AI VIET NAM – COURSE 2022

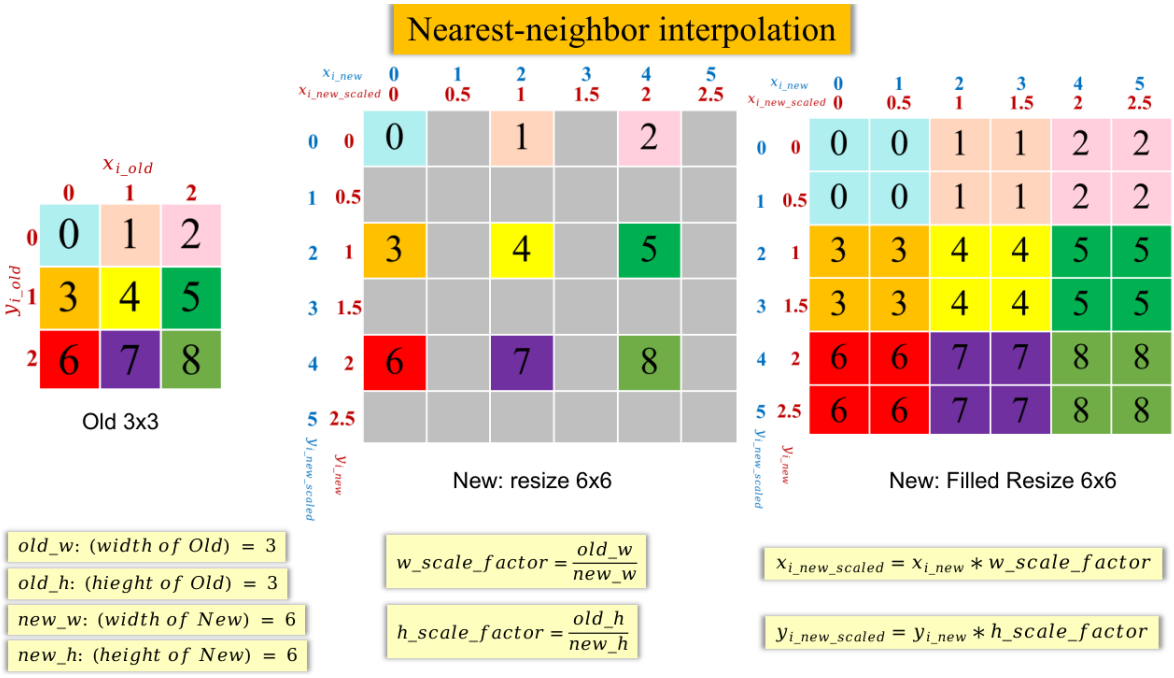
Calculus - Exercise 3

Interpolation và Linear Regression

Ngày 2 tháng 8 năm 2022

NOTE: Các bạn lưu ý truy cập giá trị của ma trận hoặc ảnh thì index phải luôn được convert sang kiểu integer

1. **Nearest Neighbor Interpolation: Thực hiện tăng kích thước của ảnh bằng thuật toán Nearest Neighbor Interpolation**



Hình 1: Ví dụ thuật toán Nearest Neighbor Interpolation tăng kích thước ảnh từ 3x3 lên 6x6

NOTE: Các bạn thực hiện theo các yêu cầu sau

1.1 Viết function *resize_nearest_neighbor_interpolation_gray(image, new_h, new_w)* để resize ảnh xám (1 channel) tăng lên theo một kích thước mới

- **Input**: nhận 3 tham số image: ảnh đầu vào (là ma trận có dim=2 vd (3x3)). new_h: chiều cao mới của ảnh sau khi tăng kích thước, new_w: chiều dài mới của ảnh sau khi tăng kích thước
- **Output**: Kết quả ảnh sau khi tăng kích thước theo new_h và new_w dùng Nearest Neighbor Interpolation.
- **Step1**: Lấy chiều cao (old_h) và chiều dài (old_w) của ảnh input

