M2: Application of Machine Learning - Coursework

William Knottenbelt, wdk24

Word Count: 2915

1 Denoising Diffusion Probabilistic Models

a) Theory

Diffusion models were introduced by Sohl-Dickstein et al. (2015) [1] as a way to generate samples from complex data distributions. They were later improved and popularised by Ho et al. (2020) [2], achieved extremely high quality image synthesis by making some refinements and simplifications, terming the models "Denoising Diffusion Probabilistic Models" (DDPM). At a high level, the method works by iteratively adding noise to an image until it consists of only noise, and training a neural network to reverse this process starting from an image containing only noise. The models have a probabilistic interpretation based on variational inference, which relies on the image being degraded by adding Guassian noise for T time-steps. The latent state of the image at time step $t \in \{1, 2, ..., T\}$ is given by

$$\mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{z}_{t-1} + \sqrt{\beta_t}\epsilon_t, \tag{1}$$

where $\mathbf{z}_0 \equiv \mathbf{x}$ is the original image, $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$ is random noise (standard normally distributed), and the set of weights $\{\beta_t\}_{t=1}^T$ are collectively called the noise schedule. This can be re-parametrised using $\alpha_t = \prod_{t=1}^T(1-\beta_t)$, such that

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\epsilon, \tag{2}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$. The noise schedule should satisfy $\alpha_T \approx 0$, such that the fully-degraded state is standard normally distributed $\mathbf{z}_T \sim \mathcal{N}(0, \mathbb{I})$. In this report, we use a noise schedule of $T = 1000$ time-steps, with β_t linearly increasing from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.

We train a neural network, $\mathbf{g}(\mathbf{z}_t, t, \phi)$, to predict the noise ϵ in equation (2) based on the latent state $\mathbf{z}_t$, using algorithm 1. The loss function is constructed to optimize the evidence lower bound (ELBO), which is necessarily lower than the log-likelihood of the training data $\{\mathbf{x}_i\}_{i=1}^N$. It relies on the assumption that the distribution of $\mathbf{z}_{t-1}|\mathbf{z}_t$ is (Ho et al., section 3.2 [2])

$$p_\phi(\mathbf{z}_{t-1}|\mathbf{z}_t) = \mathcal{N}_{\mathbf{z}_{t-1}}(\boldsymbol{\mu}_\phi(\mathbf{z}_t), \sigma_t^2\mathbb{I}), \tag{3}$$

where $\boldsymbol{\mu}_\phi(\mathbf{z}_t) = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}(\mathbf{z}_t, t, \phi)$ is the learned mean of the distribution, and σ_t is chosen by the user. Good empirical results have been found using $\sigma_t^2 = \beta_t$ [2], hence this is what we use (see step 7 in algorithm 2).

We can generate new data from the trained network using algorithm 2, which begins with pure noise and iterates through a series of progressively less-noisy states by sampling from the learned distribution $p_\phi(\mathbf{z}_{t-1}|\mathbf{z}_t)$, until we reach a sample $\mathbf{z}_0 \equiv \mathbf{x}$.

Algorithm 1 Training Denoising Diffusion Probabilistic Model

```
1: **Input:** Minibatch $\mathcal{B} = \{\mathbf{x}_i\}_{i=1}^{|\mathcal{B}|}$, noise schedule $\{\alpha_t\}_{t=1}^T$
2: **Output:** Model parameters $\phi$
3: **for** $\mathbf{x}_i$ in $\mathcal{B}$ **do**
4:     Sample $t \sim$ Uniform$(\{1, \dots, T\})$
5:     Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$
6:     Compute individual loss $l_i = \|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\epsilon, t, \phi) - \epsilon\|^2$
7: **end for**
8: Update $\phi$ to minimize $\mathcal{L}(\phi) = \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} l_i$ using gradient descent
```

Algorithm 2 Sampling from Denoising Diffusion Probabilistic Model

```
1: **Input:** Trained model parameters $\phi$, noise schedule $\{\beta_t, \alpha_t\}_{t=1}^T$
2: **Output:** Sample $\mathbf{x}$
3: $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$                                ▷ Start with pure noise
4: **for** $t = T, T-1, \dots, 2$ **do**
5:     Compute $\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}(\mathbf{z}_t, t, \phi)$
6:     Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$
7:     Compute $\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sqrt{\beta_t}\epsilon$
8: **end for**
9: $\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}(\mathbf{z}_1, 1, \phi)]$
10: **return** $\mathbf{x}$
```

For the restoration network, $\mathbf{g}(\mathbf{z}_t, t, \phi)$, we used a convolutional neural network (CNN) with LayerNorm to stabilise training and a GELU activation function. Time-encoding was including by mapping the time, t, to a sinusoidal embedding, which is then passed through a fully-connected neural network to a learned time-embedding with the same number of dimensions as channels in the first hidden layer of the CNN. Each dimension of this embedding is then added to all neurons in the channel it corresponds to in the first hidden layer.

b) Training

We trained two DDPM models on the MNIST dataset [3], using the default split of 60,000 training images and 10,000 testing images provided by PyTorch torchvision. Each image in MNIST is a 28x28 grayscale image representing a handwritten digit from 0 to 9. The first model used the restoration network described in part 1(a), with four hidden layers, configured with 16, 32, 32 and 16 channels, respectively. This consists of a total of 291217 trainable parameters. The second model used the same restoration network, except with five hidden layers, made up of 16, 32, 64, 32 and 16 channels, respectively. This has almost twice the capacity of the first model, containing a total of 542161 trainable parameters. We trained each model for 30 epochs using a batch size of 128 and the Adam [4] optimizer with a learning rate of 2×10^{-4}. To visualise model performance during training, we evaluated the average loss over the full test set after each epoch and generated 16 samples. The test loss was calculated in the same way as in algorithm 1.

