Homework 2: Conditional WGAN-GP on MNIST

CSC 8851 – Deep Learning - Fall 2025

Learning Objectives

- Implement and train a class-conditional Wasserstein GAN with Gradient Penalty (WGAN-GP) on MNIST.
- Use a projection discriminator and a label-conditional generator.
- Plot the critic scores, gradient penalty, and losses.
- Produce high-quality conditional samples, and explore latent interpolations and truncation.

Assignment Tasks (100 pts)

Part 1: Model and Losses (40 pts)

- (20 pts) Implement a conditional generator $G(z, y)$ and a projection critic $D(x, y)$ for MNIST resized to 32×32 resolution.
  - **Generator $G(z, y)$ (MNIST 32×32).** Label embedding $e_g(y) \in \mathbb{R}^{32}$ concatenated with noise $z \in \mathbb{R}^{64}$, then:
    - FC: $(64{+}32) \to 4 \times 4 \times 128$; reshape to $(128, 4, 4)$.
    - Upsample $\times 2$ (nearest) $\to$ Conv($128 \to 64$, 3×3, $s{=}1$, $p{=}1$) $\to$ BN $\to$ ReLU $(4 \to 8)$
    - Upsample $\times 2$ $\to$ Conv($64 \to 32$, 3×3) $\to$ BN $\to$ ReLU $(8 \to 16)$
    - Upsample $\times 2$ $\to$ Conv($32 \to 16$, 3×3) $\to$ BN $\to$ ReLU $(16 \to 32)$
    - Conv($16 \to 1$, 3×3) $\to$ Tanh (output in $[-1, 1]$).

    Notes: BatchNorm (BN) only in G; activations ReLU; initialize Conv/Linear with Kaiming (He) normal; no spectral/weight norm needed for this assignment.
  - **Critic $D(x, y)$ with projection.** Input $x \in \mathbb{R}^{1\times32\times32}$ with pixels normalized to $[-1, 1]$. Trunk (before projection) produces $h \in \mathbb{R}^{B\times C\times4\times4}$ with C=`proj_dim`=128:
    - Conv($1 \to 32$, 4×4, $s{=}2$, $p{=}1$) $\to$ LeakyReLU(0.2) $(32 \to 16)$
    - Conv($32 \to 64$, 4×4, $s{=}2$, $p{=}1$) $\to$ LeakyReLU(0.2) $(16 \to 8)$
    - Conv($64 \to 128$, 4×4, $s{=}2$, $p{=}1$) $\to$ LeakyReLU(0.2) $(8 \to 4)$
    - Global sum pool over 4×4: $f(x) = \sum_{u,v} h_{:,:,u,v} \in \mathbb{R}^{B\times C}$.
    - **Projection:** learn a class embedding $e(y) \in \mathbb{R}^C$ and a linear head $w \in \mathbb{R}^C$. The final score is
    $$D(x, y) \; = \; w^\top f(x) \; + \; \langle f(x), e(y) \rangle \; \in \; \mathbb{R}^B.$$

    Notes: Do *not* use sigmoid in D (WGAN-GP needs raw scores). Keep critic norm-free.
- (10 pts) Wasserstein loss with Gradient Penalty:
$$\mathcal{L}_C = -\Big(\mathbb{E}[D(x, y)] - \mathbb{E}[D(G(z, y), y)]\Big) + \lambda_{\text{gp}} \cdot \underbrace{\mathbb{E}\Big[(\|\nabla_{\hat{x}} D(\hat{x}, y)\|_2 - 1)^2\Big]}_{\text{GP}}, \tag{1}$$
$$\mathcal{L}_G = -\mathbb{E}[D(G(z, y), y)]. \tag{2}$$
  where $\hat{x} = \epsilon x + (1 - \epsilon) G(z, y)$ with $\epsilon \sim \text{Uniform}(0, 1)$.

- (10 pts) Generator EMA (Exponential Moving Average): Maintain a second, non-trainable copy of the generator G_{ema} whose weights track G via an exponential moving average. Generating using G_{ema} typically yields cleaner, less noisy images.
  - **Init.** At the start of training, set $G_{\text{ema}} \leftarrow \text{deepcopy}(G)$ and make all of its parameters non-trainable (`requires_grad = False`).
  - **Update rule** (after each `opt_G.step()`). For each floating-point entry $\theta \in$ `state_dict`(G) with EMA counterpart θ_{ema}:

$$\theta_{\text{ema}} \leftarrow \tau\,\theta_{\text{ema}} + (1-\tau)\,\theta, \quad \text{with decay } \tau \in [0,1) \text{ (e.g., } \tau = 0.999).$$

  Non-floating entries (e.g., integer counters) should be copied directly rather than averaged.

