KOREATECH 한국기술교육대학교

Real-World Data to Tensors

Oct. 2023

http://link.koreatech.ac.kr

2D Images

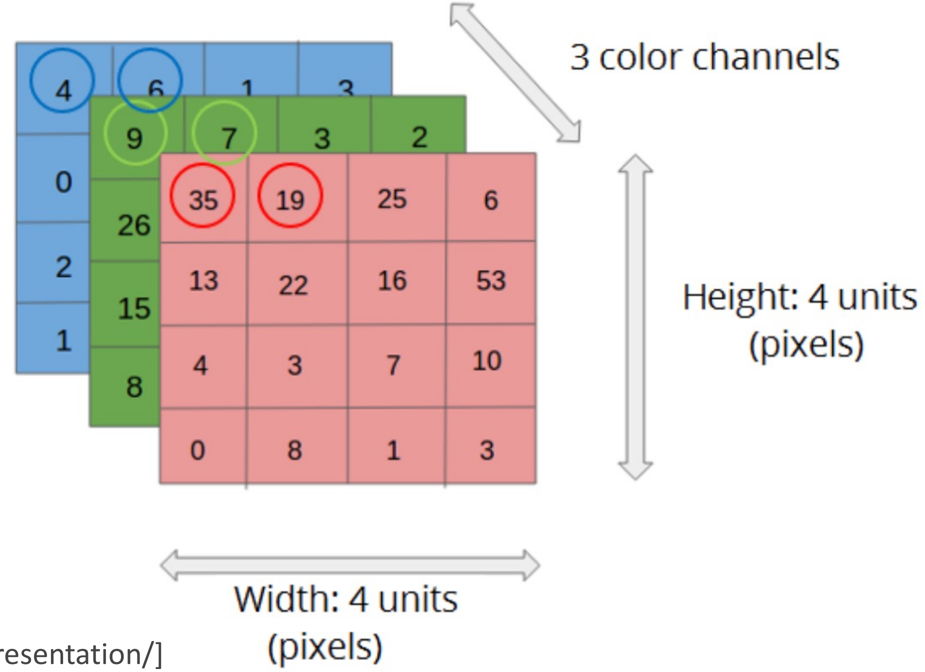
Working with Images

◈Image representation

– An image is represented as a collection of scalars arranged in a grid with a height and a width (in pixels)
  - individual pixels are often encoded using 8-bit integers, as in consumer cameras
  - In medical, scientific, and industrial applications, it is not unusual to find higher numerical precision, such as 12-bit or 16-bit

– Image color is defined by three numbers representing the intensity of red, green, and blue (RGB)
  - Three color channel (C)

– Dataset shape: ***N* × *C* × *H* × *W***
  - ***N*: Data size (number of images)**
  - ***C*: number of channels**
  - ***H*: height**
  - ***W*: width**



[Source: https://datahacker.rs/image-representation/]

Working with Images

◈Image representation example

– Loading an image file

```
# pip install imageio
import os
import imageio.v2 as imageio

img_arr = imageio.imread(
  os.path.join(os.path.pardir, os.path.pardir, "_00_data", "a_image-dog", "bobby.jpg")
)
print(type(img_arr)) # >>> <class 'numpy.ndarray'>
print(img_arr.shape) # >>> (720, 1280, 3)
print(img_arr.dtype) # >>> uint8
```

– Changing it into tensor and adjust the dimension layout

```
import torch

img = torch.from_numpy(img_arr) # H × W × C
out = img.permute(2, 0, 1)      # C × H × W
print(out.shape) # >>> (3, 720, 1280)
```

Working with Images

◈ Image representation example

– The images in a batch are stored along the first dimension

```
data_dir = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "b_image-cats")
filenames = [
    name for name in os.listdir(data_dir) if os.path.splitext(name)[-1] == '.png’
]

print(filenames)     # >>> ['cat1.png', 'cat2.png', 'cat3.png’]

from PIL import Image

for i, filename in enumerate(filenames):
    image = Image.open(os.path.join(data_dir, filename))
    image.show()
    img_arr = imageio.imread(os.path.join(data_dir, filename))
    print(img_arr.shape)   # (256, 256, 3)
    print(img_arr.dtype)   # unit8
```

Working with Images

◈Image representation example

– The images in a batch are stored along the first dimension

```
batch_size = 3
batch = torch.zeros(batch_size, 3, 256, 256, dtype=torch.uint8)

for i, filename in enumerate(filenames):
    img_arr = imageio.imread(os.path.join(data_dir, filename))
    img_t = torch.from_numpy(img_arr)
    img_t = img_t.permute(2, 0, 1)
    img_t = img_t[:3]  # Sometimes images also have an alpha channel indicating transparency
    batch[i] = img_t

print(batch.shape)     # >>> torch.Size([3, 3, 256, 256])
```

Dataset shape: ***N* × *C* × *H* × *W***

- ***N*: Data size (number of images)**
- ***C*: number of channels**
- ***H*: height**
- ***W*: width**

Working with Images

◈Image representation example

– Normalizing the data

```
# just divide the values of the pixels by 255
# 255 is the maximum representable number in 8-bit unsigned
batch = batch.float()
batch /= 255.0
print(batch.dtype)
print(batch.shape)

n_channels = batch.shape[1]

# the output has zero mean and unit standard deviation across each channel
for c in range(n_channels):
    mean = torch.mean(batch[:, c])
    std = torch.std(batch[:, c])
    batch[:, c] = (batch[:, c] - mean) / std
```

3D Images

Working with 3D Images

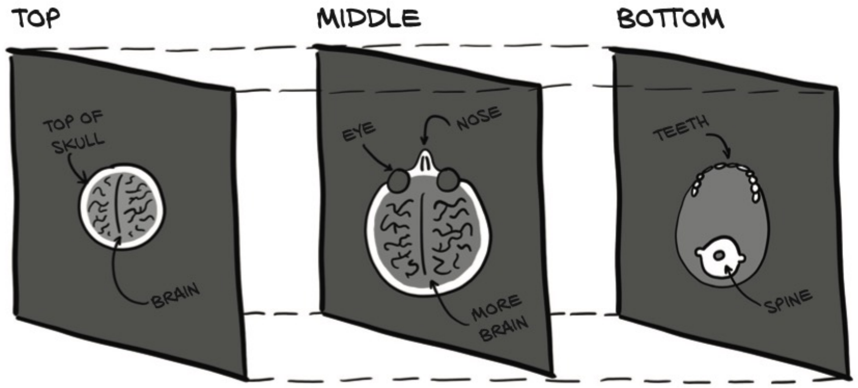
◈3D image representation

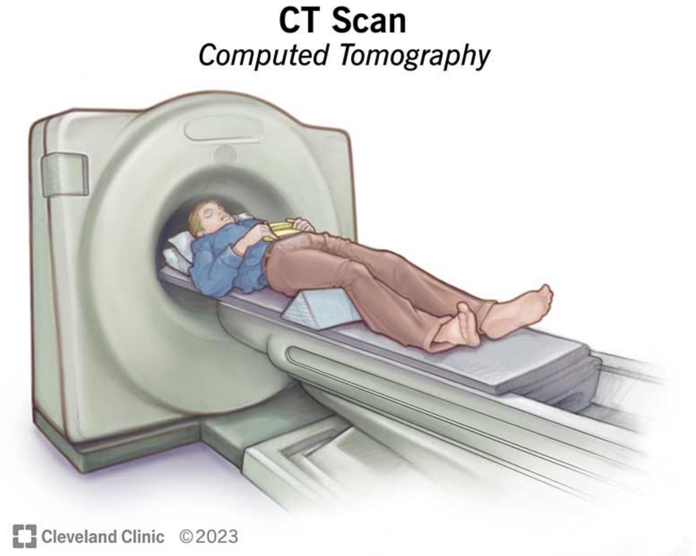
– Medical CT (computed tomography) scan images

- sequences of images stacked along the head-to-foot axis, each corresponding to a slice across the human body
- CTs have only a single intensity channel, similar to a grayscale image.
  - ➢ This means that the channel dimension can be removed out in native data formats
- By stacking individual "one-channel 2D slices" into a 4D tensor (volumetric data) having extra depth (D) dimension after the channel (C) dimension

– Dataset shape: ***N × C × D × H × W***

- ***N**: Data size (number of 3D images)*
- ***C**: number of channels*
  - ➢ *Many cases: 1*
- ***D**: depth*
- ***H**: height*
- ***W**: width*





