

# M2 Applications of Machine Learning

CRSiD: tmb76

University of Cambridge

# Contents

# Using a Diffusion Model on the MNIST Dataset

# Chapter 1

## Training a Diffusion Model

### 1.1 Denoising Diffusion Probabilistic Model (DDPM)

#### 1.1.1 Diffusion Models

Diffusion models are a class of probabilistic latent variable models. They consist of an encoder and decoder. The encoder takes the input data and maps it to a latent space in a series of steps, resulting in a series of intermediate latent variables. The encoder is similar to variational autoencoders (VAEs) in that it maps the input data to a latent space. However, the particularity of the encoder here is that the mappings it will apply at each time step are predetermined. The key part is the decoder which is trained to learn what is the reverse process of the encoder, therefore being then able to produce samples [?, p.348].

#### 1.1.2 Denoising Diffusion Probabilistic Model (DDPM)

In this report, the writing conventions of the Prince textbook will be followed [?]. The model used for this project is a DDPM. For this model, the encoder takes in input data  $\mathbf{x}$  and maps it to a latent space  $\mathbf{z}_T$ , of the same dimensionality as  $\mathbf{x}$ , in a series of steps:  $\mathbf{z}_0 \rightarrow \mathbf{z}_1 \rightarrow \dots \rightarrow \mathbf{z}_T$ . This is defined by a Markov Chain that is known, which at each step adds Gaussian Noise following a Noise or Variance Schedule,  $\beta_1, \dots, \beta_T$ . As it is a Markov Chain, and a type of variational autoencoder, the encoder can be described by an approximate probability distribution  $q$  such that [?]:

$$q(\mathbf{z}_{1:T}|\mathbf{x}) = \prod_{t=1}^T q(\mathbf{z}_t|\mathbf{z}_{1:t-1}) \quad (1.1)$$

Where the individual step is given by:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1 - \beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}) \quad (1.2)$$

In other words,  $\beta_t$  describes how much noise is going to be added to the input data at each step  $t$ . Prince’s textbook also provides a closed form expression which shows this more clearly [?]:

$$\mathbf{z}_1 = \sqrt{1 - \beta_1} \mathbf{x} + \sqrt{\beta_1} \epsilon_1 \quad (1.3)$$

And it can be shown that after  $t$  steps, this gives:

$$\mathbf{z}_t = \sqrt{\alpha_t} \mathbf{x} + \sqrt{1 - \alpha_t} \epsilon \quad (1.4)$$

where  $\alpha_t = \prod_{s=1}^t 1 - \beta_s$  and  $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ , is a sample from a standard normal distribution, and therefore is the actual noise added. The decoder is trained to learn the reverse process of the encoder, or simply how to go from  $\mathbf{z}_T$  to  $\mathbf{z}_{T-1}$ , continuing back through the latent variables to the input data  $\mathbf{x}$ . Coming back to the approximate probability distribution  $q$ , the decoder is trained to learn the reverse distributions  $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ . Approximating them as normal distributions, they can be written:

$$Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \phi_t) = \mathcal{N}_{\mathbf{z}_{t-1}}(\mathbf{f}_t[\mathbf{z}_t, \phi_t], \sigma_t^2 \mathbf{I}) \quad (1.5)$$

where  $\mathbf{f}_t$  is a neural network that takes  $\mathbf{z}_t$  as input and has parameters  $\phi_t$ , which here is just the timestep  $t$ . The reason why the model predicts the mean of the normal distribution with the variance being fixed to  $\sigma_t^2 \mathbf{I}$  is discussed in greater detail in the Ho et al. (2020) paper [?]. The training algorithm can then be written as follows:

#### Training Algorithm for DDPM reverse process [?]

**Input:** Data  $\mathbf{x}$   
**Output:**  $\mu_t = \mathbf{f}_t[\mathbf{z}_t, t]$   
**repeat**  
   **for**  $i \in \mathcal{B}$  **do** ▷ For each training example index in batch  
      $t \sim \mathcal{U}(1, \dots, T)$  ▷ Sample a random time step  
      $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  ▷ Sample noise  
      $L = \|\epsilon - \epsilon_\phi(\sqrt{\alpha_t} \mathbf{x} + \sqrt{1 - \alpha_t} \epsilon, t)\|^2$  ▷ Compute individual noise  
   **end for** ▷ Accumulate losses for batch and take gradient descent step  
**until** convergence

Predicting the noise in the algorithm instead of the mean is done by modifying the parameterization of  $\mathbf{f}_t[\mathbf{z}_t, t]$  [?].