- **Step2**: Tạo ra ma trận mới tên là new_image có kích thước là (new_h, new_w)
- **Step3**: Tính w_scale_factor và h_scale_factor như hình 1
- **Step4**: Loop hết tất cả vị trí trong new_image theo hàng ngang (x) và theo hàng dọc (y). Mỗi lần lặp sẽ thu được x_{i_new} và y_{i_new}. Sau đó dùng công thức trong hình 1 thu được $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$
- **Step5**: Làm tròn xuống (ví dụ math.floor) để làm tròn xuống thành số nguyên $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$
- **Step6**: sau đó lấy gía trị tại vị trí $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$ (đã được làm tròn xuống) của image (input) gán vào vị trí x_{i_new} và y_{i_new} của new_image
- Sau khi hoàn thành loop hết các vị trí trên new_image thì return new_image
- **NOTE:** Nếu các bạn muốn áp dụng với ảnh thật thì phải đọc ảnh lên convert sang ảnh gray rồi đưa vào function *resize_nearest_neighbor_interpolation_gray(image, new_h, new_w)* khi return thì nên convert new_image sang type uint8 để tiện cho việc hiển thị. Các bạn có thể tham khảo ví dụ code bên dưới đọc ảnh, resize và plot.
- **Example**

```
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
# generate matrix 3x3
image = np.arange(0,9).reshape(3,3).astype(np.uint8)
print(image)
>> [[0 1 2]
 [3 4 5]
 [6 7 8]]

new_image = resize_nearest_neighbor_interpolation_gray(image=image, new_h
    =6, new_w=6)
print(new_image)
>> [[0 0 1 1 2 2]
 [0 0 1 1 2 2]
 [3 3 4 4 5 5]
 [3 3 4 4 5 5]
 [6 6 7 7 8 8]
 [6 6 7 7 8 8]]

# Example with real image
# read image and convert to gray image
image = cv2.imread("/content/a.jpg", 0)
new_image = resize_nearest_neighbor_interpolation_gray(image, image.shape
    [0]*2, image.shape[1]*2)
# plot old image
plt.imshow(image, cmap='gray')
# plot new_image
plt.imshow(new_image, cmap='gray')
```

1.2 Viết function *resize_nearest_neighbor_interpolation_color(image, new_h, new_w)* để resize ảnh màu (3 channel) tăng lên theo một kích thước mới

- **Input**: nhận 3 tham số image: ảnh đầu vào (là ma trận có dim=3 vd (3x3x3)). new_h: chiều cao mới của ảnh sau khi tăng kích thước, new_w: chiều dài mới của ảnh sau khi tăng kích thước
- **Output**: Kết quả ảnh màu (3 channels) sau khi tăng kích thước theo new_h và new_w dùng Nearest Neighbor Interpolation.

- **Step1**: Lấy chiều cao (old_h), chiều dài (old_w), và số lượng kênh màu (channel) của ảnh input
- **Step2**: Tạo ra ma trận mới tên là new_image có kích thước là (new_h, new_w, channel)
- **Step3**: Tính w_scale_factor và h_scale_factor như hình 1
- **Step4**: Loop hết tất cả vị trị theo từng channel (c), mỗi channel sẽ có trong new_image theo hàng ngang (x) và theo hàng dọc (y). Mỗi lần lặp sẽ thu được c, x_{i_new} và y_{i_new}. Sau đó dùng công thức trong hình 1 thu được $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$ theo từng channel c
- **Step5**: Làm tròn xuống (ví dụ math.floor) để làm tròn xuống thành số nguyên $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$
- **Step6**: sau đó lấy gía trị tại vị trí $x_{i\_new\_scaled}$, $y_{i\_new\_scaled}$ và channel c tương ứng (đã được làm tròn xuống) của image (input) gán vào vị trí x_{i_new} và y_{i_new} theo channel c của new_image
- Sau khi hoàn thành loop hết các vị trí trên new_image thì return new_image
- **NOTE:** Nếu các bạn muốn áp dụng với ảnh thật thì phải đọc ảnh màu lên rồi đưa vào function *resize_nearest_neighbor_interpolation_color(image, new_h, new_w)* khi return thì nên convert new_image sang type uint8 để tiện cho việc hiển thị. Các bạn có thể tham khảo ví dụ code bên dưới đọc ảnh, resize và plot. Khi dùng opencv cần convert ảnh từ BGR sang RGB mới hiển thị được đúng màu với matplotlib
- **Example**