Working with 3D Images

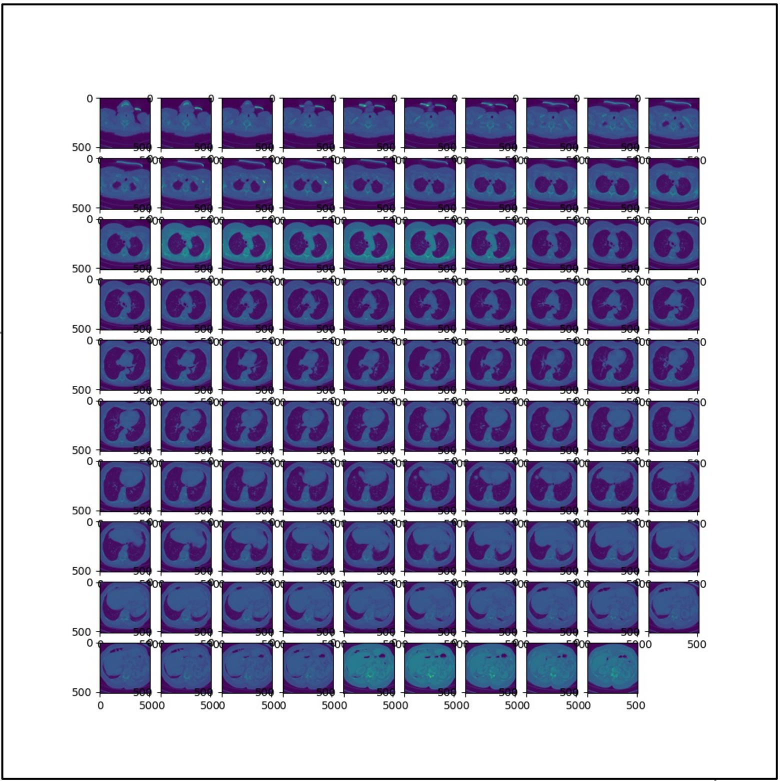
◈3D image representation example

– Loading a 3D image file

DICOM: Digital Imaging and Communications in Medicine

```
import os
import imageio.v2 as imageio
dir_path = os.path.join(
  os.path.pardir, os.path.pardir, "_00_data", "c_volumetric-dicom", "2-LUNG_3.0_B70f-04083”
)
vol_arr = imageio.volread(dir_path, format='DICOM’)
print(type(vol_arr))   # >>> <class 'imageio.core.util.Array’>: Numpy NDArray
print(vol_arr.shape)   # >>> (99, 512, 512)
print(vol_arr.dtype)   # >>> int16
print(vol_arr[0])
# >>>
# [[ -985  -990  -999 ... -1017 -1008  -971]
#  [-1016  -984  -963 ... -1000 -1009  -999]
#  ...
#  [ -920  -942  -944 ...  -893  -917  -955]
#  [ -871  -879  -905 ...  -895  -869  -867]
#  [ -876  -855  -873 ...  -933  -982  -936]]
```

Working with 3D Images



◈3D image representation example

– Visualize the 3D images

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10, 10))
for id in range(0, 99):
    fig.add_subplot(10, 10, id + 1)
    plt.imshow(vol_arr[id])
plt.show()
```

– Changing it into tensor and adjust the dimension layout

```
import torch
vol = torch.from_numpy(vol_arr).float()
vol = torch.unsqueeze(vol, 0)   # >>> channel
vol = torch.unsqueeze(vol, 1)   # >>> data size

# N × C × D × H × W
print(vol.shape) # >>> torch.Size([1, 1, 99, 512, 512])
```

Dataset shape: ***N* × *C* × *D* × *H* × *W***

- ***N*: Data size (number of 3D images)**
- ***C*: number of channels**
- ***D*: depth**
- ***H*: height**
- ***W*: width**

Working with 3D Images

◈3D image representation example

– Normalizing the data

```
mean = torch.mean(vol, dim=(3, 4), keepdim=True)
print(mean.shape)   # >>> torch.Size([1, 1, 99, 1, 1])
std = torch.std(vol, dim=(3, 4), keepdim=True)
print(std.shape)    # >>> torch.Size([1, 1, 99, 1, 1])
vol = (vol - mean) / std
print(vol.shape)    # >>> torch.Size([1, 1, 99, 512, 512])

print(vol[0, 0, 0])
# >>>
# tensor([[-1.0002, -1.0102, -1.0283,  ..., -1.0645, -1.0464, -0.9720],
#         [-1.0625, -0.9982, -0.9560,  ..., -1.0303, -1.0484, -1.0283],
#         [-1.0785, -1.0464, -1.0223,  ..., -0.9881, -1.0725, -1.0042],
#         ...,
#         [-0.8696, -0.9138, -0.9178,  ..., -0.8153, -0.8636, -0.9399],
#         [-0.7711, -0.7872, -0.8394,  ..., -0.8194, -0.7671, -0.7631],
#         [-0.7812, -0.7390, -0.7752,  ..., -0.8957, -0.9941, -0.9017]])
```

Tabular Data

Working with Tabular Data

◈Tabular data representation

– It is a table containing <u>one row per sample (or record)</u>, where <u>columns contain feature information</u> about our sample
  - The simplest form of data is sitting in a spreadsheet, CSV file, or database
– Different columns do not have the same type, but PyTorch tensors should be homogeneous
  - Information used by PyTorch neural network is typically encoded as <u>torch.float32</u>
– <u>Data munging & cleaning</u> are needed

데이터 먼징(Data Munging) 혹은 **데이터 랭글링**(Data Wrangling)이라고 불리는 이것은 원자료(raw data)를 보다 쉽게 접근하고 분석할 수 있도록 데이터를 정리하고 통합하는 과정

– Dataset shape: ***N* × *F***
  - ***N***: *Data size (number of samples)*
  - ***F***: *number of features*

Attributes

Target attribute

Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no
Claudio	$115,000	40	no	no
Robert	$29,000	23	yes	yes
Dora	$72,000	31	no	no

This is one row (example).
Feature vector is: **<Claudio,115000,40,no>**
Class label (value of Target attribute) is **no**

Working with Tabular Data

◈Tabular data representation example - I

– Loading a tabular data file

```
import csv, os
import numpy as np
wine_path = os.path.join(
  os.path.pardir, os.path.pardir, "_00_data", "d_tabular-wine", "winequality-white.csv"
)
wineq_numpy = np.loadtxt(wine_path, dtype=np.float32, delimiter=";", skiprows=1)
print(wineq_numpy.dtype)  # >>> float32
print(wineq_numpy.shape)  # >>> (4898, 12)
print(wineq_numpy)
# >>>
# [[ 7.    0.27  0.36 ...  0.45  8.8   6.  ]
#  [ 6.3   0.3   0.34 ...  0.49  9.5   6.  ]
#  ...
#  [ 5.5   0.29  0.3  ...  0.38 12.8   7.  ]
#  [ 6.    0.21  0.38 ...  0.32 11.8   6.  ]]
col_list = next(csv.reader(open(wine_path), delimiter=';'))
print(col_list)           # >>> ['fixed acidity', 'volatile acidity', ... , 'alcohol', 'quality']
```

Dataset shape: $N \times F$

- **N**: *Data size (number of samples)*
- **F**: *number of features*

Working with Tabular Data

◈Tabular data representation example - I

– Changing it into tensor and separate the target from the data

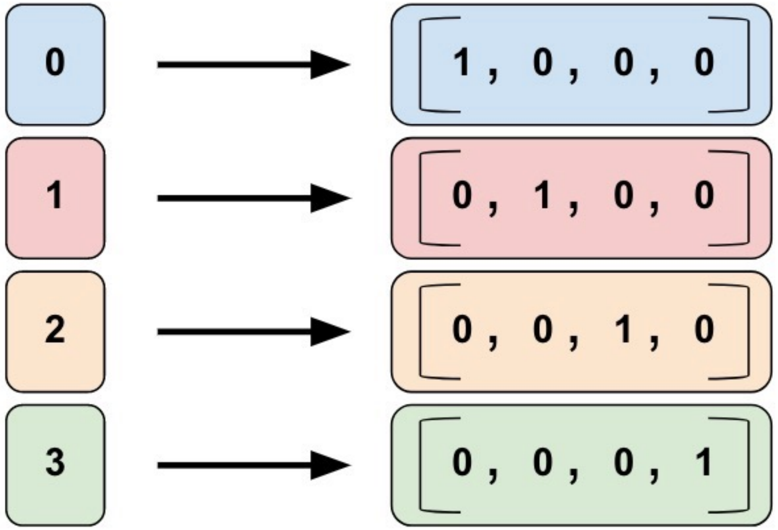
```
wineq = torch.from_numpy(wineq_numpy)
print(wineq.dtype)  # >>> torch.float32
print(wineq.shape)  # >>> torch.Size([4898, 12])

# Selects all rows and all columns except the last
data = wineq[:, :-1]
print(data.dtype)   # >>> torch.float32
print(data.shape)   # >>> torch.Size([4898, 11])

# Selects all rows and the last column
target = wineq[:, -1]
print(target.dtype) # >>> torch.float32
print(target.shape) # >>> torch.Size([4898])

# treat labels as an integer
target = target.long()
print(target.dtype) # >>> torch.int64
print(target.shape) # >>> torch.Size([4898])
Print(target)       # >>> tensor([6, 6, 6,  ..., 6, 7, 6])
```

Working with Tabular Data

◈Tabular data representation example - I

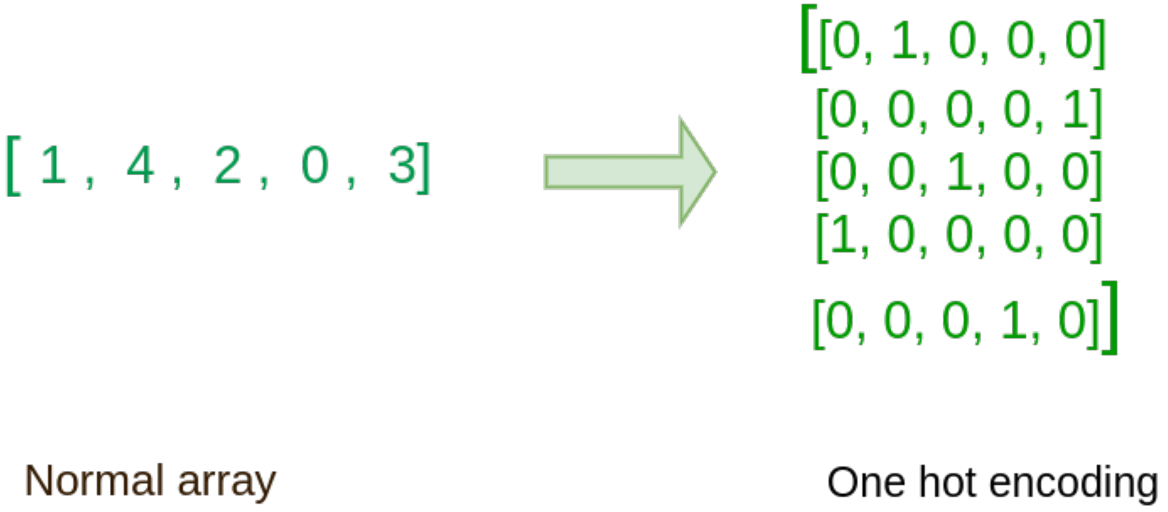
– One-hot encoding

악기	악기별 인덱스	원-핫 벡터
피아노	0	[1,0,0,0,0,0]
바이올린	1	[0,1,0,0,0,0]
비올라	2	[0,0,1,0,0,0]
첼로	3	[0,0,0,1,0,0]
드럼	4	[0,0,0,0,1,0]
트럼펫	5	[0,0,0,0,0,1]



상품 분류	가격
TV	1200000
냉장고	3500000
컴퓨터	700000
컴퓨터	1200000
냉장고	2300000
에어컨	1500000
TV	300000

TV	냉장고	컴퓨터	에어컨	가격
1	0	0	0	1200000
0	1	0	0	3500000
0	0	1	0	700000
0	0	1	0	1200000
0	1	0	0	2300000
0	0	0	1	1500000
1	0	0	0	300000



Working with Tabular Data

◈Tabular data representation example - I

– Changing the target by one-hot encoding