## 1.2 Training the Model on the MNIST Dataset

Here, the model chosen to learn prediction of the noise is a Convolutional Neural Network (CNN). It is set to have a kernel size of 7. CNN's are often used in image data processing [?, p.161], partly since they provide a way to reduce the number of weights and biases, which becomes an issue quickly in images as they are high dimensional inputs. More importantly, image recognition or prediction requires more a knowledge of what patterns define certain objects, and this whatever the position on the image. And this is something a fully connected neural network struggles with since it does not have any notion of spatial relationships between pixels, and would need to learn what a certain object looks in every rotation/position possible. This is key in the case of the MNIST dataset where only 9 objects are considered but they are found to be extremely variable, as they are handwritten.

The activation function used is GELU, which is a Gaussian Error Linear Unit, defined as:

$$\text{GELU}(x) = x\Phi(x) = x \cdot \frac{1}{2} + [1 + \text{erf}(\frac{x}{\sqrt{2}})] \quad (1.6)$$

where  $\Phi(x) = P(X \leq x)$ ,  $X \sim \mathcal{N}(0, 1)$ . It was introduced in 2016 by Hendrycks and Gimpel [?], and introduces a mix of non-deterministic masking of inputs though still depending on the input values (see Figure ??).

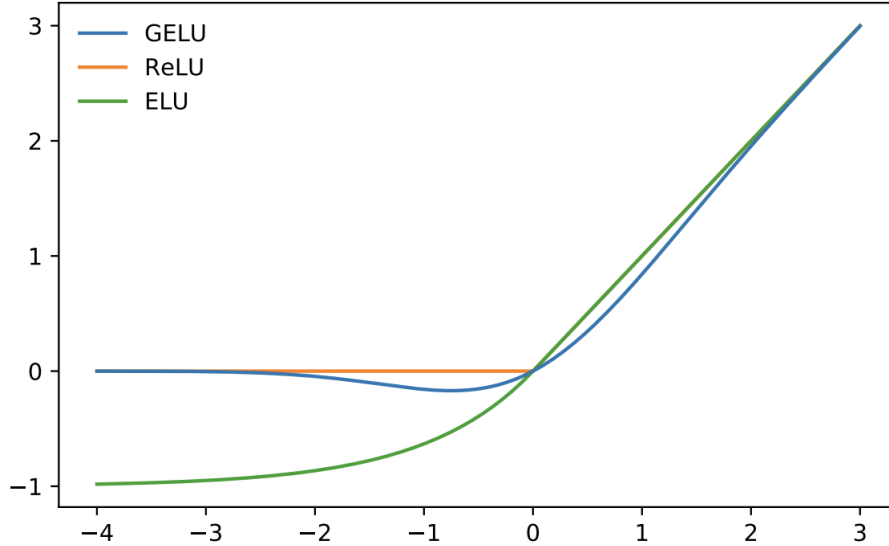


Figure 1.1: GELU activation function

The standard DDPM model was trained for 100 epochs on the MNIST dataset,

with a batch size of 128. The model was trained using the Adam optimizer with a learning rate of  $2 \times 10^{-4}$ , and the loss function used was the mean squared error.

First, the loss at each iteration was obtained and plotted in Figure ???. As can be seen the loss does decrease over time, fast at first then much slower which is expected.

