HY673 - Spring 2024 Problem Set #4
Sifakis Emmanouil, csdp1288 (10/06/2024)

Problem 1. *(Train GANs on the Fashion MNIST dataset). 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). 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.

Solution

a)

The objective of this exercise is to train a Deep Convolutional Generative Adversarial Network (DCGAN) on the Fashion MNIST dataset using the original loss function. For this reason the provided code from the Tutorial 10 was used. The only substansial change was the use of the Fashion-MNIST instead of the MNIST dataset. The Fashion MNIST dataset comprises 70,000 grayscale images in 10 categories, with 7,000 images per category. Each image is 28x28 pixels. This dataset serves as a more challenging replacement for the original MNIST dataset and is used for benchmarking machine learning algorithms. The architecture of the generator consists of one dense layer followed by three convolutional layers, while the discriminator consists of three convolutional layers followed by one dense layer. The Adam optimizer is used with a learning rate of order $O(10^{-4})$, along with dropout for regularization. The input dimension for the generator is set between 50 and 200 (100 to be precise).

The generator network transforms a noise vector (latent space input) into a 28x28 grayscale image. The input to the generator is a noise vector of dimension 100. It consists of a dense layer that transforms the noise vector to a 7x7x128 tensor, followed by three

ConvTranspose2D layers with BatchNorm and PReLU activations to generate 28x28 images. The discriminator network evaluates the probability that a given image is real or fake. The input to the discriminator is a 28x28 grayscale image. It consists of three Conv2D layers with BatchNorm, PReLU activations, and Dropout, followed by a dense layer that produces a scalar output indicating real or fake.

The Binary Cross-Entropy (BCE) loss function is used to train both the generator and the discriminator. Also the Adam optimizer is employed with a learning rate of 2×10^{-4} and betas $(0.5, 0.999)$. The training loop alternates between updating the discriminator and the generator. The best generator model is saved based on the lowest generator loss observed during training. From the loss curves in Figure 1, it is evident that the generator loss is at its lowest point at the beginning of the training and increases as the epochs progress, while the discriminator loss decreases continuously. This suggests that the best generator model, in terms of loss, is achieved at the beginning of the training process. The generator starts by creating images that are good enough to fool the discriminator, but as the discriminator improves its ability to distinguish between real and fake images, the generator's task becomes more challenging, leading to an increase in generator loss.

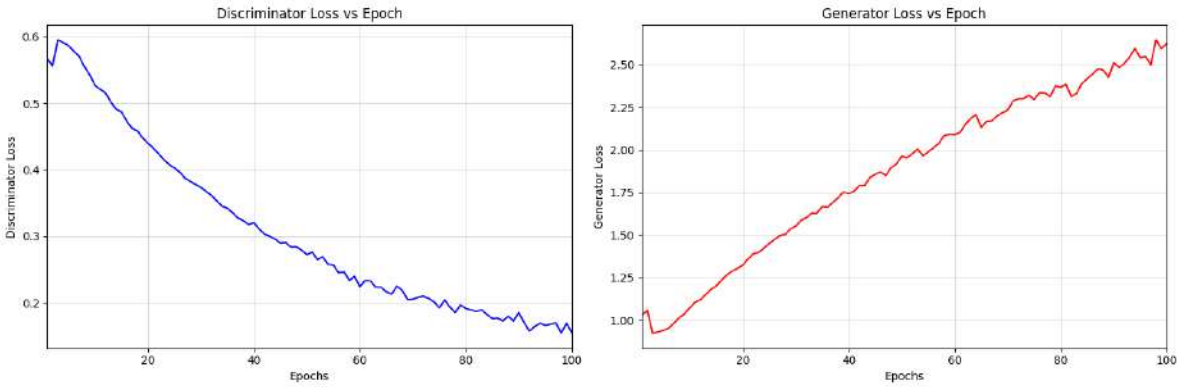

Figure 1: Discriminator and Generator Loss over Epochs (DCGAN)

For this reason, the quality of the generated images does not improve over the epochs. Below in Figure 2 are the generated images for the best epoch and for the last epoch. Given this observation, it appears that training the model for 100 epochs may be unnecessary. A fewer number of epochs might be sufficient to achieve a good balance between the generator and the discriminator. This could save computational resources and time, while still yielding high-quality generated images.

