Week 1

Python and Matrices

01418564 Deep Learning and Its Applications

Chakrit Watcharopas

ภาคต้น 2566

Outline

- Python
- NumPy and Pandas
- Matrices

Python

Why Python for This Course?

- Versatility: Python is a versatile programming language with a wide range of libraries and frameworks that support machine learning and deep learning tasks.
- Readability: Python has a clean and easy-to-read syntax, making it beginner-friendly.
- Community: Python has a large and active community, providing extensive resources and support for beginners.

Python Installation

- Installation: Follow the installation instructions provided by this link https://sites.google.com/ku.th/01418564/โนต/การตดตงซอฟตแวร
- IDE: Recommend using Jupyter Notebook or Anaconda Prompt for an interactive coding environment.

Python Basics

- Variables: Assign values to variables (e.g., x = 10).
- Data Types: Integers, floating-point numbers, strings, lists, tuples, dictionaries, etc.
- Control Flow: If-else statements and loops (for and while).
- Functions: Define and call functions to modularize code.

Python Libraries for ML and DL

- NumPy: Fundamental package for scientific computing with support for multi-dimensional arrays and mathematical functions.
- Pandas: Library for data manipulation and analysis, offering powerful data structures like DataFrames.
- Matplotlib: Plotting library for creating visualizations, such as line plots, scatter plots, and histograms.
- Scikit-learn: Machine learning library providing tools for classification, regression, clustering, and more.
- TensorFlow: A library for machine learning and artificial intelligence.

NumPy

What is NumPy?

- NumPy stands for Numerical Python.
- It is a fundamental package for scientific computing in Python.
- NumPy provides powerful data structures and functions for efficient numerical operations.

NumPy Features

- Multi-dimensional Arrays: NumPy arrays (ndarrays) are n-dimensional homogeneous arrays, enabling efficient storage and manipulation of large datasets.
- Mathematical Functions: NumPy provides a comprehensive library of mathematical functions for array operations.
- Broadcasting: NumPy allows operations on arrays of different shapes and sizes, automatically handling element-wise calculations.
- Integration with Other Libraries: NumPy seamlessly integrates with other libraries like Pandas, Matplotlib, and SciPy.

Creating NumPy Arrays

- Array Creation: NumPy arrays can be created using the np.array() function by passing a Python list, tuple, or any iterable object.

```
import numpy as np

data = [1, 2, 3, 4, 5]
arr = np.array(data)
```

NumPy Array Attributes

- Shape: The shape attribute returns a tuple indicating the dimensions of the array (e.g., (3, 4) for a 3x4 array).
- dtype: The dtype attribute shows the data type of the elements in the array (e.g., int64, float32, etc.).
- ndim: The ndim attribute returns the number of dimensions of the array.

```
print(arr.shape)   # Output: (5,)
print(arr.dtype)   # Output: int64
print(arr.ndim)    # Output: 1
```

Array Operations

- Element-wise Operations: NumPy allows performing element-wise operations on arrays, such as addition, subtraction, multiplication, and division.
- Broadcasting: Arrays of different shapes can be combined in operations, and NumPy automatically handles broadcasting.

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

result = arr1 + arr2
print(result)  # Output: [5, 7, 9]
```

NumPy Functions

- Mathematical Functions: NumPy provides a wide range of mathematical functions such as np.sin(), np.cos(), np.mean(), np.max(), etc.
- Array Manipulation: NumPy offers functions for reshaping, concatenating, and splitting arrays.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(np.mean(arr))    # Output: 3.0
print(np.max(arr))     # Output: 5
print(np.reshape(arr, (5, 1)))  # Output: [[1], [2], [3], [4], [5]]
```

Array Indexing, Slicing, and Axes

- Indexing: Access individual elements of an array using square brackets and indices.
- Slicing: Extract subsets of an array using slicing notation.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr[0])      # Output: 1
print(arr[1:4])    # Output: [2, 3, 4]
print(arr[::2])    # Output: [1, 3, 5]

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
arr.min(axis=0)    # Output: [1, 2, 3, 4, 5]
arr.min(axis=1)    # Output: [1, 6]
```

Pandas

What is Pandas?

- Pandas is an open-source data manipulation library for Python.
- It provides data structures and functions to efficiently work with structured data.
- Pandas is built on top of NumPy and is widely used in data analysis and data science.

Pandas Features

- DataFrame: The DataFrame is the central data structure in Pandas, representing a tabular, two-dimensional dataset with labeled axes (rows and columns).
- Data Manipulation: Pandas offers powerful tools for filtering, transforming, and aggregating data.
- Missing Data Handling: Pandas provides methods to handle missing data effectively.
- Data Input and Output: Pandas supports reading and writing data from various file formats, including CSV, Excel, SQL databases, and more.

Creating DataFrames

- From Lists/Arrays: Create a DataFrame from a list or array of data.
- From CSV/Excel: Read data from CSV or Excel files using Pandas' read_csv() or read_excel() functions.
- From Dictionaries: Create a DataFrame from a dictionary.

