OPPE Mock - WGAN on MNIST

Overview

Implement a **Wasserstein GAN (WGAN)** to generate MNIST digits using **PyTorch**. The objective is to demonstrate correct adversarial training under the Wasserstein framework (with either gradient penalty or weight clipping), produce intermediate diagnostic quantities, and show stable sample quality over time.

- **Dataset:** `torchvision.datasets.MNIST` (train split for training; test split only for sanity checks).
- **Image scale:** All tensors must be normalized to $[-1, 1]$ to match a `tanh` generator output.
- **Deliverables:** Inline prints/plots for the required intermediate results (dataset checks, losses, critic scores, GP/clipping stats, fixed-noise samples).
- **Duration: 2 hours**. Timebox your work; partial credit is awarded for correct intermediate results and clear logging even if training is not fully converged.
- **Randomness:** Fix a seed at the top of the notebook (e.g., `torch.manual_seed(42)`) and reuse a fixed latent batch $\mathbf{z}_{\text{fixed}}$ for periodic samples.

Colab Setup (Required)

1. In `Runtime` → `Change runtime type`, select **GPU**. At the top cell, print:
   - `torch.__version__`, `torch.cuda.is_available()`, and device name (`torch.cuda.get_device_na`
   - The fixed seed: `torch.manual_seed(42)` and, if using CUDA, `torch.cuda.manual_seed_all(42)`.
2. Allowed libraries: `torch`, `torchvision`, `numpy`, `matplotlib`, `tqdm`. *No other ML libraries.*
3. **Data transforms (example):**

```
transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),                  # [0,1]
    torchvision.transforms.Normalize(mean=(0.5,), std=(0.5,))  # -> [-1,1]
])
```

4. **Determinism (optional):** For reproducibility you may add

```
torch.backends.cudnn.benchmark = False
torch.use_deterministic_algorithms(False)  # set True only if your ops support it
```

   If deterministic mode raises errors for some ops, revert to `False`.

5. **Fixed noise:** Create and store once:

```
dz = 128  # or 64/100 per spec
z_fixed = torch.randn(64, dz, device='cuda')
```

Problem Statement

Train a **WGAN** on MNIST using PyTorch. You may choose either WGAN-GP (preferred) or WGAN with weight clipping (acceptable). Your implementation must obey the Wasserstein training objective and log the requested intermediate values.

Formal Objective (for clarity)

Let D_ψ denote the critic (a 1-Lipschitz function) and G_θ the generator mapping $z \sim p_z$ to images. The target is to minimize the Wasserstein-1 distance $W_1(p_{\text{data}}, p_G)$. In practice:

$$\max_{\psi \in \mathcal{L}} \ \mathbb{E}_{x \sim p_{\text{data}}}[D_\psi(x)] \ - \ \mathbb{E}_{z \sim p_z}[D_\psi(G_\theta(z))], \qquad \min_\theta \ -\mathbb{E}_{z \sim p_z}[D_\psi(G_\theta(z))],$$

where $\mathcal{L}$ enforces 1-Lipschitzness of D.

Preferred: WGAN-GP ($\lambda = 10$). Enforce 1-Lipschitz via a gradient penalty on random interpolates $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$:

$$L_D \ = \ \mathbb{E}_{\tilde{x} \sim p_G}[D(\tilde{x})] \ - \ \mathbb{E}_{x \sim p_{\text{data}}}[D(x)] \ + \ \lambda \mathbb{E}_{\hat{x}} \left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1 \right)^2, \quad L_G \ = \ -\mathbb{E}_z[D(G(z))].$$

Acceptable alternative: WGAN with weight clipping. Enforce 1-Lipschitz by clipping critic weights $w \leftarrow \text{clip}(w, -c, c)$ each critic step (e.g., $c = \pm 0.01$). Then

$$L_D \ = \ \mathbb{E}_{\tilde{x} \sim p_G}[D(\tilde{x})] \ - \ \mathbb{E}_{x \sim p_{\text{data}}}[D(x)], \qquad L_G \ = \ -\mathbb{E}_z[D(G(z))].$$

If you choose clipping, state it clearly in your notebook header and report clipping utilization statistics as requested.