Fig. 1 depicts the train and test loss over the training process for the low-capacity model. We see that both the train and test loss decrease sharply for the first few epochs, then decrease more gradually but still consistently for the remainder of the process. The test loss is initially lower than the train loss, which is likely a result of the fact that the train loss is an average over all 468 training steps in the epoch, whereas the test loss is evaluated at the end. There is a very high rate of improvement at the beginning of the training process, so it is natural that the model improves substantially over the first epoch, causing this initial disparity between the train and test loss. After around 7 epochs, the test loss almost perfectly aligns with the train loss for the remainder of the process. This indicates good generalization of the model, and no over-fitting. We plot the train/test loss for the high-capacity model in Fig. 2, and observe that the plot is almost identical to that of the low-capacity model. This implies that increasing the model capacity has not caused over-fitting. The final test loss for the high-capacity model was 0.0185, while for the low-capacity model it was 0.021, which indicates an improvement in model performance by increasing the capacity. The high-capacity model has extra flexibility to learn more complex features, allowing it to capture the data distribution better.

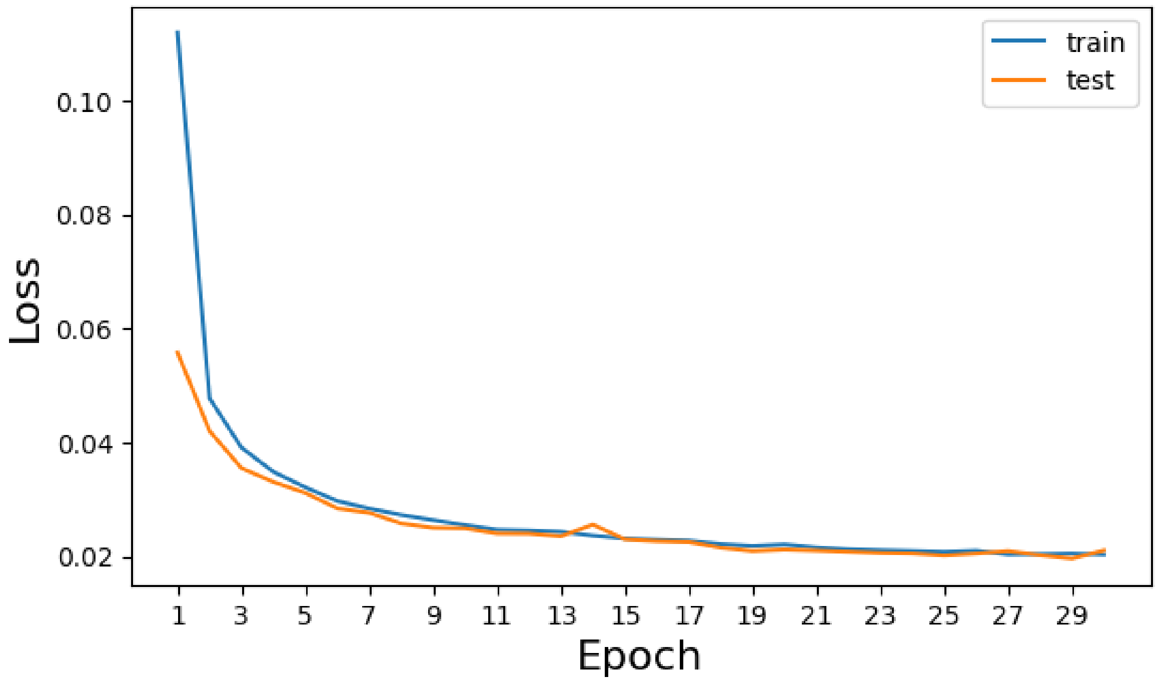

Figure 1: Plot of the average train and test loss of the low-capacity DDPM model, over the training process.

In Fig. 3, we show samples from the low-capacity model after training for 1, 5, 10, and 25 epochs. A similar plot for the high-capacity model is depicted in Fig. 4. As we can see, after the first epoch, the low-capacity model generates mostly random blobs/speckles on a blank background. By contrast, the high-capacity model generates much noisier random samples. This is likely a result of different random initializations of the models. Alternatively, since real digits are closer to random blobs on a blank background than complete noise, this may indicate that the higher-capacity model is more difficult to train. The increased complexity in the parameter space may have caused a more uneven loss surface, leading to a less optimized model after the first epoch. After 5 epochs, we see that the samples from both models start to take shape as noisy symbols, which indicates that the models are starting to capture broad features of the underlying data distribution (eg. symbolic shape, blank

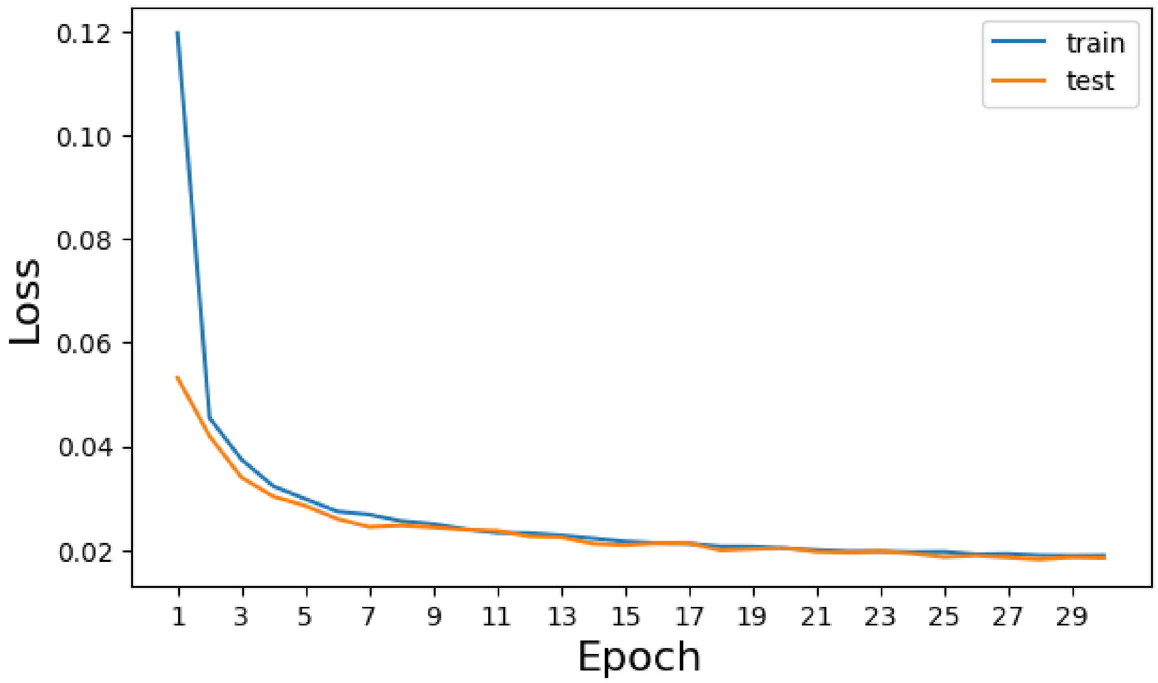

Figure 2: Plot of the average train and test loss of the high-capacity DDPM model, over the training process.

