M2: Application of Machine Learning

Max Talberg

March 26, 2024

Contents

0.1 Training a Diffusion Model

0.1.1 A regular denoising diffusion model

A 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 instances the MNIST data set.

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:

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

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

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 forward process is a Markov chain as the z_t only depends on the previous step z_{t-1}. This iterative process adds noise onto the sample data x. The diffusion model must understand the encoding phase in order to reverse this process. The diffusion model must understand the encoding phase in order 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 recovering the training data. After training the model can generate data from a sample of noise, z_T to z_{T-1} moving through the latent variables until x is recovered.

Training algorithm The training algorithm teaches the model to reverse the encoding process. It simulates the encoding phase by sampling a random timestep t, which corresponds to the noisy data z_t, as well as Gaussian noise ϵ. The aim of the model is 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 minimise 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, in order to successfully denoise a sample from data.

Algorithm 1 Diffusion model training

Require: Training data x
Ensure: Model parameters ϕ_t

repeat
  for $i \in \mathcal{B}$ **do**
    $t \sim \text{Uniform}[1, \ldots, T]$ ▷ Sample random timestep
    $\epsilon \sim \mathcal{N}(0, I)$ ▷ Sample noise
    $\ell_i = \left\| \sqrt{1-\beta_t} x_i + \sqrt{\beta_t} \epsilon - \phi_t \right\|^2$ ▷ Compute individual loss
  end for
  Accumulate losses for batch and take gradient step
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. The new latent variable receives additional noise from a Gaussian distribution, this is done to simulate the stochastic nature of the forward process improving the quality of samples generated. This process of reversing the encoding is run until the noise is decoded back into data from z_1 to x.

Algorithm 2 Sampling

Require: Model, g_θ, ϕ_t
Ensure: Sample, x

$z_T \sim \text{Norm}_z[0, I]$ ▷ Sample last latent variable
for $t = T, \ldots, 2$ **do**
  $\hat{z}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \left(z_t - \frac{\beta_t}{\sqrt{1-\alpha_t \beta_t}} g_\theta(z_t, \phi_t) \right)$ ▷ Predict previous latent variable
  $\epsilon \sim \text{Norm}_\epsilon[0, I]$ ▷ Draw new noise vector
  $z_{t-1} = \hat{z}_{t-1} + \sigma_t \epsilon$ ▷ Add noise to previous latent variable
end for
$x = \frac{1}{\sqrt{1-\beta_1}} z_1 - \frac{\sqrt{\beta_1}}{\sqrt{1-\alpha_1 \beta_1}} g_\theta(z_1, \phi_1)$ ▷ Generate sample from z_1 without noise

Model The model 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.

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, this consists of multiple CNN blocks. The DDPM has a scheduler which is 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 and 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 a number of 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 backpropogation and the Adam optimiser updates the model parameters to minimise loss.

After each training epoch the models performance is evaluated by generating samples. This is done by reversing the diffusion process starting from random noise. These samples depict the models learning progress.

Throughout the training the loss and evaluated samples are logged to track progress.

Hyperparameter tuning Below is the training performance of the loss and the quality of samples generated with the initial default hyperparameters: A diffusion model with a CNN contating hidden layers $16, 32, 32, 16$ and Adam optimiser with a learning rate of 2×10^{-4} was trained for 100 epochs with varying β parameters and timesteps T.

