TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP. HỒ CHÍ MINH

VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO

-----🕮-----

BÀI TẬP E-LEARNING 06 (GROUP)

Lớp học phần: Trí tuệ nhân tạo (010412103305)

Giảng viên hướng dẫn**: Hoàng Đức Quý**

Nhóm thực hiện**: 3**

SVTH**: Quách Phú Thuận** MSSV: **2251120446**

SVTH: **Cao Bảo Gia Luật** MSSV: **22H1120131**

SVTH: **Lê Phạm Thanh Duy** MSSV: **22H1120123**

Thành phố Hồ Chí Minh, ngày 10 tháng 01 năm 2026

MỤC LỤC

I - Các phần logic chung dùng trong bài tập ... 1

1.1. Dữ liệu và ký hiệu vector hóa ... 1

1.2. Hàm dự đoán theo mô hình tuyến tính ... 2

1.3. Hàm loss và cost ... 2

1.4. Yêu cầu khởi tạo tham số ... 3

II - Câu 1: Áp dụng SGD để huấn luyện mô hình với MSE ... 3

2.1. Công thức toán học của SGD với MSE ... 3

2.2. Giải thích logic triển khai ... 4

III - Câu 2. Áp dụng Batch GD để huấn luyện mô hình với MSE ... 5

3.1. Công thức toán học của Batch GD với MSE ... 5

3.2. Giải thích logic triển khai ... 6

IV - Câu 3: Vẽ đồ thị loss của SGD và Batch GD trên cùng một hình và nhận xét ... 7

4.1. Cách tạo dữ liệu loss ... 7

4.2. Vẽ đồ thị và giải thích ... 7

4.3. Kết quả tham số học được ... 8

4.4. Nhận xét ... 8

V - Câu 4: Thay MSE bằng MAE rồi làm lại Câu 3 và nhận xét ... 9

5.1. Công thức và ý nghĩa của MAE ... 9

5.2. SGD với MAE và giải thích ... 9

5.3. Batch GD với MAE và giải thích ... 10

5.4. Vẽ đồ thị MAE SGD và Batch GD trên cùng một hình ... 11

5.5. Kết quả tham số học được ... 12

5.6. Nhận xét ... 12

VI. Link source code chi tiết bài tập ... 13

Phần I - Các phần logic chung dùng trong bài tập

1.1. Dữ liệu và ký hiệu vector hóa

Bộ dữ liệu có 200 mẫu với bốn cột **TV**, **Radio**, **Newspaper**, **Sales**. Ba cột đầu là đặc trưng và **Sales** là nhãn cần dự đoán. Ký hiệu số mẫu là m và số tham số sau khi vector hóa là $d = 4$. Quy ước:

$$x_1 = \text{TV}, x_2 = \text{Radio}, x_3 = \text{Newspaper}, y = \text{Sales}.$$

Vì đã gộp bias vào vector tham số nên đã dùng ký hiệu đúng theo mô hình vector hóa vectorization:

$$\vec{x}^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \end{bmatrix}, \vec{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Khi đó mô hình tuyến tính được viết gọn:

$$\hat{y}^{(i)} = f_{\vec{w}}(\vec{x}^{(i)}) = \vec{w} \cdot \vec{x}^{(i)}.$$

Nếu gom toàn bộ m mẫu vào ma trận đặc trưng $X \in \mathbb{R}^{m\times 4}$ và vector nhãn $y \in \mathbb{R}^m$ thì dự đoán cho toàn bộ dữ liệu là:

$$\hat{y} = X\vec{w}$$

```
df = pd.read_csv("./advertising.csv")

X_raw = df[["TV", "Radio", "Newspaper"]].to_numpy(dtype=float) # (m, 3)
y = df["Sales"].to_numpy(dtype=float) # (m,)

m = X_raw.shape[0]

X = np.c_[np.ones(m), X_raw] # (m, 4) với cột đầu là 1
d = X.shape[1] # d = 4
```