background etc.). The samples from the high-capacity model at this stage look slightly more digit-like, which could indicate that it has surpassed the low-capacity model. After 10 epochs, the low-capacity model still generates mostly meaningless symbols, except some of the samples are broadly taking the shape of real numbers. The symbols also appear to have a higher fidelity of those after 5 epochs. At this stage, the high-capacity model still generates mostly symbols, but with higher fidelity than after 5 epochs. Finally, after 25 epochs, there is no clear difference in the samples between the low-capacity and high-capacity model, with both models generating a mix of symbols and numbers. The numbers do not appear particularly realistic but are visually identifiable.

c) Analysis

To judge model performance, we generated 100 samples from each fully-trained model. Fig. 5 shows the samples for the low-capacity model, and Fig. 6 shows those for the high-capacity model. We see that neither model reached consistent generation of visually-identifiable numbers, with the samples being a mix of meaningless symbols and actual digits. We see that the low-capacity model generates mostly meaningless symbols, which suggests that while it was able to capture broad features of the MNIST dataset (symbolic objects on a blank background, with a handwritten feel), it mostly failed to capture particular features that define specific digits. The samples produced by the high-capacity model contain a higher frequency of actual digits, which implies that increasing the capacity has allowed the model to capture more complex features that define specific digits. However, many of the digits appear distorted and low-fidelity.

To judge the models quantitatively, we calculated the Frechet-Inception-Distance (FID) score [7],

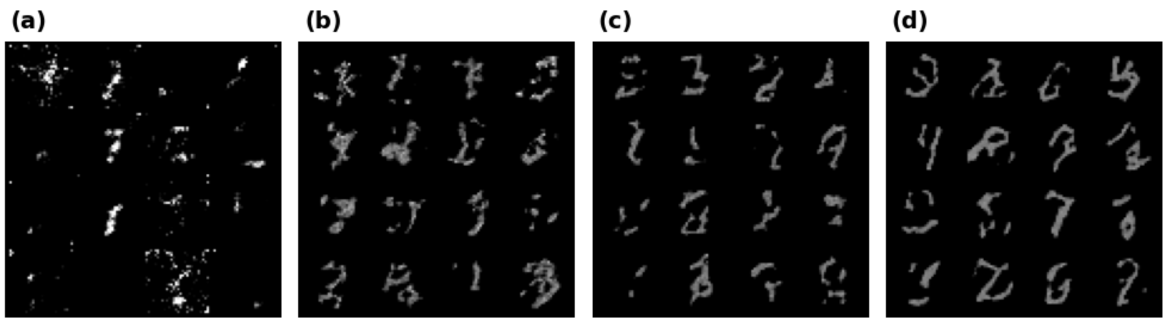

Figure 3: Samples generated by the low-capacity DDPM model after training for (a) 1, (b) 5, (c) 10, and (d) 25 epochs.

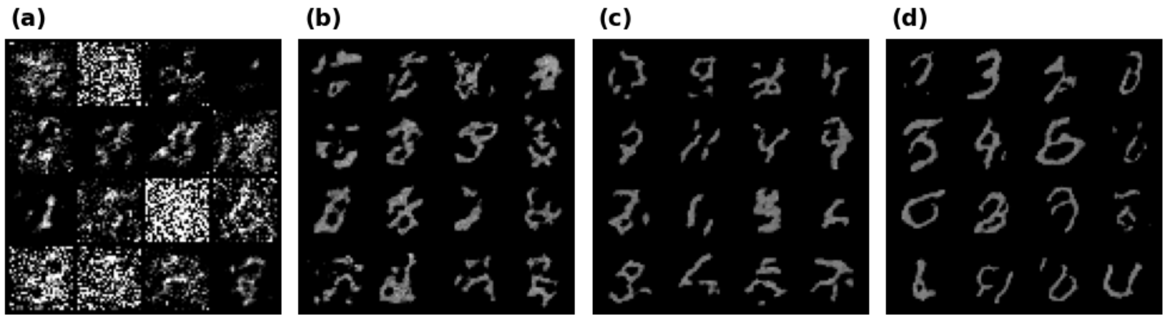

Figure 4: Samples generated by the high-capacity DDPM model after training for (a) 1, (b) 5, (c) 10, and (d) 25 epochs.

of these 100 samples with respect to 500 images from the MNIST test set. Introduced by Heusel et al. (2017) [7], the FID calculation involves using a pre-trained Inception v3 network [8] to construct 2048-dimensional feature vectors for all real and generated images. The FID score is then calculated as

$$FID = ||\mu_r - \mu_g||^2 + \mathrm{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}) \tag{4}$$

where $||\mu_r - \mu_g||^2$ is the squared Euclidean distance between the mean vectors of the real (μ_r) and generated (μ_g) image features, Σ_r and Σ_g are the covariance matrices for the real and generated image features, respectively, and $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix. A lower FID score indicates higher similarity between the distributions of the real and generated data.

Using the lighting TorchMetrics implementation [9], we calculated $FID = 111$ for the low-capacity model, and $FID = 95$ for the high-capacity model. This indicates that indeed, the high-capacity model has captured the underlying data distribution better than the low-capacity model, allowing it to generate images that are more similar to the real handwritten digits in the MNIST dataset.

