Introduction to Deep Learning (CS474)

Lecture 4

Outline

- Representing data through Pytorch **Tensor**
    - Representing tabular data
    - Working with time series

Representing tabular data

- The simplest form of data we'll encounter on a machine learning job is sitting in a spreadsheet, CSV file, or database.
- Whatever the medium, it's a table containing one row per sample (or record), where columns contain one piece of information about our sample.
- Tabular data is typically not homogeneous: different columns don't have the same type. We might have a column showing the weight of apples and another encoding their color in a label.
- PyTorch tensors, on the other hand, are homogeneous.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Representing tabular data

- Let’s see how we can load the data using Python and then turn it into a **PyTorch tensor**. Python offers several options for quickly loading a CSV file.

- Three popular options are
  - The csv module that ships with Python
  - NumPy
  - Pandas

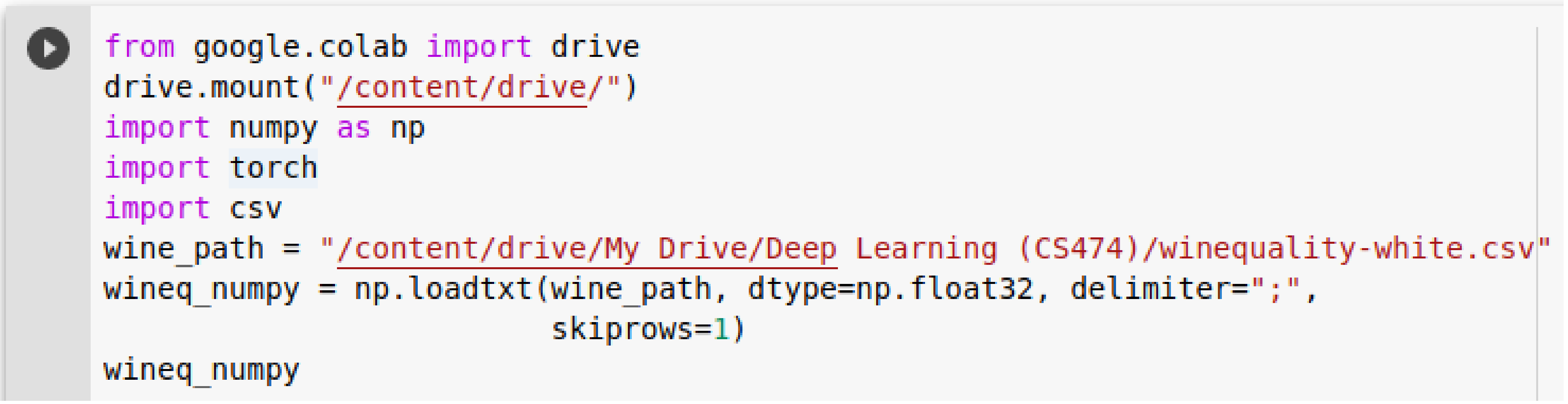
(The third option is the most time and memory-efficient.)

- The Wine Quality dataset is a freely available (http://mng.bz/90Ol)

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Representing tabular data

- Let’s see the following code:



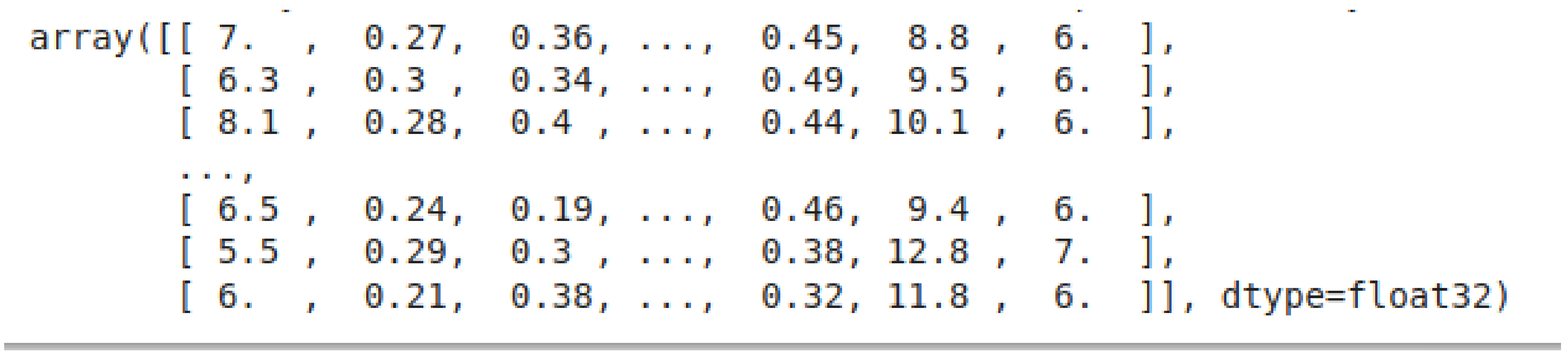
```
from google.colab import drive
drive.mount("/content/drive/")
import numpy as np
import torch
import csv
wine_path = "/content/drive/My Drive/Deep Learning (CS474)/winequality-white.csv"
wineq_numpy = np.loadtxt(wine_path, dtype=np.float32, delimiter=";",
                         skiprows=1)
wineq_numpy
```

- We just prescribe what the type of the 2D array should be (32-bit floating-point), the delimiter used to separate values in each row, and the fact that the first line should not be read since it contains the column names.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Representing tabular data

- Output of the previous code:



```
array([[ 7.   ,  0.27,  0.36, ...,  0.45,  8.8 ,  6.  ],
       [ 6.3 ,  0.3 ,  0.34, ...,  0.49,  9.5 ,  6.  ],
       [ 8.1 ,  0.28,  0.4 , ...,  0.44, 10.1 ,  6.  ],
       ...,
       [ 6.5 ,  0.24,  0.19, ...,  0.46,  9.4 ,  6.  ],
       [ 5.5 ,  0.29,  0.3 , ...,  0.38, 12.8 ,  7.  ],
       [ 6.  ,  0.21,  0.38, ...,  0.32, 11.8 ,  6.  ]], dtype=float32)
```

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Representing tabular data

- Convert the NumPy array to a PyTorch tensor:

```
wineq = torch.from_numpy(wineq_numpy)
wineq.shape, wineq.dtype
```

- **Output**:

  (torch.Size([4898, 12]), torch.float32)

- At this point, we have a floating-point torch.Tensor

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Representing tabular data

- The first kind is **continuous values**. These are the most intuitive when represented as numbers. They are strictly ordered, and a difference between various values has a strict meaning.

- Next we have **ordinal values**. The strict ordering we have with continuous values remains, but the fixed relationship between values no longer applies. A good example of this is ordering a small, medium, or large drink, with small mapped to the value 1, medium 2, and large 3. The large drink is bigger than the medium, in the same way that 3 is bigger than 2, but it doesn't tell us anything about how much bigger.

- Finally, **categorical values** have neither ordering nor numerical meaning to their values. These are often just enumerations of possibilities assigned arbitrary numbers.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Representing tabular data

