Prepared for submission to University of Cambridge

M2: Applications of Machine Learning - Coursework Assignment

Adnan Siddiquei

University of Cambridge

E-mail: as3438@cam.ac.uk

Contents

1 Overview of Diffusion Model Implementation

1.1 coursework_starter.ipynb

The code provided in the coursework_starter.ipynb notebook trains a regular denoising diffusion probabilistic model (DDPM) on the MNIST dataset, using the `pytorch` library. This section provides a detailed explanation of how the code given coursework_starter.ipynb code works. The notation used in this section adheres to the conventions used in Ch.18 of Understanding Deep Learning, Prince [1].

1.1.1 The data

The code loads the MNIST dataset (60000 images) and preprocesses it with `transforms.toTensor` which convert the image to a tensor and scales it to the range $[0, 1]$, followed by `transforms.Normalize((0.5,),(1.0))` which shifts the range to $[-0.5, 0.5]$. The scaling and range shift can help improve the training dynamics. Given that MNIST images are grayscale (a single channel), the input dimensions are 1x28x28. The dataset is then loaded into a `DataLoader` object with a batch size of 128, yielding 468 batches per epoch.

1.1.2 The CNN Model

The code defines a Convolutional Neural Network `CNN` class which is the neural network that is used to learn the diffusion process - the noise that is added to the image at each step. This is instantiated with 5 hidden layers, each outputting: 16, 32, 32, 16, 1 channels respectively, with a kernel size of 7 at the first 4 layers, and a kernel size of 3 at the last layer. Each of the first 4 hidden layers (defined in the `CNNBlock` class) are composed of a 2D convolutional layer (with 0-padding on the edges) followed by a normalisation layer (`LayerNorm` which normalises each input to standard normal) and a GELU activation function. The normalisation stabilises the activations and places them in the transition range of the GELU function, which allows for full use of the GELU's characteristics.

Decoder models for the diffusion process can either be a set of multiple models, each trained to revert the noise at a particular step, or a single model trained to revert the noise at any given step by injecting information about the time step into the model. This CNN is the latter, and the `forward` function adds a time step dependent embedding to the pre-activations of the second layer so that the CNN can learn to revert the noise at any given step.

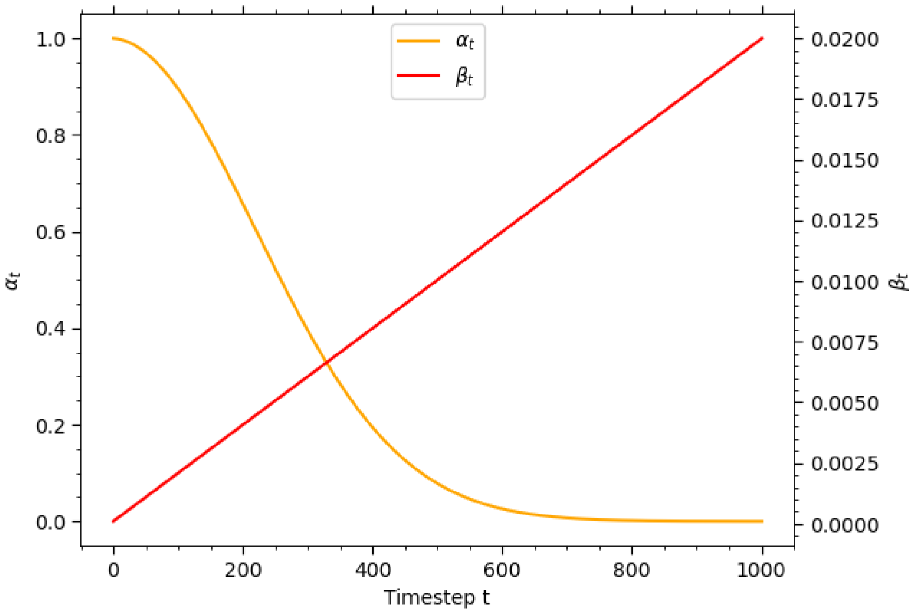


Figure 1: A plot of β_t and α_t for the linear noise schedule used in coursework_starter.ipynb. This figure illustrates that MNIST images transition for noiseless to standard normally distributed noise non-linearly over 1000 steps, with most of the noise being added by about the 600'th step.

