AI VIET NAM – COURSE 2022

Data Structure – Exercise (Week 1)

Ngày 30 tháng 5 năm 2022

1. Cho một list các số nguyên **num_lis**t và một sliding window (các bạn có thể tạm hiểu sliding window như là một list có kích thước nhỏ hơn **num_list**) có kích thước size **k** di chuyển từ trái sang phải. Mỗi lần dịch chuyển 1 vị trí sang phải có thể nhìn thấy được **k** số trong **num_list** và tìm số lớn nhất trong **k** số này sau mỗi lần trượt. **k** phải lớn hơn hoặc bằng 1. Các bạn hãy viết chương trình Python giải quyết vấn đề trên.

   - **Input:** num_list $= [3, 4, 5, 1, -44, 5, 10, 12, 33, 1]$, $k = 3$
   - **Output:** $[5, 5, 5, 5, 10, 12, 33, 33]$
   - Với input trên, quá trình tính toán của chương trình có thể được mô phỏng như sau (các bạn không nhất thiết phải code phần mô phỏng này):
     - **[3, 4, 5]**$, 1, -44, 5, 10, 12, 33, 1 \longrightarrow max = 5$
     - $3,$ **[4, 5, 1]**$, -44, 5, 10, 12, 33, 1 \longrightarrow max = 5$
     - $3, 4,$ **[5, 1, -44]**$, 5, 10, 12, 33, 1 \longrightarrow max = 5$
     - $3, 4, 5,$ **[1, -44, 5]**$, 10, 12, 33, 1 \longrightarrow max = 5$
     - $3, 4, 5, 1,$ **[-44, 5 , 10]**$, 12, 33, 1 \longrightarrow max = 10$
     - $3, 4, 5, 1, -44,$ **[5, 10, 12]**$, 33, 1 \longrightarrow max = 12$
     - $3, 4, 5, 1, -44, 5,$ **[10, 12, 33]**$, 1 \longrightarrow max = 33$
     - $3, 4, 5, 1, -44, 5, 10,$ **[12, 33, 1]** $\longrightarrow max = 33$

2. Cho 2 list các số nguyên là **num_list1** và **num_list2**, các bạn hãy viết chương trình trả về list các số đều có trong 2 list đầu vào, thứ tự không quan trọng. **Lưu ý**, không được sử dụng hàm intersection() có sẵn của Python.

   - **Case 1:**
     - **Input:** $nums1 = [1, 2, 2, 1]$, $nums2 = [2, 2]$
     - **Output:** $[2, 2]$
   - **Case 2:**
     - **Input:** $nums1 = [4, 9, 5], nums2 = [9, 4, 9, 8, 4]$
     - **Output:** $[4, 9]$ hoặc $[9, 4]$

3. Cho các hàm toán học với công thức như sau:

   - MAE $= \dfrac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
   - MSE $= \dfrac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

- RMSE $= \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
- Huber Loss (với $\delta = 0.5$) $= \frac{1}{n}\sum_{i=1}^{n} \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases}$

Các công thức trên trong lĩnh vực Máy học (Machine Learning) còn được gọi là các hàm mất mát (Loss Function), các hàm này đóng vai trò như là một thước đo sự khác biệt giữa giá trị dự đoán của mô hình máy học so với giá trị thực tế của một mẫu dữ liệu (samples). **Trong đó:** n là **số lượng samples (num_samples)**, với $i(0 < i \leq n)$ là thứ tự của mỗi sample cụ thể. Ở đây các bạn có thể hiểu là cứ mỗi i thì sẽ **có** 1 **cặp** y_i **là target và** $\hat{y}$ **là predict**.

Các bạn hãy viết chương trình Python với một số yêu cầu như sau:

- Định nghĩa một hàm trả về 2 **list** y (target) và $\hat{y}$ (prediction). Mỗi list chứa số lượng các element bằng với **num_sample** và các element này dược tạo ngẫu nhiên trong khoảng $[0, 10)$.
- – **Input:**
    - ∗ Người dùng **nhập số lượng sample (num_samples) được tạo ra (chỉ nhận integer numbers)**
    - ∗ Người dùng **nhập loss name (MAE, MSE, RMSE, Huber_Loss)**
  - **Output:** Print ra **loss name và kết quả loss cuối cùng**. Loss name là loss mà người dùng chọn
- Phải kiểm tra **num_samples** có hợp lệ hay không (**num_samples** phải là số nguyên dương). Nếu không hợp lệ thì in ra màn hình một chuỗi **'number of samples must be a postive integer number'** và dừng chương trình.
- Phải kiểm tra tên của hàm mất mát có hợp lệ hay không **(MAE, MSE, RMSE, Huber_Loss)**. Nếu không in ra màn hình một chuỗi **"loss name loss is not supported"**.
- Phải kiểm tra số lượng element của y và $\hat{y}$ có bằng nhau hay không? Và số lượng này có bằng **num_sample** hay không? Nếu không thì in ra màn hình một chuỗi **'The number of samples is incorrect'** và dừng chương trình.
- Khi đã qua các vòng check điều kiện, thực hiện tính giá trị hàm mất mát với 2 list y và $\hat{y}$ theo tên hàm mất mát người dùng nhập. Sau đó in tên của hàm mất mát đã chọn cùng kết quả trả về của hàm.
- Khi khởi tạo tự động xong y và $\hat{y}$ thì in kết quả ra màn hình như trong ví dụ ở code listing 1 (target cho y và predict cho $\hat{y}$).

```
exercise1()
>> Input number of samples (positive integer number) which are generated: 5
Input loss name: MAE
target: [5.312817926824581, 1.8851779941522784, 5.31777783817793,
    4.291091534773635, 5.436601786280094]
predict: [3.996040971399059, 0.18450975801382552, 0.6649914880642105,
    3.3606067157683697, 7.961606576438889]
MAE: 2.2251442301683513


exercise1()
>> Input number of samples (positive integer number) which are generated: 4
number of samples must be a positive integer number
```

```
12
13
14 exercise1()
15 >> Input number of samples (positive integer number) which are generated: 0
16 number of samples must be a positive integer number
```

Code Listing 1: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

4. Cho một số hàm toán học và đạo hàm của nó như sau:

   - **Tanh:**
     - $tanh(x) = \dfrac{2}{1+e^{-2x}} - 1$
     - $tanh'(x) = 1 - tanh(x)^2$
   - **Sigmoid:**
     - $sigmoid(x) = \dfrac{1}{1+e^{-x}}$
     - $sigmoid'(x) = sigmoid(x) \times (1 - sigmoid(x))$
   - **Rectified Linear Unit (ReLU):**
     - $ReLU(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$
     - $ReLU'(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$
   - **Parametric Rectified Linear Unit (PReLU):**
     - Với $\alpha = 0.25$:
     - $PReLU(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$
     - $PReLU'(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$
   - **Leaky Rectified Linear Unit (LeakyReLU):**
     - Với $\alpha = 0.01$:
     - $LeakyReLU(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$
     - $LeakyReLU'(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$
   - **Exponential Linear Unit (ELU):**
     - Với $\alpha = 0.01$:
     - $ELU(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$
     - $ELU'(x) = \begin{cases} \alpha e^x & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$
   - **Softplus:**
     - $softplus(x) = \log(1 + e^x)$
     - $softplus'(x) = \dfrac{1}{1+e^{-x}}$

- **Exponential:**
  - $exponential(x) = e^x$
  - $exponential'(x) = e^x$
- **Scaled Exponential Linear Unit (SELU):**
  - Với $\lambda = 1.05$ và $\alpha = 1.67$:
  - $SELU(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$
  - $SELU'(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^x & \text{if } x \leq 0 \end{cases}$
- **Gaussian Error Linear Unit (GELU):**
  - $GELU(x) = 0.5x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$
  - $GELU'(x) = 0.5 \tanh(0.0356774x^3 + 0.797885x) + (0.0535161x^3 + 0.398942x) \operatorname{sech}^2(0.0356774x^3 + 0.797885x) + 0.5$
- **Hard Sigmoid:**
  - $hardSigmoid(x) = \begin{cases} 0 & \text{if } x < -2.5 \\ 0.2x + 0.5 & \text{if } -2.5 \leq x \leq 2.5 \\ 1 & \text{if } 2.5 < x \end{cases}$
  - $hardSigmoid'(x) = \begin{cases} 0 & \text{if } x < -2.5 \\ 0.2 & \text{if } -2.5 \leq x \leq 2.5 \\ 0 & \text{if } 2.5 < x \end{cases}$
- **Softsign:**
  - $softsign(x) = \frac{x}{1 + |x|}$
  - $softsign'(x) = \frac{1}{(1 + |x|)^2}$
- **Swish:**
  - $swish(x) = x \times \frac{1}{1 + e^{-x}}$
  - $swish'(x) = swish(x) + \frac{1}{1 + e^{-x}}(1 - swish(x))$

