AI VIET NAM – COURSE 2022

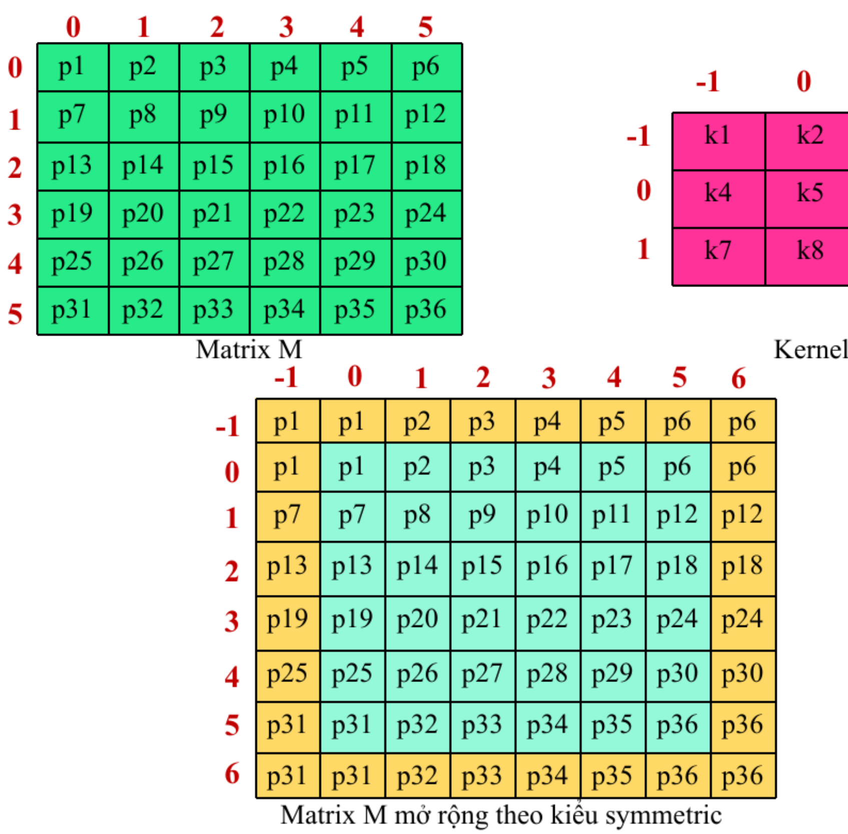
Calculus - Exercise 2

(Derivative and its Applications)

Ngày 25 tháng 7 năm 2022

1. **Sobel Edge Detector: Trình bày chi tiết convolution operation (2D convolution) với một ma trận M (6x6) cho trước bằng hai Sobel convolution kernels ($\mathbf{K_x}$, and $\mathbf{K_y}$) và thực hiện code bằng python**

$$\mathbf{M} = \begin{bmatrix} 4 & 5 & 2 & 4 & 8 & 5 \\ 8 & 11 & 8 & 11 & 12 & 13 \\ 12 & 14 & 17 & 16 & 16 & 20 \\ 21 & 19 & 21 & 23 & 25 & 25 \\ 28 & 26 & 29 & 30 & 30 & 32 \\ 34 & 32 & 32 & 33 & 38 & 36 \end{bmatrix}, \mathbf{K_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \mathbf{K_y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



Hình 1: Ví dụ tọa độ của matrix $\mathbf{M}$ và $\mathbf{M}$ mở rộng theo kiểu symmetric và kernel $\mathbf{K_x}$ và $\mathbf{K_y}$

$$\textbf{Convolution Formula:} \quad G_{x \text{ or } y} = F * I(x, y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} F(i, j) I(x - i, y - j)$$