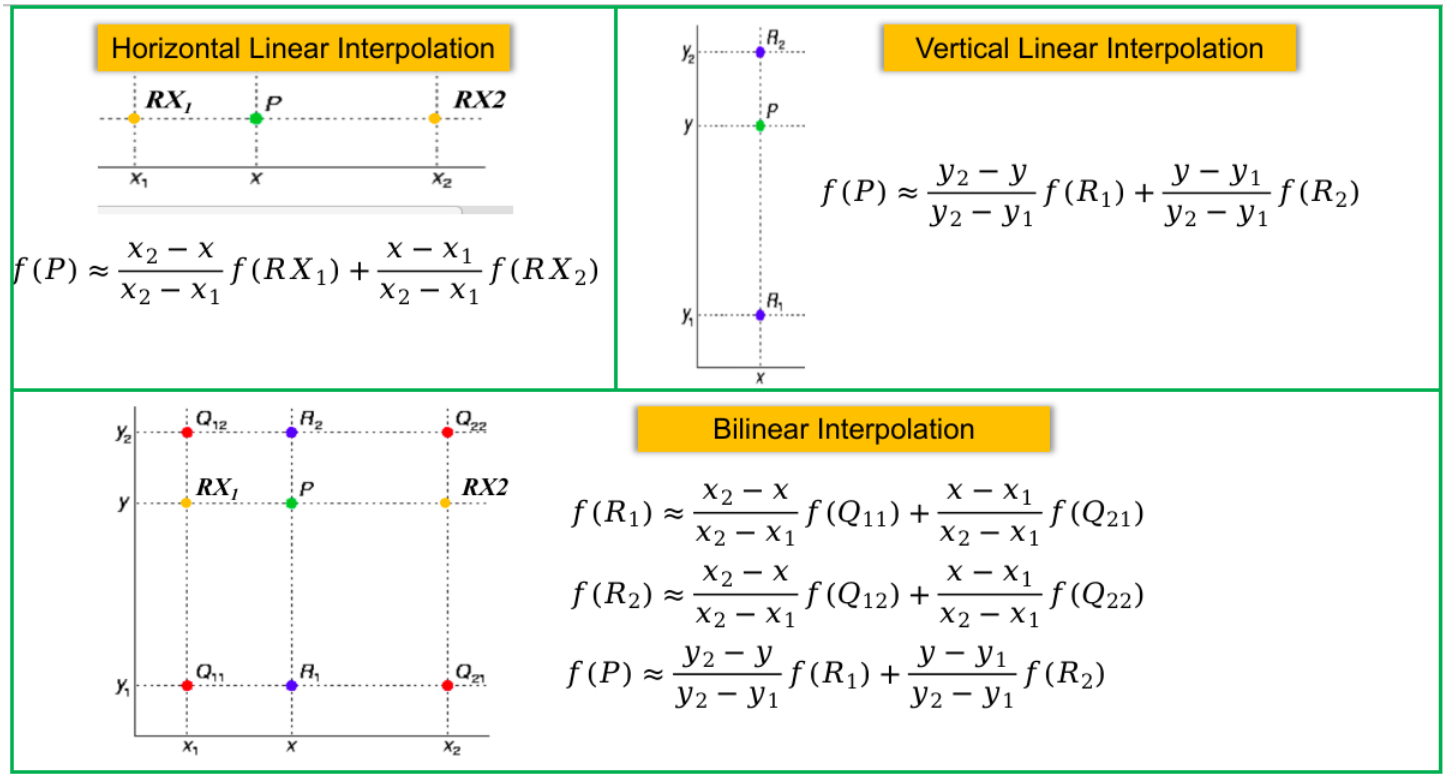
```
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt

# Example with real image
# read image and convert to gray image
image = cv2.imread("/content/a.jpg")
new_image = resize_nearest_neighbor_interpolation_color(image, image.shape
    [0]*2, image.shape[1]*2)
# plot old image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
# plot new_image
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
```

2. **Bilinear Interpolation: Thực hiện tăng kích thước của ảnh bằng thuật toán Bilinear Interpolation**