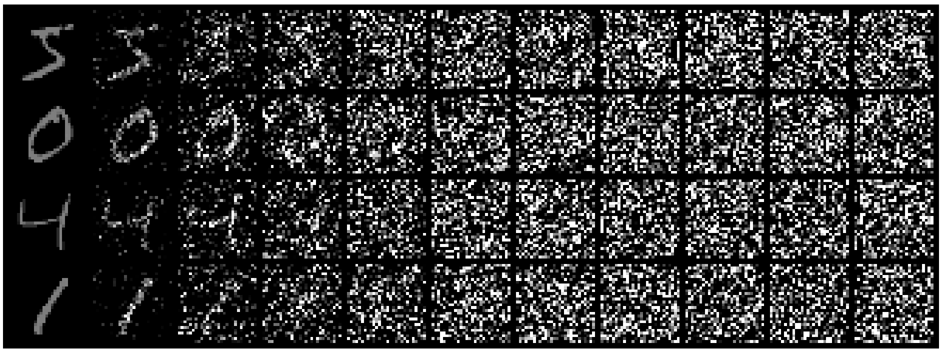


Figure 2: An illustration of the linear noise schedule applied to the first 4 MNIST images in the datastet. The first column shows the original 28x28 image, and the subsequent 10 columns show the image after 100 steps of the diffusion process. The final column shows the image after 1000 steps.

1.1.3 The DDPM Model

The training and sampling process is defined within the `DDPM` class, which contains a `forward` function that defines the forward pass of the model and a `sample` function that generates new MNIST data samples by iterating backwards through the diffusion process. The DDPM is instantiated with the CNN defined above and a linear noise schedule with β_t in the range $[10^{-4}, 0.02]$ across 1000 steps, as shown in Figure (1), where β_t and α_t are as defined in Ch.18 Prince [1].

The `forward` function samples 128 (the batch size) random α_t values from the noise

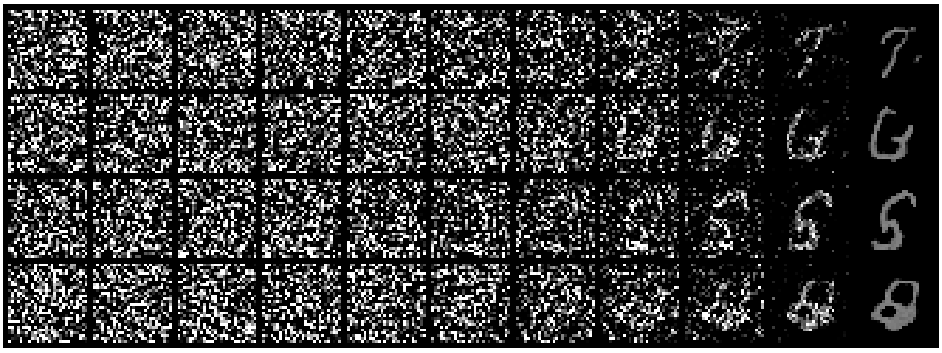


Figure 3: An illustration of the generation process. This is the generation of 4 new MNIST samples from the model trained with the default parameters in the coursework_starter.ipynb notebook. The first column shows the randomly sampled noise, the next 10 columns illustrate the image after 100 steps of decoding, with the final column showing the final generated image.

schedule and a standard normally distributed noise tensor of shape `(128,1,28,28)`, and degrades every image in the batch with the noise according to Equation (1.1) [1].

$$z_t = \sqrt{\alpha_t} \cdot \mathbf{x} + \sqrt{1 - \alpha_t} \cdot \epsilon \tag{1.1}$$

Figure (2) illustrates the diffusion process. This batch of 128 noisy MNIST images (degraded to different extents) is then passed through the CNN (along with the corresponding time steps) to make a prediction on the noise that was added to the image at the corresponding step. The function returns the loss between the actual noise and this predicted noise as the mean squared error between the two.