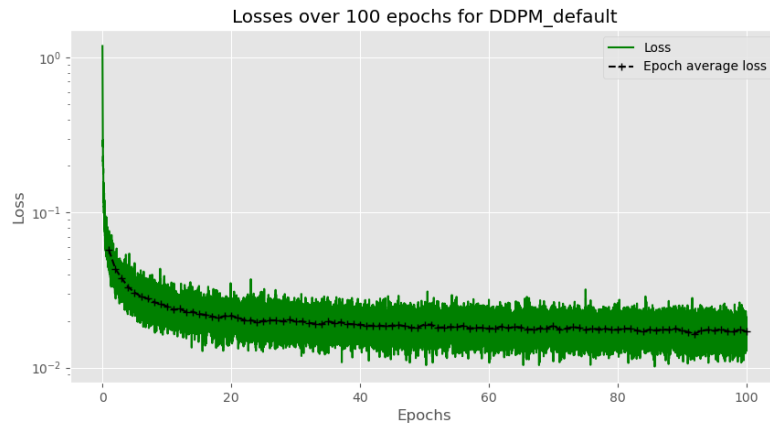


Figure 1.2: Loss at each iteration of the training process, and the average loss over each epoch

One issue that arises in the context of the MNIST dataset is that the MSE is susceptible to be very low even though the samples generated are not good. This is because the MSE is purely looking at the problem quantitatively. To get a qualitative sense of the samples generated, 16 samples were generated for different epochs using the sampling algorithm described in the Prince textbook [?, p. 363], by giving the model a pure Gaussian noise input and letting it gradually denoise it. The samples generated at epoch 1, 20, 40 and 60 are shown in Figure ??.

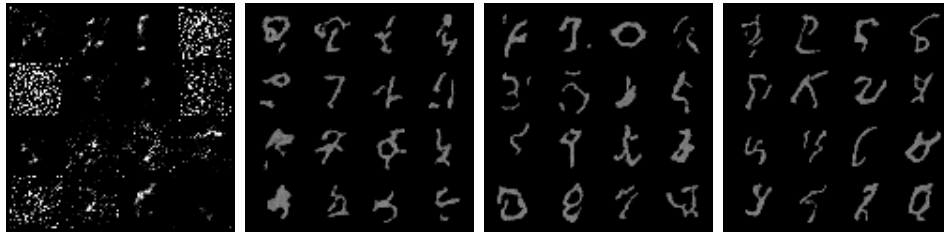


Figure 1.3: Samples generated by the DDPM model at epochs 1, 20, 40 and 60



As expected the samples generated at epoch 1 are very close to being just noise, though some samples show some patterns appearing. By epoch 20, the model is consistently generating symbols, with some resembling numbers. And it then takes a longer time to get to a point where the samples are consistently numbers. This comes back to the discussion above, as the symbols still provide a low MSE, but are not good samples, and the model is purely based on MSE.

To quantitatively evaluate the quality of the samples, the Fréchet Inception Distance (FID) score was used. The FID score measures how similar two sets of images are by comparing the feature representations of the images [?]. The lower the score the better. The FID score was calculated for the samples generated at each epoch, and the results are shown in Figure ?? . A more robust estimate is computed once all epochs are run, on a larger sample of generated images.

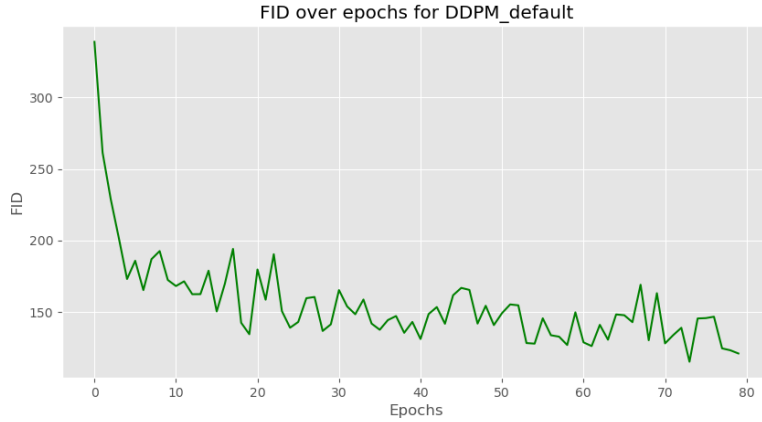


Figure 1.4: FID score at each epoch of the training process

As expected, the FID score decreases over time. However, these still show that the model is not generating good samples, as the FID score is still quite high. And even at the end of training for the last epoch, the FID score was ... .