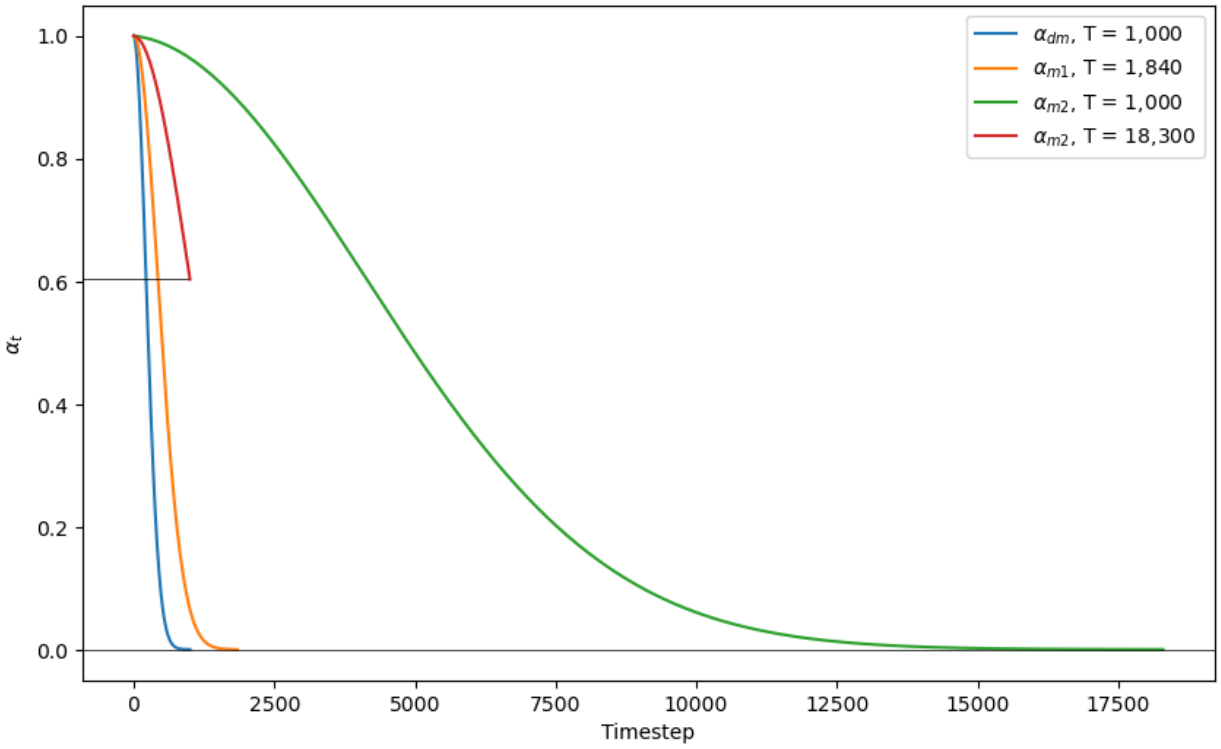

Figure 1: This figure illustrates the trajectories of the alpha parameter for three diffusion models as a function of timesteps. The standard diffusion model (df), Model 1 (m1) and Model 2 (m2) are represented by different colors. Model 2 is plotted twice to emphasise the need for a larger number of timesteps (18,300) compared to the standard (1,000) for the alpha value to approach zero, indicating the sensitivity of the model's convergence and the corresponding adjustment in timesteps required for accurate modeling.

The forward diffusion process gradually adds noise to the data, defined by the variance schedule β_t at each timestep. α_t values are the cumulative product of $(1-\beta_t)$ all the way up to T represented by, $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$. The diffusion model interpolates between two beta values β_1 and β_2 with the following relationship:

$$\beta_t = (\beta_2 - \beta_1) \cdot \frac{\text{arange}(0, T+1)}{T} + \beta_1, \tag{1}$$

$$\alpha_t = \exp\left(\text{cumsum}\left(\log(1-\beta_t)\right)\right). \tag{2}$$

As such varying β values require different timesteps in order to complete the encoding process in the training loop such that the model learns to fully denoise an image. The model was trained with the default β parameters and two variations representing model 1 and model 2. A smaller range of diffusion steps relates to incrementally smaller additions of noise.The plot depicts the α_t values and how they need different time steps for a full encoding, model 2 is also run for $T = 1000$ to convey the importance, from the plot this is only encoded to 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.