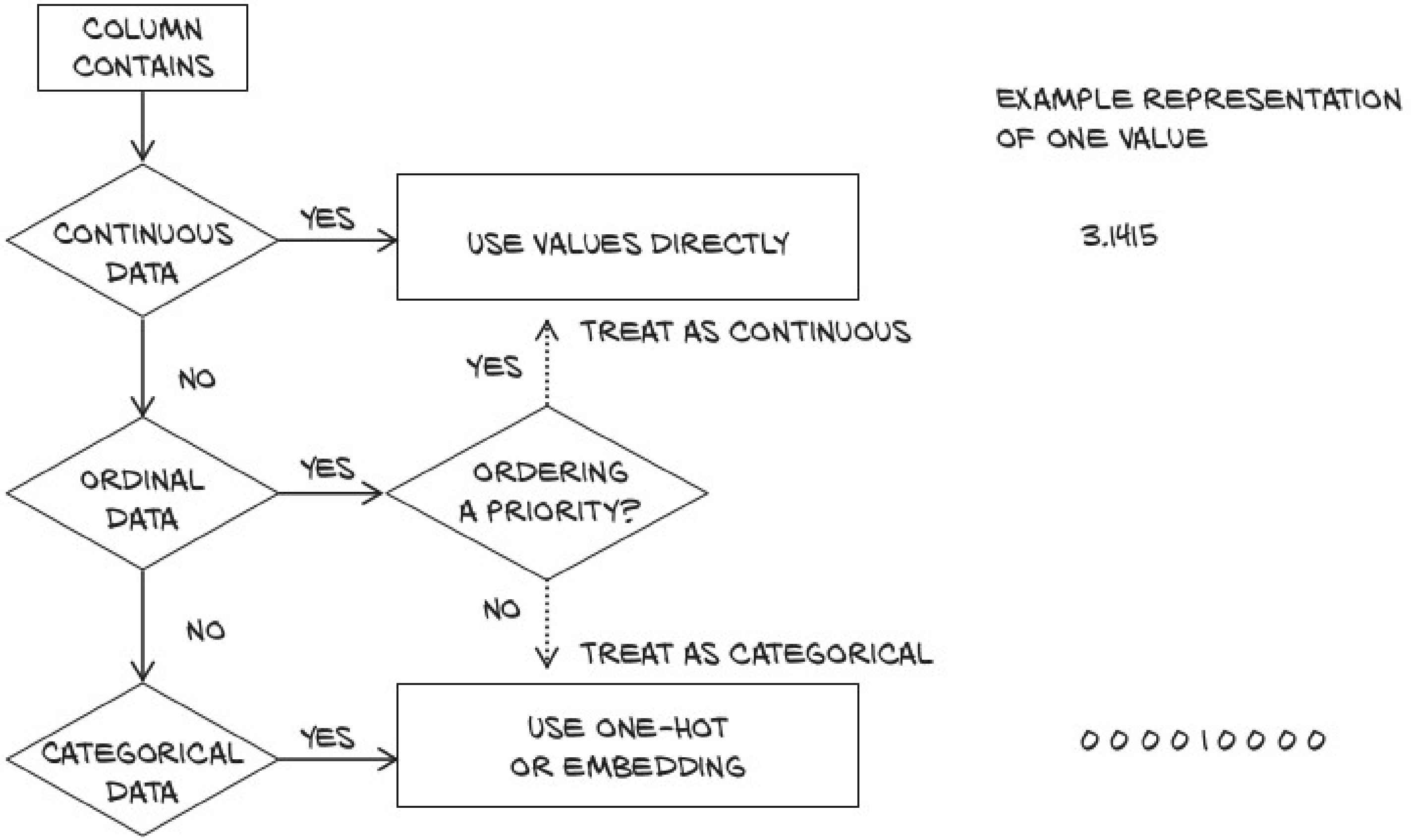


Fig. 1: How to treat columns with continuous, ordinal, and categorical data

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Working with time series

- In the previous dataset, every row in the table was independent from the others; their order did not matter. Or, equivalently, there was no column that encoded information about what rows came earlier and what came later.

- We will switch to another interesting dataset: data from a Washington, D.C., bike-sharing system reporting the hourly count of rental bikes in 2011–2012 in the Capital Bikeshare system, along with weather and seasonal information.