Comparing to figures obtained in the Ho et al. (2020) paper [?], or the Bansal et al. (2022) paper [?], it can be seen that the score obtained here indicates quite bad performance. However, it is important to note that the model was trained for only 100 epochs, with quite a shallow CNN. Further, the FID here was used regardless of the digit, which may be different to in the papers mentioned. The rationale here is that the FID score being based on the distribution of computer vision features [?], it should still be able to capture the quality of the samples generated when using it for all digits together. Finally, it is not a perfectly objective metric so it will, mainly be used for comparison of the different models trained in this report. In that aspect, it will be a more robust metric.

Additionally, the Inception Score (IS) was calculated for the samples generated at each epoch. The IS is a metric that measures the quality of generated images, by looking at the diversity and quality of the generated images. More specifically, and in this context, it will measure the variety of images/digits generated, but also how much each image looks like a digit. The IS score has lowest value 1.0 and the larger it is, the better.

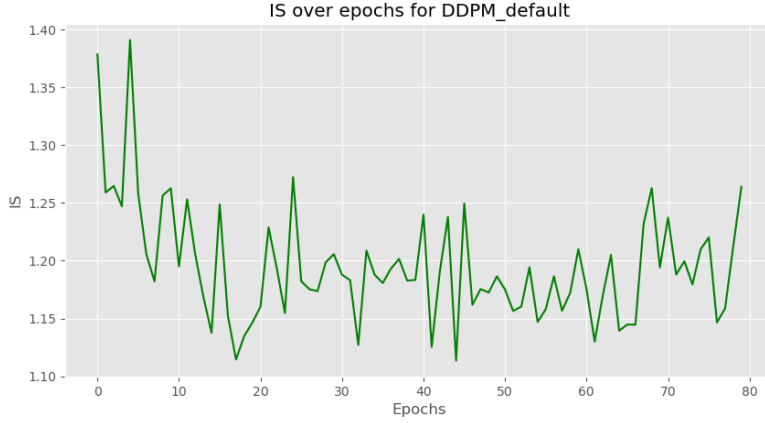


Figure 1.5: IS score at each epoch of the training process for the DDPM model

Figure ?? shows the IS score at each epoch of the training process. And the final epoch score was found to be: . Here, the IS score decreases over the epochs, which is not what is expected. Once again, this result has to be considered with caution, as because of computing efficiency, it is computed with a small sample of generated images. The more important point is that this metric only compares the generated images to themselves, so it is only a measure of the diversity and quality of the generated images.

### 1.3 Running the Model for Different Hyperparameters

For the previous trained mode, a set of hyperparameters were used (see Table ??). The noise schedule  $\beta$ 's are set by a tuple, which sets the range of values that the noise can take. They are then defined as:  $\beta_t = \frac{(\beta_2 - \beta_1) \times t}{T + \beta_1}$ , for  $t = 0, \dots, T$ . The number of timesteps is set to 1000, and the CNN was set to have 4 hidden layers with 16, 32, 32, and 16 hidden units respectively.

In this section, another set of hyperparameters is chosen and the DDPM model is trained again with these hyperparameters. The new hyperparameters are shown

Hyperparameter	Value/Choice
$\beta$ 's	$(10^{-4}, 0.02)$
Number of timesteps	1000
Learning Rate	$2 \times 10^{-4}$
Number of hidden layers & units	(16, 32, 32, 16)
Batch Size	128
Activation function	GELU

Table 1.1: Hyperparameters used for the training of the DDPM model

in Table ??.

Hyperparameter	Value/Choice
$\beta$ 's	$(10^{-4}, 0.02)$
Number of timesteps	100
Learning Rate	$6 \times 10^{-4}$
Number of hidden layers & units	(16, 32, 32, 16)
Batch Size	128
Activation function	SELU

Table 1.2: New hyperparameters used for the training of the DDPM model

By making the number of timesteps smaller, the model will have a "steeper" learning curve, as it will have to learn to denoise the image in fewer steps. The learning rate is increased to  $6 \times 10^{-4}$  to encourage the model to learn faster. Now, whether this learning will be in the right direction is another question, but the idea is to see if the model can do well with a harder set of hyperparameters. Finally, the activation function is changed to SELU, which is a Scaled Exponential Linear Unit, which resembles the GELU function but without the "dip" below 0 (see Figure ??). Overall, these hyperparameters are chosen to see if the model can perform as well with a harder set of hyperparameters, or if more capacity is needed if the model is to learn well.

