

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](mailto:as3438@cam.ac.uk)

---

## Contents

<b>1</b>	<b>Training a Diffusion Model</b>	<b>1</b>
1.1	coursework_starter.ipynb	1
1.1.1	The data	1
1.1.2	The CNN Model	1
1.1.3	The DDPM Model	2
1.1.4	The training loop	2

---

## 1 Training a Diffusion Model

### 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].

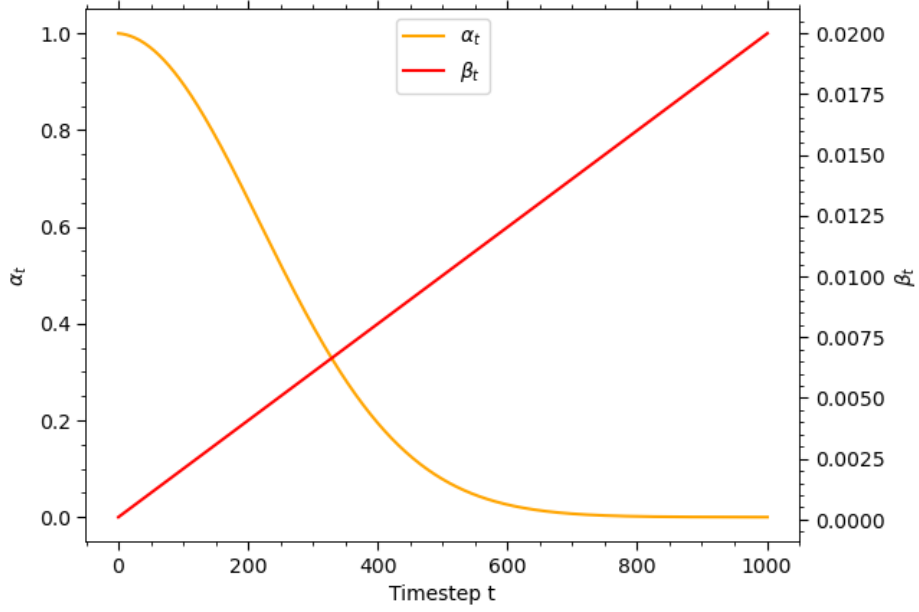
#### 1.1.1 The data

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  $1 \times 28 \times 28$ . 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  $\beta_t$  and  $\alpha_t$  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.

### 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  $\beta_t$  in the range  $[10^{-4}, 0.02]$  across 1000 steps, as shown in Figure (1), where  $\beta_t$  and  $\alpha_t$  are as defined in Ch.18 Prince [1].

The `forward` function samples 128 (the batch size) random  $\alpha_t$  values from the noise 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].

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

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.

### 1.1.4 The training loop

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

epoch.

## References

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