HY673 - Spring 2026 Problem Set #4
Yannis Pantazis (06/05/2026)

Instructions

- **Due date:** Friday May 29th, 2026
- Submission via e-mail to the class account: hy673@csd.uoc.gr
- Provide one file with the written solutions.
- Provide one folder with code.
  - The name of each file in the folder should indicate the respective exercise.
  - Your code should run on Colab.
- All assignments in this course are individual, not group assignments. You may freely discuss homework assignments with your classmates. The final solutions, however, must be written entirely on your own.
- You are allowed to use generative AI tools (e.g., ChatGPT, Gemini) only for grammatical corrections unless explicitly permitted otherwise. To maintain academic integrity, students must disclose any use of AI-generated material.

Problem 1 (Conditional Diffusion Model for MNIST digit dataset – 25 points)**.** *This exercise is an extension of the tutorial on diffusion models* ***https://github.com/mikerapt/hy673_tutorials/tree/main/Tutorial-9*** *.*

(a) Modify the code from the tutorial on diffusion models for MNIST digit generation and implement a conditional version of it. Add the conditional one-hot encoding vector as input to the U-Net.

(b) Implement the conditional diffusion model using the classifier-free approach (slide 32 in Lecture 13 and ***https://arxiv.org/pdf/2207.12598.pdf*** *).*

Problem 2 (Classifier guidance with a pretrained diffusion model – 20 points)**.** *In this exercise you will perform conditional generation by guiding an* unconditional *pretrained diffusion model with a separately trained classifier (Dhariwal & Nichol, 2021,* ***https://arxiv.org/pdf/2105.05233.pdf*** *). The key idea is that the score of the conditional distribution can be decomposed as*

$$\nabla_{x_t} \log p(x_t \mid y) \;=\; \nabla_{x_t} \log p(x_t) \;+\; \nabla_{x_t} \log p(y \mid x_t),$$

so a classifier $p_\phi(y \mid x_t)$ *trained on* noisy *inputs* x_t *can be used to steer the reverse process of an unconditional model toward class* y *without retraining the diffusion model itself.*

*(a) Take the unconditional diffusion model trained in the tutorial (**https://github.com/mikerapt/hy673_tutorials/tree/main/Tutorial-9**) on MNIST – or train one from scratch if you prefer – and **keep its weights frozen** throughout this exercise.*

(b) Train a classifier $p_\phi(y \mid x_t, t)$ on noisy *MNIST images. For each training image x_0 with label y, sample $t \sim \mathcal{U}\{1, \dots, T\}$, generate x_t using the forward diffusion process, and train the classifier to predict y from (x_t, t). The classifier must be conditioned on t (e.g., via a time embedding) since the noise level varies.*

(c) Modify the reverse sampling loop so that at each step the predicted mean is shifted by $s \cdot \Sigma_t \nabla_{x_t} \log p_\phi(y \mid x_t, t)$, where $s > 0$ is the guidance scale. Generate samples for at least three values of s (e.g., $s \in \{0, 1, 5, 10\}$) and for several target classes. Compare the visual quality and class-conditional accuracy across guidance scales and discuss the diversity-fidelity trade-off.

(d) Briefly compare classifier guidance (this exercise) with the classifier-free guidance approach of Problem 1(b). Comment on advantages and disadvantages of each in terms of training cost, flexibility, and sample quality.

Problem 3 (Train GANs on the Fashion MNIST dataset – 20 points)**.** *In this exercise you will generate images from the Fashion MNIST dataset. The FashionMNIST can be simply imported using the torchvision module via:*

```
from torchvision.datasets import FashionMNIST
dataset = FashionMNIST(rootdir, train=True, download=True)
```

(a) Train a deep convolutional generative adversarial network (DCGAN) using the original loss function on Fashion MNIST dataset. The architecture of the generator will consist of one dense and three convolutional layers while the architecture of the discriminator will consist of three convolutional and one dense layer (i.e., reverse order)[1]*. Use Adam optimizer with learning rate of order $O(10^{-4})$ as well as dropout for the training. Set the input dimension for the generator (i.e., the dimension of the noise vector) in the range between 50 and 200.*

*(b) Using the same architectures, train WGAN with Wasserstein loss function. You should change the optimizer to RMSProp and apply parameter clipping in order to enforce the Lipschitz constraint. For RMSProp check **https://pytorch.org/docs/stable/generated/torch.optim.RMSprop.html** while parameter clipping can be enforced via:*

```
for p in discriminator.parameters():
    p.clamp_(-max_amp, max_amp)
```

where 'max_amp' is a hyperparameter that determines the clipping severity. You may explore values in the range between 0.1 and 0.001.

(c) Generate samples from both models and comment on the generated images.

[1] In case of limited resources, the architectures could have just one conv layer

Problem 4 (Conditional GAN and WGAN-GP on Fashion MNIST – 20 points)**.** *This exercise extends Problem 4 by exploring two important refinements of GANs:* conditional *generation and the* gradient penalty *variant of WGAN. Use the same Fashion MNIST dataset and base architectures as in Problem 4.*

(a) Implement a conditional DCGAN (cGAN, Mirza & Osindero, 2014, `https://arxiv.org/pdf/1411.1784.pdf` *) on Fashion MNIST. Concatenate the one-hot class label to the generator's noise vector at the input, and inject the label into the discriminator (e.g., as an additional input channel after broadcasting the one-hot vector to the spatial size of the image, or by concatenating it to the dense layer). Train with the original GAN loss as in Problem 4(a).*

(b) Replace the weight clipping of Problem 4(b) with the gradient penalty *of WGAN-GP (Gulrajani et al., 2017,* `https://arxiv.org/pdf/1704.00028.pdf` *). The critic loss becomes*

$$L_D \;=\; \mathbb{E}_{\tilde{x}\sim p_g}[D(\tilde{x})] \;-\; \mathbb{E}_{x\sim p_r}[D(x)] \;+\; \lambda\,\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}\Big[\big(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\big)^2\Big],$$

where $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$ *with* $\epsilon \sim \mathcal{U}[0,1]$ *and* $\lambda \approx 10$*. You should also remove dropout/batch normalization from the critic (replace with layer normalization or instance normalization if needed) and switch the optimizer back to Adam with* $\beta_1 = 0.5$*,* $\beta_2 = 0.9$*.*

(c) Generate (i) class-conditional samples from the cGAN of part (a) for every Fashion MNIST class, and (ii) unconditional samples from the WGAN-GP of part (b). Compute the Fréchet Inception Distance (FID) between 10,000 *generated samples and* 10,000 *real test images for the three models you have trained so far: DCGAN (Problem 4(a)), WGAN with clipping (Problem 4(b)), and WGAN-GP (this problem). You may use* `torchmetrics.image.fid.FrechetInceptionDistance` *or* `pytorch-fid`*. Report the FID scores in a table and discuss the results: does the gradient penalty improve over weight clipping? How does the conditional model affect sample diversity within each class?*

Problem 5 (Create different avatars from your photo – 15 points)**.** *Make a photo of yours and go to an online site that generates avatars and create your own avatar. Generate at least five avatars using different conditions regarding appearance, colors, style and backgrounds. Explore the space of avatar making websites (at least two of them) and assess the quality and coherence of the generated avatars for each one.*