1.2. Hàm dự đoán theo mô hình tuyến tính

Theo ký hiệu $\hat{y} = X\vec{w}$, hàm dự đoán chỉ cần nhân ma trận. Hàm này nhận vào ma trận X và vector tham số $\vec{w}$, sau đó trả về $\hat{y}$ là vector dự đoán cho toàn bộ mẫu. Khi dùng ở SGD cho một mẫu, lấy một hàng $X[i]$ và nhân với $\vec{w}$ để tạo $\hat{y}^{(i)}$.

```
def predict(X: np.ndarray, w: np.ndarray) -> np.ndarray:
 return X @ w
```

1.3. Hàm loss và cost

Loss cho một mẫu thường được chọn là bình phương sai số có hệ số $\frac{1}{2}$.

$$L^{(i)} = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2.$$

Từ đó cost MSE trên toàn bộ dữ liệu là trung bình loss.

$$J_{\text{MSE}}(\vec{w}) = \frac{1}{m}\sum_{i=1}^{m} L^{(i)} = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2.$$

Sai số được ký hiệu là $e = \hat{y} - y$ và cost MSE tính bằng trung bình của $0.5 \cdot e^2$.

```
def mse_cost(X: np.ndarray, y: np.ndarray, w: np.ndarray) -> float:
 e = predict(X, w) - y
 return float(np.mean(0.5 * (e ** 2)))
```

Đề bài cũng yêu cầu thay MSE bằng MAE. Khi đó cost MAE là:

$$J_{\text{MAE}}(\vec{w}) = \frac{1}{m}\sum_{i=1}^{m} |\,\hat{y}^{(i)} - y^{(i)}\,|$$

MAE được tính bằng trung bình của trị tuyệt đối sai số.

```
def mae_cost(X: np.ndarray, y: np.ndarray, w: np.ndarray) -> float:
 e = predict(X, w) - y
 return float(np.mean(np.abs(e)))
```

1.4. Yêu cầu khởi tạo tham số

Đề bài yêu cầu learning rate $lr = 10^{-5}$, số epoch bằng 100, khởi tạo w ngẫu nhiên trong khoảng $(0,1)$ và bias bằng $b = 0$. Với cách vector hóa đang dùng thì điều kiện này tương đương với việc đặt $w_1, w_2, w_3 \sim U(0,1)$ và ép phần tử đầu của vector bằng 0, nghĩa là $w_0 = b = 0$. Phần khởi tạo có dạng tạo vector ngẫu nhiên rồi gán lại phần tử đầu.

```
rng = np.random.default_rng(seed)
w = rng.uniform(0.0, 1.0, size=d)
w[0] = 0.0
```

Phần II - Câu 1: Áp dụng SGD để huấn luyện mô hình với MSE

2.1. Công thức toán học của SGD với MSE

Với một mẫu thứ i, ký hiệu sai số là

$$e^{(i)} = \hat{y}^{(i)} - y^{(i)} = \vec{w} \cdot \vec{x}^{(i)} - y^{(i)}$$

Loss của mẫu thứ i là

$$L^{(i)} = \frac{1}{2}(e^{(i)})^2.$$

Lấy đạo hàm riêng của $L^{(i)}$ theo từng tham số w_j với $j = 0,1,2,3$ (trong đó $w_0 = b$):

$$\frac{\partial L^{(i)}}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\frac{1}{2}\left(e^{(i)}\right)^2\right] = e^{(i)}\frac{\partial e^{(i)}}{\partial w_j} = e^{(i)} \cdot x_j^{(i)}$$

Trong đó $x_0^{(i)} = 1$ (bias term). Quy tắc cập nhật SGD tại mẫu thứ i cho từng tham số là:

$$w_j \leftarrow w_j - lr \cdot e^{(i)} \cdot x_j^{(i)}, j = 0,1,2,3.$$

Vì $\vec{w}$ chứa cả b nên phép cập nhật này đồng thời cập nhật b, w_1, w_2, w_3 đúng ý nghĩa simultaneous update.

