CS5787: Exercises 3

`https://github.com/shl225/CS5787_HW3`

Sean Hardesty Lewis
shl225

1 Theory [50 pts]

1. **[40pts]** Consider the following 3 distance measures between the distributions p and q:

- **KL (Kullback-Leibler Divergence)**:

$$D_{\text{KL}}(p||q) = \int_{x \in X} p(x) \log \frac{p(x)}{q(x)}\, dx$$

Note that it is always either positive, or 0 iff $\forall x \in X, p(x) = q(x)$.
One of its disadvantages is that it is asymmetric.

- **JS (Jensen-Shannon Divergence)** – a symmetric (and smoother) version of the KL divergence:

$$D_{\text{JS}}(p||q) = \frac{1}{2} D_{\text{KL}}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2} D_{\text{KL}}\left(q \middle\| \frac{p+q}{2}\right)$$

- **Wasserstein Distance** (also known as "Earth Mover distance"), defined for continuous domain:

$$W(p,q) = \inf_{\gamma \sim \Pi(p,q)} \sum_{x,y} \gamma(x,y) \|x-y\| = \inf_{\gamma \sim \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|]$$

where $\Pi(p,q)$ is the set of all possible joint distributions of p and q, γ describes a specific one, and x and y are values drawn from p and q, respectively. Intuitively, it measures the minimal cost of "moving" p to become q.

Suppose we have two 2-dimensional probability distributions, P and Q:

- $\forall(x,y) \in P, x = 0$ and $y \sim U(0,1)$
- $\forall(x,y) \in Q, x = \theta, 0 \le \theta \le 1,$ and $y \sim U(0,1)$

θ is a given constant.

Obviously, when $\theta \neq 0$, there is no overlap between P and Q.

a. **[20pts]** For the case where $\theta \neq 0$, calculate the distance between P and Q by each of the three measurements.

Calculations for $\theta \neq 0$

1. Kullback-Leibler (KL) Divergence:
Since P and Q have disjoint supports when $\theta \neq 0$, the KL divergence is infinite.

$$D_{\text{KL}}(P||Q) = \infty$$

2. Jensen-Shannon (JS) Divergence:

$$D_{\text{JS}}(P||Q) = \log 2$$

Calculation:

- The mixed distribution $M = \frac{1}{2}(P+Q)$.
- At points where P has support (i.e., $x = 0$), $M(x,y) = \frac{1}{2}p(x,y)$.
- So the KL divergence from P to M is:

$$D_{\text{KL}}(P||M) = \int p(x,y) \log \frac{p(x,y)}{M(x,y)}\, dxdy = \int p(x,y) \log 2\, dxdy = \log 2$$

- Similarly, $D_{\text{KL}}(Q||M) = \log 2$.
- Thus:

$$D_{\text{JS}}(P||Q) = \frac{1}{2}(\log 2 + \log 2) = \log 2$$

3. Wasserstein Distance:

$$W(P,Q) = \theta$$

Calculation:

- The optimal transport plan is to move each point $(0,y)$ in P to (θ,y) in Q.
- The cost for each point is $\|(0,y) - (\theta,y)\| = \theta$.
- So the expected cost (distance) is:

$$W(P,Q) = \int_0^1 \theta \cdot dy = \theta \cdot (1-0) = \theta$$

b. **[10pts]** Repeat question a for the case where $\theta = 0$.

Calculations for $\theta = 0$

When $\theta = 0$, P and Q are identical distributions.

1. Kullback-Leibler (KL) Divergence:

$$D_{\text{KL}}(P||Q) = 0$$

2. Jensen-Shannon (JS) Divergence:

$$D_{\text{JS}}(P||Q) = 0$$

3. Wasserstein Distance:

$$W(P,Q) = 0$$

c. **[10pts]** Following your answers, what is the advantage of the Wasserstein Distance over the previous two?

The Wasserstein distance reflects the actual geometric distance between distributions. It captures the notion of how much "effort" is required to transform one distribution into another. In this case, as θ increases, the Wasserstein distance increases linearly with θ, accurately reflecting the separation between P and Q.