Các hàm trên trong lĩnh vực Học sâu (Deep Learning) còn được gọi là các Hàm kích hoạt (Activation Function), đóng vai trò rất quan trọng trong các kiến trúc mạng nơ-ron (Neural Network). Các bạn hãy thực hiện viết chương trình Python khai báo các hàm trên với một số yêu cầu sau:

- Xây dựng một hàm tổng hợp các hàm kích hoạt, cho phép lựa chọn hàm kích hoạt mong muốn thông qua tham số đầu vào và trả về giá trị hàm kích hoạt và giá trị đạo hàm của hàm kích hoạt tương ứng. Đồng thời, in ra hình ảnh vẽ trên đồ thị các kết quả trên. Hàm vẽ đồ thị sẽ được cài đặt như sau (các bạn cần tùy chỉnh lại tên biến sao cho phù hợp với code của mình):

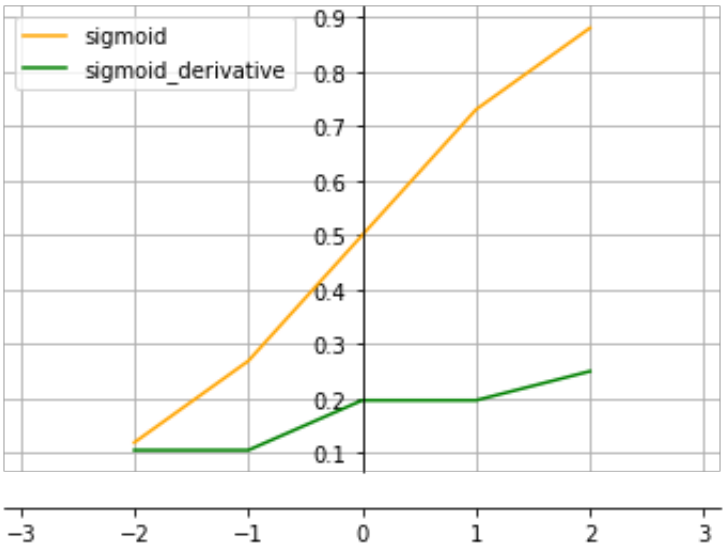
```
import numpy as np
import matplotlib.pyplot as plt

def plot(data, output, output_derivative, activation_name):
  x = np.array(list(sorted(data)))
```

```
  y = np.array(list(sorted(output)))
  y_derivative = np.array(list(sorted(output_derivative)))
  ax = plt.gca()
  ax.spines['top'].set_color('none')
  ax.spines['left'].set_position('zero')
  ax.spines['right'].set_color('none')
  ax.spines['bottom'].set_position('zero')
  plt.xlim(-np.pi, np.pi)
  plt.plot(x, y, color='orange', label=activation_name)
  plt.plot(x, y_derivative, color='green', label=f'{activation_name}
    _derivative')
  plt.grid(True)
  plt.legend()
  plt.show()

```

Code Listing 2: Hàm vẽ đồ thị kết quả tính toán hàm kích hoạt và đạo hàm của nó

- – **Input:** Một list các số bất kì và tên hàm kích hoạt mong muốn sử dụng.
  – **Output:** Giá trị hàm kích hoạt, giá trị đạo hàm hàm kích hoạt và hình vẽ của kết quả trên trục tọa độ.
- Kết quả trả về mẫu (các bạn không nhất thiết phải làm giống với ví dụ mẫu, chỉ cần đảm bảo output yêu cầu):

```
data = [-2, -1, 0, 1, 2]
exercise3(data, activation_name='sigmoid')
>>>
Activation function:  sigmoid
Original data: [-2, -1, 0, 1, 2]
sigmoid(data) =  [0.11920292202211757, 0.2689414213699951, 0.5,
    0.7310585786300049, 0.8807970779778823]
sigmoid_derivative(data) =  [0.10499358540350653, 0.19661193324148185, 0.25,
    0.19661193324148185, 0.10499358540350662]

