M2: Application of Machine Learning

Max Talberg

March 28, 2024
Word count: 2,920

Contents

0.1 Training a Diffusion Model

0.1.1 A regular denoising diffusion model

A denoising diffusion model is comprised of two phases: an encoding (forward diffusion) phase that maps data to a noise distribution and a decoding (reverse diffusion) phase designed to reverse the forward diffusion to generate data, in this instance the MNIST data set. Below is an introduction to denoising diffusion models and insight into the method used to implement such a model.

Encoding (forward process) During the encoding phase, the model maps stochastic Gaussian noise to the data over a series of steps, T. The data sample x is transformed into a series of intermediate latent variables $z_1, z_2, ..., z_T$ according to:

$$z_1 = \sqrt{1-\beta_1} \cdot x + \sqrt{\beta_1} \cdot \epsilon_1 \tag{1}$$

$$z_t = \sqrt{1-\beta_t} \cdot z_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t \quad \forall t \in \{2, \dots, T\} \tag{2}$$

where ϵ_t is noise drawn from a Gaussian distribution and β_t is a hyperparameter that determines the rate at which the noise is integrated into the data. This expression becomes:

$$z_t = \sqrt{\alpha_t} \cdot x + \sqrt{1-\alpha_t} \cdot \epsilon_t \tag{3}$$

where α_t values are the cumulative product of $(1-\beta_t)$ all the way up to T represented by, $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$. This allows for easy-to-generate samples, without worrying about the history of the evolution. This forward process is a Markov chain as the z_t only depends on the previous step z_{t-1}. This iterative process allows noise to be added to the sample data x for any given timestep t. The diffusion model must understand the encoding phase to accurately reverse this process during decoding.

Decoding (reverse process) The decoding phase, learned by the model during training, involves reversing the encoding process to reconstruct the data from noisy latent variables. During training, the model learns to reverse the noise added during the encoding phase with the aim of recovering an image from Gaussian noise. After training the model can generate data from a sample of noise, z_T to z_{T-1} moving through the latent variables until the original image x is recovered.

Training algorithm The training algorithm teaches the model this decoding process. Training simulates the encoding phase by sampling a random timestep t, which corresponds to noisy data z_t and some additional Gaussian noise ϵ. The model aims to predict the noise ϵ that was added to z_{t-1} to get z_t. This is achieved with a mean squared error (mse) loss function that minimises the difference in the predicted noise and the actual noise ϵ. The loss function is optimised through gradient descent and backpropagation to minimise this loss. The goal is for the model to accurately predict the noise ϵ added at each step, to successfully denoise a sample from data, these steps are shown in Algorithm 1 from Prince (2023).

Algorithm 1 Diffusion model training

```
Require: Training data $x$
Ensure: Model parameters $\phi_t$
1: repeat
2:     for $i \in \mathcal{B}$ do
3:         $t \sim \text{Uniform}[1, \dots, T]$                                  ▷ Sample random timestep
4:         $\epsilon \sim \mathcal{N}(0, I)$                                        ▷ Sample noise
5:         $\ell_i = \left\| g_t\left[\sqrt{\alpha_t}x_i + \sqrt{1-\alpha_t}\epsilon, \phi_t\right] - \epsilon \right\|^2$     ▷ Compute individual loss
6:     end for
7:     Accumulate losses for batch and take gradient step
8: until converged
```

Sampling algorithm The sampling algorithm utilises this trained model to generate new data from a sample of the noisy latent variable z_T. The model predicts the previous latent variable z_{T-1} by subtracting the predicted noise from the model. To simulate the stochastic nature of the forward process the new latent variable receives additional noise from a Gaussian distribution. This process of reversing the encoding is run until the noise is decoded back into data from z_T to x. These steps are illustrated in Algorithm 2 from Prince (2023).

Algorithm 2 Sampling

```
Require: Model, $g_t\ [\cdot, \phi_t\ ]$
Ensure: Sample, $x$
1: $z_T \sim \text{Norm}_z[0, I]$                                              ▷ Sample last latent variable
2: for $t = T, \ldots, 2$ do
3:     $\hat{z}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}} g_t[z_t, \phi_t]$      ▷ Predict previous latent variable
4:     $\epsilon \sim \text{Norm}_\epsilon[0, I]$                                     ▷ Draw new noise vector
5:     $z_{t-1} = \hat{z}_{t-1} + \sigma_t \epsilon$                          ▷ Add noise to previous latent variable
6: end for
7: $x = \frac{1}{\sqrt{1-\beta_1}} z_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}} g_t(z_1, \phi_1)$        ▷ Generate sample from $z_1$ without noise
```

Model The model, g_t, in this context, is a Convolutional Neural Network (CNN) that constitutes the Denoising Diffusion Probabilistic Model (DDPM). The encoding phase is used to generate training data and the decoding phase generates new samples. The model consists of a CNN and DDPM component, the CNN is a 2D model that learns to decode data and the DDPM links this together with the training and sampling algorithms to create a complete diffusion model that generates samples from the MNIST data.

Model performance The performance of a diffusion model can be assessed both qualitatively and quantitatively. Qualitatively the quality of images can be assessed at various epochs to determine how well the diffusion model is conditionally or unconditionally generating images, similarly, images can be looked at during the diffusion process at different timesteps. Quantitatively the training loss and the Fréchet Inception Distance (FID) provide a formal comparison of model performance.

The training loss, represented by line 5 in Algorithm 1, measures the difference between the noise predicted by the CNN model g_t and the actual noise ϵ. As the model is run the gradient step taken with respect to this loss will adjust the model parameters ϕ_t, to minimise this loss. The training loss tracks how well the model is learning to fit the noise distribution, this is a good measure of how well the DDPM is accurately predicting noise, lower scores indicate better image quality.