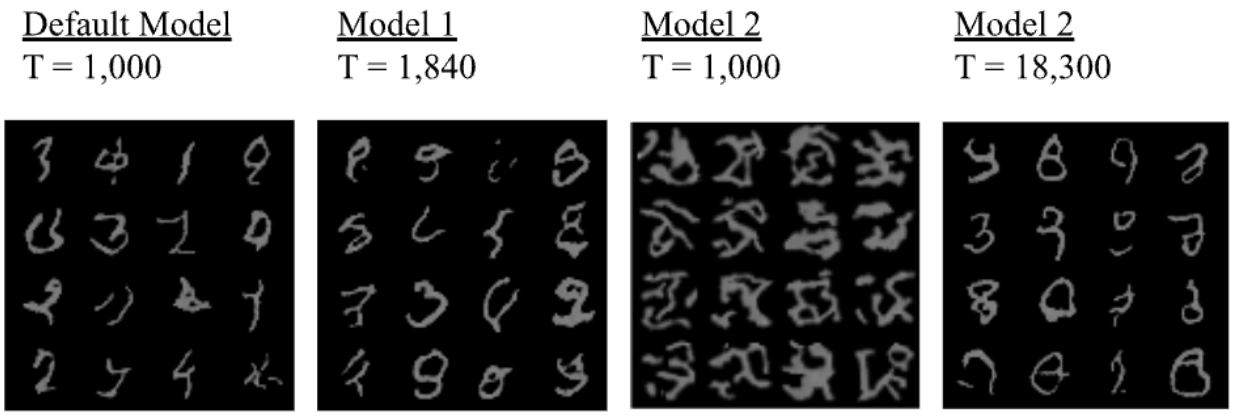

Figure 2: Generated digit images at 100 epochs for three diffusion models with distinct hyperparameters. Results are displayed from the Default Model, Model 1, and Model 2, to illustrate the effects of varying hyperparameters. The improvement in image clarity as the number of epochs grows is evident.

