

## 0 Instructions

Homework is due Tuesday, April 16, 2024 at 23:59pm Central Time. Please refer to <https://courses.grainger.illinois.edu/cs446/sp2024/homework/hw/index.html> for course policy on homeworks and submission instructions.

## 1 GAN: 5pts

A Generative Adversarial Network (GAN) consists of two parts:

- **Generator**  $\mathcal{D}$  generates a data sample from a random noise. It is trained to fit the real data distribution so that the synthesized data is close to real data.
- **Discriminator**  $\mathcal{G}$  predicts the probability of a sample coming from real data distribution.

The training objective is formulated as a minimax game:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{x \sim p_r(x)} [\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - \mathcal{D}(\mathcal{G}(z)))] . \quad (1)$$

$p_r$  is the real data distribution.  $p_z$  is the distribution that we sample from, which is usually the standard Gaussian distribution.

1. (2pts) Suppose that the generator  $\mathcal{G}$  is fixed, what is the optimal value for the discriminator, i.e.  $\mathcal{D}^*$ ? Write  $\mathcal{D}^*$  in terms of  $p_r(x)$  and  $p_g(x)$ .  $p_g$  is the generated distribution, or the distribution of  $\mathcal{G}(z)$  in Eq. 1.
2. (2pts) Show that when  $\mathcal{D}$  reaches optimal, optimizing Eq. 1 is the same as minimizing the Jensen-Shannon (JS) divergence.
3. (1pts) Explain why the original GAN has the problem of vanishing gradients.

Hint: What will happen when  $\mathcal{D}$  perfectly classifies generated samples from real data?

## 2 Diffusion Model: 11pts

In the forward diffusion process, we gradually convert a data sample  $\mathbf{x}_0 \sim q_0(\mathbf{x})$  by adding a small Gaussian noise in each step. We define

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}).$$

In the reverse diffusion process, we generate  $\mathbf{x}_0$  by passing the noise through  $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ .

1. (1pts) A diffusion model shares similarities to VAE in that it also has an encoder stage (forward diffusion process) which encodes data sample  $\mathbf{x}_0$  to noise  $\mathbf{x}_T$  and a decoder stage (backward diffusion process) which decodes the noise to a data sample. When training a diffusion model, we can also maximize ELBO to optimize the log-likelihood  $\mathbb{E}_{\mathbf{x}_0 \sim q_0(\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0)]$ . Write down the formula for ELBO for  $\log p_\theta(\mathbf{x}_0)$  in diffusion model, and indicate which distribution the expectation is with respect to using the subscript to  $\mathbb{E}[\cdot]$ .
2. (1pts) One way to evaluate generative models is to report the likelihood (or density)  $p_\theta(\mathbf{x}_0)$  where  $\mathbf{x}_0$  is sampled from the test set. Can we directly estimate the density in diffusion models?
3. (3pts) Derive  $q(\mathbf{x}_t|\mathbf{x}_0)$  as a function of  $\beta_i, i = 0, 1, \dots, t$  and  $\mathbf{x}_0$ .

**Hint:** Use the reparameterization trick.

4. (3pts) The training objective contains the KL-divergence term

$$KL(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)).$$

Derive the mean  $\mu_\theta(\mathbf{x}_t, \mathbf{x}_0)$  of  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ .

**Hint:** Using Bayes rule,  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}$ .

**Hint:**  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  is also a Gaussian distribution.

5. (3pts) We can understand diffusion models from the perspective of score-based generative models. In score-based generative models, we estimate the score function  $s_\theta(\mathbf{x}, \delta) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}, \delta)$ . The loss function is

$$\mathcal{L} = \sum_{t=1}^T \frac{\lambda_t}{2} \mathbb{E}_{\mathbf{x} \sim q_0(\mathbf{x}), \tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \delta_t^2 \mathbf{I})} \left[ \left\| s_\theta(\tilde{\mathbf{x}}, \delta_t) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\delta_t^2} \right\|_2^2 \right].$$

After obtaining  $s_\theta(\mathbf{x}, \delta)$ , we sample with Langevin dynamics.

Suppose we have trained  $s_\theta(\mathbf{x}, \delta)$  to approximate unconditional  $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}, \delta)$ , where

$\mathbf{x} \in \mathbb{R}^d$ . Let's consider the posterior sampling  $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{x}_{\text{known}})$ : we want to sample  $\mathbf{x}$  given partial observation  $\mathbf{x}_{\text{known}} \in \mathbb{R}^d$  and the observation mask  $\mathbf{M} \in \{0,1\}^d$  so that the sample aligns with the partial observation.  $\mathbf{M}_i = 1$  indicates that the  $i$ -th dimension in  $\mathbf{x}_{\text{known}}$  is observed.