- Our goal will be to take a flat, 2D dataset and transform it into a 3D one.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Working with time series

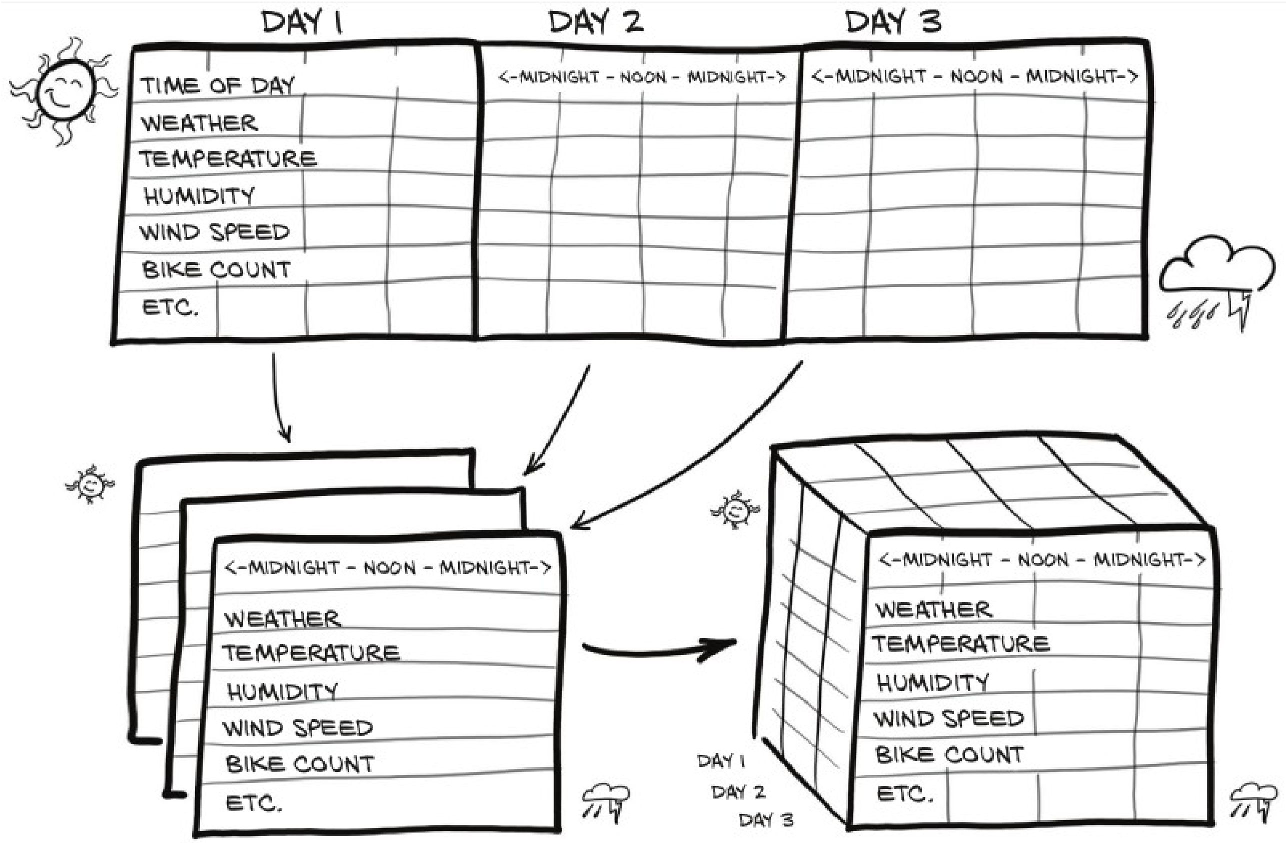
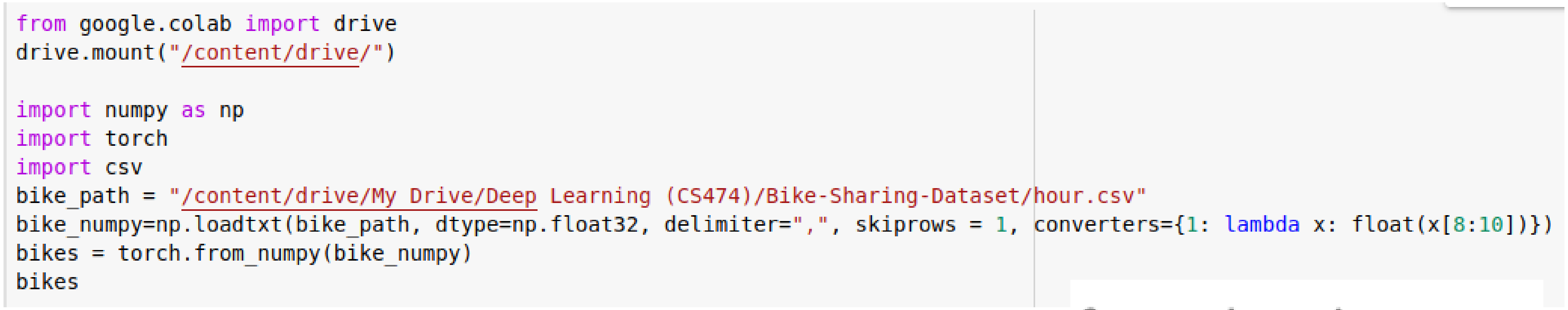