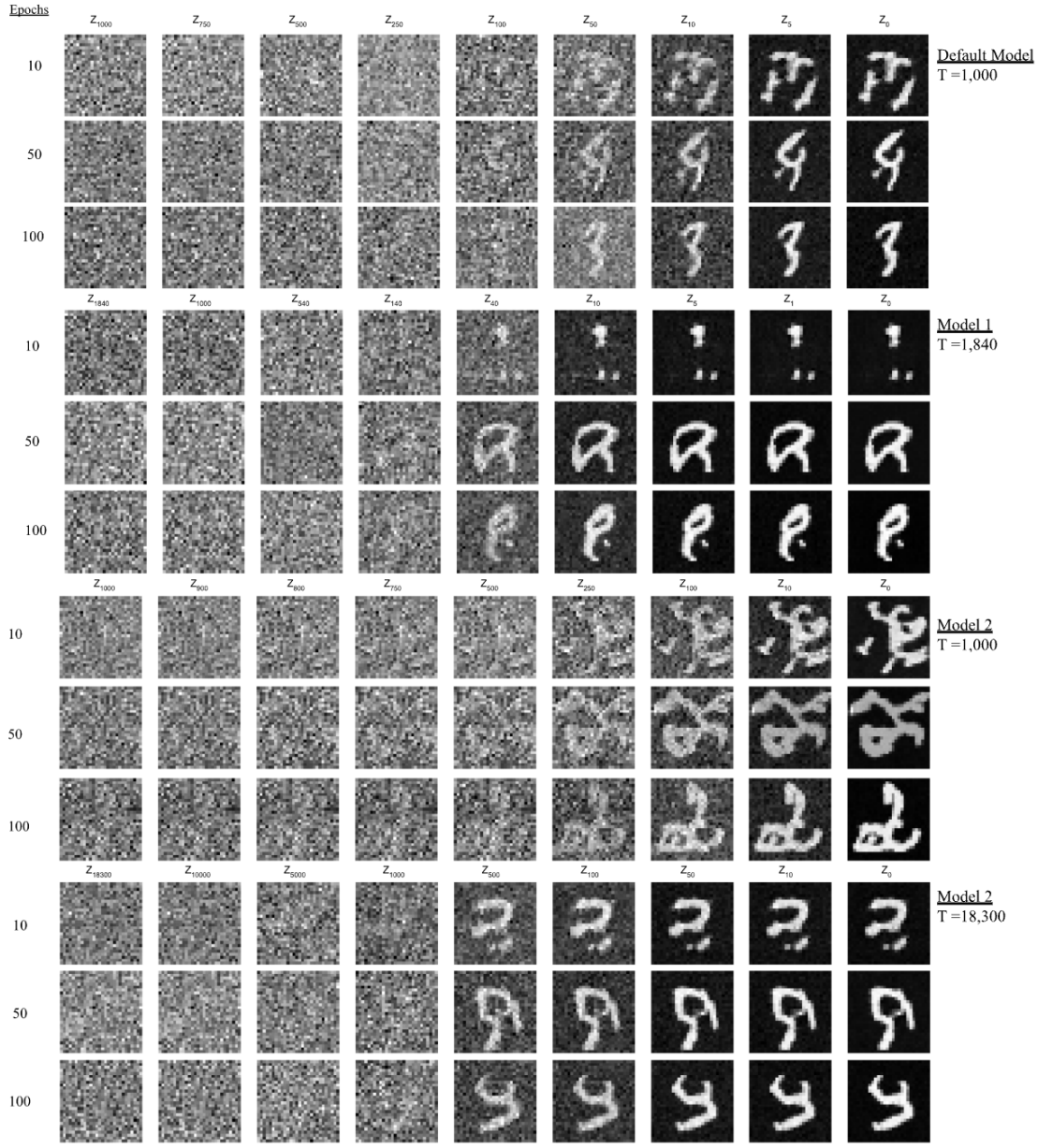

Figure 3: Visual comparison of the evolution of generated digit image quality at epochs 10, 50, and 100 for diffusion models with varying hyperparameters. The rows display results from the Default Model, Model 1, and Model 2 respectively, with columns representing images at increasing epoch counts. Clearer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

Quantitative Results Below is analysis of two different types of loss functions. First the training loss from the mse. Then Fréchet Inception Distance (FID), which compares the loss between...

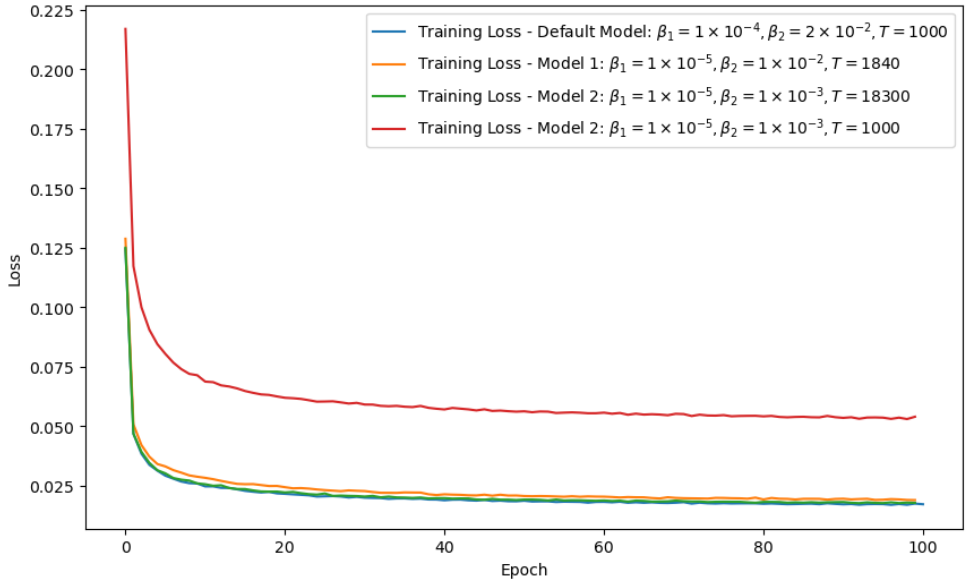

Figure 4: This figure illustrates the evolution of the training loss for three different convolutional neural network models, with distinct β and T hyperparameters. All the models exhibit an initial decrease in loss, stabilising after 20 epoch. All the models with the correct timestep converged to a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