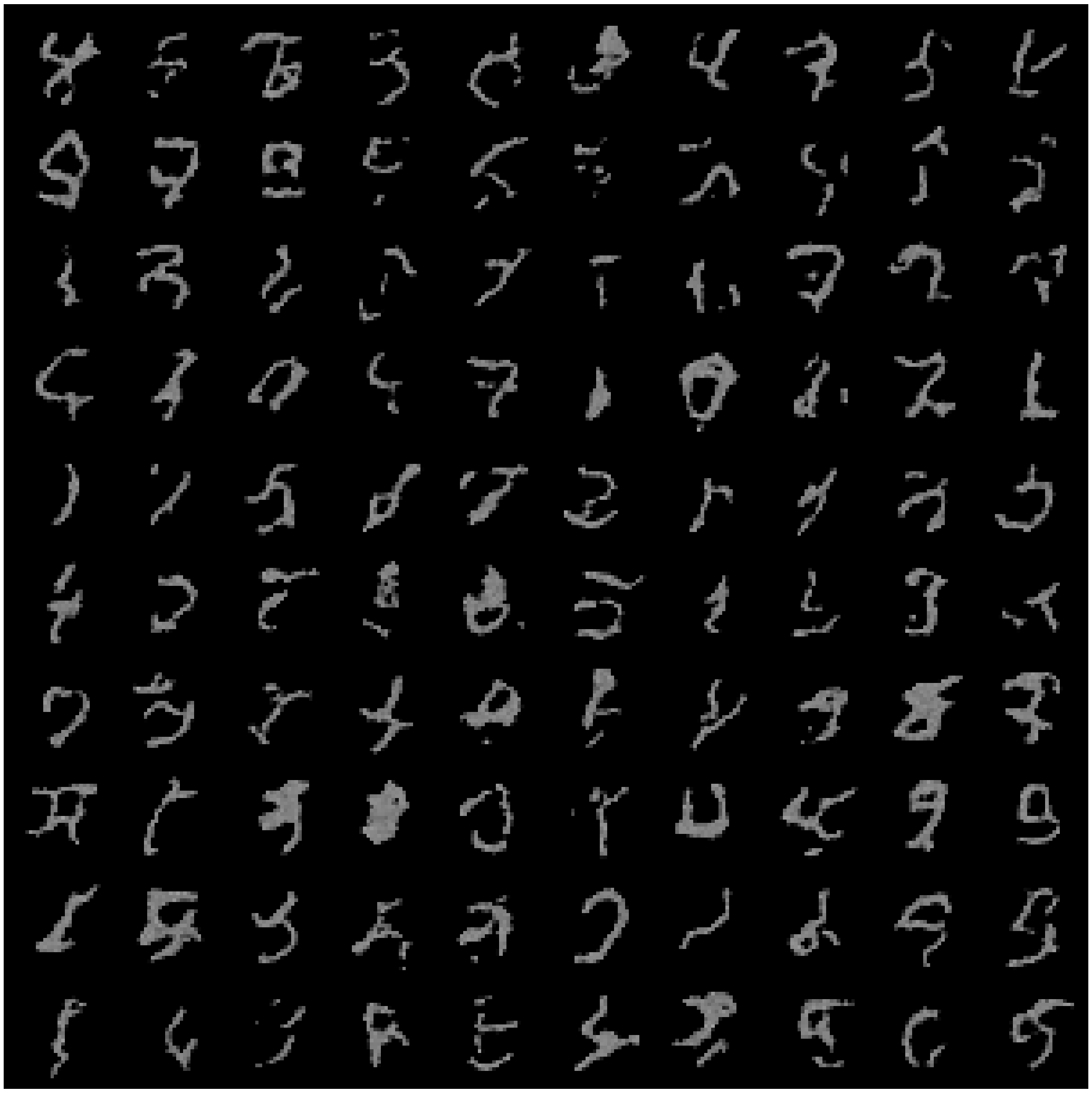

Figure 5: 100 samples generated from the low-capacity DDPM model, trained for 30 epochs.