Figure : Transformation Process

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Working with time series



```
from google.colab import drive
drive.mount("/content/drive/")

import numpy as np
import torch
import csv
bike_path = "/content/drive/My Drive/Deep Learning (CS474)/Bike-Sharing-Dataset/hour.csv"
bike_numpy=np.loadtxt(bike_path, dtype=np.float32, delimiter=",", skiprows = 1, converters={1: lambda x: float(x[8:10])})
bikes = torch.from_numpy(bike_numpy)
bikes
```

Converts date strings to numbers corresponding to the day of the month in column 1

Working with time series

For every hour, the dataset reports the following variables:

- 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`

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

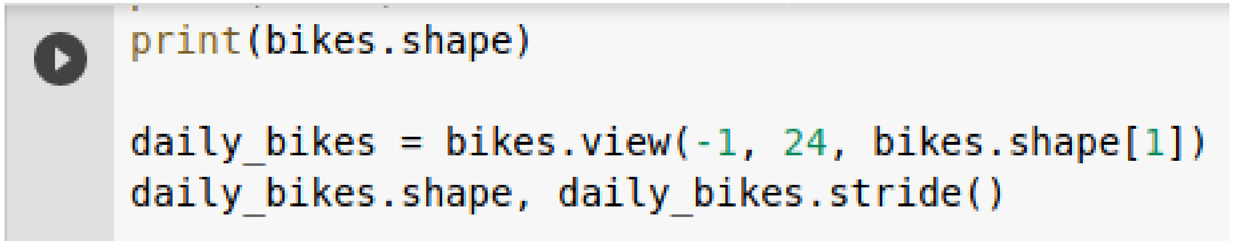
Working with time series

- In a time series dataset such as this one, rows represent successive time-points: there is a dimension along which they are ordered. Sure, we could treat each row as independent and try to predict the number of circulating bikes based on, say, a particular time of day regardless of what happened earlier.

- However, the existence of an ordering gives us the opportunity to exploit causal relationships across time.

Working with time series

- Continuing with last *notebook*;



```
print(bikes.shape)

daily_bikes = bikes.view(-1, 24, bikes.shape[1])
daily_bikes.shape, daily_bikes.stride()
```

```
torch.Size([17520, 17])
(torch.Size([730, 24, 17]), (408, 17, 1))
```

- That’s 17,520 hours, 17 columns. We have reshaped the data to have 3 axes—day, hour, and then our 17 columns.

Working with time series

- First, bikes.shape[1] is 17, the number of columns in the bikes tensor.

- Calling `view` on a tensor returns a new tensor that changes the number of dimensions and the striding information, without changing the storage.

- We see that the rightmost dimension is the number of columns in the original dataset. Then, in the middle dimension, we have time, split into chunks of 24 sequential hours. In other words, we now have N sequences of L hours in a day, for C channels.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.