Model & Training Requirements

- **Generator (G):** Input $z \sim \mathcal{N}(0, I)$ of dimension $d_z \in \{64, 100, 128\}$. Output a single-channel image tensor $1 \times 28 \times 28$ with a final `tanh` layer so outputs are in $[-1, 1]$.
- **Critic (D):** Outputs a *raw scalar score* (no sigmoid/softmax). The absolute value is not meaningful; only $\mathbb{E}[D(x)] - \mathbb{E}[D(G(z))]$ is.
- **Optimizers:**
  - **WGAN-GP (recommended):** `Adam` with $\text{lr} = 2 \times 10^{-4}$, $\beta_1 = 0.0$, $\beta_2 = 0.9$.
  - **WGAN-Clip:** `RMSprop` or `Adam` acceptable; if clipping, use small lr (e.g., 5×10^{-5} to 1×10^{-4}).
- **Training loop:** Use $n_{\text{critic}} = 5$ critic updates per generator update. Batch size ≥ 64. Shuffle data every epoch.
- **Transforms:** Ensure normalization to $[-1, 1]$. A typical pipeline is `ToTensor()` then `Normalize((0.5,), (0.5,))`.
- **Initialization:** Kaiming/He or Xavier for conv/linear layers are acceptable. Ensure the final `tanh` layer in G is not followed by normalization.
- **Gradient Penalty (if GP):** Draw $\epsilon \sim \mathcal{U}[0, 1]$ per sample and compute $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$. Backpropagate $\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1 \right)^2$ with $\lambda = 10$.
- **Clipping (if Clip):** After each critic update, clip *all* critic parameters into $[-c, c]$ (e.g., $c = 0.01$). Log the fraction of weights that hit the boundary.
- **Logging (minimum):**
  - Running means of critic loss L_D, generator loss L_G.
  - Batch means $\mathbb{E}[D(x)]$ and $\mathbb{E}[D(G(z))]$; their difference is an estimator of the Wasserstein gap.

- If GP: running mean of the gradient penalty value; optional histogram of $\|\nabla_{\hat{x}} D(\hat{x})\|_2$.
- Save/display generated grids from the fixed noise $\mathbf{z}_{\text{fixed}}$ at regular intervals (e.g., iteration 0, 2,000, 10,000, final).

- **Recommended schedule for the 2 hr window (guideline only):**
  - 0–15 min: Colab + GPU check, dataset transform, model stubs, seed, $\mathbf{z}_{\text{fixed}}$.
  - 15–35 min: Implement critic step (with GP or clipping) and generator step; test one mini-batch.
  - 35–90 min: Train, log curves and intermediate quantities; checkpoint sample grids.
  - 90–120 min: Compute/print the requested numerical answers; tidy plots/labels.

Implementation-Agnostic Numerical Questions for WGAN / WGAN-GP

Answer with numbers only (add units if any). Round to 3 decimals unless stated. Where randomness is involved, **fix your seed** and show the code that computes each value. Examples below are *illustrative*; your answers must come from *your* run.

E1. Dataset Size and Imbalance. Let c_k be the number of training samples for class $k \in \{0, \dots, 9\}$ in your loaded dataset. Report:

$$\text{Total} = \sum_{k=0}^{9} c_k, \quad \Delta = \max_k c_k - \min_k c_k, \quad \text{Imbalance} = \frac{\Delta}{\frac{1}{10}\sum_k c_k} \times 100\ \%.$$

Output: (Total, Δ, Imbalance%).
Example (worked): If $c = \{6000, 5900, 6050, 5950, 6100, 5800, 6150, 6000, 5900, 6150\}$ then Total $= 60000$, $\Delta = 6150 - 5800 = 350$, Imbalance $= 350/6000 \times 100 = \mathbf{5.833}\%$.

E2. Normalization Check (first training batch). Let $X \in \mathbb{R}^{B\times C\times H\times W}$ be the first *post-transform* training batch (e.g., C=1, H=W=28). Report:

$$\mu = \text{mean}(X),\ \sigma = \text{std}(X),\ m = \min(X),\ M = \max(X),\ p = \frac{\#\{\,|X| \le 0.5\,\}}{\#X}.$$

Output: (μ, σ, m, M, p).
Example (worked): If $\mu = -0.037$, $\sigma = 0.821$, $m = -1.000$, $M = 1.000$, and 62% of pixels satisfy $|x| \le 0.5$, output ($\mathbf{-0.037}$, $\mathbf{0.821}$, $\mathbf{-1.000}$, $\mathbf{1.000}$, $\mathbf{0.620}$).

E3. Model Size and Memory. Let n_G and n_D be the *trainable* parameter counts of your generator and critic. Report the ratio and float32 memory:

$$r = \frac{n_G}{n_D}, \qquad \text{MB} = \frac{4\,(n_G + n_D)}{10^6}.$$

Output: (n_G, n_D, r, MB).
Example (worked): If n_G=973,313 and n_D=94,721, then $r = 10.279$, MB $= 4.272$.

E4. One Training Cycle Deltas ($n_{\textbf{critic}}$=5). Run exactly 5 critic updates followed by 1 generator update on your current batch. Let $L_D^{(i)}$ be the critic loss at critic step i, and θ_D, θ_G be concatenated parameter vectors (same order before/after updates). Report:

$$\overline{L_D} = \frac{1}{5}\sum_{i=1}^{5} L_D^{(i)}, \quad \Delta_D = \|\theta_D^{\text{after}} - \theta_D^{\text{before}}\|_2, \quad \Delta_G = \|\theta_G^{\text{after}} - \theta_G^{\text{before}}\|_2.$$

Output: $(\overline{L_D},\ \Delta_D,\ \Delta_G)$.
Example (worked): If $L_D = [0.62, 0.58, 0.55, 0.60, 0.57]$, then $\overline{L_D} = 0.584$. If $\Delta_D = 0.732$, $\Delta_G = 0.541$, output $(\mathbf{0.584},\ \mathbf{0.732},\ \mathbf{0.541})$.