The FID characterises images as a distribution and computes the symmetric distance between the generated samples and real samples. The FID approximates both distributions by multivariate Gaussian and estimates the distance between them:

$$FID = \|\mu_r - \mu_g\|^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}}\right) \tag{4}$$

where μ and Σ represent the mean and covariance of the real and generated images and Tr is the trace of the matrix or PyTorch tensor in this case. FID is a strong indicator of the quality of generated images, particularly in this instance where it will be used to compare model performance.

0.1.2 Training models with different hyperparameters

High-level overview of the code The code consists of a CNN and DDPM. The CNN has a building block function that describes the convolutional layer, layer normalisation and an activation function. There is a CNN class that represents the main generative model, which consists of multiple CNN blocks. The DDPM has a scheduler, where β_t is tuned, which determines the rate of diffusion and the timestep which determines the number of steps T. The DDPM class encapsulates the diffusion process, including the forward pass for training and the sampling for generating new images.

Training process The diffusion model is trained in Python using PyTorch. The Python environment is set up with the necessary libraries and the MNIST dataset is loaded in as normalised tensors divided into batches and shuffled in preparation for training. The CNN model is then initialised with hidden layers $16, 32, 32, 16$.

The DDPM class is initialised with the CNN as the generative component and hyperparameters β_1, β_2 and the number of timesteps T. The Adam optimiser is configured with the DDPM model parameters and a learning rate $\alpha = 0.0002$.

The training loop iterates over all the batches for 100 epochs. The forward pass generates a noisy image at a random timestep t, the model predicts the noise ϵ from these images. The loss between the predicted noise and actual noise is calculated, the gradients are calculated through backpropagation and the Adam optimiser updates the model parameters to minimise loss.

After each training epoch, the model's performance is evaluated by generating samples at specific timesteps to depict the model's learning progress. Throughout training, the loss and evaluated samples are logged to track progress.

Hyperparameter tuning Below is an investigation into the performance of different models with the same initial default hyperparameters: A diffusion model with a CNN containing hidden layers $16, 32, 32, 16$ and Adam optimiser with a learning rate of 2×10^{-4}, trained for 100 epochs. The different models have varying β parameters and timesteps T.

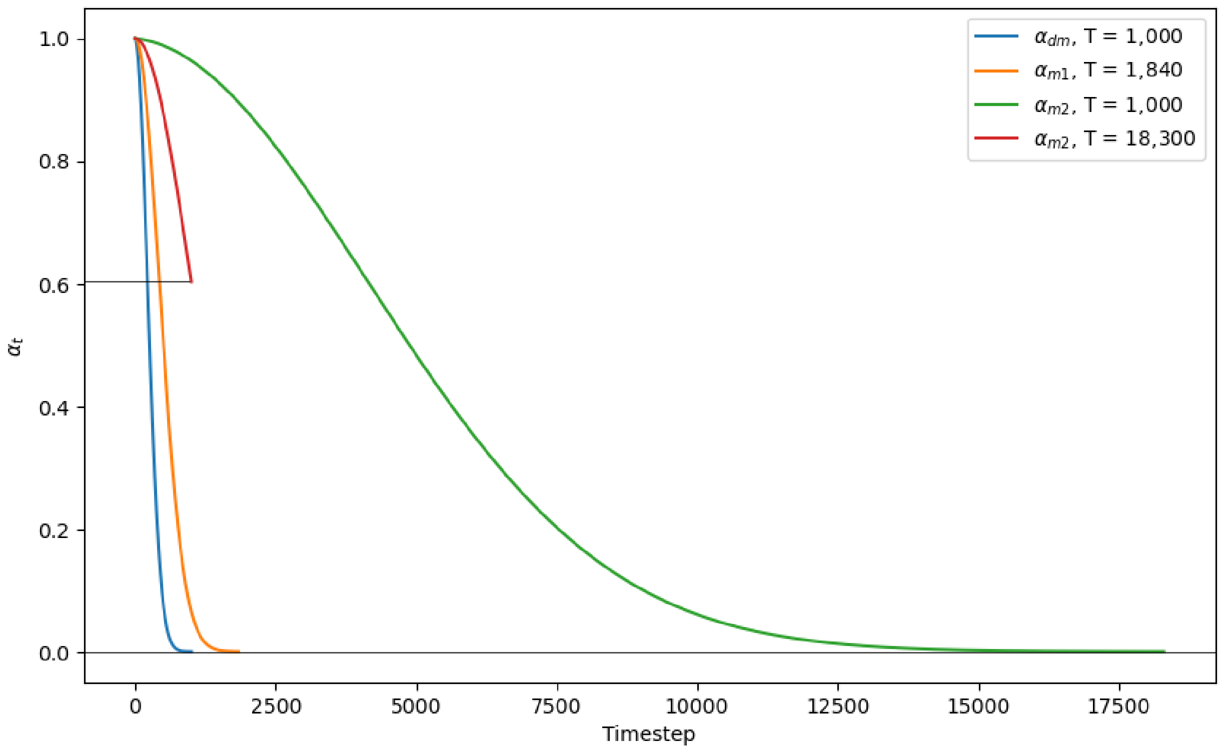

Figure 1: This figure illustrates the trajectories of the alpha parameter for three diffusion models as a function of timesteps. The standard diffusion model (α_{dm}), Model 1 (α_{m1}) and Model 2 (α_{m2}) are represented by different colors. Model 2 is plotted twice to emphasise the need for a larger number of timesteps (1,000) compared to the standard (18,300) for the alpha value to approach zero.

The forward diffusion process gradually adds noise to the data, defined by following the interpolating variance schedule between two beta values β_1 and β_2:

$$\beta_t = \frac{\beta_2 - \beta_1}{T} \cdot t + \beta_1, \tag{5}$$

where α_t is defined as:

$$\alpha_t = \exp\left(\sum_{i=0}^{t} \log(1-\beta_i)\right). \tag{6}$$

As such varying β values require different timesteps to fully denoise an image and a smaller range of diffusion steps relates to incrementally smaller additions of noise. Three models were trained: the Default Model (`diffusion_model.yaml`) with the default β parameters and two variations representing Model 1 (`model1.yaml`) and Model 2 (`model2.yaml`).