Figure 2: On the left is the generated images for the best epoch (3), where the generator has $G\_Loss = 0.92$ and $D\_Loss = 0.59$. On the right is the gererated imaged for the 100th and last epoch, where $G\_Loss = 2.62$ and $D\_Loss = 0.16$

b)

The objective of this exercise is to train a Wasserstein Generative Adversarial Network (WGAN) on the Fashion MNIST dataset using the Wasserstein loss function. The main difference from the previous exercise is regarding the loss and the optimizer. Wasserstein GAN (WGAN) aims to improve the training stability of GANs by using the Wasserstein distance instead of the Jensen-Shannon divergence. The Wasserstein loss for WGAN is defined as follows:

- **Discriminator (Critic) Loss:**

$$L_D = -\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))]$$

- **Generator Loss:**

$$L_G = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))]$$

The RMSProp optimizer is used with parameter clipping to enforce the Lipschitz constraint. Parameter clipping is performed using:

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

where ‘max_amp‘ is a hyperparameter that determines the clipping severity, with values explored in the range between 0.1 and 0.001.

In WGAN, the discriminator (critic) is trained more frequently than the generator to ensure the discriminator provides a good approximation of the Wasserstein distance. In this implementation, the discriminator is trained five times more frequently than the generator.

The training losses for both the discriminator and the generator over 100 epochs are plotted in Figure 3. Unlike in Exercise 1.a, the generator loss decreases while the discriminator loss increases. This behavior indicates that the generator improves its ability to produce

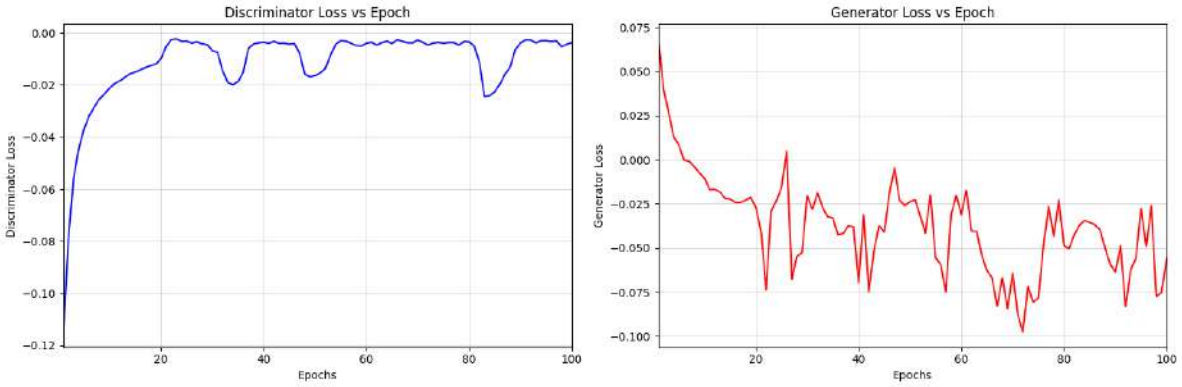

Figure 3: Discriminator and Generator Loss over Epochs (WGAN)

realistic images over time, whereas the discriminator's task becomes more challenging in distinguishing real from fake images, which is a desired outcome in WGAN training.

Below in Figure 4 it is clear that the model learns to generate images, but not substantially better the previou s question (1.a).

Figure 4: On the left is the generated images for the best epoch (72), where the generator has $G\_Loss = -0.097$ and $D\_Loss = -0.005$. On the right is the gererated imaged for the 100th and last epoch, where $G\_Loss = 0.056$ and $D\_Loss = -0.004$

c)

Although in the case of DCGAN the generator's loss was increasing and the discriminator's loss was decreasing, indicating an improvement in the discriminator's ability to distinguish between real and fake images, the situation is reversed in WGAN. In WGAN, the discriminator's loss increases, suggesting that it has difficulty differentiating real from fake images, while the generator's loss decreases, indicating improvement in generating realistic images. Despite these trends, the results from WGAN are not substantially better than those from DCGAN.