Again, the model was trained for 100 epochs, and the loss at each iteration was obtained and plotted in Figure ??.

The same decrease of the loss can be seen here though it plateaus at a higher level. Looking at the samples generated at epochs 1, 20, 40 and 60, shown in Figure ??, it can be seen that the model is generating symbols at the same times as in the previous model, but there is no improvement towards those symbols becoming digits. Even at the latest epoch, the samples are still far from looking like digits (Fig ??).

The FID score at each epoch is shown in Figure ??.

The FID score decreases over time, though it does end up higher than the previous model. The final FID score

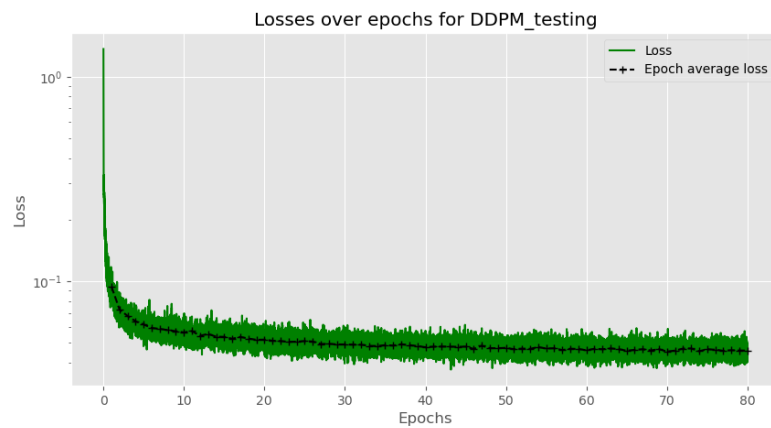


Figure 1.6: Loss at each iteration of the training process, and the average loss over each epoch

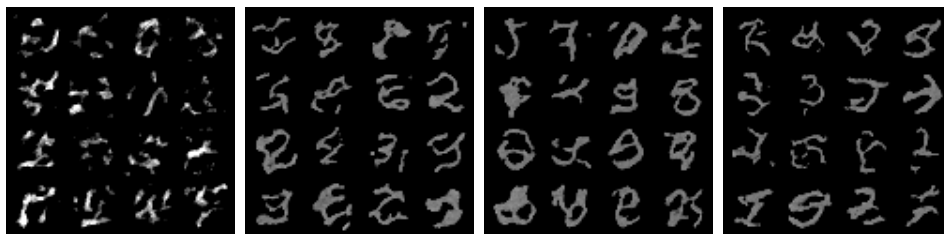


Figure 1.7: Samples generated by the DDPM model at epochs 1, 20, 40 and 60

was found to be ... . The IS score at each epoch is shown in Figure ??, and the final IS score was found to be ... .

One thing worth noting is that the FID shows consistency with the previous model run.



