Zaman Pencereleri (Time Windows)

Bu colab dosyasında bir zaman serisis makine öğrenmesi modellerini eğitmek için kullanabileceğimiz bir biçime nasıl dönüştüreceğimize bakacağız.

Bunun için elbette `tensorflow` paketlerinden faydalanacağız. TensorFlow'u içeri aktararak başlayalım.

In [1]:

```
import tensorflow as tf
```

Önceki 20 adımda verileri bir sonraki adımı tahmin etmek için bir model eğiteceğiz. Bu nedenle eğitim için 20 adımlk pencerelerden oluşan bir veri seti oluşturmamız gerekiyor.

Ama öncelikle, sıfırdan dokuza kadar sayılar içeren bir veri kümesi oluşturalım.

In [3]:

```
dataset = tf.data.Dataset.range(10)
for val in dataset:
    print(val)
```

```
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
tf.Tensor(5, shape=(), dtype=int64)
tf.Tensor(6, shape=(), dtype=int64)
tf.Tensor(7, shape=(), dtype=int64)
tf.Tensor(8, shape=(), dtype=int64)
tf.Tensor(9, shape=(), dtype=int64)
```

Toplam 10 tane tensör oluşturdul. `numpy()` işlevi ile bu tensörlerin sadece değerlerini alalım.

In [4]:

```
for val in dataset:
    print(val.numpy())
```

```
0
1
2
3
4
5
6
7
8
9
```

Oldukça iyi. Sırada `windows` metodunu çağıralım. Pencere boyutu olarak `5` değerini veriyoruz. Kaydırma değerimizi (`shift`) değerimizi `1` olarak atıyoruz. Bu sayede her pencere, önceki pencereye kıyasla bir zaman dilimi (adımı) kaydırılmış olur. `end = " "` parametresi sayesinde bir pencerenin tamamı yazdırılmadan

bir alt satıra inmesine engel oluyoruz.

In [5]:

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1)
for window_dataset in dataset:
    for val in window_dataset:
        print(val.numpy(), end=" ")
    print()
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```

Çıktılara baktığımızda son birkaç pencerenin diğer pencerelerden daha kısa olduğunu fark ediyoruz. Bunun sebebi zaman serisinin sonuna gelmemizdir. Makine öğrenmesi modelleri genellikle sabit boyutlu girdiler bekleyeceği için bu durumu düzeltmek için `drop_remainder = True` parametresini veriyoruz.

In [6]:

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
for window_dataset in dataset:
    for val in window_dataset:
        print(val.numpy(), end=" ")
    print()
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

Şuana kadar oldukça iyi gidiyoruz. Ancak, sahip olmak istediğimiz şey düzenli tensörler biçiminde veri yığınları içeren tek bir veri kümesidir. Bu durum için `flap_map` metodu yardımcı olacaktır.

In [7]:

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
for window in dataset:
    print(window.numpy())
```

```
[0 1 2 3 4]
[1 2 3 4 5]
[2 3 4 5 6]
[3 4 5 6 7]
[4 5 6 7 8]
[5 6 7 8 9]
```

Oldukça güzel, hedeflediğimiz şeyi başardık gibi görünüyor.

Makine öğrenmesi modelleri genellikle girdi özelliklerine ve **etiketlerine** sahip olmak ister. Bunu yapmak için `flat_map` metodundan hemen sonra bir `map` metodu kullanabiliriz.

In [8]:

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
for x, y in dataset:
    print(x.numpy(), y.numpy())
```

```
[0 1 2 3] [4]
[1 2 3 4] [5]
[2 3 4 5] [6]
[3 4 5 6] [7]
[4 5 6 7] [8]
[5 6 7 8] [9]
```

4 girdi ve 1 çıktı etiketi olarak tüm pencereleri parçaladık. Bununla beraber örneklerin sırasını rasgele karıştırmamız gereklidir bunu yaparsak makine öğrenmesi modelimiz sırayla ilgili yanlış bir öğrenme yapmamış olur. `shuffle` metodunu kullanarak pencereleri karıştıralım.

In [9]:

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
dataset = dataset.shuffle(buffer_size=10)
for x, y in dataset:
    print(x.numpy(), y.numpy())
```

```
[3 4 5 6] [7]
[2 3 4 5] [6]
[1 2 3 4] [5]
[4 5 6 7] [8]
[5 6 7 8] [9]
[0 1 2 3] [4]
```

Gruplama yapmak büyük seriler ile çalışırken faydalı olacaktır. Böylelikle TensorFlow mevcut girdi ile çalışırken

aynı zamanda sıradaki girdiyi yükleyebilir. Bu da işlemcimizin boş beklemeden kurtarmış olacaktır.

In [10]:

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x, y in dataset:
    print("x =", x.numpy())
    print("y =", y.numpy())
```

```
x = [[0 1 2 3]
 [2 3 4 5]]
y = [[4]
 [6]]
x = [[4 5 6 7]
 [1 2 3 4]]
y = [[8]
 [5]]
x = [[3 4 5 6]
 [5 6 7 8]]
y = [[7]
 [9]]
```

Tüm bu adımları bir araya getirelim ve sonraki colab dosyalarında kullanmak üzere bir fonksiyon tanımlayalım:

In [11]:

```
def window_dataset(series, window_size, batch_size=32,
                   shuffle_buffer=1000):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset
```