WGAN is designed to provide a more stable training process by using the Wasserstein distance, which theoretically helps in avoiding problems like mode collapse and provides more

meaningful gradients throughout the training process. However, the practical outcomes may not always align perfectly with theoretical expectations due to several factors. WGANs can be quite sensitive to the choice of hyperparameters such as the learning rate, clipping threshold, and the number of discriminator updates per generator update. Incorrect settings can lead to suboptimal training dynamics. Both WGAN and DCGAN rely heavily on the chosen architecture. If the architecture is not well-suited to the task, improvements from advanced loss functions may be marginal. The performance improvements of WGAN might be more noticeable on datasets that are significantly different from Fashion MNIST, where the distinctions between classes are more subtle and complex. These points suggest that while WGAN offers theoretical improvements over DCGAN, practical factors significantly influence these benefits. Further experiments with adjusted hyperparameters, alternative architectures, or different datasets may better leverage WGAN's capabilities.

Problem 2. *(Conditional Diffusion Model for MNIST digit dataset).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 14 and `https://arxiv.org/pdf/2207.12598.pdf`*).*

Solution

a)

The primary objective of this question is to extend the basic diffusion model, given in Tutorial 9, to incorporate class conditioning, enabling the generation of class-specific digit images. The key modification involves conditioning the U-Net architecture with one-hot encoded class labels to guide the diffusion process.

The noise scheduler function precomputes various parameters required for the diffusion process. This function remains unchanged from the original tutorial. The ChannelShuffle operation rearranges the channels of the tensor without changing its overall dimensions, facilitating better information sharing across channels. The U-Net architecture is central to the diffusion model, comprising an encoder and decoder. The encoder downsamples the input image to extract features, while the decoder upsamples these features to reconstruct the image. Both the encoder and decoder consist of several convolutional and activation layers. These components have not been altered from the original implementation.

A significant modification is introduced in the TimeEmbedding class to incorporate class labels. The one-hot encoded class labels are concatenated with the time step embeddings. This modified embedding is then processed through two linear layers with SiLU activation to produce a time embedding that conditions the diffusion process on both temporal and class-specific information.

The DiffusionModel class is updated to utilize the modified TimeEmbedding. The forward method now takes an additional input for the class labels. During the forward pass, the encoder processes the input image, and the resulting latent representation is combined

with the time embedding that includes the class information. This conditioned latent representation is then passed through the decoder to reconstruct the image.

The model is trained on the MNIST dataset using a custom training loop. The loss function is Mean Squared Error (MSE), and the optimizer is AdamW. During training, the model's performance is periodically evaluated, and the best model based on the lowest loss is saved.

The training losses over iterations (100 epochs) are plotted to visualize the learning process. The plot shows a steady decrease in loss, indicating effective learning.

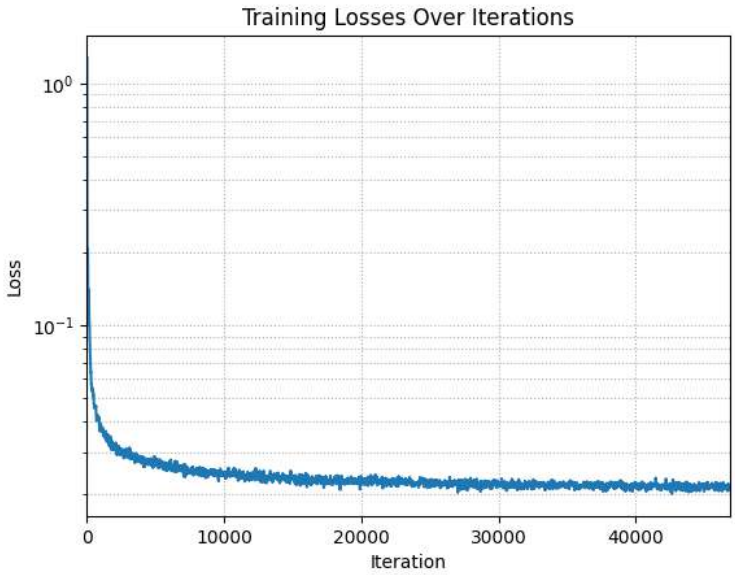

Figure 5: Training Losses Over Iterations (Conditional Diffusion Model)

The model is used to generate samples of digits conditioned on specific class labels. The generated samples demonstrate the model's ability to produce digit images corresponding to the given labels.