   NOTE: Các bạn thực hiện theo các yêu cầu sau

   2.1 Viết function *resize_bilinear_interpolation_gray(image, new_h, new_w)* để resize ảnh xám (1 channel) tăng lên theo một kích thước mới

   - **Input**: nhận 3 tham số image: ảnh đầu vào (là ma trận có dim=2 vd (3x3)). new_h: chiều cao mới của ảnh sau khi tăng kích thước, new_w: chiều dài mới của ảnh sau khi tăng kích thước
   - **Output**: Kết quả ảnh sau khi tăng kích thước theo new_h và new_w dùng Bilinear Interpolation.
   - **Step1**: Lấy chiều cao (old_h) và chiều dài (old_w) của ảnh input
   - **Step2**: Tạo ra ma trận mới tên là new_image có kích thước là (new_h, new_w)
   - **Step3**: Tính w_scale_factor và h_scale_factor như hình 1



Hình 2: Ví dụ thuật toán Bilinear Interpolation tăng kích thước ảnh từ 3x3 lên 6x6

- **Step4**: Loop hết tất cả vị trị trong new_image theo hàng ngang (x) và theo hàng dọc (y). Mỗi lần lặp sẽ thu được x_{i_new} và y_{i_new}. Sau đó dùng công thức trong hình 1 thu được $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$
- **Step5**: Sau đó dùng công thức sau để tìm tọa độ các điểm hỗ trợ thực hiện bilinear interpolation

```
x1 = math.floor(xi_new_scaled)
x2 = min(old_w - 1, math.ceil(xi_new_scaled))
y2 = math.floor(yi_new_scaled)
y1 = min(old_h - 1, math.ceil(yi_new_scaled))
```

- **Step6**: Sẽ có 4 trường hợp xảy ra và cần áp dụng công thức như hình 2 trong đó Q11, Q12, Q22, Q21, R1, R2, RX1, RX2 là những điểm có tọa độ theo format (x, y). f(điểm) là gía trị pixel (hoặc giá trị của elmenet trong ma trận) theo điểm cần tìm. Tại đây giá trị sau khi interpolation của điểm cần tìm là f(P).
  - **Case1** (x2 == x1) and (y1 == y2):
    - f(P) chính là giá trị của image tại vị trí $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$
  - **Case2** (x2 == x1): (Sử dụng (vertical) linear interpolation hình 2)
    - Tìm f(R1) = giá trị của image tại điểm có tọa độ $x_{i\_new\_scaled}$ và y1
    - Tìm f(R2) = giá trị của image tại điểm có tọa độ $x_{i\_new\_scaled}$ và y2
    - Dùng công thức (vertical) linear interpolation để tính f(P)
  - **Case3** (y1 == y2): (Sử dụng (horizontal) linear interpolation hình 2)
    - Tìm f(RX1) = giá trị của image tại điểm có tọa độ $y_{i\_new\_scaled}$ và x1
    - Tìm f(RX2) = giá trị của image tại điểm có tọa độ $y_{i\_new\_scaled}$ và x2

    - ∗ Dùng công thức (horizontal) linear interpolation để tính f(P)
  - **Case4** trường hợp còn lại: (Sử dụng bilinear interpolation hình 2)
    - ∗ Dùng x1, y1, x2, y2 cùng với image để kiếm được f(Q11), f(Q12), f(Q21), f(Q22)
    - ∗ Sau đó dùng thông tin này đi kiếm f(R1), f(R2) để kiếm được f(P)
- **Step7**: Gán gía trị f(P) vào new_image theo vị trí x_{i_new} và y_{i_new}
- Sau khi hoàn thành loop hết các vị trí trên new_image thì return new_image
- **NOTE:** Nếu các bạn muốn áp dụng với ảnh thật thì phải đọc ảnh lên convert sang ảnh gray rồi đưa vào function *resize_bilinear_interpolation_gray(image, new_h, new_w)* khi return thì nên convert new_image sang type uint8 để tiện cho việc hiển thị. Các bạn có thể tham khảo ví dụ code bên dưới đọc ảnh, resize và plot.
- **Example**

```
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
# generate matrix 3x3
image = np.arange(0,9).reshape(3,3).astype(np.uint8)
print(image)
>> [[0 1 2]
 [3 4 5]
 [6 7 8]]

new_image = resize_bilinear_interpolation_gray(image=image, new_h=6, new_w
    =6)
print(new_image)
>> [[0 0 1 1 2 2]
 [1 2 2 3 3 3]
 [3 3 4 4 5 5]
 [4 5 5 6 6 6]
 [6 6 7 7 8 8]
 [6 6 7 7 8 8]]

# Example with real image
# read image and convert to gray image
image = cv2.imread("/content/a.jpg", 0)
new_image = resize_bilinear_interpolation_gray(image, image.shape[0]*2,
    image.shape[1]*2)
# plot old image
plt.imshow(image, cmap='gray')
# plot new_image
plt.imshow(new_image, cmap='gray')
```