```
eye_matrix = torch.eye(10)
# We use the 'target' tensor as indices to extract the corresponding rows from the identity matrix
# It can generate the one-hot vectors for each element in the 'target' tensor
onehot_target = eye_matrix[target]

print(onehot_target.shape) # >>> torch.Size([4898, 10])
print(onehot_target[0])    # >>> tensor([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.])
print(onehot_target[1])    # >>> tensor([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.])
print(onehot_target[-2])   # >>> tensor([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.])
print(onehot_target)
# >>>
# tensor([[0., 0., 0.,  ..., 0., 0., 0.],
#         [0., 0., 0.,  ..., 0., 0., 0.],
#         [0., 0., 0.,  ..., 0., 0., 0.],
#         ...,
#         [0., 0., 0.,  ..., 0., 0., 0.],
#         [0., 0., 0.,  ..., 1., 0., 0.],
#         [0., 0., 0.,  ..., 0., 0., 0.]])
```

Working with Tabular Data

◈Tabular data representation example - I

– Normalizing the data

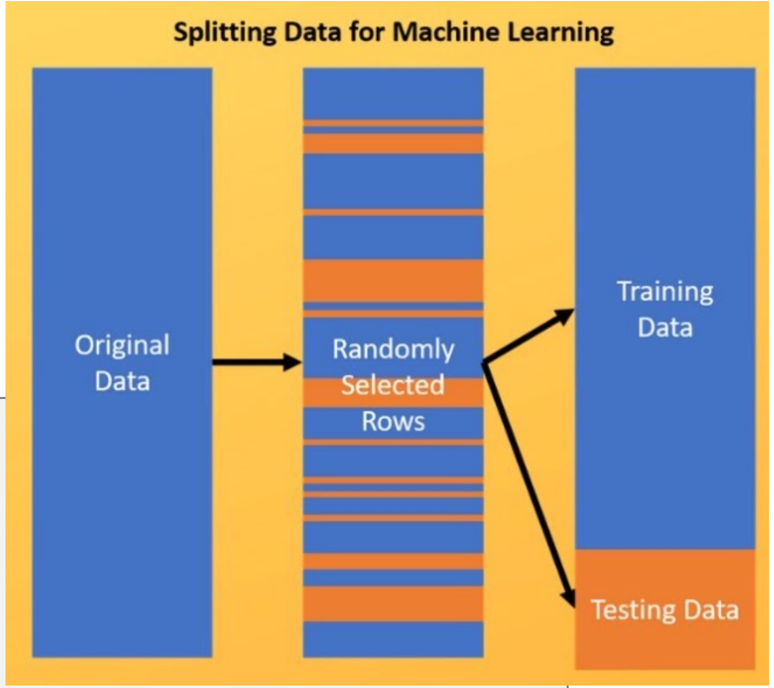
```
data_mean = torch.mean(data, dim=0)      # >>> torch.Size([4898, 11]) → torch.Size([11])
data_var = torch.var(data, dim=0)        # >>> torch.Size([4898, 11]) → torch.Size([11])
data = (data - data_mean) / torch.sqrt(data_var)

print(data)
# >>> tensor([
#         [ 1.7208e-01, -8.1761e-02,  2.1326e-01,  ..., -1.2468e+00, -3.4915e-01, -1.3930e+00],
#         [-6.5743e-01,  2.1587e-01,  4.7996e-02,  ...,  7.3995e-01, 1.3422e-03, -8.2419e-01],
#         ...,
#         [-1.6054e+00,  1.1666e-01, -2.8253e-01,  ...,  1.0049e+00, -9.6251e-01,  1.8574e+00],
#         [-1.0129e+00, -6.7703e-01,  3.7852e-01,  ...,  4.7505e-01, -1.4882e+00,  1.0448e+00]
#     ])
```

link_dl/_01_code/_03_real_world_data_to_tensors/c_tabular_wine_data.py

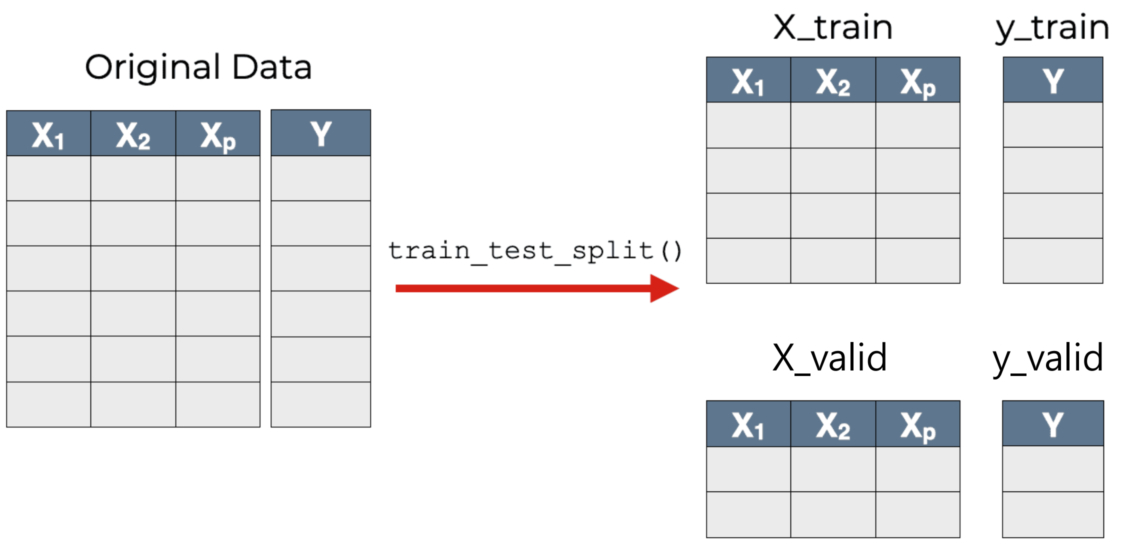
Working with Tabular Data

◈Tabular data representation example - I

– train_test_split




```
from sklearn.model_selection import train_test_split

X_train, X_valid, y_train, y_valid = train_test_split(
  data, onehot_target, test_size=0.2
)

print(X_train.shape)  # >>> torch.Size([3918, 11])
print(y_train.shape)  # >>> torch.Size([3918, 10])

print(X_valid.shape)   # >>> torch.Size([980, 11])
print(y_valid.shape)   # >>> torch.Size([980, 10])
```



Working with Tabular Data

◈Tabular data representation example - II

– The California Housing Dataset (https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset)

- Number of Instances: 20640
- Attributes or features (8)
  - MedInc: median income in block
  - HouseAge: median house age in block
  - AveRooms: average number of rooms
  - AveBedrms: average number of bedrooms
  - Population: block population
  - AveOccup: average number of household members
  - Latitude: house block latitude (위도)
  - Longitude: house block longitude (경도)
- Target
  - Median house value (for a California district block) - it is expressed in $100,000

a group of people residing within a home

Working with Tabular Data

◈Tabular data representation example - II

– Using prepared data (The California Housing Dataset)