Figure 1.8: Samples generated by the DDPM model at epoch 100

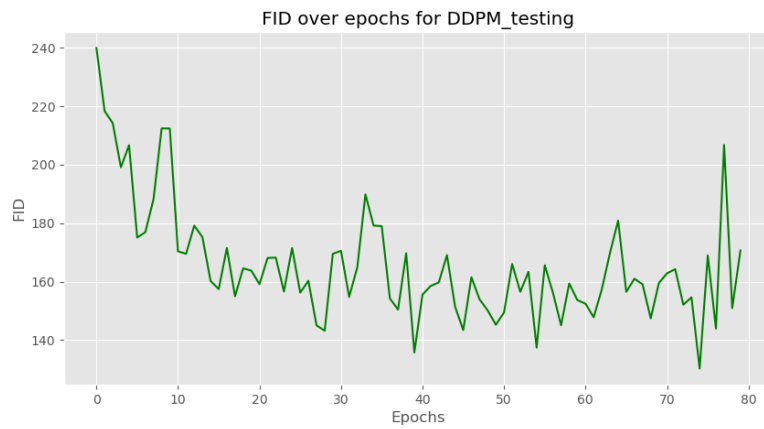


Figure 1.9: FID score at each epoch of the training process

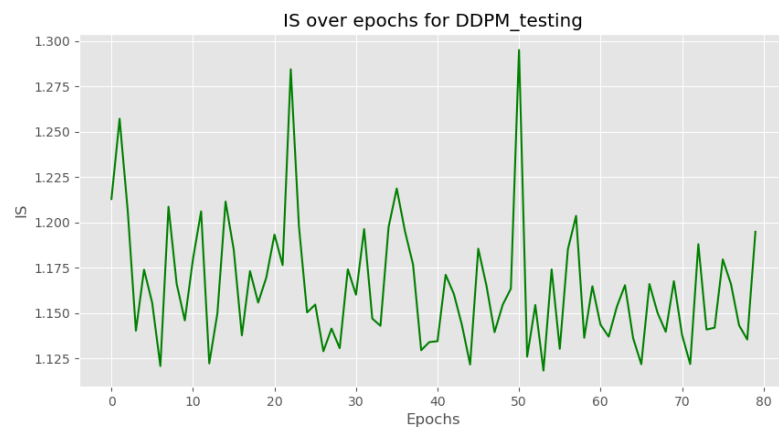


Figure 1.10: IS score at each epoch of the training process for the DDPM model

## Chapter 2

# Custom Degradation Function

In Bansal et al. (2022, [?]), a conceptual summary of degradation functions is given. Starting with image  $\mathbf{x} \in \mathbb{R}$ , the degradation of the image, or forward process of the encoder for the DDPM described in section 1.1.2, can be considered as follows:  $\mathbf{x}_t = D(\mathbf{x}, t)$ , where  $D$  is the degradation operator and  $t$  is the severity of the degradation. In other words,  $D(\mathbf{x}, 0) = \mathbf{x}$ . In Chapter 1, the  $D$  operator consisted of adding Gaussian noise with variance described by the Variance/Noise Schedule  $\beta_{1,...,T}$ . In this chapter, a custom degradation function, or operator  $D$ , will be described and used to train the model on the MNIST dataset. The important part of the degradation function is that an inverse process,  $R$ , is required to invert  $D$  and satisfies:  $R(\mathbf{x}_t, t) \approx \mathbf{x}$ , and  $R(\mathbf{x}_t, 1) \approx \mathbf{x}_{t-1}$ . The degradation function will be described in the next section. And as was discussed in Chapter 1, this is implemented through a neural network parameterized by  $\phi$  and trained to minimize the loss  $L = \|\mathbf{x} - R_\phi(D(\mathbf{x}, t), t)\|$ , taking an  $l_1$  norm [?].

### 2.1 A Row/Column Averaging degradation function

Taking inspiration from the super-resolution degradation function described in Bansal et al. (2022) [?], a degradation function that averages the rows and columns of the image is proposed. An example of the degradation function for columns is described below:

### Column Averaging Degradation Function in the forward process

**Input:** Data  $\mathbf{x}$   
**Output:**  $\mathbf{x}_0 = \mathbf{f}_t[\mathbf{z}_t, t]$   
**repeat**  
**for**  $i \in \mathcal{B}$  **do** ▷ For each training example index in batch  
 $t \sim \mathcal{U}(1, \dots, T)$  ▷ Sample a random time step  
**for**  $j \in \text{Column Schedule}[1, \dots, t]$  **do** ▷ Degradation of columns  
 $\mathbf{z}_t[:, j] = \frac{1}{28} \sum_{k=1}^{28} \mathbf{x}[k, j]$   
**end for**  
 $L = \|\mathbf{x} - \mathbf{f}_t[\mathbf{z}_t, t]\|^2$  ▷ Predict the original image  
**end for** ▷ Accumulate losses for batch and take gradient descent step  
**until** convergence

In this forward process the column or row schedule is defined in 3 ways:

- Randomly: an order of rows covering all values from 1 to 28 is randomly chosen
- Outside-In: [1,28,2,27,3,26,...]
- Inside-Out: [14,15,13,16,12,17,...]