2.2. Giải thích logic triển khai

Hàm *train_sgd_mse* thực hiện đúng công thức trên: giữ đúng $lr = 10^{-5}$, số epoch bằng 100 và dùng seed để tái hiện kết quả.

```
def train_sgd_mse(X, y, lr=1e-5, epochs=100, seed=42):
 m, d = X.shape
 rng = np.random.default_rng(seed)

 w = rng.uniform(0.0, 1.0, size=d)
 w[0] = 0.0

 idx = np.arange(m)
 losses = []

 losses.append(mse_cost(X, y, w))

 for _ in range(epochs):
 rng.shuffle(idx)
 for i in idx:
 e_i = float(X[i] @ w - y[i])
 grad = e_i * X[i]
 w = w - lr * grad
 losses.append(mse_cost(X, y, w))

 return w, np.array(losses)
```

Hàm bắt đầu bằng việc lấy kích thước m, d của ma trận X để biết số tham số cần tối ưu. Sau đó hàm tạo bộ sinh số ngẫu nhiên rng để khởi tạo $\vec{w}$ sao cho ba trọng số w_1, w_2, w_3 nằm trong khoảng (0,1) và ép phần tử đầu w_0 bằng 0 để thỏa điều kiện $b = 0$ của đề bài.

Mảng *idx* chứa các chỉ số từ 0 đến $m - 1$ để biểu diễn thứ tự duyệt mẫu. Mỗi epoch, gọi *rng.shuffle(idx)* để xáo trộn thứ tự duyệt mẫu để giữ đúng tinh thần SGD vì gradient của mỗi bước dựa trên một mẫu riêng và thứ tự ngẫu nhiên giúp giảm nguy cơ bị lệch theo thứ tự dữ liệu.

Trong vòng lặp duyệt mẫu, tính sai số mẫu thứ *i* bằng *e_i = X[i] @ w - y[i]*. Biểu thức *X[i] @ w* chính là $\hat{y}^{(i)} = \vec{w} \cdot \vec{x}^{(i)}$ do X[i] là hàng thứ *i* của ma trận *X* và đã bao gồm phần tử đầu bằng 1. Sau đó tính gradient *grad = e_i * X[i]* tương ứng với công thức $\frac{\partial L^{(i)}}{\partial \vec{w}} = e^{(i)} \vec{x}^{(i)}$. Cuối cùng cập nhật *w = w - lr * grad* đúng theo quy tắc SGD. Sau khi duyệt hết m mẫu, tính lại cost MSE trên toàn bộ tập bằng *mse_cost(X, y, w)* và lưu vào *losses*. Việc lưu loss theo epoch giúp phục vụ Câu 3 khi vẽ đường cong hội tụ.

```
losses.append(mse_cost(X, y, w))  # Loss ban đầu

    for _ in range(epochs):
        rng.shuffle(idx)  # Shuffle để duyệt mẫu ngẫu nhiên mỗi epoch
        for i in idx:
            e_i = float(X[i] @ w - y[i])  # e^{(i)} = ŷ^{(i)} - y^{(i)}
            grad = e_i * X[i]  # ∂L^{(i)}/∂w = e^{(i)} · x^{(i)}
            w = w - lr * grad  # Cập nhật w_j ← w_j - lr · grad_j
        losses.append(mse_cost(X, y, w))  # Loss sau epoch
```

Phần III - Câu 2. Áp dụng Batch GD để huấn luyện mô hình với MSE

3.1. Công thức toán học của Batch GD với MSE

Với toàn bộ dữ liệu, đã viết sai số vector

$$e = \hat{\mathbf{y}} - y = X\vec{w} - y$$

Cost MSE là

$$J_{\text{MSE}}(\vec{w}) = \frac{1}{2m}\sum_{i=1}^{m}(e^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Đạo hàm riêng của J_{MSE} theo từng tham số w_j với $j = 0,1,2,3$:

$$\frac{\partial J_{\text{MSE}}}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left[\frac{1}{2}\left(e^{(i)}\right)^2\right] = \frac{1}{m}\sum_{i=1}^{m} e^{(i)} \cdot x_j^{(i)}$$

Batch GD cập nhật theo công thức cho từng tham số:

$$w_j \leftarrow w_j - lr \cdot \frac{1}{m}\sum_{i=1}^{m} e^{(i)} \cdot x_j^{(i)}, j = 0,1,2,3$$