```
import torch
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()
print(housing.keys())
# >>> dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR’])

print(type(housing.data))           # >>> <class 'numpy.ndarray'>
print(housing.data.dtype)           # >>> float64
print(housing.data.shape)           # >>> (20640, 8)
print(housing.feature_names)
# >>> ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude’]

print(housing.target.shape)         # >>> (20640,)
print(housing.target_names)         # >>> ['MedHouseVal']
```

Dataset shape: *N × F*

- ***N*: *Data size (number of samples)***
- ***F*: *number of features***

Working with Tabular Data

◈Tabular data representation example - II

– Normalizing the data

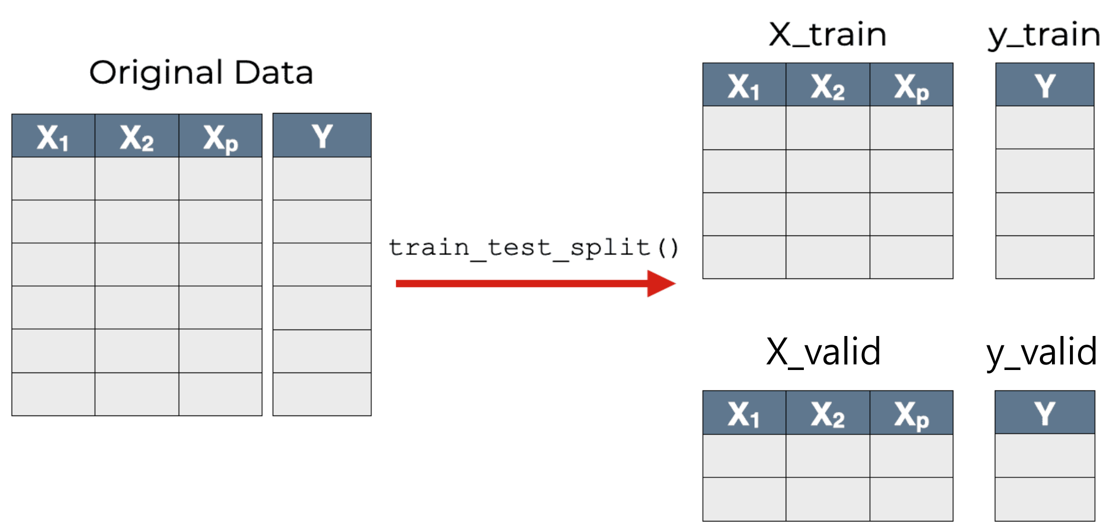
```
import numpy as np

print(housing.data.min(), housing.data.max())
# >>> -124.35 35682.0

data_mean = np.mean(housing.data, axis=0)
data_var = np.var(housing.data, axis=0)
data = (housing.data - data_mean) / np.sqrt(data_var)
target = housing.target

print(data.min(), data.max())
# >>> -2.385992341673387 7 119.41910318829312
```

Working with Tabular Data

◈Tabular data representation example - II

– Separate the target from the data and change them into tensors

```
from sklearn.model_selection import train_test_split

X_train, X_valid, y_train, y_valid = train_test_split(data, target, test_size=0.2)

X_train = torch.from_numpy(X_train)
X_valid = torch.from_numpy(X_valid)
y_train = torch.from_numpy(y_train)
y_valid = torch.from_numpy(y_valid)

print(X_train.shape)  # >>> torch.Size([14448, 8])
print(y_train.shape)  # >>> torch.Size([14448])

print(X_valid.shape)   # >>> torch.Size([6192, 8])
print(y_valid.shape)   # >>> torch.Size([6192])
```



Time Series Data

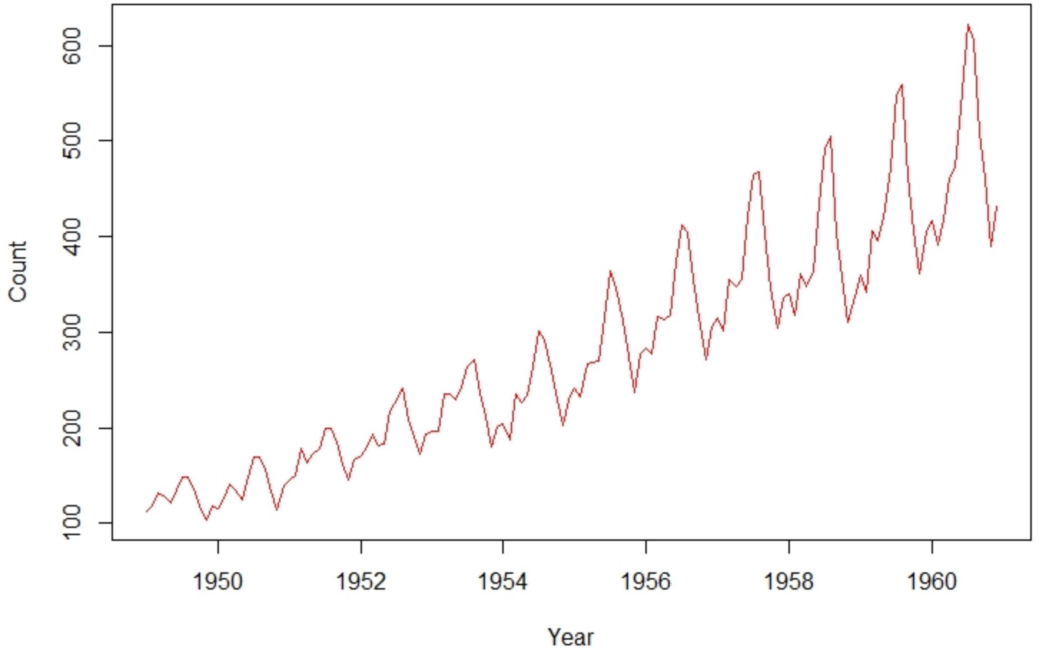
Working with Time Series Data

◈Time series data representation

- Time series data
  - A sequence of data points collected over time intervals
  - The time intervals
    1) equally spaced as in the case of periodic metrics, or
    2) unequally spaced as in the case of events

[Important feature with time series]
- Successive observations are usually NOT independent
- future values can be predicted from past observations

- Dataset shape: ***N × L × F***
  - ***N**: Data size (days)*
  - ***L**: Length (hours)*
  - ***F**: features*

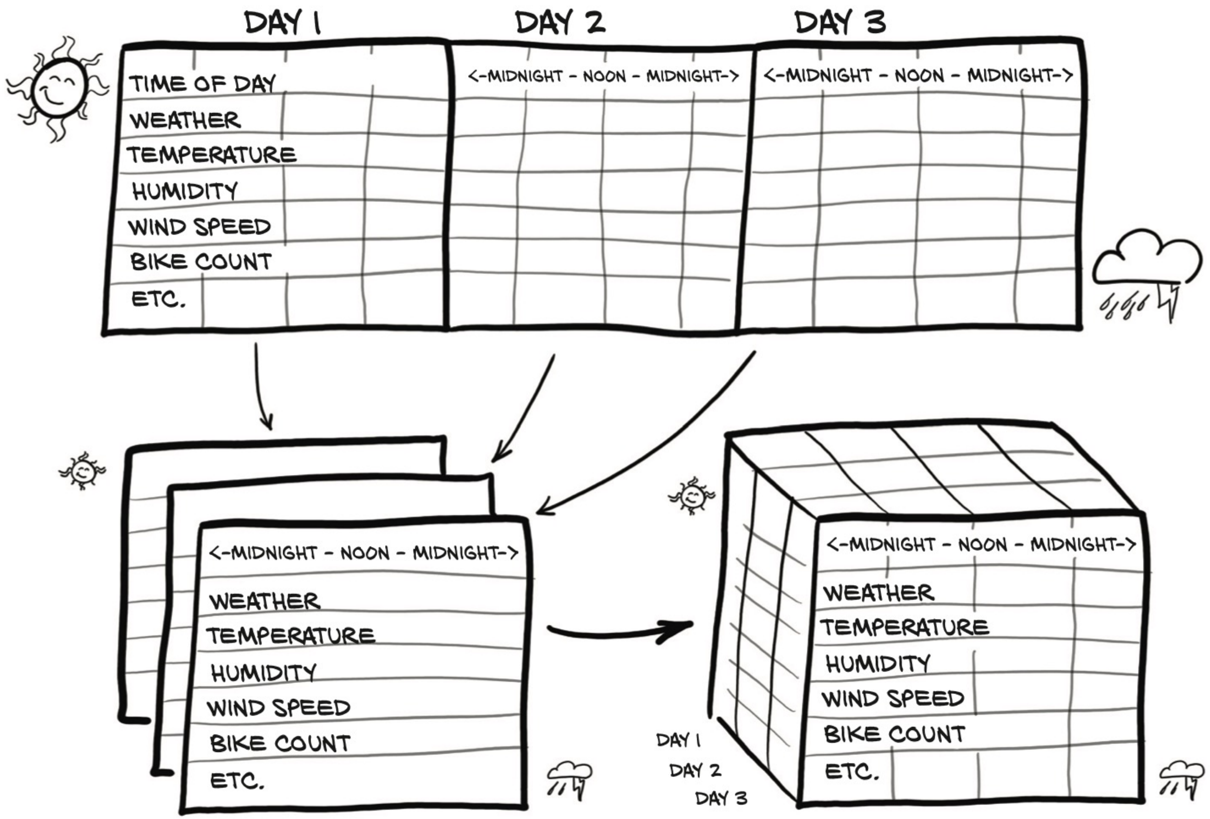


- Examples of time series analysis
  - Weather prediction, earthquake prediction, and energy consumption prediction

Working with Time Series Data

◈Time series data representation

- Bike-sharing data (Washington, D.C., 2011 – 2012) - http://mng.bz/jgOx
- Data munging goal
  - To take a flat 2D dataset and transform it into a 3D one (***N* × *L* × *F***)



Working with Time Series Data

- Time series data representation
  - Bike-sharing data (Washington, D.C., 2011 - 2012) - http://mng.bz/jgOx
  - Each row is a separate hour of data
    - For every hour (row), the dataset reports the following features
    - Num of features: 17
      - Numerical data: 16
      - Nominal data: 1
        - » Whether situation
          - 1: clear
          - 2: mist
          - 3: light rain/snow
          - 4: heavy rain/snow

- Index of record: instant
- Day of month: day
- Season: season (1: spring, 2: summer, 3: fall, 4: winter)
- Year: yr (0: 2011, 1: 2012)
- Month: mnth (1 to 12)
- Hour: hr (0 to 23)
- Holiday status: holiday
- Day of the week: weekday
- Working day status: workingday
- Weather situation: weathersit (1: clear, 2:mist, 3: light rain/snow, 4: heavy rain/snow)
- Temperature in °C: temp
- Perceived temperature in °C: atemp
- Humidity: hum
- Wind speed: windspeed
- Number of casual users: casual
- Number of registered users: registered
- Count of rental bikes: cnt will be used as target

Working with Time Series Data

◈Time series data representation

– Loading the bike-sharing data and change it into tensor