Figure 1 depicts how α_t values need different timesteps to fully encode. Model 2, initially run with $T = 18,300$, is run for $T = 1,000$ to convey this idea. From Figure 1 it's clear this model is only encoded about 40% of the way to Gaussian noise. These models are analysed qualitatively and quantitatively below.

Qualitative Results Below is a selection of samples created during the model training. The selected samples aim to provide a means of comparison of the effects of varying hyperparameters.

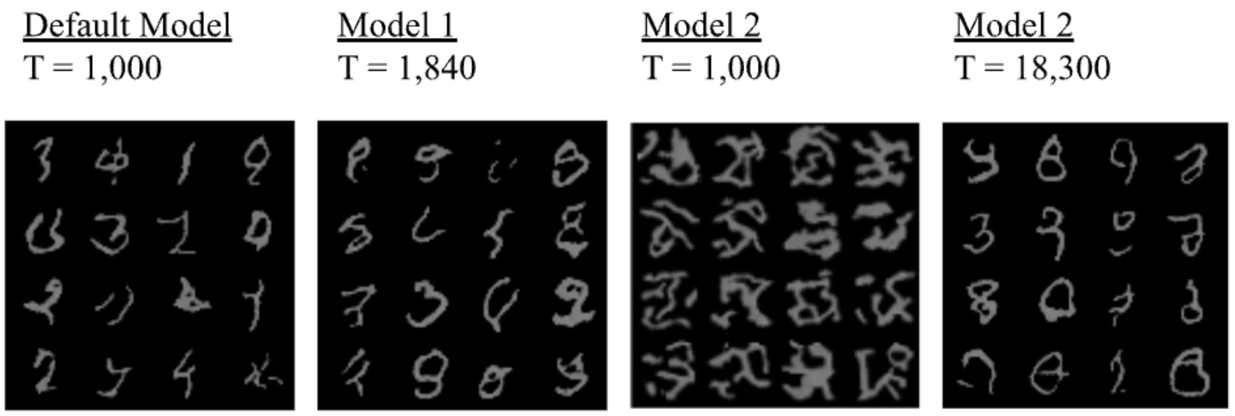

Figure 2: MNIST outcomes of a 4 x 4 grid containing 16 indices at 100 epochs for three diffusion models with distinct hyperparameters. Results are displayed from the Default Model, Model 1, and Model 2 to illustrate the effects of varying hyperparameters. The improvement in image clarity as the number of epochs grows is evident.

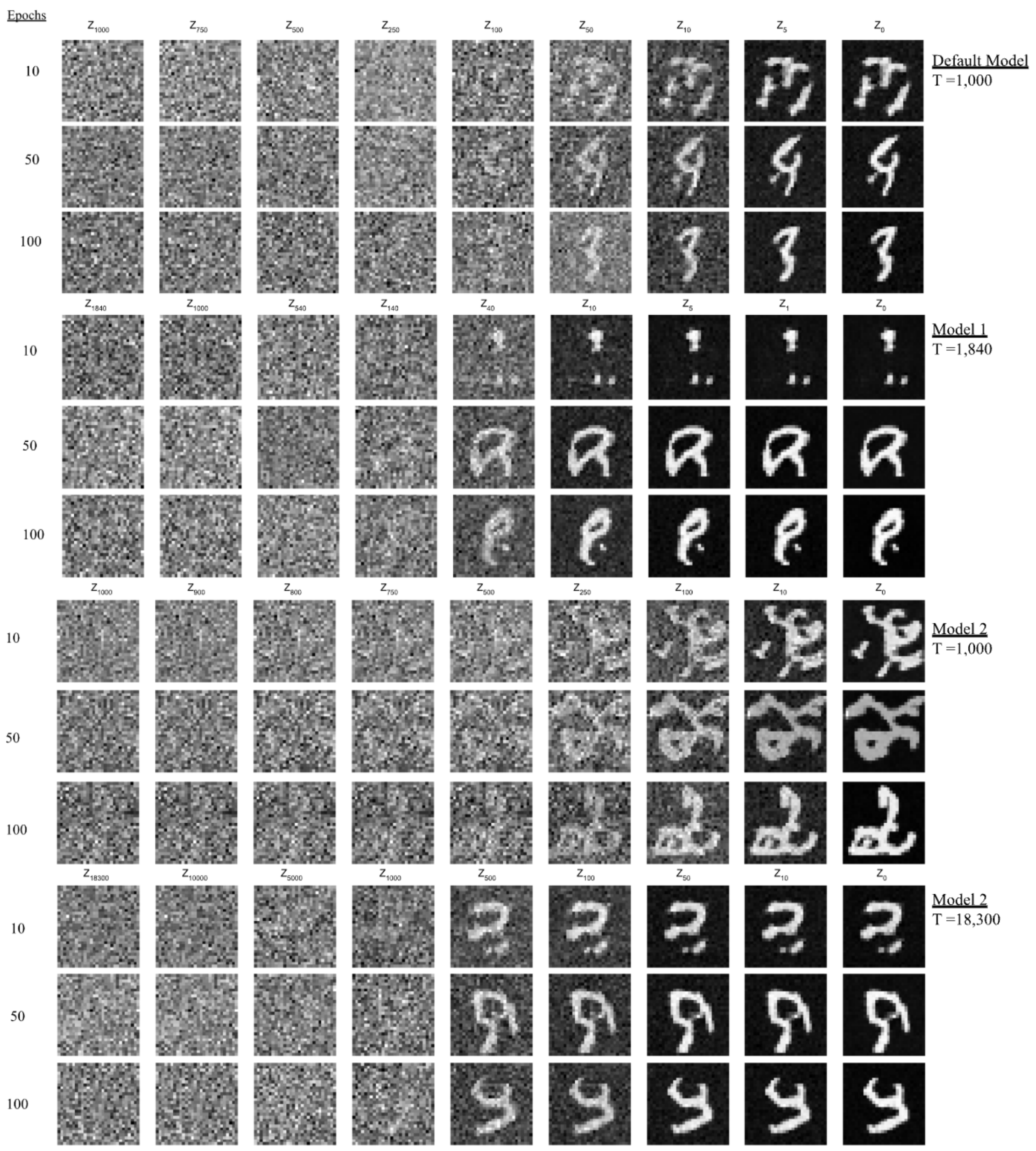
Epochs
z1000 z750 z500 z250 z100 z50 z10 z5 z0
10
50
100
Default Model
T =1,000
z1840 z1000 z540 z140 z40 z10 z5 z1 z0
Model 1
T =1,840
z1000 z900 z800 z750 z500 z250 z100 z10 z0
Model 2
T =1,000
z18300 z10000 z5000 z1000 z500 z100 z50 z10 z0
Model 2
T =18,300