  Example pseudocode for EMA:

```
def ema_update(model, ema_model, decay=0.999):
    with torch.no_grad():
        msd = model.state_dict()
        esd = ema_model.state_dict()
        for k in esd.keys():
            if k in msd:
                if esd[k].dtype.is_floating_point:
                    esd[k].mul_(decay).add_(msd[k] * (1.0 - decay))
                else:
                    esd[k] = msd[k]
        ema_model.load_state_dict(esd)
```

  Example pseudocode for Gradient Penalty:

```
# real: [B, 1, H, W], labels y: [B]
# fake_det is detached from G for the critic update
z = torch.randn(B, z_dim, device=dev)
fake_det = G(z, y).detach()

# interpolate
eps = torch.rand(B, 1, 1, 1, device=dev)
x_hat = (eps * real + (1 - eps) * fake_det).requires_grad_(True)

# critic outputs
real_score = D(real, y).mean()
fake_score = D(fake_det, y).mean()
hat_score  = D(x_hat, y).mean()

# gradients wrt inputs
grad = torch.autograd.grad(
    outputs=hat_score, inputs=x_hat,
    create_graph=True, retain_graph=True, only_inputs=True
)[0]                                  # shape: [B, 1, H, W]
```

```
grad_norm = grad.view(B, -1).norm(2, dim=1)  # L2 per sample

gp = ((grad_norm - 1.0) ** 2).mean()
loss_C = -(real_score - fake_score) + lambda_gp * gp
```

Part 2: Training (20 pts)

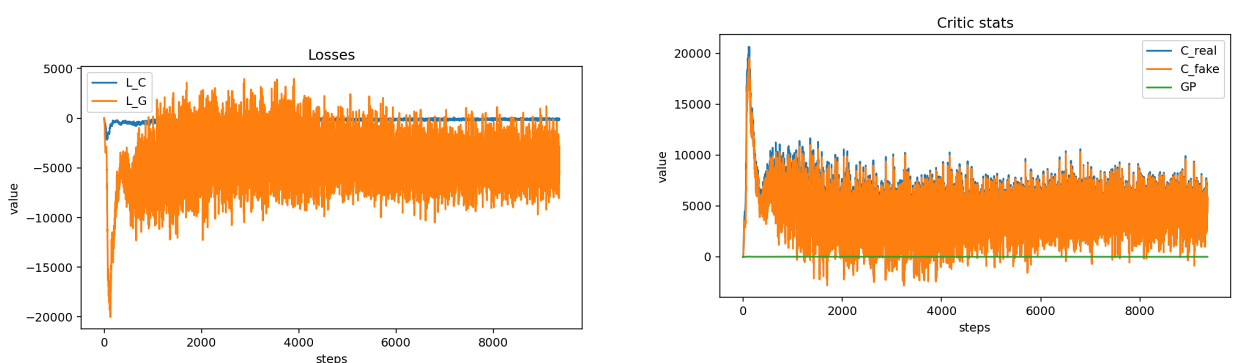
- (10 pts) Train for 10–20 epochs. Recommended: Adam with lr$= 2\times10^{-4}$, betas$(0.0, 0.9)$; λ_{gp}=10; n_{critic}=2–3 (where you update the critic $D(x, y)$ n_{critic} times for every update of $G(z, y)$); batch size ≈ 128. Plot a set of generated samples after every epoch of training using the same latents z using G_{ema}.
- (10 pts) Plot curves for: $\mathbb{E}[D(x, y)]$, $\mathbb{E}[D(G(z, y), y)]$, GP, $\mathcal{L}_C$, $\mathcal{L}_G$ (see Fig. 1).

Part 3: Conditional Samples (20 pts)

- At the end of training, generate a $10 \times N$ grid of conditional samples using your best sampler (e.g., EMA generator). See Fig. 2.