```
import os
import numpy as np
import torch
torch.set_printoptions(edgeitems=2, threshold=50, linewidth=75)

bikes_path = os.path.join(
  os.path.pardir, os.path.pardir, "_00_data", "e_time-series-bike-sharing-dataset",
  "hour-fixed.csv”
)

bikes_numpy = np.loadtxt(
    fname=bikes_path, dtype=np.float32, delimiter=",", skiprows=1,
    converters={
        1: lambda x: float(x[8:10])
    } # 1: Column Index, 2011-01-07 --> 07 --> 7.0 (from date to day)
)
bikes = torch.from_numpy(bikes_numpy)
print(bikes.shape) # >>> torch.Size([17520, 17])
```

Working with Time Series Data

◈Time series data representation

– 1) Convert it into daily data and 2) divide them into data and target

```
daily_bikes = bikes.view(-1, 24, bikes.shape[1])
print(daily_bikes.shape)
# >>> torch.Size([730, 24, 17]) ← 730 days

daily_bikes_data = daily_bikes[:, :, :-1]
daily_bikes_target = daily_bikes[:, :, -1].unsqueeze(dim=-1)

print(daily_bikes_data.shape)
# >>> torch.Size([730, 24, 16])

print(daily_bikes_target.shape)
# >>> torch.Size([730, 24, 1])
```

Working with Time Series Data

◈Time series data representation

– (Temporarlily) Change nominal data by one-hot encoding

```
first_day_data = daily_bikes_data[0]
print(first_day_data.shape)    # >>> torch.Size([24, 16])

# Whether situation: 1: clear, 2:mist, 3: light rain/snow, 4: heavy rain/snow
print(first_day_data[:, 9].long())
# >>> tensor([1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2])
eye_matrix = torch.eye(4)
weather_onehot = eye_matrix[first_day_data[:, 9].long() - 1]
print(weather_onehot.shape)    # >>> torch.Size([24, 4])

first_day_data_torch = torch.cat(
    tensors=(first_day_data, weather_onehot), dim=1
)
print(first_day_data_torch.shape)   # >>> torch.Size([24, 20])
```

```
tensor([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])
```

Working with Time Series Data

◈Time series data representation

– For all data, change nominal data by one-hot encoding

```
day_data_torch_list = []
for daily_idx in range(daily_bikes_data.shape[0]):        # range(730)
    day = daily_bikes_data[daily_idx]                      # day.shape: [24, 17]
    weather_onehot = eye_matrix[day[:, 9].long() - 1]
    day_data_torch = torch.cat(tensors=(day, weather_onehot), dim=1) # day_torch.shape: [24, 20]
    day_data_torch_list.append(day_data_torch)

print(len(day_data_torch_list))      # >>> 730

daily_bikes_data = torch.stack(day_data_torch_list, dim=0)

print(daily_bikes_data.shape)  # >>> torch.Size([730, 24, 20])
```

Working with Time Series Data

◈Time series data representation

– 1) Remove the original nominal feature, and 2) normalize the temperature feature

```
print(daily_bikes_data[:, :, :9].shape, daily_bikes_data[:, :, 10:].shape)
# >>> torch.Size([730, 24, 9]) torch.Size([730, 24, 10])

daily_bikes_data = torch.cat(
    [daily_bikes_data[:, :, :9], daily_bikes_data[:, :, 10:]], dim=2
)
print(daily_bikes_data.shape)    # >>> torch.Size([730, 24, 19])

temperatures = daily_bikes_data[:, :, 9]
daily_bikes_data[:, :, 9] = /
    (daily_bikes_data[:, :, 9] - torch.mean(temperatures)) / torch.std(temperatures)

daily_bikes_data = daily_bikes_data.transpose(1, 2)

print(daily_bikes_data.shape)  # >>> torch.Size([730, 24, 19])
```

Dataset shape: $N \times L \times F$

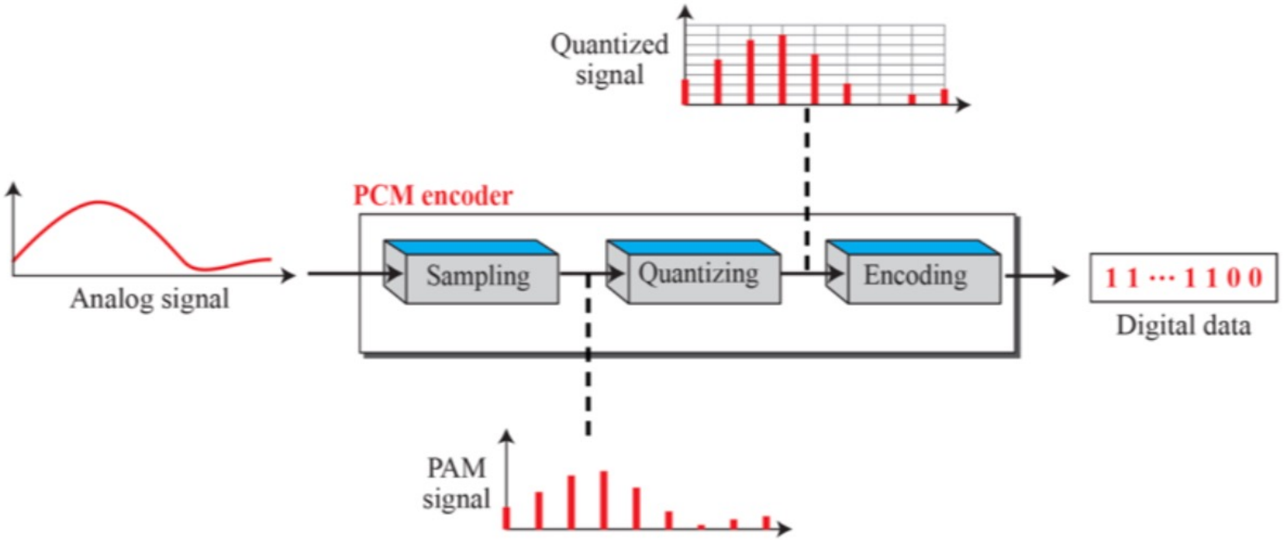
- *N: Data size (days)*
- *L: Length (hours)*
- *F: features*

Audio Data

Working with Audio Data

◈Audio data representation

– Pulse Code Modulation (PCM)
  - A continuous sound signal is both sampled at each time point and quantized in amplitude
  - It is represented as a vector of numbers



– The highest possible quality → Sampling rate: over 40,000 times per second (40,000 Hz)
  - Audio CD sampling frequency: 44,100 Hz
  - 1 Hour and 2 channels data size: 2 * 16 * 44100 * 3600 = 5080320000 bit = 605.6 MB (if stored without compression)

– Dataset shape: ***N* × *C* × *L***
  - ***N**: Data size (number of audios)*
  - ***C**: Channel*
  - ***L**: Length*

Working with Audio Data

◈Audio data representation

- There are a plethora of audio formats, WAV, AIFF, MP3, AAC being the most popular, where raw audio signals are typically encoded in compressed form

- ESC-50 sound repository
  - https://github.com/karoldvl/ESC-50
  - We can download audio files in the `audio` directory.

- `scipy.io.wavfile.read()`
  - In order to load the sound we resort to SciPy, specifically, which has the nice property to return data as a NumPy array:

Working with Audio Data

◈Audio data representation example

```
import torch
import os
import scipy.io.wavfile as wavfile

audio_1_path = os.path.join(
  os.path.pardir, os.path.pardir, "_00_data", "f_audio-chirp", "1-100038-A-14.wav”
)
audio_2_path = os.path.join(
  os.path.pardir, os.path.pardir, "_00_data", "f_audio-chirp", "1-100210-A-36.wav”
)

# mono channel (1 channel) sound
freq_1, waveform_arr_1 = wavfile.read(audio_path_1)
print(freq_1)                     # >>> 44100
print(type(waveform_arr_1))       # >>> <class 'numpy.ndarray'>
print(len(waveform_arr_1))        # >>> 220500 ← 44100 * 5 sec.
print(waveform_arr_1)             # >>> [ -388 -3387 -4634 ...  2289  1327    90]
```

Working with Audio Data

◈Audio data representation example

```
freq_2, waveform_arr_2 = wavfile.read(audio_2_path)

waveform = torch.empty(2, 1, 220_500)
waveform[0, 0] = torch.from_numpy(waveform_arr_1).float()
waveform[1, 0] = torch.from_numpy(waveform_arr_2).float()