Figure 3: Visual comparison of the evolution of generated digit image quality at epochs 10, 50, and 100 for diffusion models with varying hyperparameters. The rows display results from the Default Model, Model 1, and Model 2 respectively, with columns representing images at increasing epoch counts. Clearer and more recognisable digits emerge as the models progress through epochs.

Quantitative Results Below are the results of the training loss and FID scores.

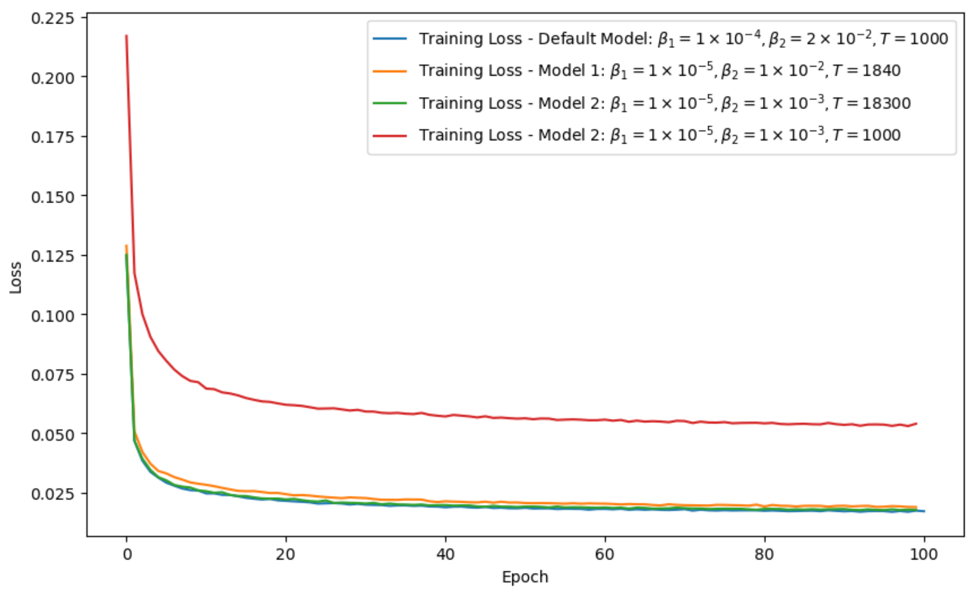

Figure 4: This figure illustrates the evolution of the training loss for three different models, with distinct β and T hyperparameters. All the models exhibit an initial decrease in loss, stabilising after 20 epochs. All the models with the correct timestep converged to a similar loss and stayed there, whereas Model 2 with a timestep of 1,000 maintained a slightly larger loss.

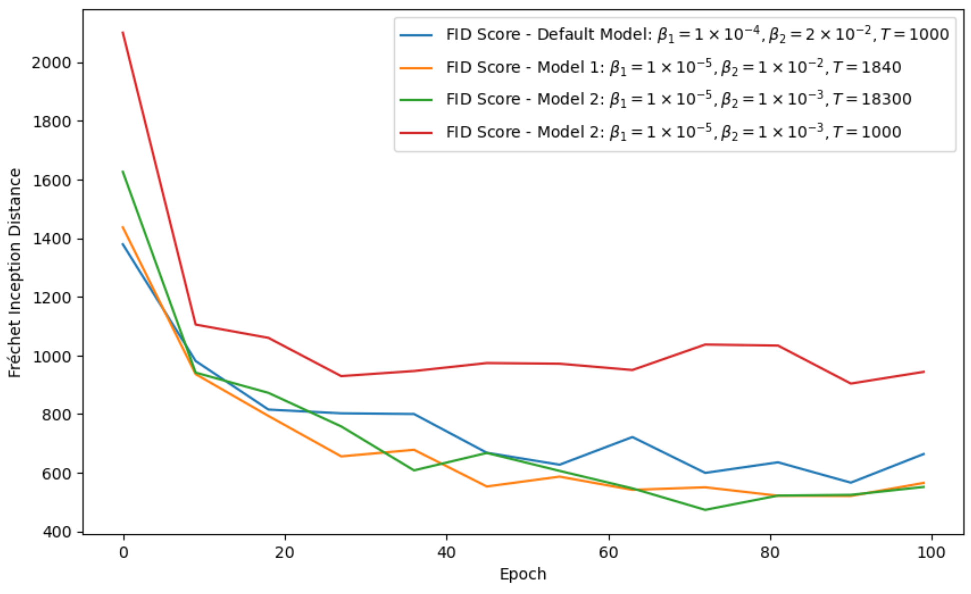

Figure 5: This figure illustrates the evolution of the FID score for three different models, with distinct β and T hyperparameters. All the models with the correct timestep converged to a similar FID score. Model 2 trained on 1,000 timesteps achieved the highest FID score.