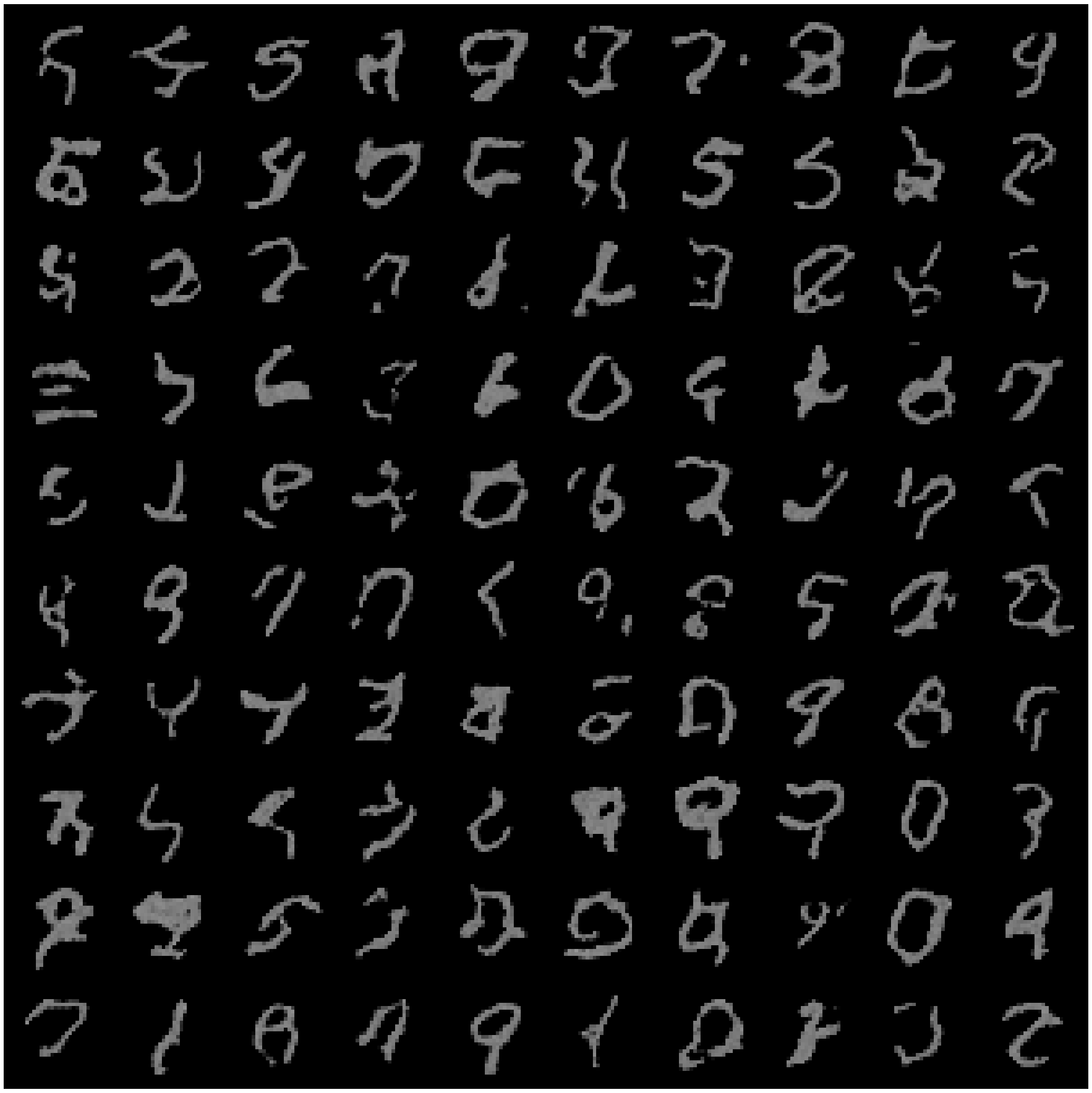

Figure 6: 100 samples generated from the high-capacity DDPM model, trained for 30 epochs.

2 Cold Diffusion

a) Custom degradation: Eraser

Bansal et al. [5] showed that it is possible to construct a diffusion model using arbitrary, even deterministic, image degradation strategies. Formally, we can define an arbitrary degradation operator D, such that the latent state of an image $\mathbf{x}$ at time-step $t \in \{0, 1, ..., T\}$ is given by $\mathbf{z}_t = D(\mathbf{x}, t)$. The degradation operator should satisfy $D(\mathbf{x}, 0) = \mathbf{x}$, and it should be possible to sample unconditionally from the distribution of fully-degraded images $\mathcal{Z}$ (where $D(\mathbf{x}, T) \sim \mathcal{Z}$).

We train a restoration network to predict the original image from the latent states, $\mathbf{g}(\mathbf{z}_t, t, \phi) \approx \mathbf{x}$, using algorithm 3. We can then sample from the trained model using algorithm 4, which involves sampling $z_T \sim \mathcal{Z}$ and iterating through progressively less degraded states.

Algorithm 3 Training Cold Diffusion

```
1: Input: Minibatch $\mathcal{B} = \{\mathbf{x}_i\}_{i=1}^{|\mathcal{B}|}$, degradation operator $D$
2: Output: Model parameters $\phi$
3: for $\mathbf{x}_i$ in $\mathcal{B}$ do
4:     Sample $t \sim \text{Uniform}(\{1, \dots, T\})$
5:     Compute individual loss $l_i = \|\mathbf{g}(D(\mathbf{x}_i, t), t, \phi) - \mathbf{x}_i\|^2$
6: end for
7: Update $\phi$ to minimize $\mathcal{L}(\phi) = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} l_i$ using gradient descent
```

Algorithm 4 Sampling for Cold Diffusion

```
1: Input: Trained model parameters $\phi$, degradation operator $D$
2: Output: Sample $\mathbf{x}$
3: Sample background colour $c \sim \text{Uniform}([-0.5, 0.5])$
4: Sample $\mathbf{z}_T \sim \mathcal{Z}$                    ▷ Start with fully degraded image
5: for $t = T, T-1, \dots, 2$ do
6:     Compute $\hat{\mathbf{x}} = \mathbf{g}(\mathbf{z}_t, t, \phi)$
7:     Compute $\mathbf{z}_{t-1} = \mathbf{z}_t - D(\hat{\mathbf{x}}, t) + D(\hat{\mathbf{x}}, t-1)$
8: end for
9: $\mathbf{x} = \mathbf{g}(\mathbf{z}_1, 1, \phi)$
10: return $\mathbf{x}$
```

In this vein, we designed a novel degradation strategy to have the visual effect of an eraser 'rubbing out' the image. Specifically, the effect is that the head of an eraser is moved across the image in a zig-zag pattern, leaving an empty background image behind. The trajectory of the eraser is defined by a sequence of pixels $\{q_i\}_{i=1}^{M}$, where M is the number of pixels in the trajectory. The pixel q_i is the center of the eraser head at time-step $t = \frac{i}{M}T$. The trajectory is fixed, and can be seen in Fig. 7(a).

Similar to the 'inpainting' degradation in Bansal et al. [5], the degradation is implemented by multiplying the input images, $\mathbf{x}$, by a sequence of masks $\{\mathbf{e}_i\}_{i=1}^{M}$. The images are degraded via $\mathbf{x} \cdot \bar{\mathbf{e}}_t$, where $\cdot$ denotes entry-wise multiplication and

$$\bar{\mathbf{e}}_t = \prod_{i=1}^{s_t M} \mathbf{e}_i,$$

where $s_t = \frac{t}{T} \in [0, 1]$ is the severity of the degradation. The mask $\mathbf{e}_i$ is the eraser head, which consists of a disk of radius 3 consisting of 0s, centered on pixel q_i. This is multiplied by a 2D Gaussian curve of standard deviation 3 centred on pixel q_i, which is normalised to have a peak of 1 and subtracted from 1 to have a central value of 0. The disk is necessary to ensure the removal of all information, while the Gaussian allows the edges of the disk have a smooth fall-off, which was empirically found to improve performance of the model. An individual eraser head mask is visualised in Fig. 7(b).