Show the conditional score function estimation  $s_\theta(\mathbf{x}, \delta|\mathbf{x}_{\text{known}})$  as a function of  $s_\theta(\mathbf{x}, \delta)$ ,  $\mathbf{x}$ ,  $\mathbf{x}_{\text{known}}$  and  $\mathbf{M}$ .

**Hint:**  $s_\theta(\mathbf{x}, \delta|\mathbf{x}_{\text{known}})$  estimates  $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}, \delta|\mathbf{x}_{\text{known}})$ . Using Bayes' rule,  $p_\theta(\mathbf{x}, \delta|\mathbf{x}_{\text{known}}) = \frac{p(\mathbf{x}_{\text{known}}|\mathbf{x})p_\theta(\mathbf{x}, \delta)}{p(\mathbf{x}_{\text{known}})}$ .

**Hint:** You can assume that  $p(\mathbf{x}_{\text{known}}|\mathbf{x}) \propto \exp(-\|(\mathbf{x} - \mathbf{x}_{\text{known}}) \odot \mathbf{M}\|_2^2)$ . Here,  $\odot$  is the element-wise product.

**Remark:** With the derived  $s_\theta(\mathbf{x}, \delta|\mathbf{x}_{\text{known}})$ , we can sample from the posterior distribution without re-training. This can be applied to tasks, e.g. image inpainting.

### 3 Unsupervised learning / contrastive learning: 4 pts

True or false questions. If it is false, explain the reason in a few sentences. Each question is worth 1 point.

1. In unsupervised learning, we can evaluate the effectiveness of unsupervised learning by selecting several important downstream tasks and reporting performance on them.
2. Choosing a good pretraining task is essential for unsupervised learning. MAE and BERT are both examples of self-prediction. MAE uses the same mask-out rate during training as BERT.
3. Minimizing the InfoNCE loss maximizes a lower bound on mutual information.
4. We cannot use CLIP to classify images without finetuning on labelled image classification dataset.

## 4 Coding: GAN, 10pts

In this problem, you need to implement a Generative Adversarial Network and train it on MNIST digits.

Table 1: **Discriminator Architecture**

<i>Layer No.</i>	<i>Layer Type</i>	<i>Kernel Size</i>	<i>Stride</i>	<i>Padding</i>	<i>Output Channels</i>
1	conv2d	3	1	1	2
2	ReLU	-	-	-	2
3	MaxPool	2	2	-	2
4	conv2d	3	1	1	4
5	ReLU	-	-	-	4
6	MaxPool	2	2	-	4
7	conv2d	3	1	0	8
8	ReLU	-	-	-	8
9	Linear	-	-	-	1

Table 2: **Generator Architecture**

<i>Layer No.</i>	<i>Layer Type</i>	<i>Kernel Size</i>	<i>Stride</i>	<i>Padding</i>	<i>Bias</i>	<i>Output Channels</i>
1	Linear	-	-	-	✓	1568
2	LeakyReLU(0.2)	-	-	-	-	1568
3	Upsample(scale=2)	-	-	-	✗	32
4	conv2d	3	1	1	✓	16
5	LeakyReLU(0.2)	-	-	-	-	16
6	Upsample(scale=2)	-	-	-	✗	16
7	conv2d	3	1	1	✓	8
8	LeakyReLU(0.2)	-	-	-	-	8
9	conv2d	3	1	1	✓	1
10	sigmoid	-	-	-	-	1

1. Implement a discriminator **DNet** in `hw5_gan.py` with architecture in Tab. 1. Layers contain bias if corresponding `torch` function has an option for adding one.

**Remark 1:** From layer 8 to layer 9, you need to flatten each data entry from a matrix to a vector.

2. Implement a generator **GNet** in `hw5_gan.py` with architecture in Tab. 2.

**Remark 2:** From layer 2 to layer 3, you need to reshape each data to size  $(32, 7, 7)$  in the format of *CHW*. Note,  $1568 = 32 \times 7 \times 7$ .

**Remark 3:** For (a) and (b), please define layers in `__init__` with **exactly the same** order as they appear in Tab. 1 and Tab. 2.