print(waveform.shape)              # >>> torch.Size([2, 1, 220500])
```

Dataset shape: ***N* × *C* × *L***

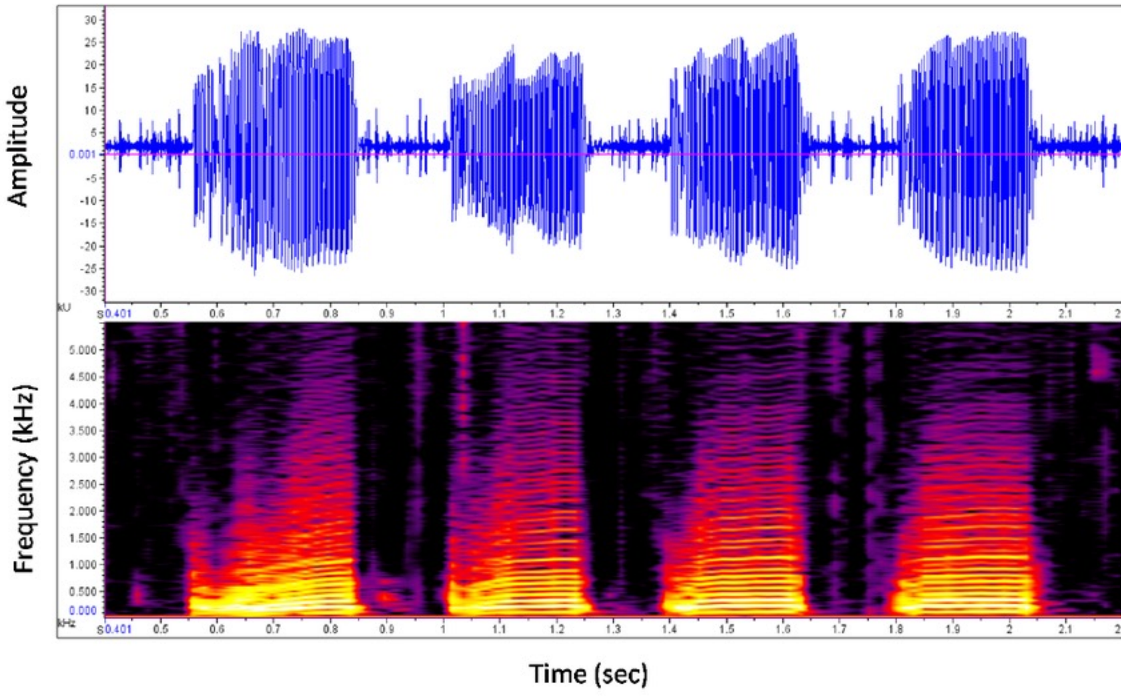
- *N: Data size (number of audios)*
- *C: Channel*
- *L: Length*

Working with Audio Data

◈Audio data representation - II

– Spectrogram

- a representation of the intensity at each frequency at each point in time
  - ➢ 오디오 신호나 소리의 주파수와 시간에 대한 시각적 표현
  - ➢ Fourier transform is usually used



– Dataset shape: $N \times C \times F \times T$

- *N: Data size (number of audios)*
- *C: Channel*
- *F: Frequency*
- *T: Time*

Working with Audio Data

◈Audio data representation - II example

```
from scipy import signal

_, _, sp_arr_1 = signal.spectrogram(waveform_arr_1, freq_1)
_, _, sp_arr_2 = signal.spectrogram(waveform_arr_2, freq_2)

sp_1 = torch.from_numpy(sp_arr_1)
sp_2 = torch.from_numpy(sp_arr_2)
print(sp_1.shape)          # >>> torch.Size([129, 984]) ← Frequency & Time
print(sp_2.shape)          # >>> torch.Size([129, 984]) ← Frequency & Time

sp_left_t = torch.from_numpy(sp_arr_1)
sp_right_t = torch.from_numpy(sp_arr_2)
print(sp_left_t.shape)     # >>> torch.Size([129, 984])
print(sp_right_t.shape)    # >>> torch.Size([129, 984])

sp_t = torch.stack((sp_left_t, sp_right_t), dim=0).unsqueeze(dim=0)
print(sp_t.shape)          # >>> torch.Size([1, 2, 129, 984])
```

Dataset shape: $N \times C \times F \times T$

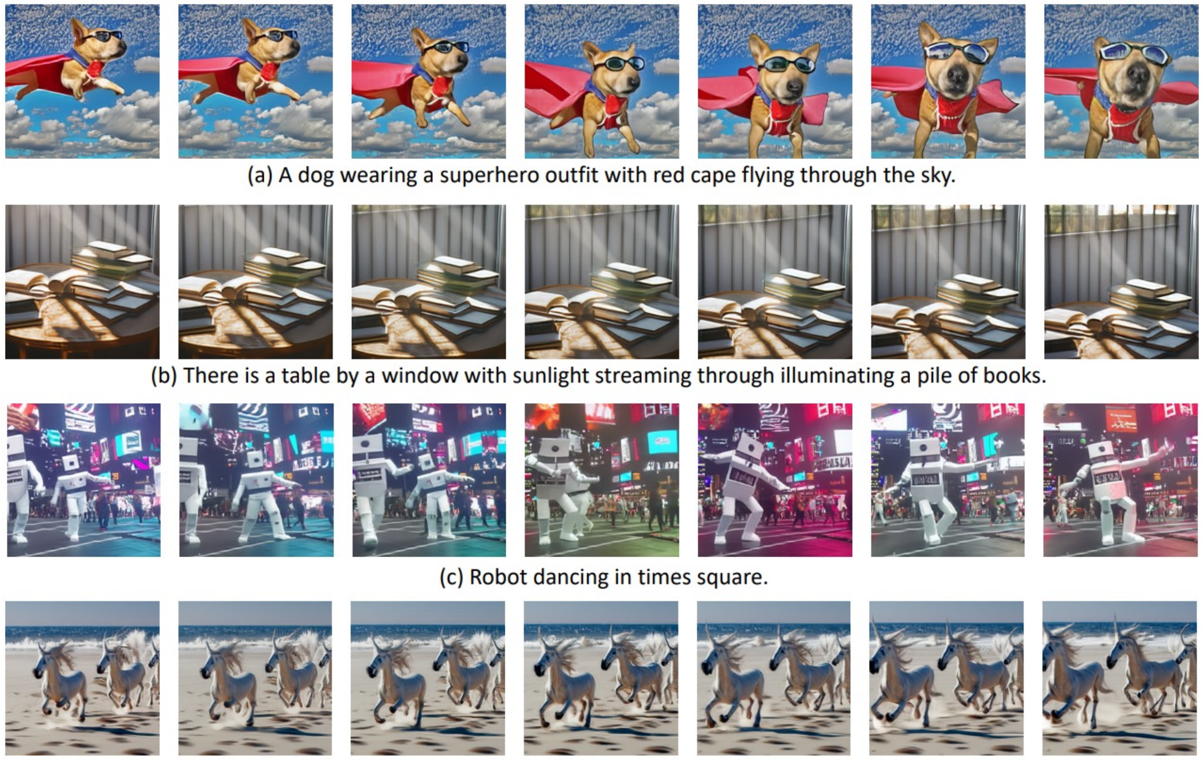
- *N: Data size (number of audios)*
- *C: Channel*
- *F: Frequency*
- *T: Time*

Video Data

Working with Video Data

◈Video data representation

- Video data can be seen as equivalent to volumetric data, with \`depth\` replaced by the \`time\` dimension.
- Dataset shape: *N × C × T × H × W*
  - *N: Data size (number of videos)*
  - *C: Channel*
  - *T: Time (or frames)*
  - *H: height*
  - *W: width*



(a) A dog wearing a superhero outfit with red cape flying through the sky.
(b) There is a table by a window with sunlight streaming through illuminating a pile of books.
(c) Robot dancing in times square.
(d) Unicorns running along a beach, highly detailed.

[Source: https://arxiv.org/abs/2209.14792]

- Codec module install needed usually
  - pip install imageio[ffmpeg]
  - ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video

Working with Video Data

◈ Time series data representation

– get meta information by using imageio.get_reader()

```
import numpy as np
import torch
import os
import imageio

video_path = os.path.join(
  os.path.pardir, os.path.pardir, "_00_data", "g_video-cockatoo", "cockatoo.mp4”
)
reader = imageio.get_reader(video_path)
print(type(reader)) # >>> <class 'imageio.plugins.ffmpeg.FfmpegFormat.Reader'>
meta = reader.get_meta_data()
print(meta)
# >>>
# {'plugin': 'ffmpeg', 'nframes': inf,
#  'ffmpeg_version': '6.0 built with Apple clang version 14.0.3 (clang-1403.0.22.14.1)’,
#  'codec': 'h264', 'pix_fmt': 'yuv420p(tv, bt709, progressive)', 'audio_codec': 'aac’,
#  'fps': 29.53, 'source_size': (480, 360), 'size': (480, 360), 'rotate': 0,
#  'duration': 17.93}
```

Working with Video Data

◈Time series data representation

– Construct video data from 'imageio.plugins.ffmpeg.FfmpegFormat.Reader' object

```
for i, frame in enumerate(reader):
  frame = torch.from_numpy(frame).float()  # frame.shape: [360, 480, 3]
  print(i, frame.shape)                    # i, torch.Size([360, 480, 3])

n_channels = 3
n_frames = 529
video = torch.empty(1, n_frames, n_channels, *meta['size'])  # (1, 529, 3, 480, 360)
print(video.shape)

for i, frame in enumerate(reader):
  frame = torch.from_numpy(frame).float()       # frame.shape: [360, 480, 3]
  frame = torch.permute(frame, dims=(2, 1, 0))  # frame.shape: [3, 480, 360]
  video[0, i] = frame

video = video.permute(dims=(0, 2, 1, 3, 4))
print(video.shape)
```

Dataset & DataLoader

Dataset & DataLoader

- torch.utils.data.Dataset
  - stores data samples and expected target values (labels)
  - return one sample at a time
- torch.utils.data.Dataloader
  - groups data in batches, iterate them, and enables multiprocessing

```
● dataset = MyDataset(file)
● dataloader = DataLoader(dataset, batch_size, shuffle=True)
```

Training: True
Testing: False

Dataset & DataLoader

◈ torch.utils.data.Dataset

– stores data samples and expected target values (labels)

– return one sample at a time

```
from torch.utils.data import Dataset

class MyDataset(Dataset):
    def __init__(self, ...):
        self.data = ...
        self.target = ...