An important quality of the degradation is that it should produce a diverse distribution of latent images, $\mathcal{Z}$, since the generation process is deterministic for a given latent sample. Hence, to generate diverse images, the starting images must be diverse. To satisfy this criteria, we have the eraser 'rub out' the image leaving a random background given by $\mathbf{b} \sim \mathcal{N}(c\mathbf{1}, \sigma_b^2 \mathbb{I})$, where c is a uniformly distributed solid colour $c \sim \text{Uniform}([-0.5, 0.5])$, and $\mathbf{1}$ is full of 1s, with the dimensions of $\mathbf{x}$. This gives the final form of the eraser degradation operator as

$$D(\mathbf{x}, t) = \mathbf{x} \cdot \bar{\mathbf{e}}_t + \mathbf{b} \cdot (1 - \bar{\mathbf{e}}_t) \tag{5}$$

The pixel trajectory $\{q_i\}_{i=1}^{M}$ is chosen such that $\bar{\mathbf{e}}_T$ consists of only zeros, thereby ensuring that the image gets completely erased. This leaves the fully-degraded image $D(\mathbf{x}, T) = \mathbf{b} \sim \mathcal{N}(c\mathbf{1}, \sigma_b^2 \mathbb{I})$. We used $\sigma_b = 0.05$, hence our latent distribution is $\mathcal{Z} = \mathcal{N}(c\mathbf{1}, 0.0025\mathbb{I})$ with $c \sim \text{Uniform}([-0.5, 0.5])$. The various components making up the eraser degradation are visualised in Fig. 7.

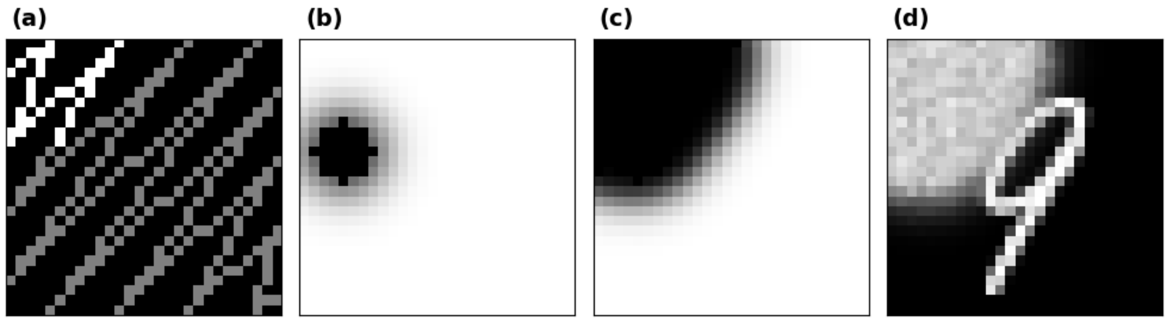

Figure 7: Depiction of how the eraser degradation is constructed, for severity 0.16. (a) shows the trajectory of the eraser, where the white pixels are the trajectory up to severity 0.16, and the grey pixels show the remainder of the trajectory. (b) shows an single eraser-head mask, $\mathbf{e}$, at the position defined by severity 0.16. (c) shows the cumulative product of all eraser-head masks, $\bar{\mathbf{e}}$, in the trajectory up to severity 0.16. (d) shows the actual degradation of an image from MNIST, where the cumulative eraser mask is applied and leaves behind a random noisy background.

b) Training and Analysis

We trained a model using $T = 100$ time-steps of the eraser degradation operator with algorithm 3, on the same MNIST training set used in section 1. We used the restoration network $\mathbf{g}(\mathbf{z}_t, t, \phi)$ described in 1(a), with four hidden layers, configured with 16, 32, 32 and 16 channels, respectively. We trained the model for 30 epochs with a batch size of 128 and the Adam optimizer with a learning rate of 2×10^{-4}, evaluating the test loss and generating 16 samples with algorithm 4 after each epoch.

In Fig. 8, we plot the train/test loss over the training process. We see that the training loss decreases sharply at the start of the training process, and more gradually for later epochs. The test loss follows a similar trend, and is close to the train loss for all epochs except the first. This indicates good generalisation and lack of over-fitting. An interesting feature is that the test loss appears less stable, oscillating above and below the training loss over the full process. This may suggest that the model parameters are moving between different local minima, some of which generalise better or worse than others. This might indicate that the learning rate is too high, causing the model to jump around the parameter space without converging to a single minima. In Fig. 9 we visualise samples from the model after training for 1, 5, 10 and 25 epochs. We see that after the first epoch, the model has already started to generate faded symbols, most of which look very similar. This suggests that the latent distribution, $\mathcal{Z}$, is not sufficiently diverse. After 5 epochs, the generated symbols become higher fidelity, but are still mostly meaningless. After 10 epochs, the generated samples include some identifiable digits, and other symbols that are less recognisable but still have a vague shape of digits. By the 25th epoch, the samples are mostly digits, with just a few meaningless symbols.