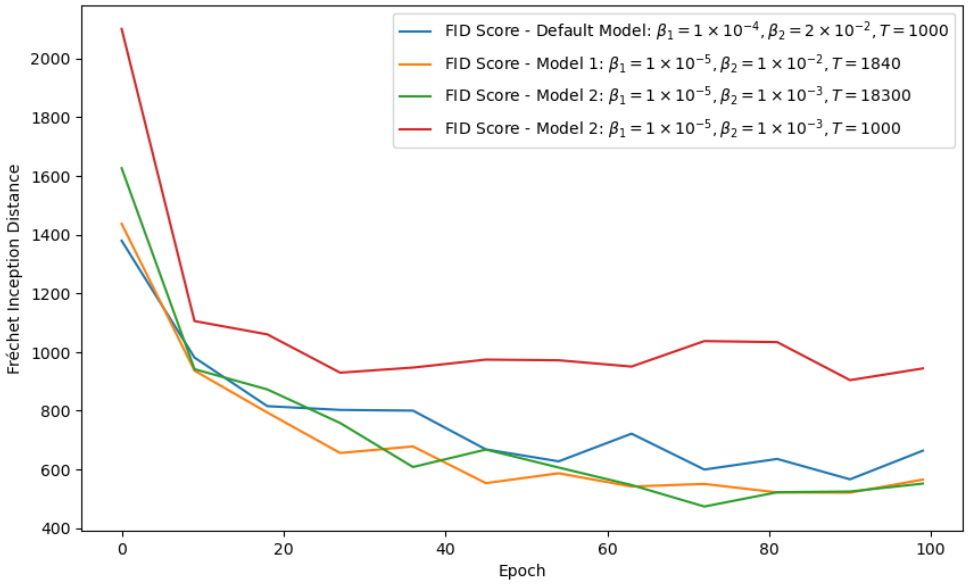

Figure 5: The FID scores provide insight into the image quality produced by each diffusion model. Lower scores indicate better image quality, with Model 2 trained on 18,300 timesteps scoring lowest. All the models with the correct timestep converged to a similar FID score. Model 2 trsined on 1,000 timesteps acheived the highest score as expected.

0.1.3 Analysis of different models

Timestep analysis Figure 1 depicts how model 2 with a timestep of 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.

Qualitative analysis Figure 2 depicts the MNIST outcomes of a 4 x 4 grid containing 16 indices at 100 epochs for all the models. As predicted Model 2 with 1,000 timesteps has not properly learn diffusion process and as a result has generated an interesting mixture of shapes with a mild resemblance of numbers. The Default Model and Model 1 have similar final mages with a the majority of integers resembling numbers. Model 2 appears to have performed best, although considering the increase in timesteps to the other models this improvement is not much more improved.

Similarly Figure 3 depicts all the models, this time at varying timesteps at the sampling (denoising) stage. It appears Model 2 with the correct number of timesteps denoises first, potentially due to the large number of timesteps meaning the model has time to learn the noise schedule and creates a finer image.

Quantitative analysis Supporting the qualitative argument figures 4 and 5 suggest that Model 2 with 1000 steps performs worst. The training loss depicts no noticeable difference in the other models, only Model 2 appears to have marginally lower loss. The FID score supports this suggesting the models with slightly lower β parameters performed better, with Model 2 performing best. Although considering the number of timesteps for this performance, suggests Model 1 is arguably a better model moving forwards.

0.2 Custom Degradation

Following the successful implementation of a diffusion model with Guassian noise as the degradation strategy an investigation into custom cold diffusion methods was conducted.

0.2.1 Custom degradation strategy

Custom degradation Bansal et al. (2022) proposed various degradation techniques beyond traditional Guassian noise. They demonstrated that diffusion model are able to generate images with deterministic degradation's 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. Therefore proposed a new sampling algorithm for custom degradation.

Algorithm 3 Improved Sampling for Cold Diffusion

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

Instead of adding Gaussian noise into the image at each timestep and predicting the stochastic noise added, the new training algorithm learns the deterministic degradation's. 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 had kernel size and sigma parameters which represents the standard deviation of the Gaussian blur. The kernel size was fixed to encompass the whole image and the the sigma parameter has a linear relationship to the current timestep. Resulting in a directly proportional deterministic decay of information.

Algorithm 4 Deterministic Forward Blurring

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

Cold diffusion is implemented using a forward function and a sampling function. The forward function iteratively blurs an image following the linear blur schedule, the CNN model learns this forward diffusion.

The sampling algorithm follows Algorithm 3, starting from a fully blurred image at $t = T$ the algorithm iterativley restores the image at the current timestep using the trained CNN model and degrades this estimate to correct for accumulated error. This iteration continues until the final timestep T.

0.2.2 Implementation of degradation strategy

Blur schedule The Gaussian blur degradation strategy is then implemented to investigate the performance of custom degradation strategy.