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return {
            'input' : self.data[idx],
            'target' : self.target[idx]
        }
```

- Read data & target
- Preprocess them

- Return the size of the dataset

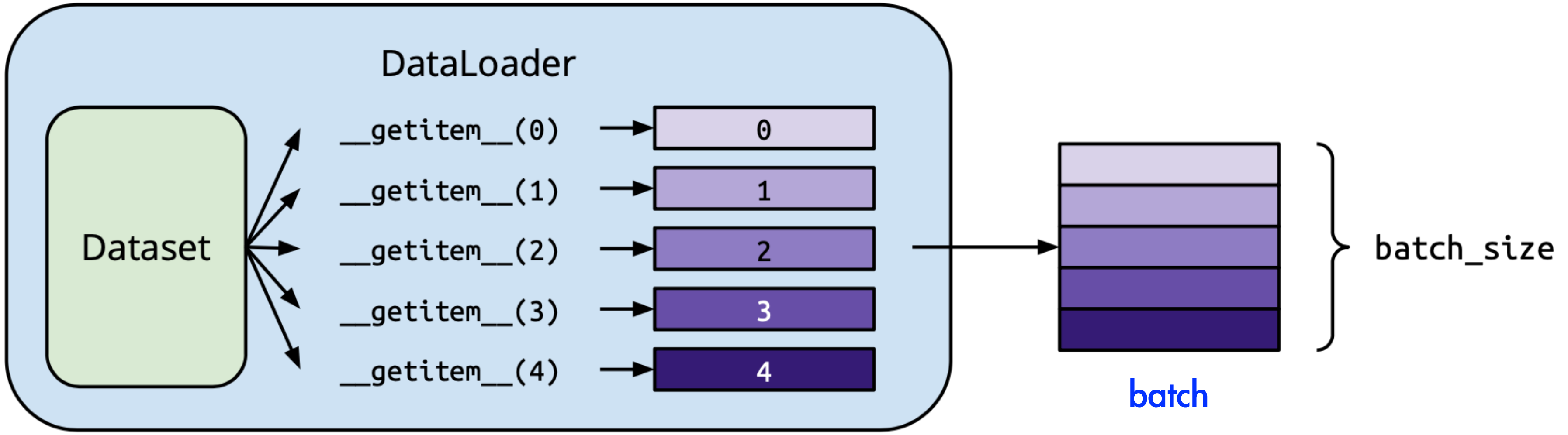
- Return one sample at a time

Dataset & DataLoader

◆torch.utils.data.Dataloader

– groups data in batches, iterate them, and enables multiprocessing

```
from torch.utils.data import DataLoader

my_mydataset = MyDataset()
data_loader = DataLoader(dataset=my_dataset, batch_size=5)

for idx, batch in enumerate(data_loader):
    # do something with idx & batch
```



Dataset & DataLoader

◈ torch.utils.data.Dataloader

- While training a deep learning model, we typically want to pass samples in 'minibatches’
- suffle=True (default: False)
  - We can reshuffle the data at every epoch to reduce model overfitting
- num_workers=N (default: 0)
  - We can use Python's multi-processing to speed up data retrieval
    - N: the number of subprocesses to create for data loading
  - If num_workers=0, it implies that the data will be loaded in the main process only

```
from torch.utils.data import DataLoader

my_mydataset = MyDataset()
data_loader = DataLoader(dataset=my_dataset, batch_size=5, suffle=True, num_workers=4)

for idx, batch in enumerate(data_loader):
    # do something with idx & batch
```

Dataset & DataLoader

◆torch.utils.data.Dataloader

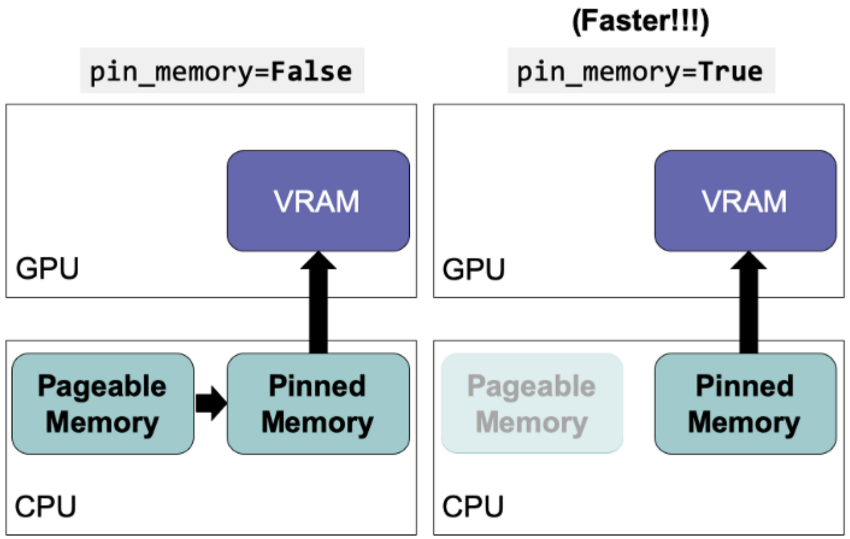
– drop_last=True (default: False)

- If it is set to True, the last incomplete batch of data is dropped in case the dataset size is not divisible by the batch size
- The reason why one would want to drop the last batch is that the gradient based on very small batches could lead to noisy updates to the parameters which could slow down the training of the network

```
from torch.utils.data import DataLoader

my_mydataset = MyDataset()
data_loader = DataLoader(dataset=my_dataset, batch_size=5, suffle=True, num_workers=4, drop_last=True)

for idx, batch in enumerate(data_loader):
    # do something with idx & batch
```


[source] https://www.scaler.com/topics/pytorch/dataloader-in-pytorch/

Dataset & DataLoader



◈torch.utils.data.Dataloader

- pin_memory=True (default: False)
  - GPUs cannot access the data directly from the pageable memory of the CPU
  - 'pin_memory=True' directly allocates the staging memory (CUDA-pinned memory) for the data
  - It saves the time taken in transferring the data from the CPU's pageable memory to staging memory (i.e., pinned memory a.k.a., page-locked memory, i. e., memory that is guaranteed to be resident in physical memory at all times)

```
from torch.utils.data import DataLoader

my_mydataset = MyDataset()
data_loader = DataLoader(
    dataset=my_dataset, batch_size=5, suffle=True, num_workers=4, drop_last=True, pin_memory=True
)
for idx, batch in enumerate(data_loader):
    # do something with idx & batch
```

Dataset & DataLoader

◆torch.utils.data.random_split

- Randomly split a dataset into non-overlapping new datasets of given lengths
- If a list of fractions that sum up to 1 is given, the lengths will be computed automatically
- Optionally fix the generator for reproducible results, e.g.:

```
import torch
from torch.utils.data import random_split

generator1 = torch.Generator().manual_seed(42)
generator2 = torch.Generator().manual_seed(42)

train_data, test_data = random_split(range(10), [7, 3], generator=generator1)
print(list(train_data), list(test_data))
# >>> [2, 6, 1, 8, 4, 5, 0] [9, 3, 7]

train_data, validation_data, test_data = random_split(range(30), [0.7, 0.2, 0.1], generator=generator2)
print(len(train_data), len(validation_data), len(test_data))
# >>> 21 6 3
```

link_dl/_01_code/_03_real_world_data_to_tensors/h_linear_regression_dataset_dataloader.py

Dataset & DataLoader Examples

- Linear Regression Dataset Example (1/2)

```
import torch
from torch.utils.data import Dataset, DataLoader, random_split

class LinearRegressionDataset(Dataset):
    def __init__(self, N=50, m=-3, b=2, *args, **kwargs):
        # N: number of samples, e.g. 50, m: slope & b: offset
        super().__init__(*args, **kwargs)
        self.x = torch.rand(N, 2)
        self.noise = torch.rand(N) * 0.2
        self.m = m
        self.b = b
        self.y = (torch.sum(self.x * self.m) + self.b + self.noise).unsqueeze(-1)

    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        y = torch.sum(self.x[idx] * self.m) + self.b + self.noise[idx]
        return {'input': self.x[idx], 'target': y.unsqueeze(-1)}


    def __str__(self):
        return "Data Size: {0}, Input Shape: {1}, Target Shape: {2}".format(
            len(self.x), self.x.shape, self.y.shape
        )
```

Dataset & DataLoader Examples

◈Linear Regression Dataset Example (2/2)

```
if __name__ == "__main__":
    linear_regression_dataset = LinearRegressionDataset()
    print(linear_regression_dataset)
    # >>> Data Size: 50, Input Shape: torch.Size([50, 2]), Target Shape: torch.Size([50, 1])

    for idx, sample in enumerate(linear_regression_dataset):
        print("{0} - {1}: {2}".format(idx, sample['input'], sample['target']))

    train_dataset, validation_dataset, test_dataset = random_split(
        linear_regression_dataset, [0.7, 0.2, 0.1]
    )
    print(len(train_dataset), len(validation_dataset), len(test_dataset))
    # >>> 35 10 5

    train_data_loader = DataLoader(dataset=train_dataset, batch_size=4, shuffle=True)

    for idx, batch in enumerate(train_data_loader):
        print("{0} - {1}: {2}".format(idx, batch['input'], batch['target']))
```

Dataset & DataLoader Examples

◈Dog-Cat 2D Image Dataset Example (1/3)

```
import os
from PIL import Image
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms

class DogCat2DImageDataset(Dataset):
    def __init__(self):
        self.image_transforms = transforms.Compose([
            transforms.Resize(size=(256, 256)),
            transforms.ToTensor()
        ])

        dogs_dir = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "a_image-dog")
        cats_dir = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "b_image-cats")

        image_lst = [
            Image.open(os.path.join(dogs_dir, "bobby.jpg")),  # (1280, 720, 3)
            Image.open(os.path.join(cats_dir, "cat1.png")),  # (256, 256, 3)
            Image.open(os.path.join(cats_dir, "cat2.png")),  # (256, 256, 3)
            Image.open(os.path.join(cats_dir, "cat3.png"))  # (256, 256, 3)
        ]
```

Dataset & DataLoader Examples

◈Dog-Cat 2D Image Dataset Example (2/3)