- **NOTE: Convolution Formula đối với trường hợp này:**
  - F là kernel $\mathbf{K_x}$ hoặc $\mathbf{K_y}$
  - i và j trong range $[-N, N]$. Bài tập này là trong range $[-1, 1]$
  - $I(x, y)$ là matrix $\mathbf{M}$
  - x và y trong range $[0, 5]$
- **NOTE: Các bạn thực hiện theo các yêu cầu sau**
  - **Step0**: Viết biểu thức tính $G_{x \text{ or } y}$ cho từng vị trí x, từ 0 đến 5 và y từ 0 đến 1. Dùng Convolution Formula cho ma trận **M** mở rộng theo kiểu symmetric
  - **Step1**: Tìm một phần $\mathbf{G_x}$ bằng cách dùng Convolution Formula cho ma trận **M** mở rộng theo kiểu symmetric với kernel $\mathbf{K_x}$. (Thế số từ ma trận **M** và $\mathbf{K_x}$ vào các biểu thức ở Step0 tại các vị trí (x=0, y=1), (x=0, y=2), (x=0, y=3), (x=0, y=4), (x=0, y=5))
  - **Step2**: Tìm một phần $\mathbf{G_y}$ bằng cách dùng Convolution Formula cho ma trận **M** mở rộng theo kiểu symmetric với kernel $\mathbf{K_y}$. (Thế số từ ma trận **M** và $\mathbf{K_y}$ vào các biểu thức ở Step0 tại các vị trí (x=0, y=1), (x=0, y=2), (x=0, y=3), (x=0, y=4), (x=0, y=5))
  - **Step3**: Tìm $\mathbf{G} = |\mathbf{G_x}| + |\mathbf{G_y}|$, tại các vị trí (x=0, y=1), (x=0, y=2), (x=0, y=3), (x=0, y=4), (x=0, y=5)
  - **Step4**: Tạo ra 2D-array kernel $\mathbf{K_x}$ bằng Python
  - **Step5**: Tạo ra 2D-array kernel $\mathbf{K_y}$ bằng Python
  - **Step6**: Dùng method convolve2d(in1, in2, mode="same", boundary="symm") từ thư viện scipy (scipy.signal.convolve2d) để tính được toàn bộ $\mathbf{G_x}$ (6x6) và $\mathbf{G_y}$ (6x6). (ví dụ: in1 là ma trận M, in2 là $\mathbf{K_x}$ hoặc $\mathbf{K_y}$)
  - **Step7**: Viết function *compute_sobel_edges(matrix)* nhận chỉ một input (matrix) là ma trận tương tự ma trận **M**. Sau đó thực hiện tuần tự Step3, Step4, Step5 (in1 là matrix) và return $\mathbf{G} = |\mathbf{G_x}| + |\mathbf{G_y}|$
  - Trình bày chi tiết convolution operation Step0, Step1, Step2 và Step3 (Có thể viết bằng latex sau đó gửi file (dạng ảnh hoặc pdf), hoặc viết bằng markdown trên google colab)
  - Sau đó các bạn viết code cho function *compute_sobel_edges(matrix)* (Step4, Step5, Step6, và Step7), tạo ra ma trận **M** và kiểm tra kết quả với Step3.
- **Example**
  - **Trình bày convolution operation**
    - **Step0**:
      - $G_{x \text{ or } y}$ **tại x = 0, y = 0**: F(-1,-1)I(1,1) + F(-1,0)I(1,0) + F(-1, 1)I(1,-1) + F(0, -1)I(0,1) + F(0,0)I(0,0) + F(0,1)I(0,-1) + F(1,-1)I(-1,1) + F(1,0)I(-1,0) + F(1,1)I(-1,-1) = k1*p8 + k2*p7 + k3*p7 + k4*p2 + k5*p1 + k6*p1 + k7*p2 + k8*p1 + k9*p1
      - Tính tương tự cho các x và y theo range như sau x từ 0 đến 5 và y từ 0 đến 1.
    - **Step1**:
      - G_x **tại x = 0, y = 0**: -11 + 0 + 8 + -10 + 0 + 8 + -5 + 0 + 4 = -6
      - Tính tương tự cho các x và y tại (x=0, y=1), (x=0, y=2), (x=0, y=3), (x=0, y=4), (x=0, y=5)
    - **Step2**:

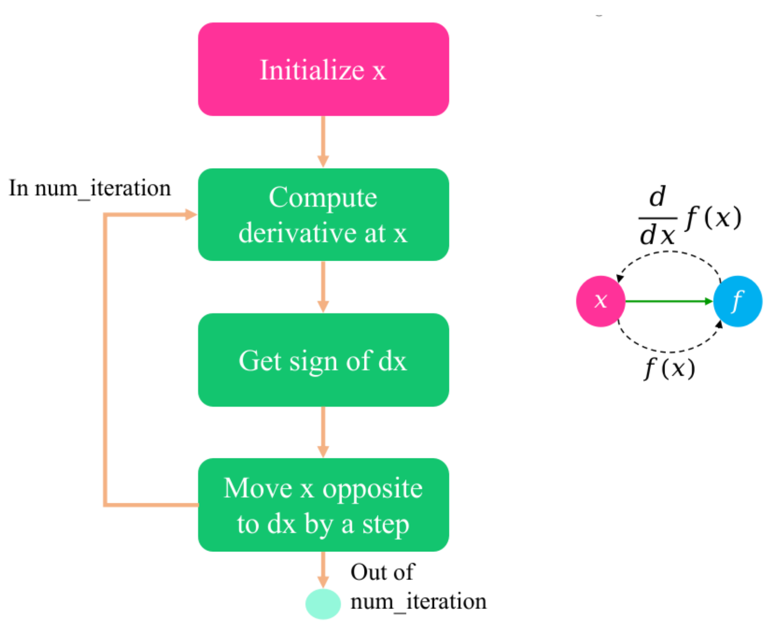
- G_y **tại x = 0, y = 0**: 11 + 16 + 8 + 0 + 0 + 0 + -5 + -8 + -4 = 18
- Tính tương tự cho các x và y tại (x=0, y=1), (x=0, y=2), (x=0, y=3), (x=0, y=4), (x=0, y=5)

* **Step3**:
  - G **tại x = 0, y = 0**: |-6| + |-18| = 24
  - Tính tương tự cho các x và y tại (x=0, y=1), (x=0, y=2), (x=0, y=3), (x=0, y=4), (x=0, y=5)

```
import numpy as np
from scipy.signal import convolve2d

M = np.array(
    [[ 4,  5,  2,  4,  8,  5,],
    [ 8, 11,  8, 11, 12, 13,],
    [12, 14, 17, 16, 16, 20,],
    [21, 19, 21, 23, 25, 25,],
    [28, 26, 29, 30, 30, 32,],
    [34, 32, 32, 33, 38, 36,]]
)
G = compute_sobel_edges(matrix=M)
print(G)
>> [[24 28 28 46 28 36]
 [42 44 52 60 52 56]
 [52 52 54 56 62 58]
 [64 58 64 62 64 56]
 [60 50 58 56 56 48]
 [32 26 22 36 34 24]]
```

2. **Simple Optimization: Thực hiện thuật toán optimization đơn giản sau để tìm vị trí tại x mà $f(x)$ là minimum**



Hình 2: Simple Optimization Algorithm

NOTE: Các bạn thực hiện theo các yêu cầu sau

2.1 Viết function *find_minimum(f, x, num_iteration, step)* và dựa theo thuật toán ở hình 2 để tìm xấp xỉ x mà $f(x)$ (ví dụ $f(x) = 3x^4 - 4x^2 - 6x - 3$) là minimum.