The degradation function then, given a time step  $t$ , will average the first  $t$  rows/-columns listed in the schedule. The loss is then calculated as the MSE between the original image and the image predicted by the model from that time step, and the model is trained to minimize this loss.

For the inverse process, algorithm 2 of the Bansal et al. (2022) paper is used, as it gives better performance for cold diffusion methods [?, p. 4]. The inverse process is described as follows:

### Inverse Process for Column Averaging Degradation Function

**Input:** Degraded Data  $\mathbf{z}_T$   
**Output:** Prediction of original sample  $\mathbf{x}$   
**for**  $s = T, T-1, \dots, 1$  **do** ▷ For each time step in reverse order  
 $\hat{\mathbf{x}} \leftarrow \mathbf{f}_T[\mathbf{z}_T, T]$  ▷ Make a direct prediction  
 $\mathbf{x}_{s-1} = \mathbf{x}_s - \mathbf{D}(\hat{\mathbf{x}}, s) + \mathbf{D}(\hat{\mathbf{x}}, s-1)$  ▷ Predict the previous time step's image  
**end for**

Where  $\mathbf{D}$  is the degradation function, so  $\mathbf{D}(\hat{\mathbf{x}}, s)$  is the image  $\hat{\mathbf{x}}$  degraded at time step  $s$ , with the first  $s$  rows/columns in the schedule order averaged. The degradation was originally made to average rows/columns by groups of 4, resulting in a degraded image with 7 rows/columns. However, these led to too much information lost and more importantly only 7 time steps over which the model could learn how to de-average the rows/columns. With each row/column averaged on their own, this gives



28 time steps, which should allow the model to learn better how to de-average the rows/columns.

## 2.2 Training the modified model on the MNIST dataset

The non-grouped column averaging model is discussed here as it gave the best result. The model was trained for ... epochs, with the following hyperparameters:

Hyperparameter	Value/Choice
Row/Column Schedule	Random
Number of timesteps	28
Learning Rate	$2 \times 10^{-4}$
Number of hidden layers & units	(16, 32, 32, 16)
Batch Size	128
Activation function	GELU

Table 2.1: Hyperparameters used for the training of the column averaging cold diffusion model


As before, samples are generated for epochs 1, 20, 40 and 60, and the results are shown in Figure ?? . ... get numbers quickly but only certain ones. discussion about direction depemndent information. Model is therefore biased...

In terms of the numbers...The loss at each iteration was obtained and are plotted together in Figure ??.

The FID score at each epoch is shown in Figure ??, and the final FID score was found to be ... . The IS score at each epoch is shown in Figure ??, and the final IS score was found to be ... .

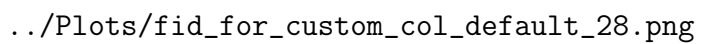
## 2.3 Comparing with the original model

Evaluate the fidelity of the samples generated by the two models, discussing any differences between them.



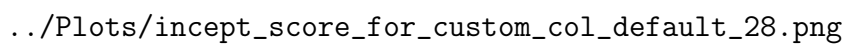
`../Plots/losses_for_custom_col_default_28.png`

Figure 2.1: Loss at each iteration of the training process, and the average loss over each epoch



../Plots/fid\_for\_custom\_col\_default\_28.png

Figure 2.2: FID score at each epoch of the training process



../Plots/incept\_score\_for\_custom\_col\_default\_28.png

# Chapter 3

## Appendix

### 3.1 Other versions of the degradation function

Here is shown results obtained for the different versions of the degradation function.

plot of original, degraded, direct sampled for col7

final loss, FID, IS for col7

plot of original, degraded, direct sampled for row7

final loss, FID, IS for row7

plot of original, degraded, direct sampled for row28

final loss, FID, IS for row28

### 3.2 AI tools use

Ask chatgpt to summarise your prompts