2.2 Viết function *resize_bilinear_interpolation_color(image, new_h, new_w)* để resize ảnh màu (3 channel) tăng lên theo một kích thước mới

- **Input**: nhận 3 tham số image: ảnh đầu vào (là ma trận có dim=3 vd (3x3x3)). new_h: chiều cao mới của ảnh sau khi tăng kích thước, new_w: chiều dài mới của ảnh sau khi tăng kích thước
- **Output**: Kết quả ảnh sau khi tăng kích thước theo new_h và new_w dùng Bilinear Interpolation.
- **Step1**: Lấy chiều cao (old_h), chiều dài (old_w) và số lượng kênh màu (channel) của ảnh input
- **Step2**: Tạo ra ma trận mới tên là new_image có kích thước là (new_h, new_w, channel)
- **Step3**: Tính w_scale_factor và h_scale_factor như hình 1

- **Step4**: Loop hết tất cả vị trí theo từng channel (c), mỗi channel sẽ có trong new_image theo hàng ngang (x) và theo hàng dọc (y). Mỗi lần lặp sẽ thu được c, x_{i_new} và y_{i_new}. Sau đó dùng công thức trong hình 1 thu được $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$ theo từng channel c
- **Step5**: Sau đó dùng công thức sau để tìm tọa độ các điểm hỗ trợ thực hiện bilinear interpolation

```
x1 = math.floor(xi_new_scaled)
x2 = min(old_w - 1, math.ceil(xi_new_scaled))
y2 = math.floor(yi_new_scaled)
y1 = min(old_h - 1, math.ceil(yi_new_scaled))
```

- **Step6**: Sẽ có 4 trường hợp xảy ra và cần áp dụng công thức như hình 2 trong đó Q11, Q12, Q22, Q21, R1, R2, RX1, RX2 là những điểm có tọa độ theo format (x, y). f(điểm) là gía trị pixel (hoặc giá trị của elmenet trong ma trận) theo điểm cần tìm. Tại đây giá trị sau khi interpolation của điểm cần tìm là f(P).
  - **Case1** (x2 == x1) and (y1 == y2):
    - f(P) chính là giá trị của image tại vị trí $x_{i\_new\_scaled}$ và $y_{i\_new\_scaled}$ với channel c tương ứng.
  - **Case2** (x2 == x1): (Sử dụng (vertical) linear interpolation hình 2)
    - Tìm f(R1) = giá trị của image tại điểm có tọa độ $x_{i\_new\_scaled}$ và y1 với channel c tương ứng.
    - Tìm f(R2) = giá trị của image tại điểm có tọa độ $x_{i\_new\_scaled}$ và y2 với channel c tương ứng.
    - Dùng công thức (vertical) linear interpolation để tính f(P)
  - **Case3** (y1 == y2): (Sử dụng (horizontal) linear interpolation hình 2)
    - Tìm f(RX1) = giá trị của image tại điểm có tọa độ $y_{i\_new\_scaled}$ và x1 với channel c tương ứng.
    - Tìm f(RX2) = giá trị của image tại điểm có tọa độ $y_{i\_new\_scaled}$ và x2 với channel c tương ứng.
    - Dùng công thức (horizontal) linear interpolation để tính f(P)
  - **Case4** trường hợp còn lại: (Sử dụng bilinear interpolation hình 2)
    - Dùng x1, y1, x2, y2 cùng với image để kiếm được f(Q11), f(Q12), f(Q21), f(Q22) với channel c tương ứng.
    - sau đó dùng thông tin này đi kiếm f(R1), f(R2) để kiếm được f(P)
- **Step7**: Gán gía trị f(P) vào new_image theo vị trí x_{i_new}, y_{i_new} và channel c tương ứng
- Sau khi hoàn thành loop hết các vị trí trên new_image thì return new_image
- **NOTE:** Nếu các bạn muốn áp dụng với ảnh thật thì phải đọc ảnh lên rồi đưa vào function *resize_nearest_neighbor_interpolation_color(image, new_h, new_w)* khi return thì nên convert new_image sang type uint8 để tiện cho việc hiển thị. Các bạn có thể tham khảo ví dụ code bên dưới đọc ảnh, resize và plot. Khi dùng opencv cần convert ảnh từ BGR sang RGB mới hiển thị được đúng màu với matplotlib
- **Example**

```
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
```