- **Input**: Nhận 4 input
  - **f**: function $f(x)$
  - **x**: giá trị khởi tạo x đầu tiên
  - **num_iteration**: Số lần lặp thuật toán để tìm x
  - **step**: độ lớn để cho một lần cập nhật x (độ lớn quãng đường đi ngược hướng với giá trị đạo hàm tại x (dx))
- **Output** giá trị xấp xỉ của x tại đó giá trị hàm $f(x)$ (hàm được truyền vào từ input) là minimum
- **Các bạn thực hiện theo các step sau**:
  - **Step1**: Thực hiện vòng lặp với num_iteration số lần lặp
  - **Step2**: Trong mỗi lần lặp tìm giá trị đạo hàm (dx) tại x của hàm $f(x)$ bằng phương pháp đạo hàm trung tâm (central difference)
  - **Step3**: Xét dấu của giá trị đạo hàm (dx) để xác định độ lớn giá trị cập nhật.
  - **Step4**: Nếu dx là số dương thì cập nhật $x = x - step$, nếu dx là số âm thì cập nhật $x = x + step$, nếu $dx = 0$ thì không thực hiện việc cập nhật
  - **Step5**: Thực hiện Step2, Step3 và Step4 để cập nhật x cho đến khi đủ num_iteration số lần lặp thì thoát loop và trả về x mới nhất

```python
# Example 2.1
import random
def f(x):
    return 3*x**4 - 4*x**2 - 6*x - 3

x = random.uniform(-10, 10)
print("initial x: ", x)
>> initial x:  8.033107966229196

x = find_minima(f=f, x=x, num_iteration=100, step=0.1)
print(x)
>> 1.033107966229207
```

2.2 Trình bày chi tiết simple optimization algorithm từng bước cập nhật giá trị x ở câu 2.1 với $f(x) = 3x^4 - 4x^2 - 6x - 3$ và trong năm lần tặp đầu tiên (num_iteration = 5)(Các bạn có thể xem ví dụ bên dưới). Biết rằng giá trị khởi tạo đầu tiên $x = 3.0$, step = 0.1. Sau đó các bạn dùng *find_minimum(f, x, num_iteration, step)* ở 2.1 để kiểm tra kết quả (print từng x mỗi lần cập nhật).

- **Note**: Trình bày chi tiết simple optimization algorithm có thể viết bằng latex sau đó gửi file (dạng ảnh hoặc pdf), hoặc viết bằng markdown trên google colab
- **Example:**
  - $f(x) = x^2$, num_iteration=5, step=0.1, x khởi tạo đầu tiên bằn 2.0
  - $f'(x) = 2x$
  - **Lần đầu tiên:**
    - dx = 2x = 2*2.0 = 4.0
    - dx > 0, x = x - step = 2.0 - 0.1 = 1.9
  - **Lần thứ hai:**

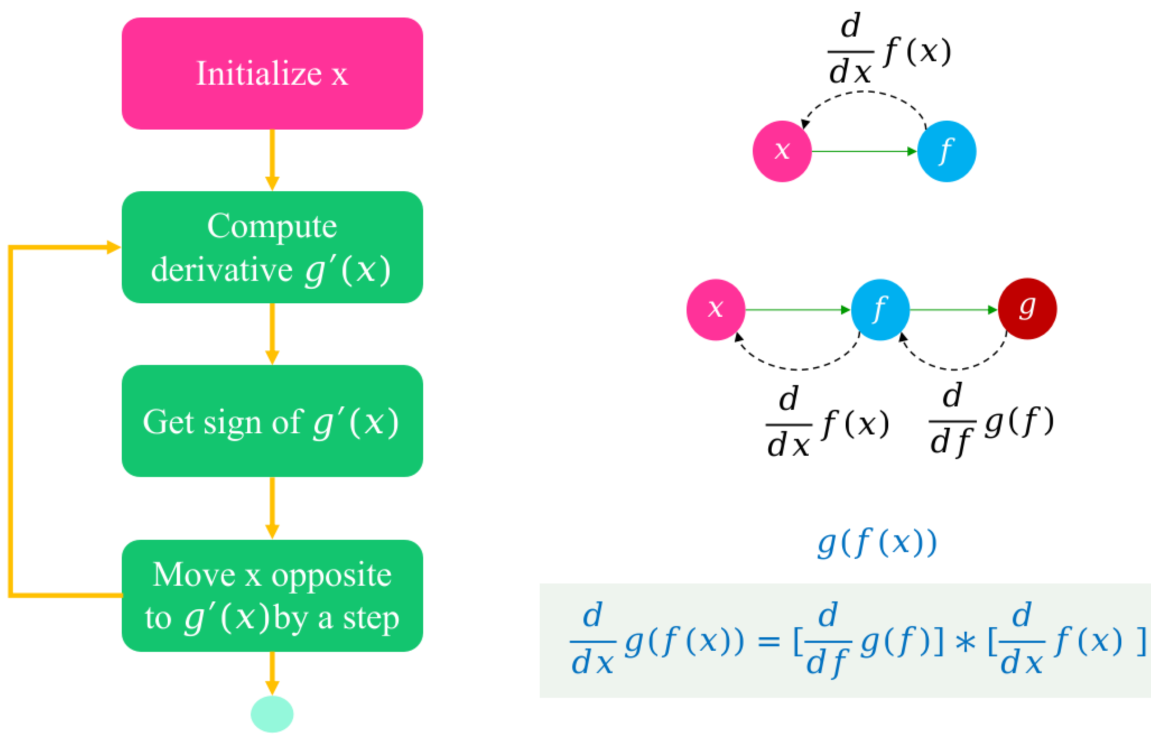
  * dx = 2x = 2*1.9 = 3.8
  * dx > 0, x = x - step = 1.9 - 0.1 = 1.8
- Thực hiện tương tự cho đến lần thứ năm
- Kiểm tra kết quả bằng function đã viết ở 2.1 (print từng x mỗi lần cập nhật)

```
# Example 2.2
def f(x):
    return x**2

x = 2.0
print("initial x: ", x)
>> initial x:  2.0

x = find_minimum(f=f, x=x, num_iteration=5, step=0.1)
print(x)
>> 1.9
1.7999999999999998
1.6999999999999997
1.5999999999999996
1.4999999999999996
```