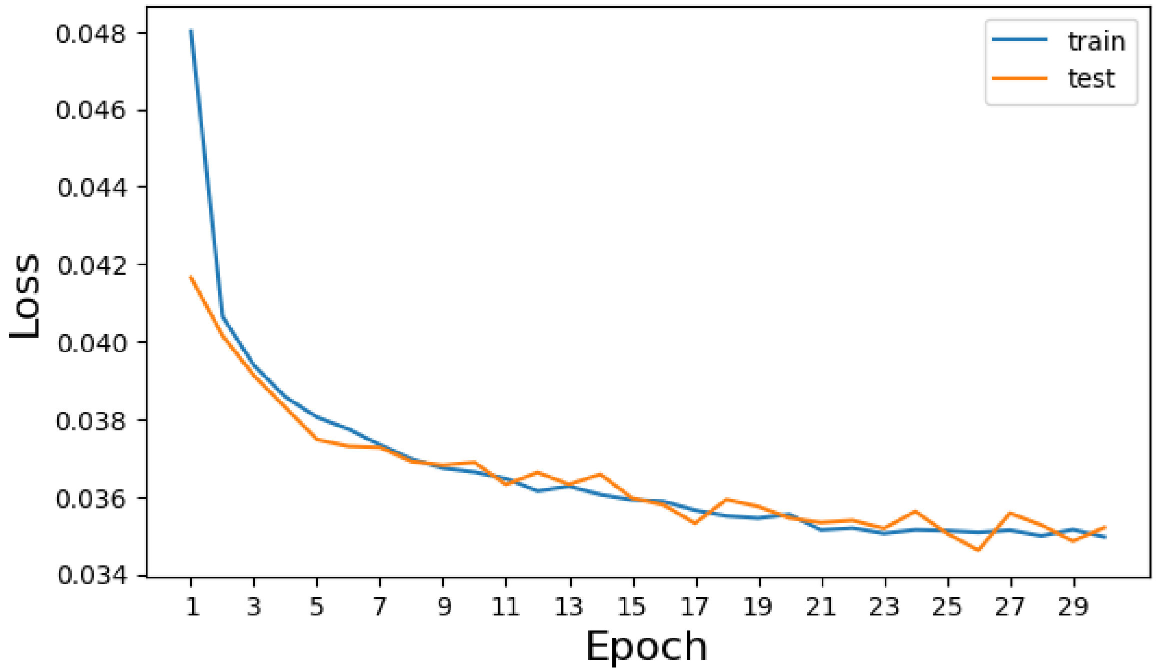

Figure 8: Plot of the average train and test loss of the cold diffusion model, over the training process.

After training, we generated 100 samples from the model, depicted in Fig. 9. We see that the model generates mostly identifiable digits with high fidelity, and a low proportion of meaningless symbols. This shows that the model successfully captured the underlying distribution of MNIST, down to the core features that characterise each digit. This shows that the restoration network is very robust, being able to reconstruct identifiable digits from images degraded by the unconventional eraser degradation. We also calculated the FID of these 100 samples with respect to 500 samples from the MNIST test set, yielding a score of 89. This indicates reasonably strong similarity between the generated and real images.

However, the diversity of the generation is fairly low, with an excessively high proportion of 1s, and many 5s and 9s. This could be a result of insufficient diversity in the latent distribution, $\mathcal{Z}$, despite the measures we took to randomise the background images. This could be improved by using more diverse backgrounds, for example by increasing the standard deviation σ_b of the background noise.

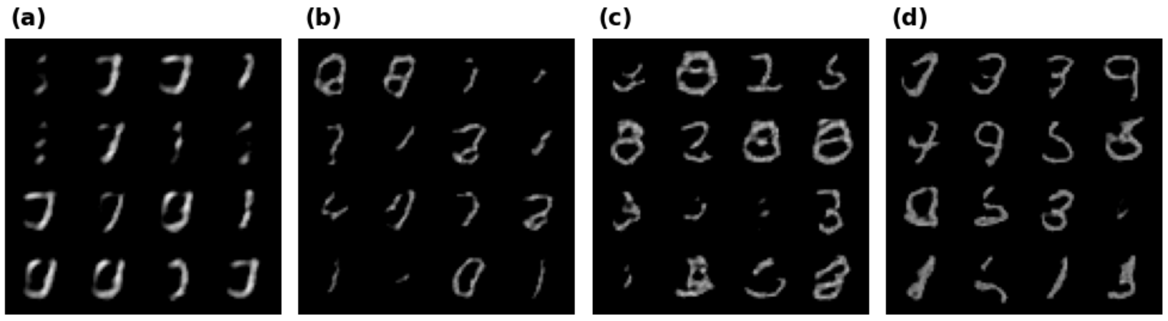

Figure 9: Samples generated by the cold diffusion model using after training for (a) 1, (b) 5, (c) 10, and (d) 25 epochs.

Another explanation is that there is a hidden inductive bias of the eraser degradation strategy which encourages generation of certain digits. It could also be that predicting 1s is a local minima of the loss function in algorithm 4 (eg. since ones intersect most other numbers and the backgrounds are consistent). Additionally, there are some meaningless symbols, and some of the digits look distorted and unrealistic. This implies that the model does have limitations in learning certain aspects of digit representation, or in dealing with the eraser degredation. To improve this model, we could try increasing the capacity of the restoration network, to see if it yields a similar performance boost as it did in section 1. We could also tune the training strategy by experimenting with other loss functions, like L1 loss, and optimizers, like stochastic gradient descent.

c) Gaussian Noise vs Eraser Comparison

As can be seen by comparing figures 5, 6 and 10, the cold diffusion model generated identifiable digits more consistently and with higher fidelity than both of the DDPM models. The digits appear more sharper, and have less noise or random artefacts. This is reinforced by the FID score of 89 achieved by the cold diffusion model, compared to 111 by the low-capacity and 95 by the high-capacity DDPM model. This suggests that for training diffusion models on the MNIST dataset, the eraser degredation strategy with $T = 100$ time-steps is superior to the standard Gaussian noise degradation for $T = 1000$ time-steps.

This enhanced performance may purely be a result of the lower number of time-steps, since it gives the model more 'practice' at each time-step, which is important when we are training for just 30 epochs. The MNIST dataset is very simplistic compared to other image datasets, thus $T = 100$ time-steps is sufficient. Alternatively, it may be that the eraser degradation is a superior degradation strategy for the MNIST dataset. For example, diffusion models trained on the MNIST dataset might favour degradations that are less stochastic than standard Gaussian noise degradation. The eraser degradation is partially stochastic due to the random background, but the trajectory and size of the eraser head is fixed. Furthermore, the cold diffusion model had the same capacity as the low-capacity DDPM model (and all the same hyper-parameters), which shows that the gain in performance from using the eraser degradation is larger than the gain from doubling the model capacity. This also proves that the capacity of the low-capacity DDPM model is sufficient to capture the complex features necessary for digit generation.

Despite the improvement in consistency and fidelity, the cold diffusion model generated less diverse samples than the DDPM models, with many of the generated images showing 1s. This may be a result of lower diversity in the distribution of fully-degraded latent images. For standard Gaussian noise degradation, z_T is normally distributed with 0 mean and unit variance, while for the eraser degradation we have $z_T \sim \mathcal{N}(c\mathbf{1}, 0.0025\mathbb{I})$ where $c \sim \text{Uniform}([-0.5, 0.5])$. The variance of 0.0025 is much lower than 1, causing a less diverse latent sample space (despite the uniformly distributed base colour). There also may be an inductive bias present in the eraser degradation that produces less diverse samples, which is not present in the standard Gaussian noise degradation.