0.1.3 Analysis of different models

Timestep analysis Figure 1 depicts how Model 2 with a timestep of $T = 1,000$ results in 60% of the original signal remaining at the end of training. This means the model only learns a fraction of the forward diffusion process, resulting in improper sample generations when compared to the model trained with the correct number of timesteps $T = 18,300$. Smaller β_1 and β_2 values require more timesteps to reach Gaussian noise as each step introduces less noise than larger parameters. As depicted by Figure 1 a substantially larger number of timesteps is required for complete diffusion of Model 2. In theory, more timesteps that take smaller steps result in the model learning more detail in the decoding process and should result in improved generation of images, although at the cost of computational overhead. In the instance of Gaussian noise, Equation 3, allows for instant sampling at any timestep t so the computational overhead is not that bad.

Qualitative analysis As suggested by the timestep analysis Model 2 with 1,000 timesteps has not properly learnt the diffusion process, Figure 2 depicts a variety of interesting symbols with a mild resemblance to numbers. The Default Model and Model 1 have similar final images with a majority resembling numbers. Model 2 with timesteps $T = 18,300$ appears to have performed best, although considering the computational cost for this increase in timesteps compared to the other models this is not much more of an improvement.

Figure 3 illustrates the evolution of generated images at different epochs. Models with smaller timestep increments appear to reconstruct digits earlier in the denoising process. This suggests a more granular noise schedule captures finer details more effectively and yields finer images. Although the final images appear to be of a similar resolution, perhaps there is still noise that is harder to resolve by eye.

Quantitative analysis Supporting the timestep and qualitative analysis, Figures 4 and 5 suggest that Model 2 with $T = 1000$ steps performs worst with a FID score almost twice that of Model 2 when trained on $T = 18,300$, the training loss tells a similar story. The training loss of the remaining models appears to be incredibly similar, with the Default Model being slightly worse. The FID score supports this result, suggesting models with slightly lower β parameters performed better, with Model 2 with $T = 18,300$ performing best closely followed by Model 1. Considering the computational overhead for Model 2, Model 1 is a good option moving forward since it formally scored similarly to Model 2 and visually produced similar quality images at a cheaper computational cost.

0.2 Custom Degradation

Following the successful implementation of a diffusion model with Gaussian noise as the degradation strategy an investigation into custom cold diffusion methods was conducted. The degradation method implemented was Gaussian blur.

0.2.1 Custom degradation strategy

Custom degradation Bansal et al. (2022) proposed various degradation techniques beyond traditional Gaussian noise. They demonstrated that diffusion models can generate images with deterministic degradations that perform as well as noise-based degradation. The paper explored several custom diffusion techniques such as blurring, masking and even an animorphosis transformation where animal faces are used in the degradation. Bansal et al. (2022) noticed traditional diffusion algorithms, similar to Algorithm 1 and 2, produced sub-optimal results with deterministic degradation. Proposing a new sampling algorithm for custom degradation (Algorithm 4).

Instead of adding Gaussian noise into the image at each timestep and predicting the stochastic noise added, the new training algorithm learns the deterministic degradations. The new algorithm utilises a restoration term R which aims to reverse the degradation at the current timestep and a degradation term D, which is added to this estimation to correct the accumulated error from the R term.

Blurring In this report, the custom degradation strategy implemented was blurring. Instead of adding Gaussian noise and learning to reverse it, a known Gaussian blur was applied to systematically remove information. Gaussian blur takes a weighted average around the pixel, the implementation of Gaussian blur has parameters kernel size and sigma. Kernel size is the size of the Gaussian kernel, this was fixed to encompass the whole image. Sigma represents the standard deviation of the Gaussian blur and was given a linear relationship to the current timestep, resulting in a directly proportional deterministic decay of information. The blur schedule followed the following linear form:

$$\sigma_{GB} = \sigma_{base} + (\sigma_{scale} \cdot t) \tag{7}$$

where $\sigma_{base} = 0.02$ and $\sigma_{scale} = 0.01$. The deterministic Gaussian blur is implemented using a forward training function and a sampling function.

Training algorithm The forward function iteratively blurs an image following the linear blur schedule (Equation 7), the CNN model learns this forward diffusion via Algorithm 3. The loss function minimises the mse of the original image and the blurred image at z_t, by doing so the model learns the deterministic steps from x to z_t. This process is shown below in Algorithm 3.

Algorithm 3 Deterministic Forward Blurring

```
Require: Training data $x$
Ensure: Model parameters $\phi_t$
1: repeat
2:     for $i \in \mathcal{B}$ do
3:         $t \sim \text{Uniform}[1, \dots, T]$                    ▷ Sample random timestep
4:         Gaussian blur $\sim \sigma(t)$                          ▷ Initialise Gaussian blur at timestep
5:         $z_{tblur} \sim$ Gaussian blur$(x_i)$                   ▷ Apply Gaussian blur to training data
6:         $\ell_i = \|x - \text{gt}(z_{tblur}, t)\|^2_{\phi_t}$   ▷ Compute MSE loss for deterministic blur
7:     end for
8:     Accumulate losses for batch and take gradient step
9: until converged
```

Sampling algorithm The sampling algorithm follows Algorithm 4, starting from a fully blurred image at $t = T$ the algorithm iteratively restores the image at the current timestep using the trained CNN model and degrades this estimate to correct for the accumulated error. This iteration continues until the final timestep T.

Algorithm 4 Improved Sampling for Cold Diffusion

```
Require: A degraded sample $x_t$
1: for $t = T, \dots, 2$ do
2:     $\hat{x}_0 \leftarrow R(x_t, t)$
3:     $x_{t-1} \leftarrow x_t - D(\hat{x}_0, t) + D(\hat{x}_0, t-1)$
4: end for
```

In the Gaussian noise model the sampling was unconditional as it started from pure Gaussian noise, in the custom degradation implementation the sampling is conditional starting from a blurred image at timestep T. To further assess the quality of the custom degradation model, unconditional diffusion was conducted starting from a black image.

0.2.2 Implementation of degradation strategy

Blur schedule Figure 6 shows the linear deterministic Gaussian blur schedule. Illustrating the transition from clear to maximally blurred states as dictated by the increasing sigma values.