3. **Simple Optimization (Chain Rule): Thực hiện thuật toán optimization đơn giản sau để tìm vị trí tại x mà $g(f(x)$ là minimum**



Hình 3: Simple Optimization Algorithm with Chain Rule

NOTE: Các bạn thực hiện theo các yêu cầu sau

3.1 Viết function *find_minimum(f, g, x, num_iteration, step)* và dựa theo thuật toán ở hình 3 để tìm xấp xỉ x mà $g(f(x))$ (ví dụ $f(x) = 2x^3 - 3x^2 + 4x$, và $g(x) = x^2 - 10x + 2$) là minimum.

- **Input**: Nhận 5 input
  - **f**: function $f(x)$
  - **g**: function $g(x)$ (lưu ý bài toán này sẽ là $g(f(x))$, x ở đây biểu diễn biến truyền vào và khác với x của $f(x)$)
  - **x**: giá trị khởi tạo x đầu tiên để đưa vào $f(x)$
  - **num_iteration**: Số lần lặp thuật toán để tìm x
  - **step**: độ lớn để cho một lần cập nhật x (độ lớn quãng đường đi ngược hướng với giá trị đạo hàm tại x của $g(f(x))$. ($g'(f(x))$ hoặc dg_dx)
- **Output** giá trị xấp xỉ của x tại đó giá trị hàm $g(f(x))$ (hàm được truyền vào từ input) là minimum
- **Các bạn thực hiện theo các step sau**:
  - **Step1**: Thực hiện vòng lặp với num_iteration số lần lặp
  - **Step2**: Trong mỗi lần lặp tìm giá trị đạo hàm (df_dx) tại x của hàm $f(x)$, giá trị đạo hàm (dg_df) tại $f(x)$ của hàm $g(f(x))$ bằng phương pháp đạo hàm trung tâm (central difference)
  - **Step3**: Tính giá trị đạo hàm (dg_dx) tại x của hàm $g(f(x))$ (dg_dx = dg_df * df_dx)
  - **Step4**: Xét dấu của giá trị đạo hàm (dg_dx) để xác định độ lớn giá trị cập nhật.
  - **Step5**: Nếu dg_dx là số dương thì cập nhật $x = x - step$, nếu dg_dx là số âm thì cập nhật $x = x + step$, nếu dg_dx = 0 thì không thực hiện việc cập nhật
  - **Step6**: Thực hiện Step2, Step3, Step4 và Step5 để cập nhật x cho đến khi đủ num_iteration số lần lặp thì thoát loop và trả về x mới nhất

```python
# Example 3.1
import random
def f(x):
    return 2*x**3 - 3*x**2 + 4*x

def g(x):
    return x**2 - 10*x + 2

x = random.uniform(-2, 2)
print("initial x: ", x)
>> initial x:  -1.2054332142134205

x = find_minimum(f=f, g=g, x=x, num_iteration=40, step=0.1)
print(x)
>> 1.3945667857865798
```