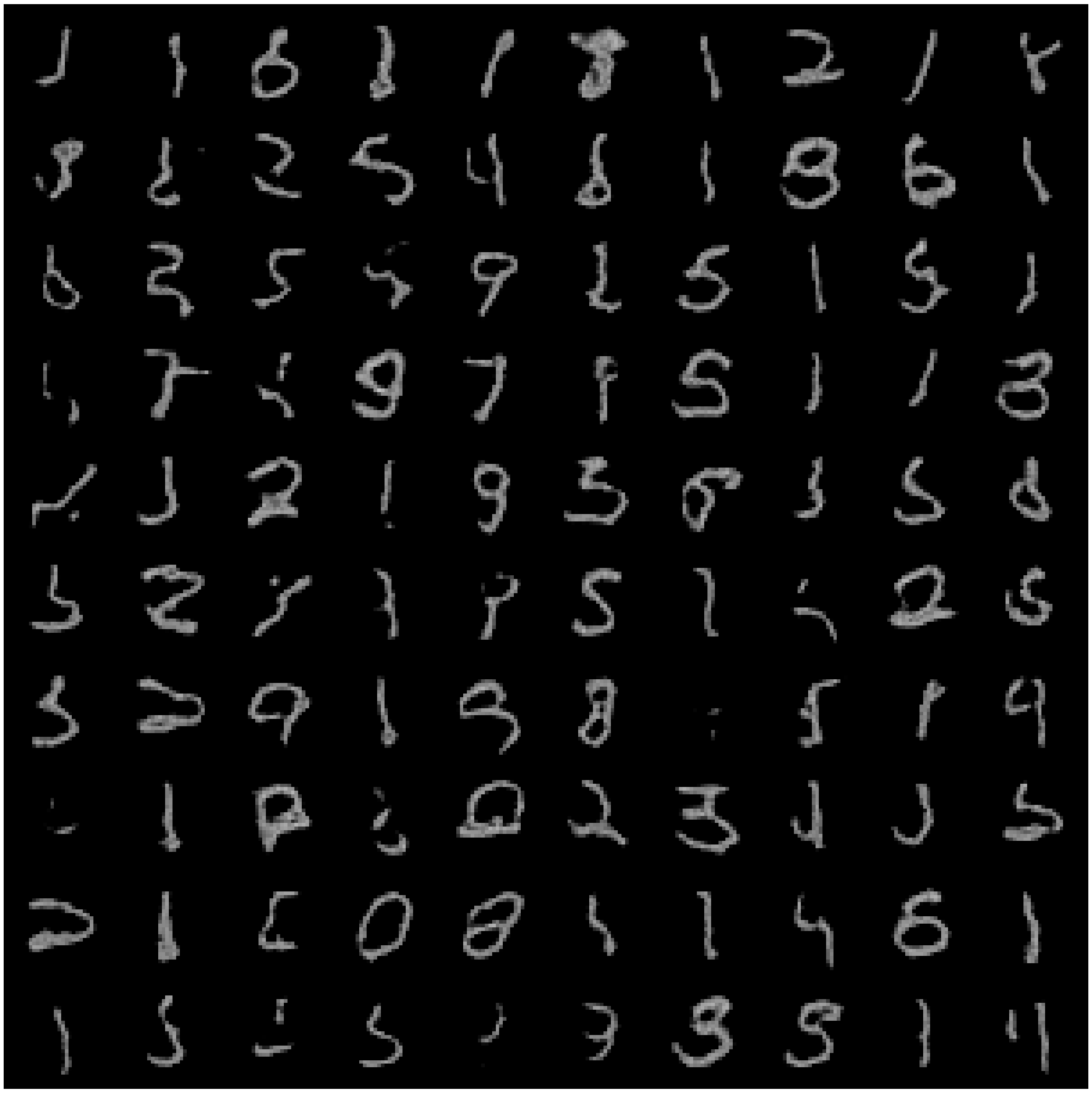

Figure 10: 100 samples generated from the cold diffusion model, trained for 30 epochs.

References

[1] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, S. Ganguli (2015). *Deep Unsupervised Learning using Nonequilibrium Thermodynamics.* Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2256-2265.

[2] J. Ho, A. Jain, P. Abbeel (2020). *Denoising Diffusion Probabilistic Models.* In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33 (NeurIPS 2020), Curran Associates, Inc., pp. 6840–6851. `https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf`.

[3] L. Deng (2012). *The mnist database of handwritten digit images for machine learning research.* IEEE Signal Processing Magazine, 29(6), pp.141–142.

[4] D.P. Kingma, J. Ba (2015). *Adam: A Method for Stochastic Optimization.* In Y. Bengio, Y. LeCun (Eds.), Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015. `http://arxiv.org/abs/1412.6980`.

[5] A. Bansal, E. Borgnia, H.-M. Chu, J. Li, H. Kazemi, F. Huang, M. Goldblum, J. Geiping, T. Goldstein (2023). *Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise.* In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Curran Associates, Inc., pp. 41259–41282. `https://proceedings.neurips.cc/paper_files/paper/2023/file/80fe51a7d8d0c73ff7439c2a2554ed53-Paper-Conference.pdf`.

[6] A.Q. Nichol, P. Dhariwal (2021). *Improved Denoising Diffusion Probabilistic Models.* Proceedings of the 38th International Conference on Machine Learning, in Proceedings of Machine Learning Research, vol. 139, pp. 8162-8171. Available from `https://proceedings.mlr.press/v139/nichol21a.html`.

[7] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter (2017). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.* In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30 (NIPS 2017), Curran Associates, Inc. `https://proceedings.neurips.cc/paper_files/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf`.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna (2016). *Rethinking the Inception Architecture for Computer Vision.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[9] N.S. Detlefsen, J. Borovec, J. Schock, A. Harsh, T. Koker, L. Di Liello, D. Stancl, C. Quan, M. Grechkin, W. Falcon (2022). *TorchMetrics - Measuring Reproducibility in PyTorch.* Available from `https://www.pytorchlightning.ai`. Code repository: `https://github.com/Lightning-AI/torchmetrics`. Released: February 11, 2022.

Appendix

A Auto-Generation Tools

Github copilot was used to assist with basic programming tasks like plotting figures. It was also occasionally used to assist with the generation of doc-strings of functions and files.