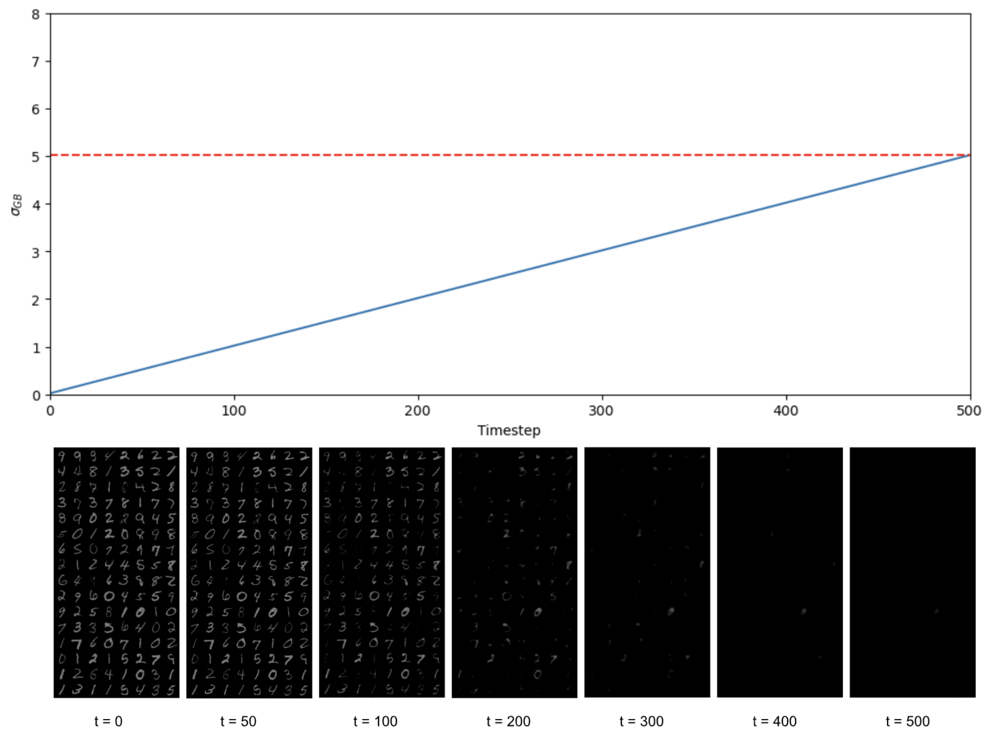

Figure 6: The graph demonstrates the linear increase of the standard deviation (σ_{GB}) used in the Gaussian blur process, corresponding to timesteps from $t = 0$ to $t = 500$. Below the graph, a series of images from the MNIST dataset is presented, showcasing the progressive blurring effect at various timesteps ($t = 0, 50, 100, 200, 300, 400, 500$).

Training custom degradation model The forward blurring process followed Algorithm 3. For a given batch a random timestep t was sampled and used to initialise the Gaussian blur kernel with a standard deviation proportional to the timestep. The initialised blur kernel was applied to the training data, x_i to provide a blurred image z_{tblur}. The mse is calculated between the blurred image and the original image and the model g_t learns the deterministic noise between the original and blurred images. This process was iteratively repeated for all the batches.

The sampling process followed Algorithm 4. Starting from a blurred image at the maximum timestep $t = T$, the model iteratively restored (R) the image using the trained CNN g_t. The restored image is then corrected for accumulated error with the degradation term (D). This process is iteratively repeated until $t = 0$ and the original image is restored.

Qualitative results Below is a selection of samples created during the model training to provide a visual means of comparison for stochastic sampling and custom degradation.