In contrast, the KL and JS divergences do not change with increasing θ when the supports are disjoint; they remain at infinity or a constant value ($\log 2$). This makes them less sensitive to the actual "distance" between non-overlapping distributions.

2. **[10pts]** Assume that we have access to an LSTM and a Transformer model. What is the computational complexity of generating a sequence of length N for both models? Explain your reasoning.

Computational Complexity of Generating a Sequence of Length N:

- **LSTM (Long Short-Term Memory):**
  - **Complexity:** $O(N)$

  - **Explanation:** LSTMs generate sequences sequentially. At each time step t, the computation depends on the previous hidden state h_{t-1}. So generating a sequence of length N requires N sequential computations.

- **Transformer:**
  - **Complexity:** $O(N^2)$
  - **Explanation:** Transformers can process all positions simultaneously thanks to their parallelizable architecture. However, the self-attention mechanism computes attention weights between all pairs of positions, leading to $O(N^2)$ complexity.

2 Practical [50 pts]

3. **[5pts]** To gain Intuition (and have fun), go to https://poloclub.github.io/ganlab/. Read the instructions, and then train a GAN by yourself. Draw a distribution and train the GAN on it. Submit in the PDF a screen shot of the result along with the epoch number and the graphs on the right.

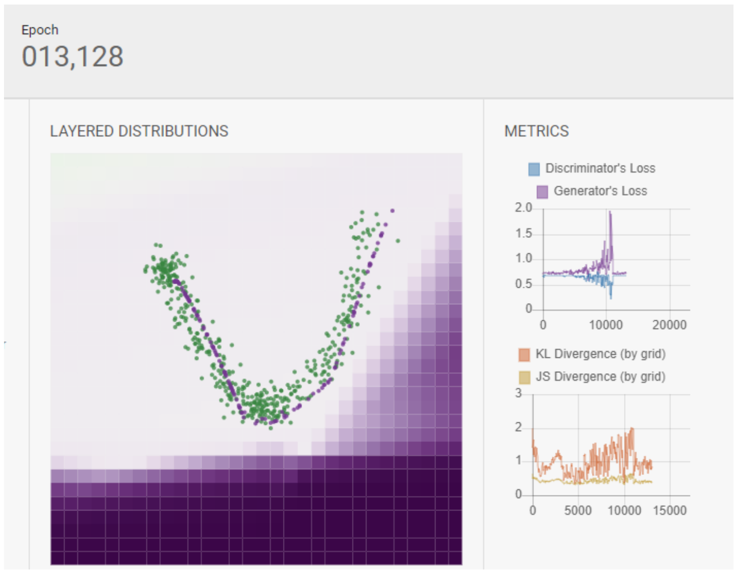

Figure 1: Converged GAN from GAN Lab.

4. **[20pts]** Here we address semi-supervised learning via a variational autoencoder. You will be implementing a part of the paper "Semi-supervised Learning with Deep Generative Models", by Kingsma et al. Read the paper, and implement the M1 scheme, as described in Algorithm1, and detailed throughout the paper. It is based on a VAE for feature extraction, and then a (transductive) SVM for classification. Implement the network suggested for MNIST, and apply it on the Fashion MNIST data set. Present the results for 100, 600, 1000 and 3000 labels, as they are presented in Table 1 in the paper. For simplicity, you may use a regular SVM (with a kernel of your choice). Take the latent representation of the labeled data as training, and then test it on the test set.

   a. We implemented the M1 model using a Variational Autoencoder for feature extraction and an SVM for classification on the Fashion MNIST dataset. The results below show how performance improves as we increase the number of labeled samples. We used Python 3.9.1 as the kernel and a fixed seed.

Table 1: Test accuracy of our implemented M1 model on Fashion MNIST.