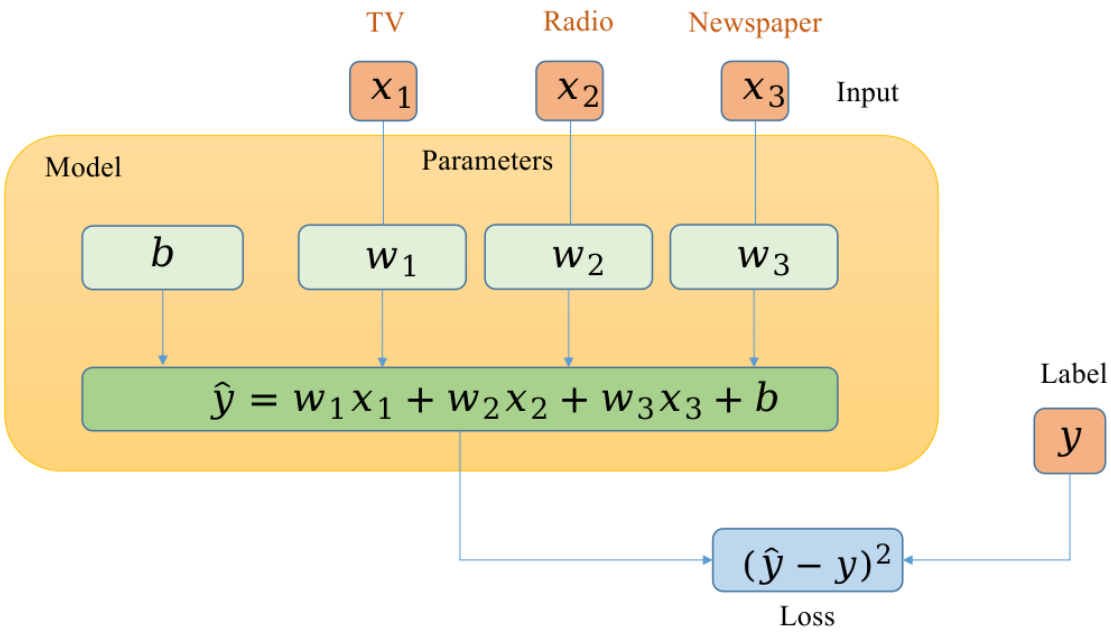
```
5
6  image = cv2.imread("/content/a.jpg")
7  new_image = resize_bilinear_interpolation_color(image, image.shape[0]*2,
       image.shape[1]*2)
8  # plot old image
9  plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
10 # plot new image
11 plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
```

3. **Linear Regression: Thực hiện thuật toán linear regresion trên tập data advertising.csv. Các bạn sẽ dựa trên 3 thông tin đầu vào là TV, Radio, Newspaper để dự đoán Sale**

Giới thiệu về tập data: Data có 200 samples (rows), gồm 4 cột thông tin Tv, Radio, Newspaper, và Sales. Đề bài yêu cầu dùng thông tin ở 3 cột đầu tiên (Tv, Radio, Newspaper) để dự đoán được cột cuối cùng (Sale) dùng linear regression model.