3.2. Giải thích logic triển khai

Hàm ***train_bgd_mse*** thực hiện đúng công thức batch phía trên.

```
def train_bgd_mse(X, y, lr=1e-5, epochs=100, seed=42):
 m, d = X.shape
 rng = np.random.default_rng(seed)

 w = rng.uniform(0.0, 1.0, size=d)
 w[0] = 0.0

 losses = []
 losses.append(mse_cost(X, y, w))

 for _ in range(epochs):
 e = X @ w - y
 grad = (X.T @ e) / m
 w = w - lr * grad
 losses.append(mse_cost(X, y, w))

 return w, np.array(losses)
```

Phần khởi tạo tham số giống Câu 1 để đảm bảo cùng điều kiện đầu vào cho hai phương pháp tối ưu, giúp so sánh trực quan hơn ở Câu 3. Ở mỗi epoch, Batch GD tính sai số vector *e = X @ w - y*, trong đó *X @ w* là ŷ, rồi tính gradient bằng *grad = (X.T @ e) / m* tương ứng với công thức $\frac{1}{m}\sum_{i=1}^{m} e^{(i)} \cdot x_j^{(i)}$ cho mỗi tham số w_j. Sau đó cập nhật w = w - lr * grad đúng theo công thức Batch GD.

Khác biệt cốt lõi so với SGD là Batch GD dùng gradient tính từ toàn bộ *m* mẫu cho mỗi lần cập nhật, do đó mỗi epoch chỉ có một lần cập nhật tham số nhưng gradient ổn định hơn.

```
losses.append(mse_cost(X, y, w))  # Loss ban đầu

    for _ in range(epochs):
        e = X @ w - y  # e = Xw - y, vector sai số toàn bộ mẫu
        grad = (X.T @ e) / m  # ∂J/∂w_j = (1/m) Σ e^{(i)} · x_j^{(i)}
        w = w - lr * grad  # Cập nhật w_j ← w_j - lr · grad_j
        losses.append(mse_cost(X, y, w))  # Loss sau epoch
```

Phần IV - Câu 3: Vẽ đồ thị loss của SGD và Batch GD trên cùng một hình và nhận xét

4.1. Cách tạo dữ liệu loss

Mỗi hàm train trả về *losses* là mảng loss được lưu theo epoch. Sau đó gọi hai hàm huấn luyện và lưu lại kết quả.

```
lr = 1e-5
epochs = 100
seed = 42

w_sgd_mse, loss_sgd_mse = train_sgd_mse(X, y, lr=lr, epochs=epochs, seed=seed)
w_bgd_mse, loss_bgd_mse = train_bgd_mse(X, y, lr=lr, epochs=epochs, seed=seed)
```

