=string)
```

13.2.2 프로토콜 버퍼 개요

프로토콜 버퍼(protocol buffer): TFRecord에 직렬화 된 포맷

```
#정의 - 프로토 파일 생성

%%writefile person.proto
syntax = "proto3";
message Person {
  string name = 1;
  int32 id = 2;
  repeated string email = 3;
}

#정의 컴파일

>>> !protoc person.proto --python_out=. --descriptor_set_out=person.desc --include_imports
>>> !ls person*
```

```
>>> from person_pb2 import Person

>>> person = Person(name="Al", id=123, email=["a@b.com"])  # Person 생성
>>> print(person)  # Person 출력

name: "Al"
id: 123
email: "a@b.com"

>>> person.name  # 필드 읽기
'Al'

>>> person.name = "Alice"  # 필드 수정
>>> person.email[0]  # 배열처럼 사용할 수 있는 반복 필드
'a@b.com'

>>> person.email.append("c@d.com")  # 이메일 추가
>>> s = person.SerializeToString()  # 바이트 문자열로 직렬화
>>> s
b'\n\x05Alice\x10{\x1a\x07a@b.com\x1a\x07c@d.com'

>>> person2 = Person()  # 새로운 Person 생성
>>> person2.ParseFromString(s)  # 바이트 문자열 파싱 (27 바이트)
27

>>> person == person2  # 동일
True
```

13.2.3 텐서플로 프로토콜 버퍼

텐서플로는 파싱 연산을 위한 특별한 프로토콜 버퍼 정의를 가지고 있음

- Example 프로토콜 버퍼 정의

```
syntax = "proto3";

message BytesList { repeated bytes value = 1; }
message FloatList { repeated float value = 1 [packed = true]; }
message Int64List { repeated int64 value = 1 [packed = true]; }
message Feature {
    oneof kind {
        BytesList bytes_list = 1;
        FloatList float_list = 2;
        Int64List int64_list = 3;
    }
};
message Features { map<string, Feature> feature = 1; };
message Example { Features features = 1; };
```

- 객체 생성 및 TFRecord 파일에 저장

```
#객체 생성
from tensorflow.train import BytesList, FloatList, Int64List
from tensorflow.train import Feature, Features, Example

person_example = Example(
    features=Features(
        feature={
            "name": Feature(bytes_list=BytesList(value=[b"Alice"])),
            "id": Feature(int64_list=Int64List(value=[123])),
            "emails": Feature(bytes_list=BytesList(value=[b"a@b.com", b"c@d.com"]))
        }))


#직렬화
with tf.io.TFRecordWriter("my_contacts.tfrecord") as f:
    f.write(person_example.SerializeToString())
```

13.2.3 Example 프로토콜 버퍼를 읽고 파싱하기

```
feature_description = {
    "name": tf.io.FixedLenFeature([], tf.string, default_value=""),  #기본값 표현
    "id": tf.io.FixedLenFeature([], tf.int64, default_value=0),
    "emails": tf.io.VarLenFeature(tf.string),                        #가변값 표현
}
for serialized_example in tf.data.TFRecordDataset(["my_contacts.tfrecord"]):
```

```
    parsed_example = tf.io.parse_single_example(serialized_example,
                                                feature_description)
```

고정 길이 특성 → 텐서로 파싱

가변 길이 특성 → 희소 텐서로 파싱

```
#희소 텐서를 밀집 텐서로 변환
>>> tf.sparse.to_dense(parsed_example["emails"], default_value=b"")
<tf.Tensor: shape=(2,), dtype=string, numpy=array([b'a@b.com', b'c@d.com'], dtype=object)>

#사실 상 희소 텐서 값을 바로 참조하는 것이 더 간단
>>> parsed_example["emails"].values
<tf.Tensor: shape=(2,), dtype=string, numpy=array([b'a@b.com', b'c@d.com'], dtype=object)>
```

이진 데이터도 포함할 수 있음

```
#이미지 넣기

#이미지 로드

data = tf.io.encode_jpeg(img)
example_with_image = Example(features=Features(feature={
    "image": Feature(bytes_list=BytesList(value=[data.numpy()]))}))
serialized_example = example_with_image.SerializeToString()
# then save to TFRecord

feature_description = { "image": tf.io.VarLenFeature(tf.string) }
example_with_image = tf.io.parse_single_example(serialized_example, feature_description)
decoded_img = tf.io.decode_jpeg(example_with_image["image"].values[0])

decoded_img = tf.io.decode_image(example_with_image["image"].values[0])

plt.imshow(decoded_img)
plt.title("Decoded Image")
plt.axis("off")
plt.show()
```

어떤 텐서라도 직렬화하고 저장할 수 있음