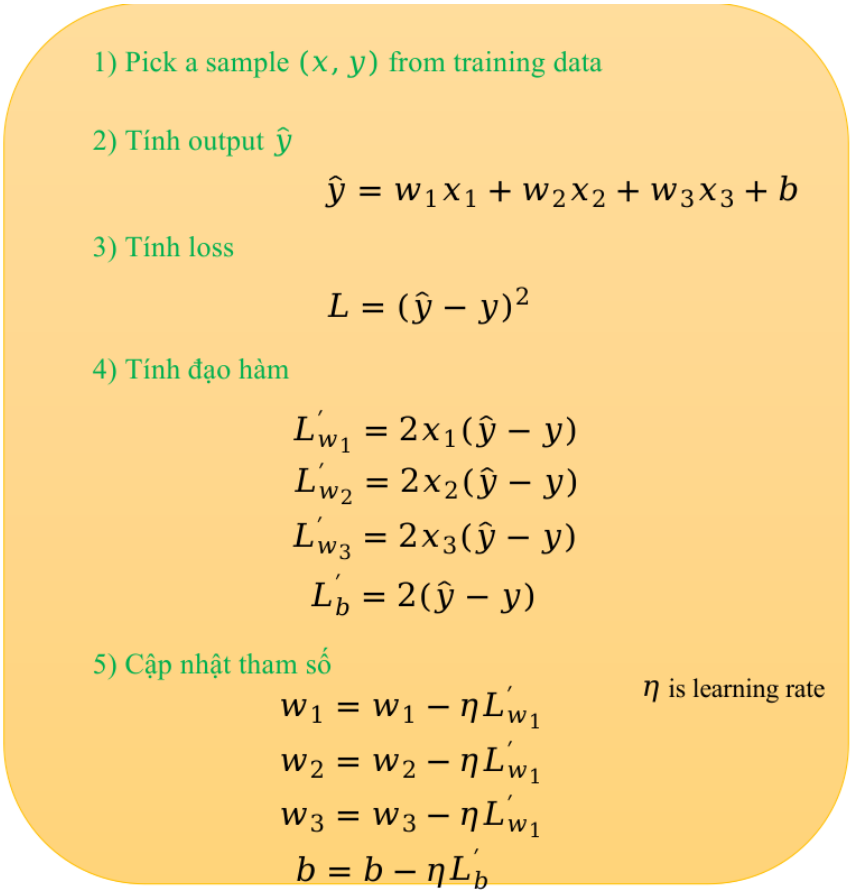
Linear Regression (theo yêu cầu của đề bài này) hình 3:

- **Input:** là một vector $\mathbf{x}$ chứa 3 element $\mathbf{x} = [x_1, x_2, x_3]$ tương ứng với các thông tin TV (x_1), Radio (x_2), Newspaper (x_3).
- **Parameters:** Gồm 4 parameter w_1, w_2, w_3, b để tạo thành một linear regression model
- **Output (kết quả dự đoán Sale):** $\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b$
- **Loss**: $\hat{y}$ là kết quả dự đoán Sale và y là giá trị thực tế. Hàm loss giúp thông báo model đang dự đoán sai bao nhiêu và nên cập nhật lại các parameter để giảm thiểu sai số này. Hàm loss ở đây là bình phương sai số của $\hat{y}$ và y. Loss $= (\hat{y} - y)^2$



Hình 3: Cách hoạt động của Linear Regression x_i là input, w_i và b là các parameter của model giúp dự đoán Sale (kếy quả dự đoán $\hat{y}$) và y là gía trị thực tế của sale