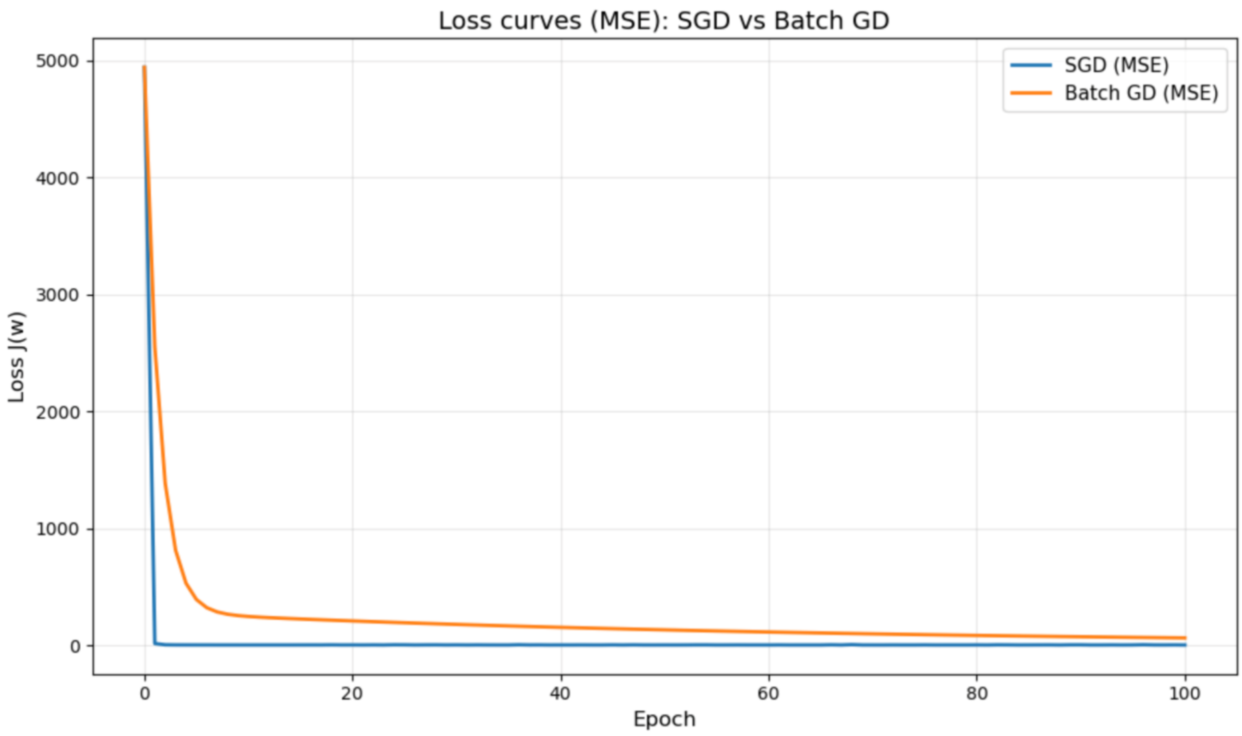
4.2. Vẽ đồ thị và giải thích

Vẽ hai đường loss trên cùng một hình bằng matplotlib, trong đó trục hoành là epoch và trục tung là giá trị cost $J(\vec{w})$.

```
plt.figure(figsize=(10, 6))
plt.plot(np.arange(len(loss_sgd_mse)), loss_sgd_mse, label="SGD (MSE)", linewidth=2)
plt.plot(np.arange(len(loss_bgd_mse)), loss_bgd_mse, label="Batch GD (MSE)", linewidth=2)
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss J(w)", fontsize=12)
plt.title("Loss curves (MSE): SGD vs Batch GD", fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Tạo một hình mới và lần lượt vẽ *loss_sgd_mse* và *loss_bgd_mse*.

Hàm *np.arange(len(loss_sgd_mse))* tạo ra chỉ số epoch tương ứng với số phần tử trong mảng loss, và điều này giúp hai đường được vẽ đúng trục thời gian. Các nhãn trục và tiêu đề được đặt rõ ràng để người xem hiểu đây là so sánh SGD và Batch GD dưới hàm mất mát MSE.



4.3. Kết quả tham số học được

```
SGD (MSE)
b  = 0.11967383063472119
w1 = 0.06516511957164679
w2 = 0.1616711041137433
w3 = 0.027222512582367613