```
import pandas as pd

data = {'Name': ['John', 'Alice', 'Bob'],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)
```

Exploring DataFrames

- Head and Tail: Use the head() and tail() methods to view the first or last few rows of the DataFrame.
- Shape: The shape attribute returns the dimensions of the DataFrame (rows, columns).
- Info: The info() method provides a summary of the DataFrame's structure and data types.

```
print(df.head())     # Output: First 5 rows of the DataFrame
print(df.shape)      # Output: (3, 2)
print(df.info())     # Output: Summary of DataFrame information
```

Data Manipulation with Pandas

- Selecting Columns: Access specific columns using column names or indices.
- Filtering Rows: Use boolean indexing to filter rows based on specific conditions.
- Aggregation: Perform calculations on groups of data using functions like sum(), mean(), etc.

```
# Selecting columns
print(df['Name'])
print(df[['Name', 'Age']])
```

```
# Filtering rows
filtered_df = df[df['Age'] > 25]
```

```
# Aggregation
average_age = df['Age'].mean()
```

Handling Missing Data

- Detecting Missing Data: Use isnull() or notnull() functions to detect missing values in the DataFrame.
- Handling Missing Data: Fill missing values using fillna() or dropna() functions.

```
import pandas as pd

# Assuming the DataFrame df is already created

# Detecting missing data
print(df.isnull())

# Handling missing data
df_filled = df.fillna(0)
df_dropped = df.dropna()
```

Data Input and Output

- Reading Data: Use read_csv(), read_excel(), or other functions to read data from different file formats.
- Writing Data: Use to_csv(), to_excel(), or other functions to write data to different file formats.

```
import pandas as pd

# Reading data
df = pd.read_csv('data.csv')

# Writing data
df.to_excel('output.xlsx', index=False)
```

Matrices

What is a Matrix?

- A matrix is a rectangular array of numbers, symbols, or expressions arranged in rows and columns.
- Matrices are usually denoted by capital letters, e.g., A, B, C.
- The individual values within a matrix are called elements.
- The element in the i-th row and j-th column of a matrix A is denoted as A[i, j].

$$B = \begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 6 & 5 \end{bmatrix}$$

Matrix Dimensions

- The size or dimensions of a matrix are described by the number of rows and columns it contains.
- An m x n matrix has m rows and n columns.
- A matrix with 3 rows and 2 columns is referred to as a 3 x 2 matrix.

Matrix Multiplication

- Matrix multiplication is an operation that combines two matrices to produce a third matrix.
- For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix.
- Suppose we have two matrices, A and B, where A is a 2 x 3 matrix and B is a 3 x 2 matrix.

$$AB = \begin{bmatrix} 1 & 3 & 5 \\ 5 & 6 & 7 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 1*2+3*3+5*6 & 1*3+3*4+5*5 \\ 5*2+6*3+7*6 & 5*3+6*4+7*5 \end{bmatrix}$$

$$= \begin{bmatrix} 41 & 40 \\ 70 & 74 \end{bmatrix}$$

Use of Matrix Multiplication - Linear Systems

- Matrix multiplication is employed in solving systems of linear equations and finding solutions to problems in physics, engineering, and economics.

$$a_1x + b_1y + c_1z = d_1$$
$$a_2x + b_2y + c_2z = d_2$$
$$a_3x + b_3y + c_3z = d_3$$

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

Properties of Matrix Multiplication

- Associativity: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
- Non-Commutativity: In general, $\mathbf{AB} \neq \mathbf{BA}$
- Distributivity: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
- Identity Matrix: $\mathbf{AI} = \mathbf{A} = \mathbf{IA}$ (where $\mathbf{I}$ is the identity matrix)
- Inverses: If a matrix $\mathbf{A}$ has an inverse $\mathbf{A}^{-1}$, then $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$

References

Varsity Tutors. *Representing Systems of Linear Equations using Matrices*. Available at: https://www.varsitytutors.com/hotmath/hotmath_help/topics/representing-systems-of-linear-equations-using-matrices , accessed June 8, 2023.