Number of Labels	Test Accuracy
100	0.4420
600	0.6783
1000	0.7289
3000	0.7591

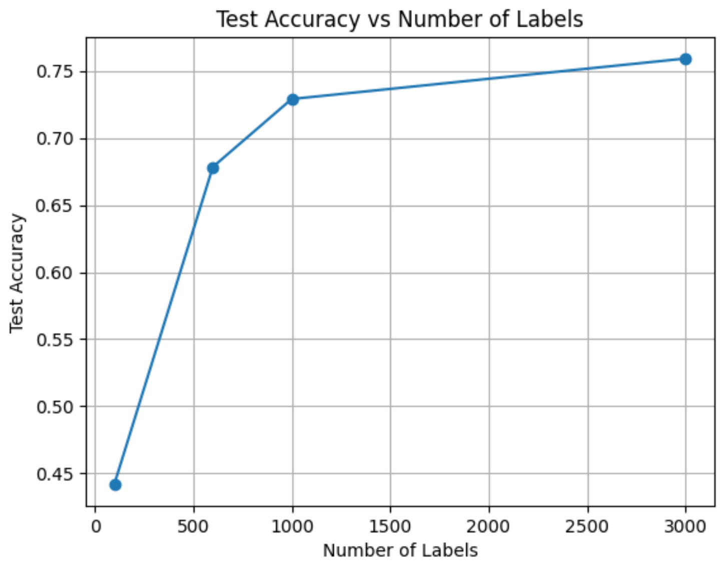

Figure 2: Test Accuracy vs Number of Labels of our M1 model on Fashion MNIST.

5. **[25pts]** Here you will get to play a bit with GANs and WGANs, by implementing a part of the paper ”Improved Training of Wasserstein GANs”, by Gulrajani et al. Read the paper, and implement DCGAN, WGAN (weight clipping), and WGAN-GP (gradient clipping). Choose two architectures from Table 1. Make appropriate modifications and apply it on the Fashion MNIST data set. Training should be supervised by the class label.

a. Plot the loss functions. There should be two plots, one for each architecture. Each plot should have three loss curves (one for each method). No need for validation / testing loss curves, as there is no “ground-truth.”

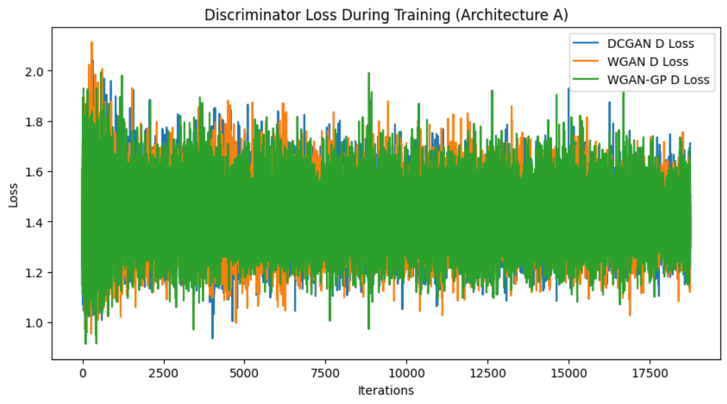

Figure 3: Discriminator Loss During Training (Architecture A).

i.

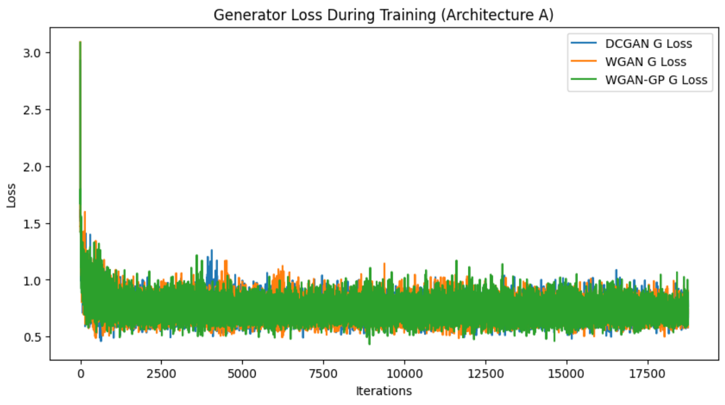

Figure 4: Generator Loss During Training (Architecture A).

ii.