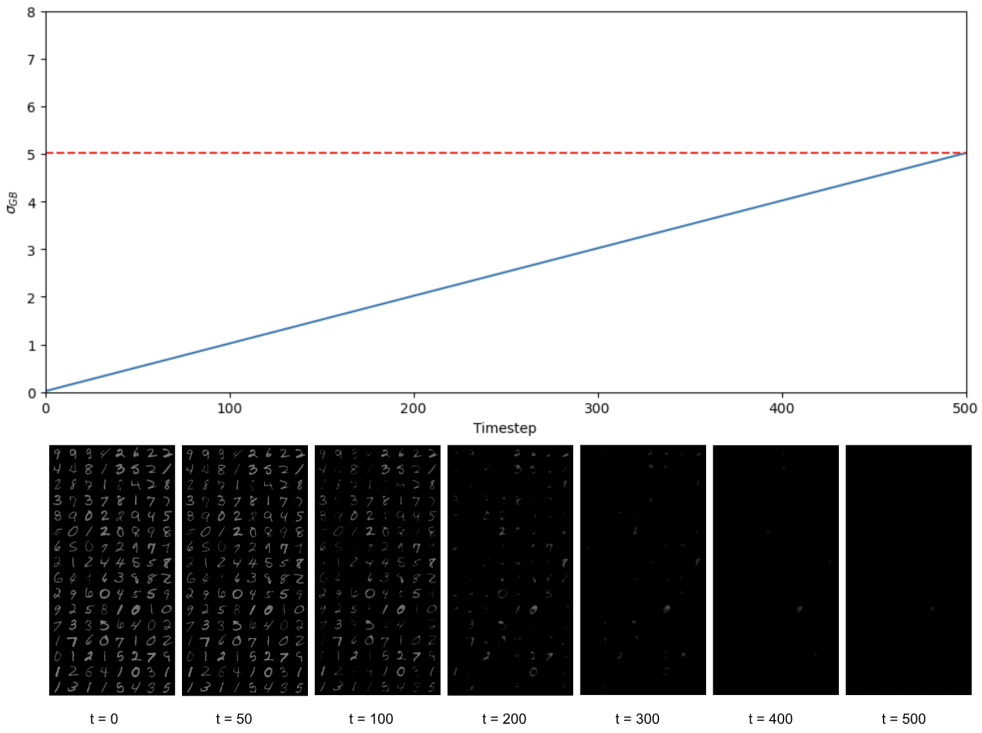

Figure 6: Linear deterministic Gaussian blur schedule. The graph demonstrates the linear increase of the standard deviation (σ_{GB}) used in the Gaussian blur process, corresponding to timesteps from $t = 0$ to $t = 500$. Below the graph, a series of images from the MNIST dataset are presented, showcasing the progressive blurring effect at various timesteps ($t = 0, 50, 100, 200, 300, 400, 500$), illustrating the transition from clear to maximally blurred states as dictated by the increasing sigma values.

Training custom degradation model The forward blurring process followed Algorithm 4. For a given batch a random timestep t was sampled, this was 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. The loss was then calculated between the blurred image and the original image. The model gt was then updated to learn the deterministic noise between the original and blurred images. This process was iteratviley repeated for all the batches.

The sampling process followed Algorithm 3. Starting from a blurred image at the maximum timestep $t = T$, the model iteratively restored (R) the image using the trained CNN gt. The restored image is then corrected for accumulated error with the degradation term. This process is iterativley repeated until $t = 0$.

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.

