Mini Project: Using Autograd to Learn Parameters
(10 points)

Overview

This project will help you practice using `torch.autograd` for automatic differentiation in PyTorch. You will write and run code in a Jupyter Notebook (`.ipynb`), then use # comments inside the notebook for short explanations, observations, and reflection.

Starter Code

Use the provided starter notebook that creates a simple linear model:

$$z = w_0 x_0 + w_1 x_1$$

with initial parameters `requires_grad=True`. It computes a loss like:

```
loss = z.sum()
```

and calls `loss.backward()` to populate gradients.

Tasks (10 points total)

T1. Modify the code to use a real dataset (4 pts).

- Replace the toy x_0, x_1 with a small custom dataset:

$$X = \{(1,2),(2,3),(3,4),(4,5)\}, \quad y = [5,\ 7,\ 9,\ 11].$$

- Compute predictions $\hat{y} = w_0 x_0 + w_1 x_1$.
- Use Mean Squared Error (MSE) loss:

```
loss = ((y_pred - y) ** 2).mean()
```

T2. Perform gradient descent updates (3 pts).

- After `loss.backward()`, update w_0, w_1 manually:

$$w := w - \eta \cdot w.grad$$

  (pick $\eta = 0.01$).
- Use `with torch.no_grad():` to disable autograd during the update step.

- Run for 50 epochs, print loss every 10 epochs.

T3. Plot and comment (2 pts).

- Plot the loss vs. epoch (line plot).
- In comment cells, briefly explain whether the loss decreases as expected.

T4. Reflection (1 pt).

- Write 3–4 comment lines at the bottom of the notebook:
  a) What did you learn about autograd?
  b) What step was confusing or new to you?
  c) How might you improve the code (e.g., optimizer, larger dataset)?

Deliverable

Submit a single Jupyter Notebook file `autograd_project.ipynb` that includes:

- Code cells with your solution and training loop,
- Plot of loss curve,
- # comments with your answers, observations, and reflection.

Grading Rubric (10 pts)

- Correct dataset integration and loss definition: **4 pts**
- Manual parameter updates and convergence: **3 pts**
- Loss curve plot and short commentary: **2 pts**
- Reflection comments: **1 pt**