```



5. **(OPTIONAL)** Trong lập trình, bài toán tìm kiếm phần tử trong list là một trong những bài toán lập trình cơ bản, với input đầu vào là một list và phần tử ta muốn tìm kiếm **x**, output sẽ

là chỉ mục **i** (index) hay còn được gọi là vị trí của phần tử **x** trong list. **Binary Search** là một trong những thuật toán giải quyết vấn đề này, với các bước thực hiện nhau:

(a) Từ list đầu vào, sắp xếp lại các phần tử của list theo thứ tự **tăng dần**. Nếu list đã được sắp xếp sẵn thì không cần thực hiện bước này.

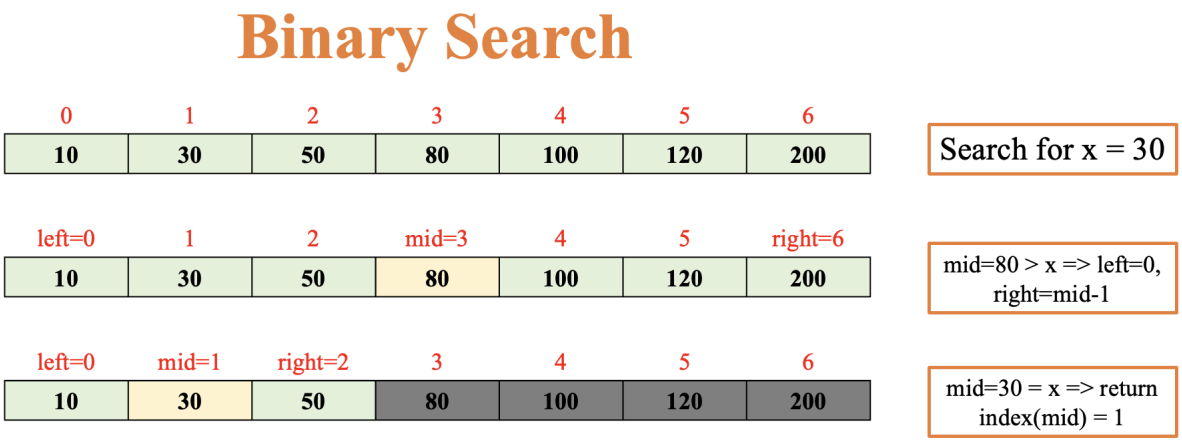
(b) Khởi tạo 2 biến **left** $= 0$ và **right** $= len(list) - 1$. Hai biến này dùng để xác định biên tìm kiếm trong list.

(c) Sử dụng phần tử ở giữa của list làm giá trị cho biến **mid** (có thể tận dụng hai biến **left**, **right** đã khởi tạo phía trên để tìm).

(d) Kiểm tra nếu giá trị của biến **mid** bằng với giá trị phần tử muốn tìm kiếm **x** $\longrightarrow$ Trả về chỉ mục của biến **mid** và kết thúc chương trình. Nếu không, ta xét tiếp như sau:

- Nếu **mid** $>$ **x**, gán biến **mid** bằng giá trị của phần tử nằm giữa vị trí **left** và vị trí **right** $= index($**mid**$) - 1$ trong list.
- Nếu **mid** $<$ **x**, gán biến **mid** bằng giá trị của phần tử nằm giữa vị trí **left** $= index($**mid**$)+1$ và **right** trong list.

(e) Lặp lại bước **(d)** cho tới khi **mid** $=$ **x**. Nếu không tìm được giá trị **mid** nào bằng với giá trị **x** cần tìm, trả về **-1**.



Hình 1: Ảnh minh họa mô phỏng quá trình xử lý của **Binary Search**

Dựa vào các bước tính toán trên, các bạn hãy viết chương trình Python khởi tạo một hàm **binary_search()** mô phỏng lại thuật toán **Binary Search** với một số yêu cầu như sau:

- **Case 1:**
  - **Input:** $lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, $x = 3$
  - **Output:** $i = 2$
- **Case 2:**
  - **Input:** $lst = [10, 30, 50, 80, 99, 100, 140, 200]$, $x = 500$
  - **Output:** $i = -1$

–Hết–