**Remark 4:** We have listed **all** layers for discriminator and generator. No need to add any extra components.

3. Implement the weight initialization function `_weight_init` in `DNet` and `GNet`: use `kaiming_uniform` for weights and 0 for the bias if the layer contains bias.

**Hint:** to iterate over all layers an `nn.Module` has, you may find `self.children()` useful. See `children()` function explained in <https://pytorch.org/docs/stable/generated/torch.nn.Module.html>.

4. Implement the loss function for the discriminator: `_get_loss_d` of `GAN` class in `hw5_gan.py`.

**Hint:** you may find `torch.nn.BCEWithLogitsLoss` useful.

5. Implement the loss function for the generator: `_get_loss_g` of `GAN` class in `hw5_gan.py`.

**Hint:** you may find `torch.nn.BCEWithLogitsLoss` useful.

6. Attach generated images after training.

**Remark 5:** the provided code default saves images during training. You can choose three of the saved ones and indicate the corresponding epochs.

**Remark 6:** with default training settings, you should obtain reasonable generated images after around 30 epochs.

## 5 Coding: Diffusion Model, 10pts

In this problem, you need to implement a score-based generative model and use it to sample from a complicated distribution of points. As introduced in lecture, sampling from this model consists of two main steps: matching the score functions  $s_\theta(\mathbf{x}, \delta)$  of each noise-perturbed distribution and then sampling from it by running Langevin dynamics.

---

**Algorithm 1** Compute denoising loss.

---

**Require:** Score function  $s_\theta$ , training sample  $\mathbf{x}$ ,  $\{\sigma_i\}_{i=1}^L$ .

- 1: Sample  $\sigma$  from  $\{\sigma_i\}_{i=1}^L$
  - 2: Sample  $\mathbf{z} \sim \mathcal{N}(0, I)$
  - 3:  $\tilde{\mathbf{x}} \leftarrow \mathbf{x} + \sigma \mathbf{z}$
  - 4:  $\lambda \leftarrow \sigma^2$
  - 5:  $\mathcal{L} \leftarrow \frac{\lambda}{2} \left\| s_\theta(\tilde{\mathbf{x}}, \sigma) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \right\|_2^2$
- return**  $\mathcal{L}$
- 

---

**Algorithm 2** Annealed Langevin dynamics.

---

**Require:**  $\{\sigma_i\}_{i=1}^L, \epsilon, T$ .

- 1: Initialize  $\tilde{\mathbf{x}}_0 \sim \mathcal{N}(0, I)$
  - 2: **for**  $i \leftarrow 1$  to  $L$  **do**
  - 3:    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$
  - 4:   **for**  $t \leftarrow 1$  to  $T$  **do**
  - 5:     Draw  $\mathbf{z}_t \sim \mathcal{N}(0, I)$
  - 6:      $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \alpha_i s_\theta(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{2\alpha_i} \mathbf{z}_t$
  - 7:   **end for**
  - 8:    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
  - 9: **end for**
- return**  $\tilde{\mathbf{x}}_T$
- 

▷  $\alpha_i$  is the step size.

1. Implement the class `ScoreNet` in `hw5_diffusion.py`. `ScoreNet` is a neural network that predicts  $s_\theta(\mathbf{x}, \delta)$ . We use 8 linear layers and `Softplus` as the activation function. In the forward pass, it takes  $\mathbf{x}$  and  $\delta$  as inputs. Read the comments in the codes for detailed instructions.
2. Implement the training step in `compute_denoising_loss`. The detailed steps are in Alg. 1. Note that Alg. 1 is for only one training sample. However, in `compute_denoising_loss`, you will be asked to apply it for all training samples and return the loss averaged over all training samples.

3. Implement the Langevin dynamics sampling in `langevin_dynamics_sample`. We use the Annealed Langevin Dynamics shown in Alg. 2.
4. You are now ready to train the model and sample by running the main function.
  - (a) (3pts) Visualize the score function using `hw5_utils.plot_score`. **Include the visualization in the report.**
  - (b) (4pts) Generate 1000 new samples with `langevin_dynamics_sample`. Plot the points at time step 0, 200, 400, 600, 800 and the final sampled points. Your final samples should roughly follow the pattern “CS446”. **Include the plots in the report.**
  - (c) (3pts) Visualize the trajectory of langevin dynamics. You could get the trajectory by setting `return_traj=True` in `langevin_dynamics_sample`. **Include the visualization of the trajectory in the report.**