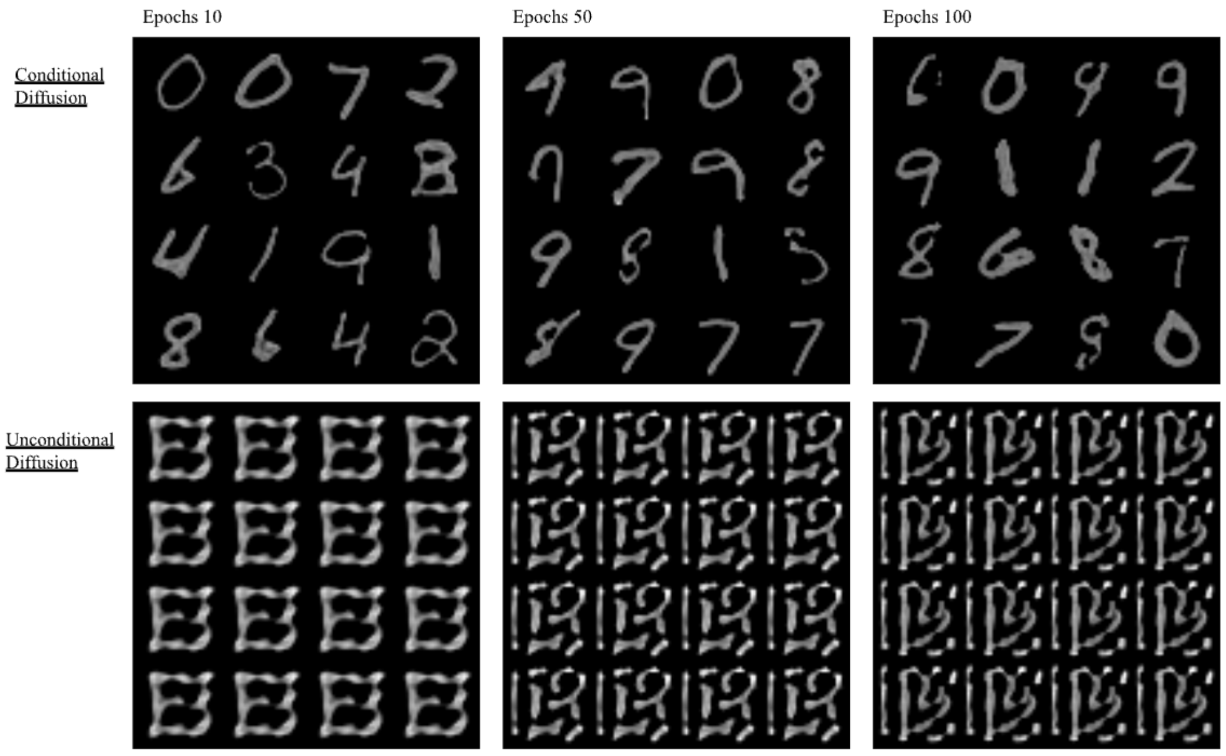

Figure 7: MNIST outcomes of a 4 x 4 grid containing 16 indices from the custom degradation model generated by conditional diffusion and unconditional diffusion of Gaussian blur. The images depict the visual reconstructions after 10, 50, and 100 epochs.

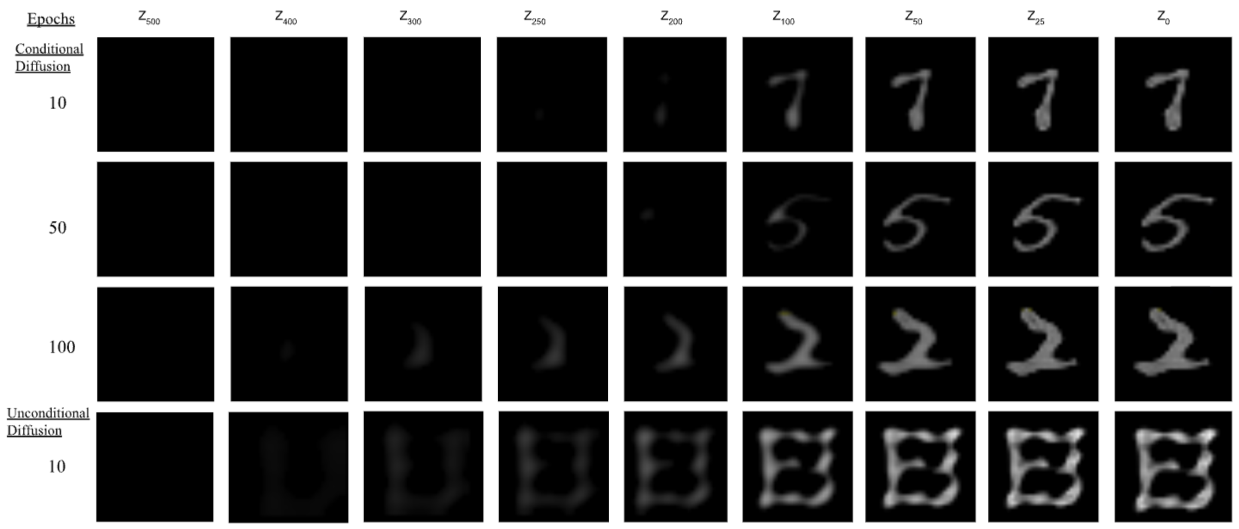

Figure 8: Visual comparison of the evolution of generated digit image quality generated by conditional diffusion and unconditional diffusion of Gaussian blur over training epochs 10, 50, and 100. From left to right, each column represents increasingly restored states of the same digit images at each epoch.

Quantitative results Below are the results of the training loss and FID scores.

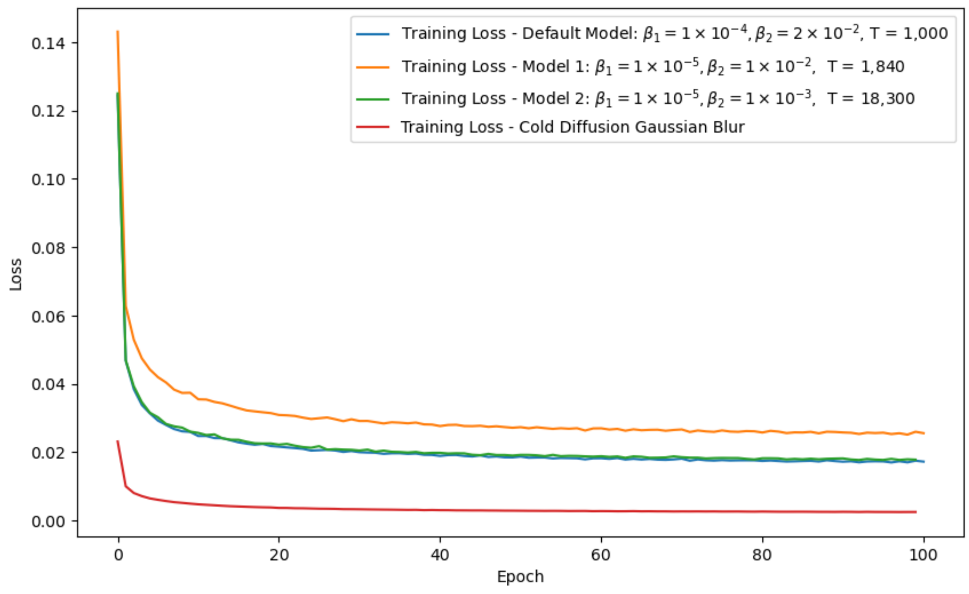

Figure 9: This figure illustrates the training loss of three stochastic models from the Gaussian noise sampling. Compared with the loss from the Gaussian blur custom degradation. All the models converge after a similar number of epochs, with the cold diffusion model reaching the lowest loss.