NOTE: Các bạn thực hiện theo các yêu cầu sau



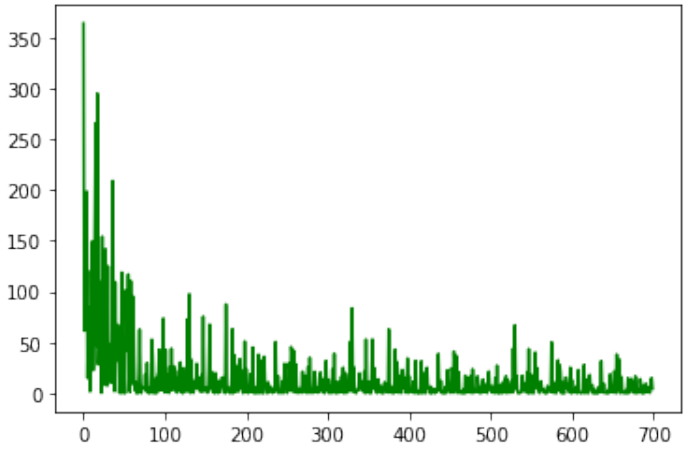
Hình 4: Các bước để thực hiện train linear regression model

- Mục tiêu chính là dựa và linear regression model đi tìm bộ parameter w_1, w_2, w_3, b để có thể predict $\hat{y}$ gần với y nhất cho từng sample (row).
- Data được đã được chuẩn bị theo template như trong link phần Linear Regression. Có 200 sample do đó **X** là ma trận (200x3) và **y** là vector có 200 element
- Các bạn viết function *implement_linear_regression(X, y, epoch_max, eta)* để thực hiện tìm bộ parameter cho linear regression model và trả về list loss ở các lần cập nhật model.
  - **Input:** Có 4 tham số **X** ma trận và **y** là vector được xử lý từ data ở trên. *epoch_max* là số lần lặp train toàn bộ data (1 epoch là cần 200 lần lặp để train một vòng toàn bộ data). *eta* (η) chính là learning rate là tốc độ học (*eta* càng lớn học càng nhanh nhưng có thể dẫn đến không ổn định và nếu quá lớn sẽ có thể không hội tụ, còn nếu quá nhỏ sẽ học rất chậm)
  - **Output**: trả về 2 thành phần. parameters là tuple (w_1, w_2, w_3, b) bộ tham số mà có thể dự đoán tốt trên tập data. Và list loss của mỗi lần cập nhật parameters
- **Các bước thực hiện theo hình 4**: Lưu ý là 1 epoch các bạn sẽ lặp hết 200 lần vì có 200 samples mỗi lần các bạn sẽ lấy 1 sample thực hiện theo các step bên dưới. Và mình cần lặp *epoch_max* lần epoch để model học tốt hơn. (Ví dụ *epoch_max=50*, thì chúng ta sẽ lặp 50 lần, mỗi lần lặp 200 lấy sample để train với data). Sau đây là các step trong 1 lần lấy 1 sample để train
  - **Step0**: Khởi tạo giá trị bất kỳ cho w_1, w_2, w_3, b và khởi tạo một list losses. Tạo 2 vòng lặp for vòng thứ nhất cho epoch_max và vòng 2 để lặp 200 samples. Sau đó mỗi sample

sẽ thực hiện các bước như bên dưới

- **Step1**: lấy một sample (1 row) từ ma trận **X** và (1 element) từ vector **y**. Sẽ lấy được x_1, x_2, x_3 và y (1 element)
- **Step2**: Dự đoán kết quả bằng các đi tìm $\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b$
- **Step3**: Tính loss bằng công thức $L = (\hat{y} - y)^2$. Append L vào list losses
- **Step4**: Tính đạo hàm loss theo từng parameters (đã có công thức sẵn như trong hình 4). Tính được $L'_{w1}, L'_{w2}, L'_{w3}, L'_b$
- **Step5**: Cập nhật parameters (đã có công thức sẵn như trong hình 4) tính được w_1, w_2, w_3, b mới
- Lặp đi lặp lại từ step 1 đến step 5 cho đến khi hết số lần lặp xong hết 2 vòng lặp ở step 0, thì trả về bộ parameters và losses. Sau đó các bạn vẽ losses (700 loss đầu) như hình 5 thấy loss giảm dần là kết quả đúng. Cách vẽ tham khảo link

```
parameters, losses = implement_linear_regression(X, y, epoch_max = 50, eta =
    0.01)

# in loss cho 700 sample    u
x_axis = list(range(700))
plt.plot(x_axis, losses[:700], color="g")
plt.show()
```



Hình 5: Loss ở 700 lần lặp đầu tiên