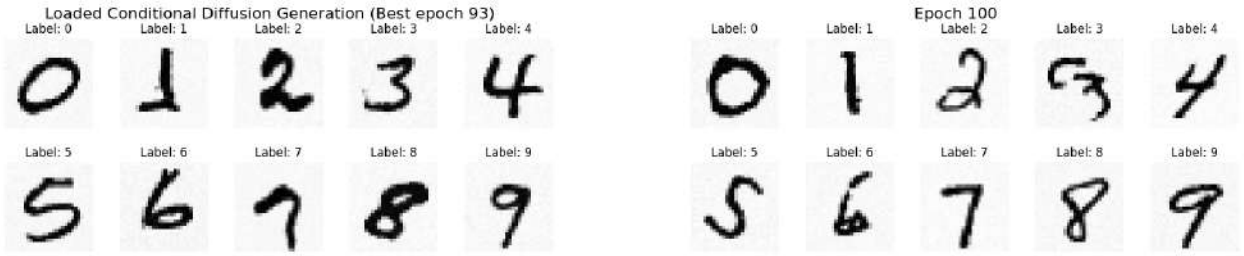

Figure 6: On the left is the generated images for the best epoch (93), where there is Training loss = 0.02. On the right there is the generated images for the last epoch (100), where there is Training loss = 0.022

The training process shows effective learning, and the generated samples validate the model's conditioning capability.

b)

This section focuses on the implementation of a classifier-free conditional diffusion model for the MNIST digit dataset. The classifier-free approach enhances the model's flexibility by allowing it to generate both conditional and unconditional samples without the need for an external classifier.

The noise scheduler, the ChannelShuffle and the U-Net architecture (including encoder, decoder and time_embedding) remain unchanged from the previous implementation. The DiffusionModel class is updated to implement classifier-free guidance. During training, the model randomly drops the conditioning labels with a specified probability (drop_prob = 0.1). This allows the model to learn to generate both conditional and unconditional samples. During sampling, the combined score from the conditional and unconditional models is used to guide the generation process. This combined score is computed as follows:

$$\epsilon_\theta(z_\lambda, c) = (1 + w)\epsilon_\theta(z_\lambda, c) - w\epsilon_\theta(z_\lambda)$$

where:

- $\epsilon_\theta(z_\lambda, c)$ is the score estimate from the conditional model.
- $\epsilon_\theta(z_\lambda)$ is the score estimate from the unconditional model.
- w is the guidance weight (w=0.5).

The updated latent variable is then calculated using the combined score.

The loss function is Mean Squared Error (MSE), and the optimizer is AdamW. During training, the model's performance is periodically evaluated, and the best model based on the lowest loss is saved. The training losses over iterations are plotted to visualize the learning process. The plot shows a steady decrease in loss, indicating effective learning.