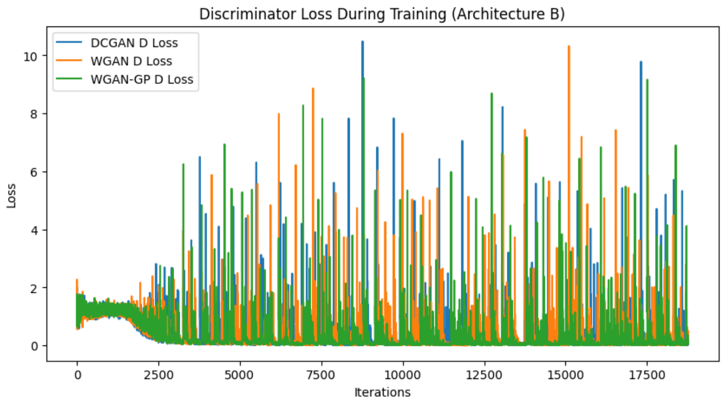

Figure 5: Discriminator Loss During Training (Architecture B).

iii.

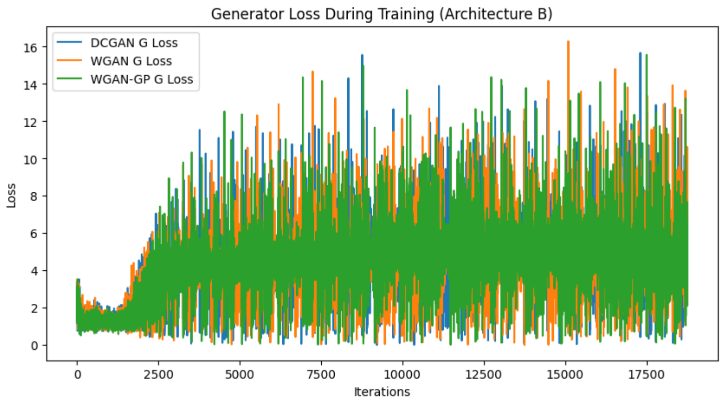

Figure 6: Generator Loss During Training (Architecture B).

iv.

b. From the better performing architecture, select two generated images from each method for two labels (e.g. six images for T-shirt and six images for Dress). Also include two real images from the corresponding label.

Architecture B was better performing due to its capability to generate higher quality images, despite having higher generator losses. This suggests that its discriminator is a bit more robust, leading to more effective training dynamics and more visually appealing and diverse outputs.

Table 2: Failed Attempts per Architecture and Model at Threshold 0.1

Architecture	**DCGAN**	**WGAN**	**WGAN-GP**
A	15,452	15,683	16,383
B	21,267	21,540	21,801

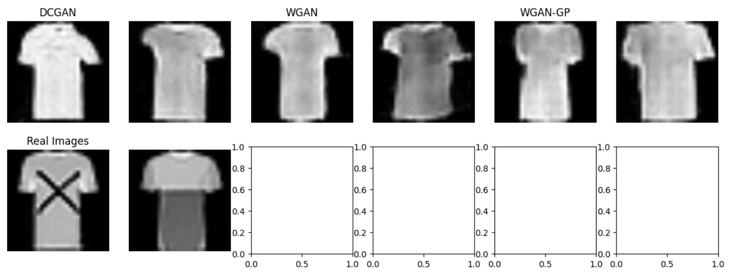

Figure 7: Generated Images for T-shirt (Architecture B).

i.

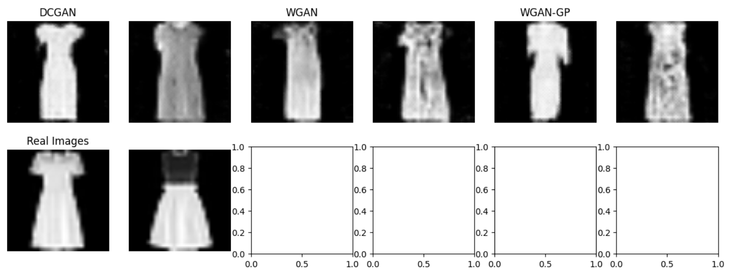

Figure 8: Generated Images for Dress (Architecture B).

ii.

For more details, code, and relevant citations, please see the Github.