The `sample` function can generate new MNIST samples by iterating backwards through the diffusion process using the learned parameters of the CNN, and a random sample of standard normal noise. Figure (3) illustrates the generation process.

1.1.4 The training loop

The training loop iterates over 100 epochs, using the Adam optimiser and a learning rate of 2×10^{-4}, saving generated samples from the model at every epoch along with the loss at each epoch.

2 Training the model

This section discusses the training of the DDPM model with varying linear noise schedules.

Three different models were trained, the first model (default model) was trained with the provided noise schedule: $\beta_t = [10^{-4}, 0.02]$ with 1000 steps; the second model (the "long model") was trained with a longer noise schedule: $\beta_t = [10^{-4}, 0.004]$ with 5000 steps which added less noise at each step; and the third model (the "short model") was trained with a shorter noise schedule: $\beta_t = [10^{-4}, 0.1]$ with 200 steps which added more noise at each step. The number of steps was amended for each model such that the final noise level was the same for each model (α_t of the order of 10^{-5} at the final time step). All models were also trained for 100 epochs with a batch size of 128, with the Adams optimiser, a learning rate of $2\ times 10^{-4}$ and the same CNN architecture described in Section (1). Figure (4) illustrates the noise schedules used for each model, Figure (5) shows the training loss for each model

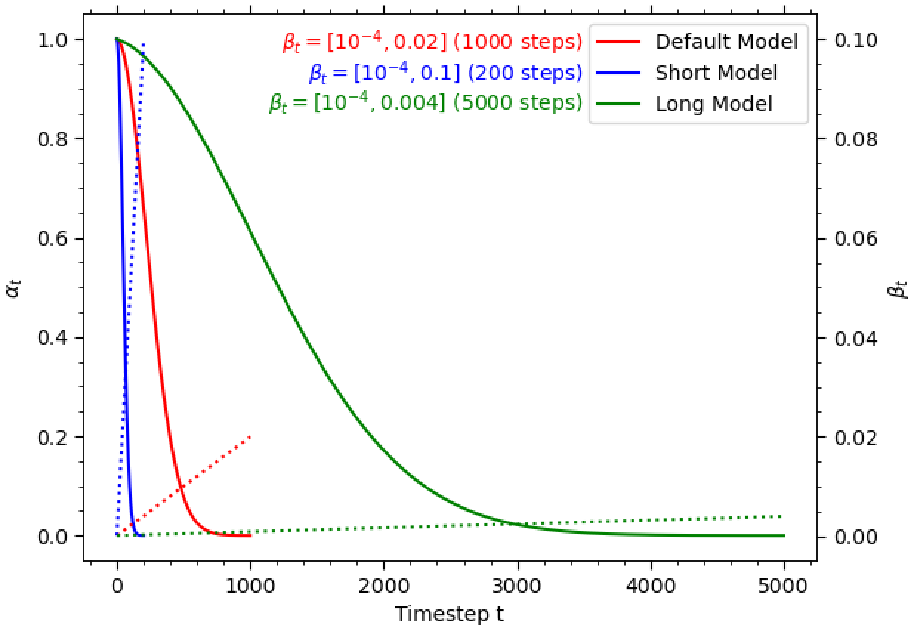


Figure 4: A plot of each noise schedule that was evaluated. The dotted lines represent the β_t values and the solid lines represent the α_t values.