```
class DogCat2DImageDataset(Dataset):
    def __init__(self):
        ...
        image_lst = [self.image_transforms(img) for img in image_lst]
        self.images = torch.stack(image_lst, dim=0)

        # 0: "dog", 1: "cat"
        self.image_labels = torch.tensor([[0], [1], [1], [1]])

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        return {'input': self.images[idx], 'target': self.image_labels[idx]}

    def __str__(self):
        str = "Data Size: {0}, Input Shape: {1}, Target Shape: {2}".format(
            len(self.images), self.images.shape, self.image_labels.shape
        )
        return str
```

Dataset & DataLoader Examples

◈Dog-Cat 2D Image Dataset Example (3/3)

```
if __name__ == "__main__":
    dog_cat_2d_image_dataset = DogCat2DImageDataset()
    print(dog_cat_2d_image_dataset)
    # >>> Data Size: 4, Input Shape: torch.Size([4, 3, 256, 256]), Target Shape: torch.Size([4, 1])

    for idx, sample in enumerate(dog_cat_2d_image_dataset):
        print("{0} - {1}: {2}".format(idx, sample['input'].shape, sample['target']))

    train_dataset, test_dataset = random_split(dog_cat_2d_image_dataset, [0.7, 0.3])

    print(len(train_dataset), len(test_dataset))
    # >>> 3 1

    train_data_loader = DataLoader(
        dataset=train_dataset, batch_size=2, shuffle=True
    )

    for idx, batch in enumerate(train_data_loader):
        print("{0} - {1}: {2}".format(idx, batch['input'].shape, batch['target']))
```

Dataset & DataLoader Examples

◈Wine Dataset Example (1/3)

```
import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader, random_split

class WineDataset(Dataset):
    def __init__(self):
        wine_path = os.path.join(
            os.path.pardir, os.path.pardir, "_00_data", "d_tabular-wine", "winequality-white.csv”
        )
        wineq_numpy = np.loadtxt(wine_path, dtype=np.float32, delimiter=";", skiprows=1)
        wineq = torch.from_numpy(wineq_numpy)

        data = wineq[:, :-1]  # Selects all rows and all columns except the last
        data_mean = torch.mean(data, dim=0)
        data_var = torch.var(data, dim=0)
        self.data = (data - data_mean) / torch.sqrt(data_var)


        target = wineq[:, -1].long()  # treat labels as an integer
        eye_matrix = torch.eye(10)
        self.target = eye_matrix[target]
```

Dataset & DataLoader Examples

◈Wine Dataset Example (2/3)

```
class WineDataset(Dataset):
    ...
    ...

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        wine_feature = self.data[idx]
        wine_target = self.target[idx]
        return {'input': wine_feature, 'target': wine_target}


    def __str__(self):
        str = "Data Size: {0}, Input Shape: {1}, Target Shape: {2}".format(
            len(self.data), self.data.shape, self.target.shape
        )
        return str
```

Dataset & DataLoader Examples

◈Wine Dataset Example (3/3)

```
if __name__ == "__main__":
    wine_dataset = WineDataset()
    print(wine_dataset)
    # >>> Data Size: 4898, Input Shape: torch.Size([4898, 11]), Target Shape: torch.Size([4898, 10])

    for idx, sample in enumerate(wine_dataset):
        print("{0} - {1}: {2}".format(idx, sample['input'].shape, sample['target'].shape))

    train_dataset, validation_dataset, test_dataset = random_split(wine_dataset, [0.7, 0.2, 0.1])
    print(len(train_dataset), len(validation_dataset), len(test_dataset))
    # >>> 3429 980 489

    train_data_loader = DataLoader(
        dataset=train_dataset, batch_size=32, shuffle=True, drop_last=True
    )

    for idx, batch in enumerate(data_loader):
        print("{0} - {1}: {2}".format(idx, batch['input'].shape, batch['target'].shape))
```

Dataset & DataLoader Examples

◈California Housing Dataset Example (1/3)

```
import numpy as np
from torch.utils.data import Dataset, DataLoader, random_split


class CaliforniaHousingDataset(Dataset):
    def __init__(self):
      from sklearn.datasets import fetch_california_housing

      housing = fetch_california_housing()
      data_mean = np.mean(housing.data, axis=0)
      data_var = np.var(housing.data, axis=0)

      self.data = torch.tensor(
          (housing.data - data_mean) / np.sqrt(data_var), dtype=torch.float32
      )

      self.target = torch.tensor(housing.target, dtype=torch.float32).unsqueeze(dim=-1)
```

Dataset & DataLoader Examples

◈California Housing Dataset Example (2/3)

```
class CaliforniaHousingDataset(Dataset):
    ...
    ...

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample_data = self.data[idx]
        sample_target = self.target[idx]
        return {'input': sample_data, 'target': sample_target}

    def __str__(self):
        str = "Data Size: {0}, Input Shape: {1}, Target Shape: {2}".format(
            len(self.data), self.data.shape, self.target.shape
        )
        return str
```

Dataset & DataLoader Examples

◈California Housing Dataset Example (3/3)

```python
if __name__ == "__main__":
    california_housing_dataset = CaliforniaHousingDataset()
    print(california_housing_dataset)
    # >>> Data Size: 20640, Input Shape: torch.Size([20640, 8]), Target Shape: torch.Size([20640, 1])

    for idx, sample in enumerate(california_housing_dataset):
        print("{0} - {1}: {2}".format(idx, sample['input'].shape, sample['target'].shape))

    train_dataset, validation_dataset, test_dataset = random_split(
        california_housing_dataset, [0.7, 0.2, 0.1]
    )
    print(len(train_dataset), len(validation_dataset), len(test_dataset))
    # >>> 14448 4128 2064

    train_data_loader = DataLoader(
        dataset=train_dataset, batch_size=32, shuffle=True, drop_last=True
    )

    for idx, batch in enumerate(data_loader):
        print("{0} - {1}: {2}".format(idx, batch['input'].shape, batch['target'].shape))
```

Dataset & DataLoader Examples

◈Bikes Dataset Example (1/5)

```
import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader, random_split

class BikesDataset(Dataset):
  def __init__(self):
    bikes_path = os.path.join(
      os.path.pardir, os.path.pardir, "_00_data", "e_time-series-bike-sharing-dataset", "hour-fixed.csv”
    )

    bikes_numpy = np.loadtxt(
      fname=bikes_path, dtype=np.float32, delimiter=",", skiprows=1,
      converters={
        1: lambda x: float(x[8:10])  # 2011-01-07 --> 07 --> 7
      }
    )
    bikes = torch.from_numpy(bikes_numpy)

    daily_bikes = bikes.view(-1, 24, bikes.shape[1])  # daily_bikes.shape: torch.Size([730, 24, 17])
    self.daily_bikes_target = daily_bikes[:, :, -1].unsqueeze(dim=-1)
```

Dataset & DataLoader Examples

◈Bikes Dataset Example (2/5)

```
class BikesDataset(Dataset):
  def __init__(self):
    ...

    self.daily_bikes_data = daily_bikes[:, :, :-1]
    eye_matrix = torch.eye(4)

    day_data_torch_list = []
    for daily_idx in range(self.daily_bikes_data.shape[0]):  # range(730)
      day = self.daily_bikes_data[daily_idx]  # day.shape: [24, 17]
      weather_onehot = eye_matrix[day[:, 9].long() - 1]
      day_data_torch = torch.cat(tensors=(day, weather_onehot), dim=1)  # day_torch.shape: [24, 21]
      day_data_torch_list.append(day_data_torch)

    self.daily_bikes_data = torch.stack(day_data_torch_list, dim=0)

    self.daily_bikes_data = torch.cat(
      [self.daily_bikes_data[:, :, :9], self.daily_bikes_data[:, :, 10:]], dim=2
    )
```

Dataset & DataLoader Examples

❖Bikes Dataset Example (3/5)

```
class BikesDataset(Dataset):
  def __init__(self):
    ...

    temperatures = self.daily_bikes_data[:, :, 9]

    self.daily_bikes_data[:, :, 9] = \
      (self.daily_bikes_data[:, :, 9] - torch.mean(temperatures)) / torch.std(temperatures)

    assert len(self.daily_bikes_data) == len(self.daily_bikes_target)
```

Dataset & DataLoader Examples

◈Bikes Dataset Example (4/5)

```
class BikesDataset(Dataset):
    def __init__(self):
        ...


    def __len__(self):
        return len(self.daily_bikes_data)

    def __getitem__(self, idx):
        bike_feature = self.daily_bikes_data[idx]
        bike_target = self.daily_bikes_target[idx]
        return {'input': bike_feature, 'target': bike_target}

    def __str__(self):
        return "Data Size: {0}, Input Shape: {1}, Target Shape: {2}".format(
            len(self.daily_bikes_data), self.daily_bikes_data.shape, self.daily_bikes_target.shape
        )
```

Dataset & DataLoader Examples

◈Bikes Dataset Example (5/5)

```
if __name__ == "__main__":
    bikes_dataset = BikesDataset()
    print(bikes_dataset)
    # >>> Data Size: 730, Input Shape: torch.Size([730, 19, 24]), Target Shape: torch.Size([730, 1, 24])

    for idx, sample in enumerate(bikes_dataset):
        print("{0} - {1}: {2}".format(idx, sample['input'].shape, sample['target'].shape))

    train_dataset, validation_dataset, test_dataset = random_split(bikes_dataset, [0.7, 0.2, 0.1])
    print(len(train_dataset), len(validation_dataset), len(test_dataset))
    # >>> 511 146 73

    train_data_loader = DataLoader(
        dataset= train_dataset, batch_size=32, shuffle=True, drop_last=True
    )

    for idx, batch in enumerate(train_data_loader):
        print("{0} - {1}: {2}".format(idx, batch['input'].shape, batch['target'].shape))
```