```
dataset = tf.data.TFRecordDataset(["my_contacts.tfrecord"]).batch(10)
for serialized_examples in dataset:
    parsed_examples = tf.io.parse_example(serialized_examples,
                                          feature_description)
```

13.2.4 SequenceExample 프로토콜 버퍼를 사용해 리스트의 리스트 다루기

```
#SequenceExample 프로토콜 버퍼 정의

message FeatureList { repeated Feature feature = 1; };
message FeatureLists { map<string, FeatureList> feature_list = 1; };
message SequenceExample {
  Features context = 1;
  FeatureLists feature_lists = 2;
};
```

Example을 만들고 직렬화하고 파싱하는 것과 비슷

- 튜플 반환
- 가변 길이의 시퀀스 → 레그드 텐서로 변환

```
parsed_context, parsed_feature_lists = tf.io.parse_single_sequence_example(
    serialized_sequence_example, context_feature_descriptions,
    sequence_feature_descriptions)
.
.
.
print(tf.RaggedTensor.from_sparse(parsed_feature_lists["content"]))
```

13.3 입력 특성 전처리

사용하고자 하는 데이터의 모든 특성을 수치 특성으로 변환 및 정규화 필요

```
#람다 층을 사용
#표준화 수행
mean = np.mean(x_train, axis = 0, keepdims=True)
stds = np.std(x_train, axis = 0, keepdims=True)
eps = keras.backend.epsilon()
model = keras.models.Squential([
    keras.layers.Lambda(lambda inputs: (inputs - means) / (stds + eps))

#완전한 사용자 정의층
class Standardization(keras.layers.Layer):
    def adapt(self, data_sample):
        self.means_ = np.mean(data_sample, axis=0, keepdims=True)
        self.stds_ = np.std(data_sample, axis=0, keepdims=True)
    def call(self, inputs):
        return (inputs - self.means_) / (self.stds_ + keras.backend.epsilon())
```

`adapt( )` 메서드

- Standardization 층 추가 전에 사용
- 데이터 샘플과 함께

```
std_layer = Standardization()
std_layers.adapt(data_sample)
```

13.3.1 원-핫 벡터를 사용해 범주형 특성 인코딩하기

범주형 특성 - 신경망 주입 전 인코딩해야 함

```
vocab = ['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']
indices = tf.range(len(vocab), dtype=tf.int64)
table_init = tf.lookup.KeyValueTensorInitializer(vocab, indices)
num_oov_buckets = 2
table = tf.lookup.StaticVocabularyTable(table_init, n_oov_buckets)
```

ovv 버킷: 샘플 데이터를 기반으로 어휘 사전을 정의하고, 샘플 데이터에 없는 다른 범주를 추가하는 것

- 데이터 셋이 너무 크거나
- 범주가 너무 자주 바뀌어서
- 전체 범주 리스트를 구하기 어려울 때 !

13.3.2 임베딩을 사용해 범주형 특성 인코딩하기

표현이 좋을수록 신경망은 정확한 예측을 만듦

→ 범주가 유용하게 표현되도록 임베딩이 훈련되는 경향

→ **표현 학습**

13.3.3 케라스 전처리 층

표준 전처리 층을 제공하기 위해 노력 중임

13.4 TF변환

전처리는 훈련과 동시에 하는 것보다 사전에 하는 것이 효율적

→ 아파치 빔이나 스파크와 같은 도구의 도움

하지만, 위와 같은 방식은 훈련/서빙 왜곡을 만들어 내어 버그나 성능 감소를 일으킴...

<개선 방향>

- 아파치 빔이나 스파크 코드로 전처리된 데이터에서 훈련된 모델을 받는다
- 앱이나 웹에 배포하기 전 전처리 담당하는 층을 동적으로 추가

→ 아파치 빔이나 스파크 코드 & 전처리 층 , 두 개의 버전으로 구분

13.5 텐서플로 데이터셋(TFDS) 프로젝트

표준 데이터 셋을 이용하고 싶다 ? → TFDS !

13.6 연습문제