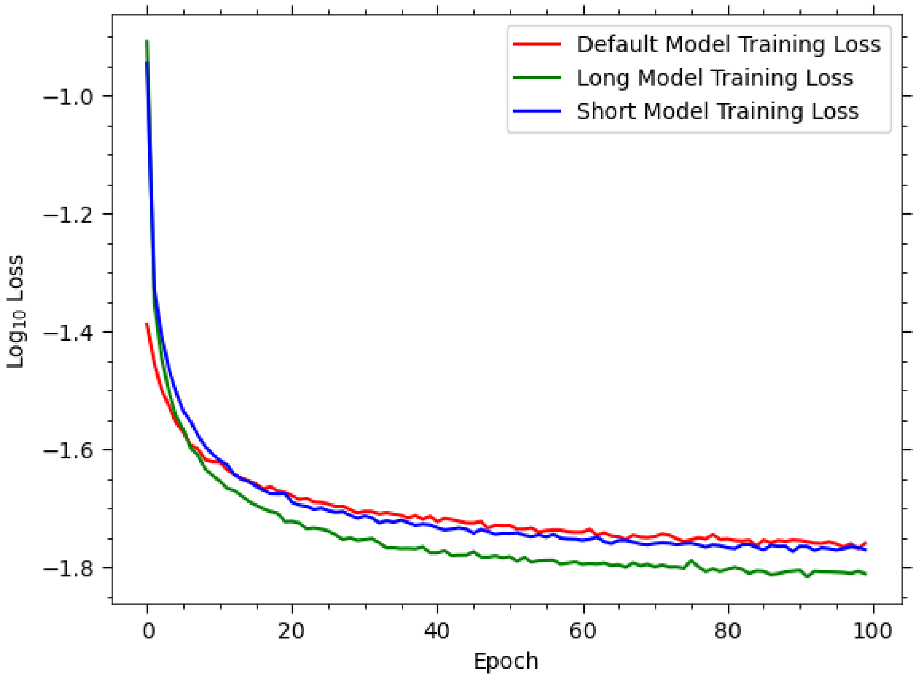
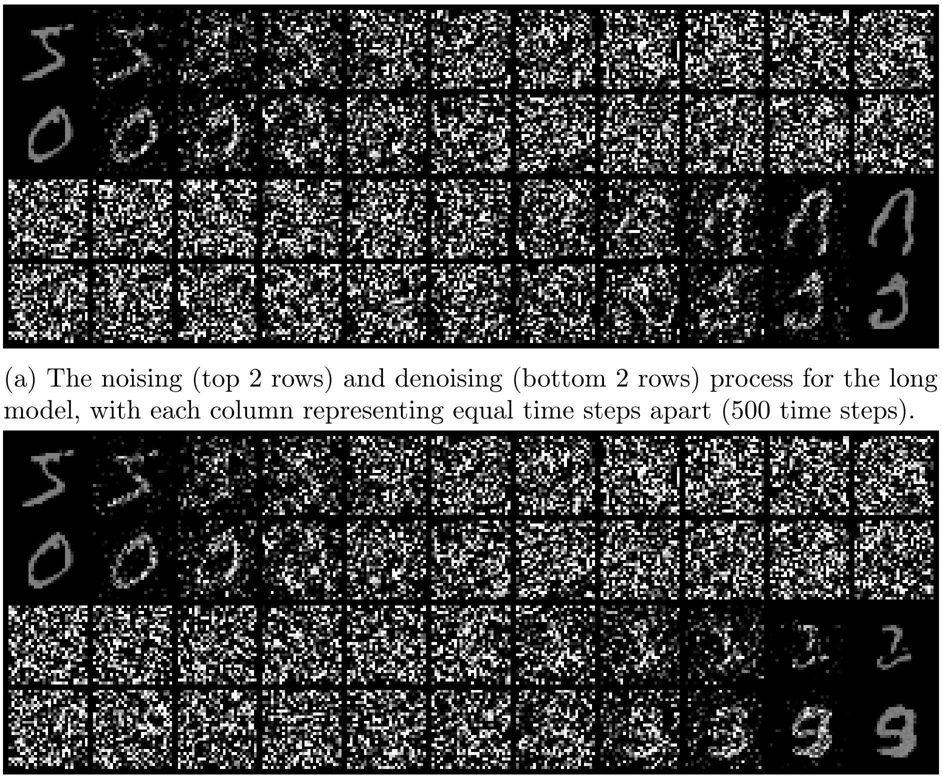