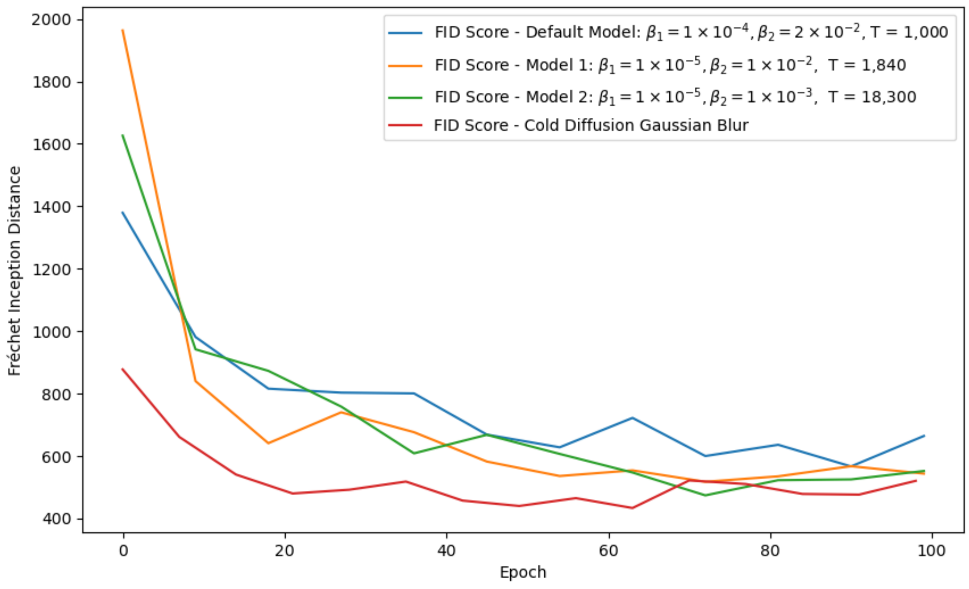

Figure 10: The FID scores of the Gaussian noise stochastic sampling models compared with the deterministic custom degradation model. The cold diffusion model converged and maintained the lowest FID score.

0.2.3 Comparison of the two degradation strategies

Timestep comparison The Gaussian noise model, Figure 1, follows a non-linear convergence schedule determined by α_t whereas the Gaussian blur model, Figure 6, follows a linear schedule. The constant increase in blur with each timestep makes this model predictable as the degree of blur at each timestep is fixed. In contrast, the noise degradation introduces randomness into the model that will be difficult for the model to reverse, although will result in the model learning more robust features.

Qualitative comparison Figures 7 and 8 show the blur models reconstructed at different epochs. The conditional diffusion model yields clear resolvable images at 10 epochs and the images at 100 epochs closely resemble the MNIST digits. In comparison to Figures 2 and 3, the conditional diffusion blur model outperforms the noise model. The unconditional diffusion blur however performs badly and generates an arbitrary shape, in Figure 7 this shape is identical across the whole 4x4 grid.

While the deterministic model achieves a near-perfect reconstruction, this is not indicative of the blur model learning the data's underlying distributions. Instead, the model may be overfitting the predictable linear degradation pattern, which limits the model's ability to sample beyond the training conditions as evident from the unconditional diffusion. The stochastic sampling is more robust and due to the introduction of randomness in training, learns more generalised features of the data and as such can unconditionally generate samples.

Quantitative comparison The training loss of the blur model decreased rapidly and maintained a value close to zero for the remaining epochs, this overfitting is likely due to the predictability of the degradation. Whereas the training loss of Gaussian noise converges quickly and maintains a larger loss for the remaining epochs. This larger training loss is a result of randomness in the Gaussian noise model, which prevents overfitting and gives the model flexibility.

Similarly, the FID score supports this argument. The blur model maintains the lowest score across all epochs, indicating the similarities in the sampled images and the real images of the distribution. Considering the low FID scores achieved by stochastic Model 1 and the compromise between computational overhead and quality of results, this model seems like the best model moving forward.

It is clear the Gaussian noise model is perfectly learning the degradation steps and as such is only able to conditionally reconstruct images. This model has no flexibility, evidenced by the unconditional reconstructed images. Despite the quantitative metrics, the blur model will generally perform worse than the noise model as it lacks robustness.

Concluding thoughts The analysis here explores the balance between predictability and robustness. The comparison shows that while the deterministic Gaussian blur model achieves a lower training loss and FID scores, it does so at the cost of overfitting and a lack of robustness. The stochastic Gaussian noise model achieved slightly worse quantitative scores, although qualitatively performed well when given an unconditional starting point which shows the robustness of this model.

Future work should investigate the effects of changing to a non-linear blur schedule, regularisation and the addition of Gaussian noise to the blur model generating a hybrid model.

Bibliography

Bansal, A., Borgnia, E., Chu, H.-M., et al. 2022, Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise. `https://arxiv.org/abs/2208.09392`

Prince, S. J. 2023, Understanding Deep Learning (MIT press)

0.3 Appendix

0.3.1 Model hyperparameters

Table 1: `diffusion_model.yaml`

Parameter	Value
epochs	100
timesteps	[1000, 750, 500, 250, 100, 50, 10, 5, 1]
interval	9
type	noise
beta1	1e-4
beta2	0.02
n_T	1000

Table 2: `model1.yaml`

Parameter	Value
epochs	100
timesteps	[1840, 1000, 540, 140, 40, 10, 5, 2, 1]
interval	9
type	noise
beta1	1e-5
beta2	0.01
n_T	1840

Table 3: `model2.yaml`

Parameter	Value
epochs	100
timesteps	[18300, 10000, 5000, 1000, 500, 100, 50, 10, 1]
interval	9
type	noise
beta1	1×10^{-5}
beta2	1×10^{-3}
n_T	18300

Table 4: `blur_model.yaml`

Parameter	Value
epochs	100
timesteps	[500, 400, 300, 250, 200, 100, 50, 25, 1]
interval	9
type	blur
n_T	500

0.3.2 Auto-generative tools

This project has utilised auto-generative tools in the development of documentation that is compatible with auto-documentation tools, latex formatting and the development of plotting functions. Example prompts used for this project:

- Generate doc-strings in NumPy format for this function.
- Generate Latex code for an algorithm.
- Generate Latex code for an equation.
- Generate Python code for a 9 by 1 subplot.