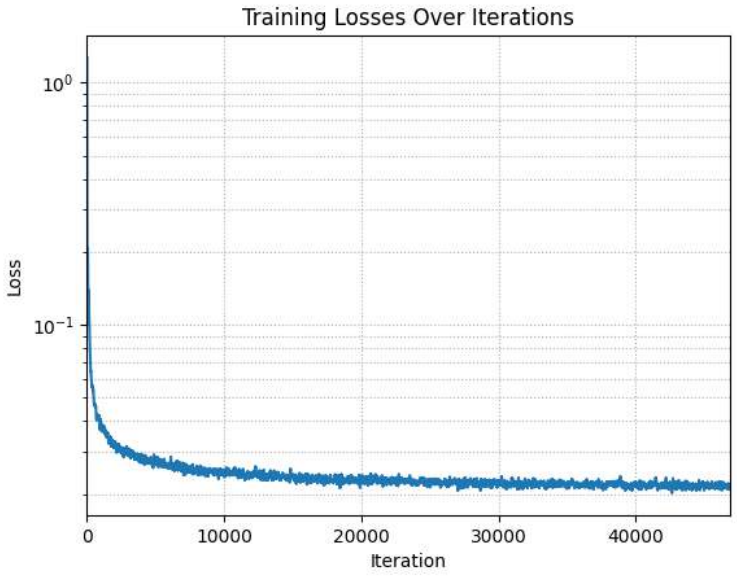

Figure 7: Training Losses Over Iterations (Conditional Diffusion Classifier free Model)

The classifier-free guidance method effectively combines conditional and unconditional scores to refine the generated samples.

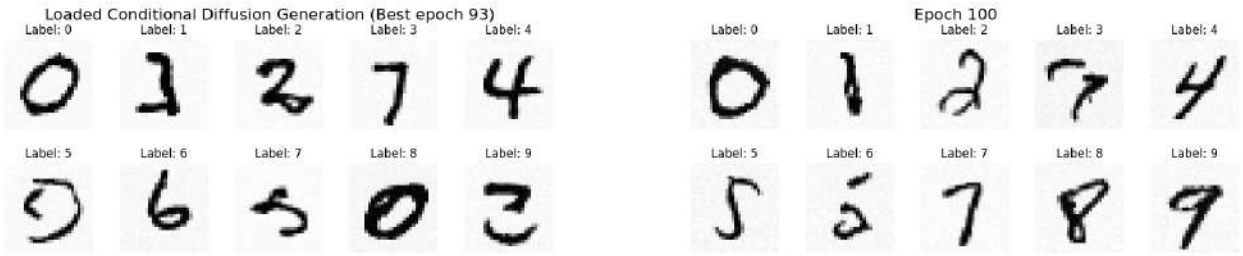

Figure 8: On the left is the generated images for the best epoch (93), where there is Training loss = 0.02. On the right there is the generated images for the last epoch (100), where there is Training loss = 0.021

The classifier-free diffusion model , as shown above, shows some confusion in the generated samples. The results were not as accurate or clear as those from the conditional diffusion model. Despite the theoretical advantages of classifier-free guidance, which aims to combine the strengths of both conditional and unconditional generation, the practical results indicated that the classifier-free model struggled with balancing these influences, leading to less accurate generation.

Problem 3. *(Create different avatars from your photo). 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.*

Solution

I used three programs to generate avatars from images of myself: two free online tools, LightX and StarryAI, and one paid tool, ChatGPT-Avatar (ChatGPT4.0). Here are the results and my assessment.

→LightX

LightX produced the best results for this exercise. By using only one picture and some presets, it quickly generated conditional avatar images based on presets such as character style, statue, superhero, etc. The process was fast, efficient, and the avatars closely resembled the original photo while adapting to the chosen styles and backgrounds.

Figure 9: Generated results using LightX for Photo n.1

Figure 10: Generated results using LightX for Photo n.2

→StarryAI

StarryAI functions more like an enhancer tool, applying a generative fill filter on top of the existing picture to create a vintage anime style (the only style with relatively good results). While it doesn't directly meet the criteria of the exercise, I found the results interesting and included them. The tool adds an artistic layer to the original image, enhancing it rather than transforming it entirely.

Figure 11: Generated results using StarryAI, vintage Anime Preset

→ ChatGPT-Avatar (ChatGPT4o)

ChatGPT-Avatar generates an inspired-by-the-photo image by using the provided photo and a prompt. This tool is more abstract compared to LightX. It incorporates elements from the original photo but doesn't maintain the raw information and characteristics of the original

face as LightX does. For example, the Pixar and cartoon avatars generated by ChatGPT-Avatar retained the details like wearing a Hawaiian shirt and being in front of leaves, while transforming them into cartoon and Pixar styles.

Figure 12: Generated results using ChatGPT 4o

All the answers to th previous exeercises are inside the provided .zip file. Below is the MEGA link where i have stored the checkpoints for the best models of the first two exercises.
→MEGA Link : ..link