E5. Empirical Wasserstein Gap (current batch). Let $D(x)$ be your critic's scalar output (no sigmoid). Compute the batch means on the *same* step:

$$\mathbb{E}[D(x_{\text{real}})], \qquad \mathbb{E}[D(G(z))], \qquad \widehat{W}_1 = \mathbb{E}[D(x_{\text{real}})] - \mathbb{E}[D(G(z))].$$

Output: $(\mathbb{E}[D(x)],\ \mathbb{E}[D(G(z))],\ \widehat{W}_1)$.
Example (worked): If 0.820 and -0.430, then $\widehat{W}_1 = 1.250$; output $(\mathbf{0.820},\ -\mathbf{0.430},\ \mathbf{1.250})$.

E6. Gradient Penalty Statistics (for WGAN-GP). Form interpolates $\hat{x} = \epsilon x + (1 - \epsilon)\tilde{x}$ with $\epsilon \sim \mathcal{U}[0, 1]$, x real, $\tilde{x}$ generated. For N=256 interpolates at the end of an epoch, compute

$$g_i = \|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2, \qquad \overline{g} = \frac{1}{N}\sum_i g_i, \qquad s_g = \text{std}(g), \qquad \overline{\text{GP}} = \lambda \cdot \frac{1}{N}\sum_i (g_i - 1)^2.$$

Output: $(\overline{g},\ s_g,\ \overline{\text{GP}})$.
Example (worked): With $g = \{0.8, 1.2, 1.0, 0.6\}$ and $\lambda = 10$: $\overline{g} = 0.900$, $s_g = 0.224$, $\overline{\text{GP}} = 0.600$.

E7. Clipping Utilization (for WGAN with weight clipping). Let w be the vector of all critic weights and $c>0$ the clip bound used (i.e., $w \leftarrow \text{clip}(w, -c, c)$). Report:

$$f = \frac{\#\{|w| = c\}}{\#w}, \qquad \overline{|w|} = \text{mean}(|w|), \qquad r_{\max} = \frac{\max|w|}{c}.$$

Output: $(f,\ \overline{|w|},\ r_{\max})$.
Example (worked): If $c = 0.01$, 18% of weights hit $\pm c$, $\overline{|w|} = 0.0065$, and $\max|w| = c$, then output $(\mathbf{0.180},\ \mathbf{0.0065},\ \mathbf{1.000})$.

E8. Fixed-Noise Drift (image-space). Keep a fixed z_{fixed} and save generated tensors (before any de-normalization) at iterations $t_1 < t_2 < t_3$; let G_t be the full stack (e.g., an 8×8 grid $\Rightarrow$ 64 images). Compute mean absolute per-pixel differences:

$$d_{12} = \text{mean}(|G_{t_2} - G_{t_1}|), \qquad d_{23} = \text{mean}(|G_{t_3} - G_{t_2}|).$$

Output: $(d_{12},\ d_{23})$.
Example (worked): For 64 images of 28×28, pixel count $= 64 \cdot 28 \cdot 28 = 50176$. If total absolute diffs are $S_{12} = 12544$, $S_{23} = 10035$, then $d_{12} = 12544/50176 = \mathbf{0.250}$, $d_{23} = 10035/50176 = \mathbf{0.200}$.

E9. Latent Traversal Correlation. Pick one latent index i, sweep z_i over 10 evenly spaced values in $[-2, 2]$ with other dims fixed. For each image j, compute its mean pixel intensity m_j (same scale as your training tensors). Report the Pearson correlation:

$$r = \frac{\sum_j (z_j - \overline{z})(m_j - \overline{m})}{\sqrt{\sum_j (z_j - \overline{z})^2}\sqrt{\sum_j (m_j - \overline{m})^2}}.$$

Output: r.
Example (worked): If m_j increases roughly linearly with z_j, you might get $r = \mathbf{0.970}$.

E10. Critic Ordering Accuracy (pairwise, no classifier). On a held-out mini-batch, form K pairs $\{(x_{\text{real}}^{(k)}, G(z)^{(k)})\}$ and evaluate your critic once. Report the fraction for which the critic ranks real above fake:

$$a = \frac{1}{K} \sum_{k=1}^{K} \mathbf{1}\{D(x_{\text{real}}^{(k)}) > D(G(z)^{(k)})\}.$$

Output: $a \in [0, 1]$.
Example (worked): If 52 of 64 pairs satisfy the inequality, $a = \mathbf{0.8125}$.

Tip. If you use WGAN-GP, answer E6 and put `N/A` for E7; if you use weight clipping, answer E7 and put `N/A` for E6. Use the same tensor scale (e.g., $[-1, 1]$) consistently in E8–E9.