Part 4: Latent Exploration and Truncation (20 pts)

- (10 pts) Latent interpolations for all digits: for each class $c \in \{0, \ldots, 9\}$, interpolate z between two random endpoints and show a row of K images. Aggregate into a $10 \times K$ panel. See Fig. 3.
- (10 pts) Truncation sweep: for each class, generate samples by scaling $z \leftarrow \psi z$ with $\psi \in \{3.0, 2.5, 2.0, 1.5, 1.0, 0.8, 0.6, 0.4, 0.2, 0.1\}$. See Fig. 4.



(a) Loss curves: $\mathcal{L}_C$ and $\mathcal{L}_G$.

(b) Critic diagnostics: $\mathbb{E}[D(x, y)]$, $\mathbb{E}[D(G(z, y), y)]$, and GP.

Figure 1: Training curves.

Submission Instructions

1. Submit a Jupyter Notebook named `CSC8851_F2025_HW2_<YourName>.ipynb`. Include all code, plots, and generated images. The notebook must run top-to-bottom.

Figure 2: Conditional generation grid at the end of training (rows = labels 0–9), produced with EMA generator.

2. Submit a PDF export of your notebook named `CSC8851_F2025_HW2_<YourName>.pdf`. DO NOT paste images of your code in the PDF. It will fail the plagiarism check. :
   - Using **nbconvert**:

     ```
     jupyter nbconvert --to pdf CSC8851_F2025_HW2_<YourName>.ipynb
     ```

   - Or convert to HTML then print to PDF:

     ```
     jupyter nbconvert --to html CSC8851_F2025_HW2_<YourName>.ipynb
     ```

     Open the HTML in a browser and print to PDF.
   - Or use the Jupyter/Colab menu: `File → Download as → PDF`.
3. Upload both files (`.ipynb` and `.pdf`) to the assignment on iCollege.

You may include additional qualitative results (e.g., failure cases). Cite any external code sources used. Keep your code and discussion concise and clear.

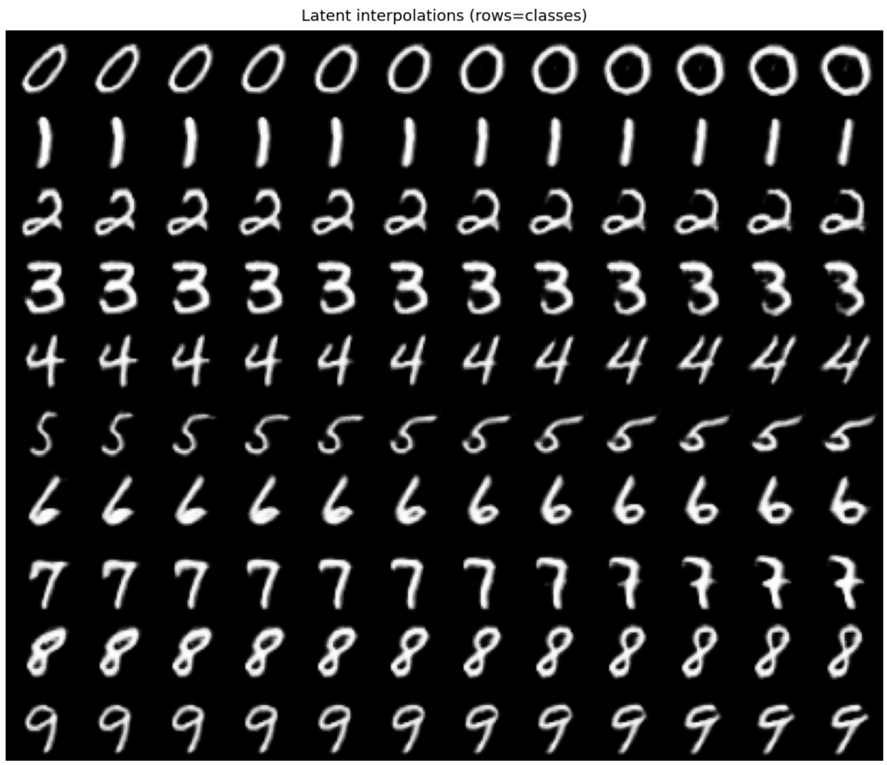


Figure 3: Latent interpolations: rows = classes 0–9, columns = steps along the interpolation.

Figure 4: Truncation sweep per class with $\psi \in \{3.0, 2.5, 2.0, 1.5, 1.0, 0.8, 0.6, 0.4, 0.2, 0.1\}$.