Batch GD (MSE)
b  = -0.0070478126467513925
w1 = -0.013710156164045239
w2 = 0.5354607694230217
w3 = 0.23933233760833886
```

4.4. Nhận xét

Với cùng learning rate rất nhỏ 10^{-5}, cả hai phương pháp đều có xu hướng làm giảm cost theo epoch nếu mô hình hội tụ. Đường của Batch GD thường mượt hơn vì mỗi bước cập nhật dùng gradient trung bình trên toàn bộ dữ liệu. Đường của SGD thường dao động hơn vì mỗi lần cập nhật dựa trên một mẫu, do đó nhiễu gradient lớn hơn, tuy nhiên SGD có thể giảm nhanh ở giai đoạn đầu do cập nhật liên tục trong một epoch. Nhận xét cuối cùng cần dựa trên chính đồ thị thu được, vì độ dao động và tốc độ giảm phụ thuộc vào dữ liệu và cách lưu loss theo epoch.

Phần V - Câu 4: Thay MSE bằng MAE rồi làm lại Câu 3 và nhận xét

5.1. Công thức và ý nghĩa của MAE

Khi thay MSE bằng MAE, cost trở thành

$$J_{\mathrm{MAE}}(\vec{w}) = \frac{1}{m}\sum_{i=1}^{m} \mid \hat{y}^{(i)} - y^{(i)} \mid$$

Đạo hàm của trị tuyệt đối không xác định tại $e^{(i)} = 0$, vì vậy trong tối ưu hóa đã dùng khái niệm subgradient. Một lựa chọn phổ biến là dùng hàm dấu

$$\mathrm{sign}(e) = \begin{cases} 1, & e > 0 \\ 0, & e = 0 \\ -1, & e < 0 \end{cases}$$

và khi đó subgradient cho một mẫu theo tham số w_j có thể viết

$$\frac{\partial}{\partial w_j} \mid e^{(i)} \mid = \mathrm{sign}(e^{(i)}) \cdot x_j^{(i)}, j = 0,1,2,3$$

Với toàn bộ dữ liệu, đạo hàm riêng của J_{MAE} theo tham số w_j là:

$$\frac{\partial J_{\mathrm{MAE}}}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m} \mathrm{sign}\,(e^{(i)}) \cdot x_j^{(i)}, j = 0,1,2,3$$

5.2. SGD với MAE và giải thích

SGD với MAE dùng *np.sign(e_i)* để thay cho $e^{(i)}$ trong gradient của MSE.

```python
def train_sgd_mae(X, y, lr=1e-5, epochs=100, seed=42):
 m, d = X.shape
 rng = np.random.default_rng(seed)

 w = rng.uniform(0.0, 1.0, size=d)
 w[0] = 0.0

 idx = np.arange(m)
 losses = []
 losses.append(mae_cost(X, y, w))

 for _ in range(epochs):
 rng.shuffle(idx)
 for i in idx:
 e_i = float(X[i] @ w - y[i])
 grad = np.sign(e_i) * X[i]
 w = w - lr * grad
 losses.append(mae_cost(X, y, w))

 return w, np.array(losses)
```

Phần tính e_i giống Câu 1 vì sai số vẫn là $\hat{y}^{(i)} - y^{(i)}$. Điểm thay đổi nằm ở gradient, thay vì nhân với $e^{(i)}$ như MSE thì MAE nhân với sign(e_i) để phản ánh hướng tăng giảm của trị tuyệt đối. Vì np.sign(0)=0, khi một mẫu được dự đoán đúng hoàn toàn thì subgradient bằng không và mẫu đó không tạo lực cập nhật tham số.

5.3. Batch GD với MAE và giải thích

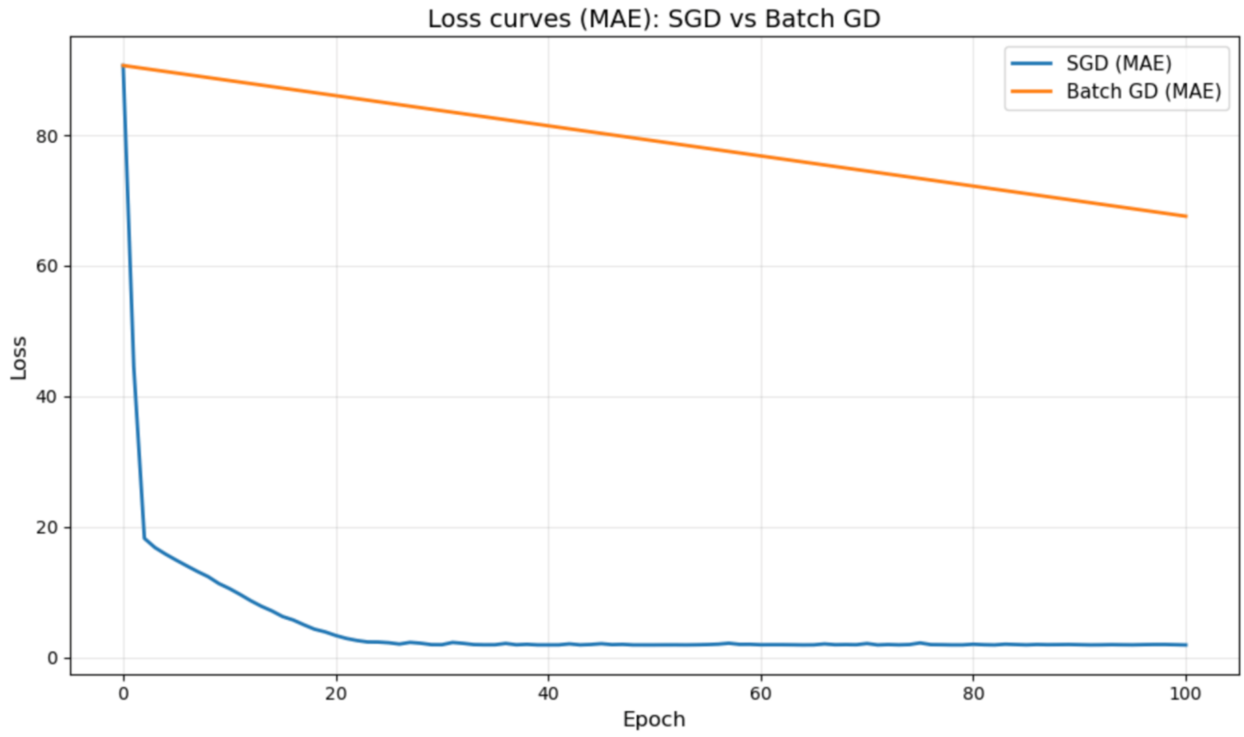
Batch GD với MAE tính *sign* trên toàn bộ vector sai số, sau đó tính tổng $\frac{1}{m}\sum_{i=1}^{m} \text{sign}\,(e^{(i)}) \cdot x_j^{(i)}$ cho mỗi tham số w_j.