3.2 Trình bày chi tiết simple optimization algorithm từng bước cập nhật giá trị x ở câu 3.1 với $f(x) = 2x^3 - 3x^2 + 4x$, và $g(x) = x^2 - 10x + 2$ và trong năm lần tặp đầu tiên (num_iteration = 5)(Các bạn có thể xem ví dụ bên dưới). Biết rằng giá trị khởi tạo đầu tiên x = -1.5, step = 0.1. Sau đó các bạn dùng *find_minimum(f, g, x, num_iteration, step)* ở 3.1 để kiểm tra kết quả (print từng df_dx, dg_df, dg_dx và x mỗi lần cập nhật).

- **Note**: Trình bày chi tiết simple optimization algorithm (chain rule) có thể viết bằng latex sau đó gửi file (dạng ảnh hoặc pdf), hoặc viết bằng markdown trên google colab
- **Example:**

– $f(x) = x^2 + 3x$, $g(x) = x^3 + x + 2$, num_iteration=5, step=0.1, x khởi tạo đầu tiên bằn 1.0
– $f'(x) = 2x + 3$, $g'(x) = 3x^2 + 1$, $g'(f(x)) = 3(x^2 + 3x)(2x + 3) + (2x + 3)$
– **Lần đầu tiên:**
  ∗ df_dx = 2x +3 = 2*1.0 +3 = 5.0
  ∗ dg_df = 3*$f^2(x)$ + 1 = 3*$f^2(1.0)$ + 1 = 3*4^2 +1 = 49.0
  ∗ dg_dx = dg_df*df_dx = 49.0 * 5.0 = 245.0
  ∗ dg_dx > 0, x = x - step = 1.0 - 0.1 = 0.9
– **Lần đầu tiên:**
  ∗ df_dx = 2x +3 = 2*0.9 +3 = 4.8
  ∗ dg_df = 3*$f^2(x)$ + 1 = 3*$f^2(0.9)$ + 1 ≈ 3*3.51^2 +1 ≈ 37.9603
  ∗ dg_dx = dg_df*df_dx = 37.9603 * 4.8 ≈ 182.20943
  ∗ dg_dx > 0, x = x - step = 0.9 - 0.1 = 0.8
– Thực hiện tương tự cho đến lần thứ năm
– Kiểm tra kết quả bằng function đã viết ở 3.1 (print từng df_dx, dg_df, dg_dx và x mỗi lần cập nhật)

```python
# Example 3.2
def f(x):
    return x**2 + 3*x

def g(x):
    return x**3 + x + 2

x = 1.0
print("initial x: ", x)
>> initial x:  1.0

x = find_minimum(f=f, g=g, x=x, num_iteration=5, step=0.1)
print(x)
>> df_dx 5.000000413701855
dg_df 48.99987970929942
dg_dx 244.99941881783823
0.9
df_dx 4.800000397153781
dg_df 37.960319332341896
dg_dx 182.20954787132544
0.8
df_dx 4.600000380605707
dg_df 28.724826961479266
dg_dx 132.13421495563767
0.7000000000000001
df_dx 4.400000364057632
dg_df 21.124293425600627
dg_dx 92.94689876310301
0.6000000000000001
df_dx 4.200000347509558
dg_df 14.996821562363039
dg_dx 62.986655773463596
0.5000000000000001
```