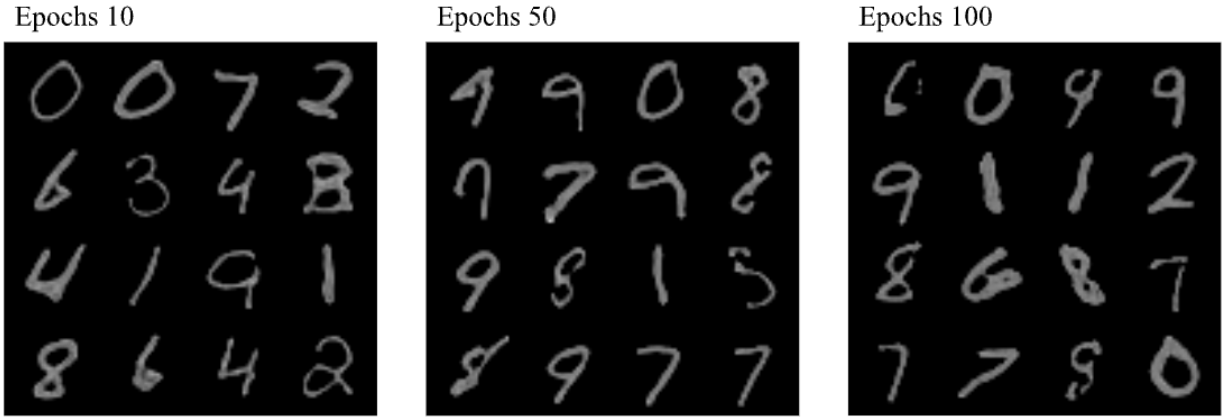

Figure 7: Generated images of digits using the custom degradation model at different training stages. The images depict the visual quality and clarity of digit reconstructions after 10, 50, and 100 epochs, demonstrating the model's progressive improvement in deblurring and restoration accuracy.

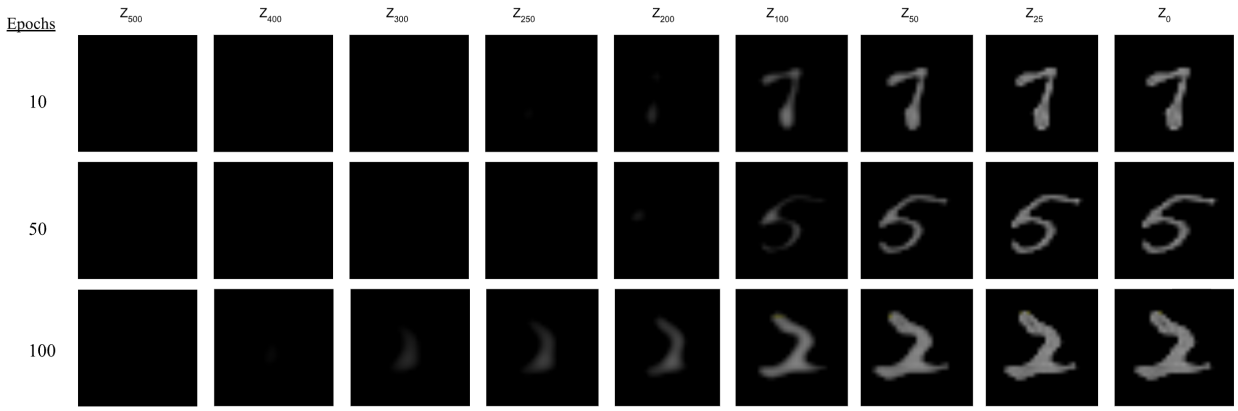

Figure 8: Visual comparison of the evolution of generated digit image quality over training epochs 10, 50, and 100. From left to right, each column represents increasingly restored states of the same digit images at each epoch, showcasing the model's capability to refine and clarify details through the training process.

Quantitative results Below is analysis of the training loss and the FID.