```python
def train_bgd_mae(X, y, lr=1e-5, epochs=100, seed=42):
 m, d = X.shape
 rng = np.random.default_rng(seed)

 w = rng.uniform(0.0, 1.0, size=d)
 w[0] = 0.0

 losses = []
 losses.append(mae_cost(X, y, w))

 for _ in range(epochs):
 e = X @ w - y
 grad = (X.T @ np.sign(e)) / m
 w = w - lr * grad
 losses.append(mae_cost(X, y, w))

 return w, np.array(losses)
```

5.4. Vẽ đồ thị MAE SGD và Batch GD trên cùng một hình

Vẽ loss MAE tương tự Câu 3.

```python
plt.figure(figsize=(10, 6))
plt.plot(np.arange(len(loss_sgd_mae)), loss_sgd_mae, label="SGD (MAE)", linewidth=2)
plt.plot(np.arange(len(loss_bgd_mae)), loss_bgd_mae, label="Batch GD (MAE)", linewidth=2)
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title("Loss curves (MAE): SGD vs Batch GD", fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



5.5. Kết quả tham số học được

```
SGD (MAE)
b  = 0.026359999999999096
w1 = 0.06469043975205598
w2 = 0.17343991991137472
w3 = 0.027462029059365158

Batch GD (MAE)
b  = -0.0010000000000000002
w1 = 0.2918359397520526
w2 = 0.8353339199113818
w3 = 0.666814029059359
```

5.6. Nhận xét

Khi dùng MAE, gradient theo nghĩa subgradient chỉ phụ thuộc vào dấu của sai số, vì vậy biên độ cập nhật không tăng khi sai số lớn như MSE. Điều này làm MAE thường bền hơn với ngoại lệ, nhưng tốc độ hội tụ và hình dạng đường loss có thể khác so với MSE. Batch GD với MAE thường cho đường loss ổn định hơn, còn SGD với MAE có thể dao động do cập nhật theo từng mẫu. Nhận xét cuối cùng cần dựa trên đồ thị thực tế vì mức độ dao động và tốc độ giảm phụ thuộc dữ liệu và điều kiện khởi tạo.

Phần VI - Link source code chi tiết bài tập

Github: https://github.com/DevLoopsX/UTH-AI-assignments/tree/main/Solutions/BT06-Linear_Regression