Figure 5: A plot of the training loss for each model over 100 epochs.

over 100 epochs, and Figure (6) illustrates the encoding and decoding process for the long and short models. The Frechet Inception Distance (FID) was calculated for each model as the model was trained, as shown in Figure (7). Traditionally, the FID is calculated with



(a) The noising (top 2 rows) and denoising (bottom 2 rows) process for the long model, with each column representing equal time steps apart (500 time steps).

(b) The noising (top 2 rows) and denoising (bottom 2 rows) process for the short model, with each column representing equal time steps apart (20 time steps).

Figure 6: The noising and denoising process for the long and short models. These images illustrate the same process illustrated for the default model in Figure (2) and Figure (3).

the InceptionV3 model, however, given the narrow context of the MNIST dataset, a simple CNN classifier model was trained on the MNIST dataset and the 1024-length feature vector at the penultimate layer of this classifier was used to calculate the FID instead. The MNIST classifier was trained for 7 epochs and attained an accuracy of 99.2% on the test set, which meant the penultimate layer was a good feature extractor for the MNIST dataset. Figure (7) illustrates that neither model achieved any significant improvement over the others over the 100 epochs, and as expected, the loss curve on Figure (5) matches perfectly with the FID curve.

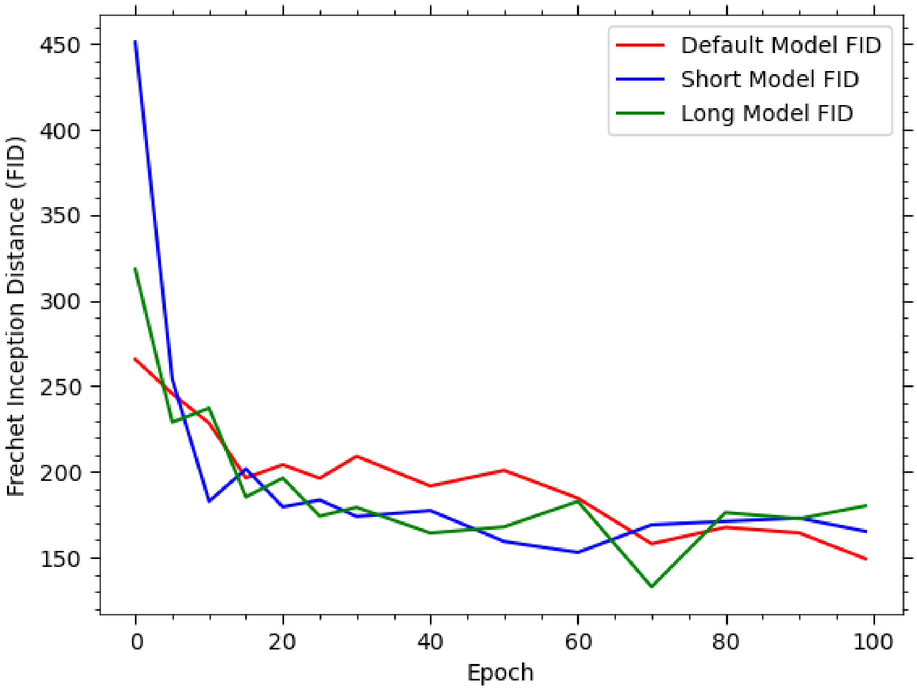


Figure 7: A plot of the Frechet Inception Distance (FID) for the generated samples from each model over 100 epochs. The FID was calculated using the first 512 samples from the MNIST dataset compared against 32 samples generated by each model at varying epochs.

References

[1] Simon J.D. Prince, *Understanding Deep Learning*. The MIT Press, 2023. Available at https://udlbook.github.io/udlbook/ [Accessed: 20-Mar-2024].