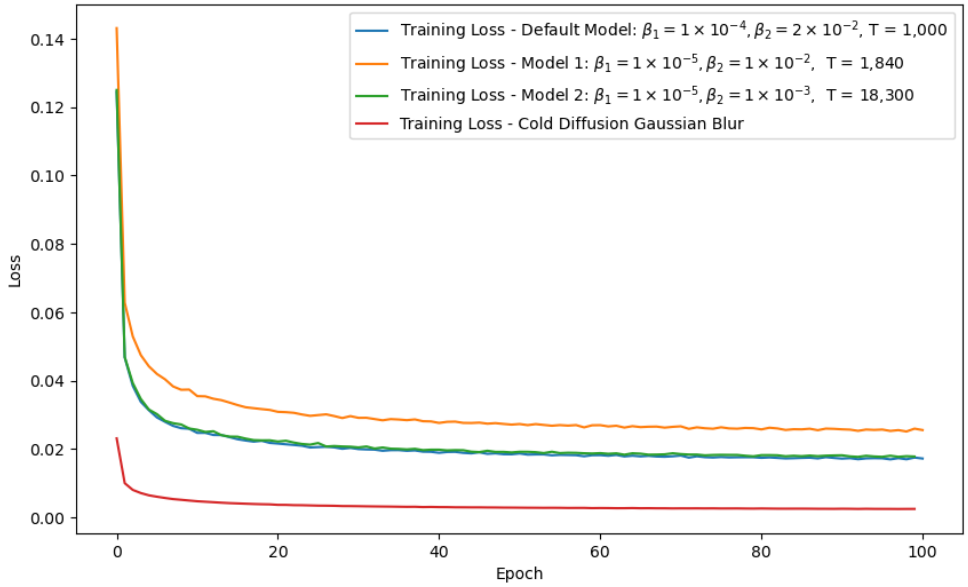

Figure 9: This figure illustrates the evolution of the training loss of three stochastic models from the Gaussian noise sampling. Compared with the loss from the Gaussian blur custom degradation. All the models converge after a similar number of epochs, with the cold diffusion model reaching the lowest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

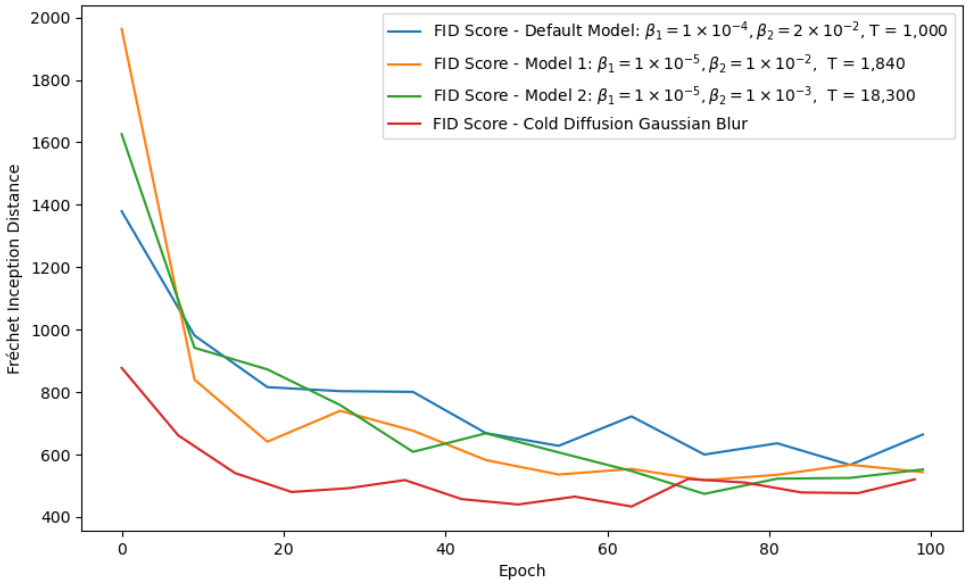

Figure 10: The FID scores of the Gaussian noise stochastic sampling models compared with the deterministic custom degradation model. The cold diffusion model converged too and maintained the lowest FID score.

0.2.3 Comparison of the two degradation strategies

Timestep analysis Figure 1 depicts how model 2 with a timestep of 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.

Qualitative analysis Figure 2 depicts the MNIST outcomes of a 4 x 4 grid containing 16 indices at 100 epochs for all the models. As predicted Model 2 with 1,000 timesteps has not properly learn diffusion process and as a result has generated an interesting mixture of shapes with a mild resemblance of numbers. The Default Model and Model 1 have similar final mages with a the majority of integers resembling numbers. Model 2 appears to have performed best, although considering the increase in timesteps to the other models this improvement is not much more improved.

Similarly Figure 3 depicts all the models, this time at varying timesteps at the sampling (denoising) stage. It appears Model 2 with the correct number of timesteps denoises first, potentially due to the large number of timesteps meaning the model has time to learn the noise schedule and creates a finer image.

Quantitative analysis Supporting the qualitative argument figures 4 and 5 suggest that Model 2 with 1000 steps performs worst. The training loss depicts no noticeable difference in the other models, only Model 2 appears to have marginally lower loss. The FID score supports this suggesting the models with slightly lower β parameters performed better, with Model 2 performing best. Although considering the number of timesteps for this performance, suggests Model 1 is arguably a better model moving forwards.

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

0.3 Appendix