Generative Neural Networks - Diffusion Models

Nicolas Johansson
Chalmers University of Technology
MPDSC
nicolasj@chalmers.se

June 2, 2024

1 Introduction

With the advance of computing power and generative neural networks, generating images has become more and more popular. Modern generated images are of very high quality, often indistinguishable from the data used to train the model. One of the state-of-the-art techniques for generating such images is known as *diffusion probabilistic models*.

The basic idea of diffusion models is to add noise to a dataset and then learning to remove it. Adding noise to the data is known as the *forward process* $q(x_t|x_{t-1})$. This process is known, and typically consists of successively adding noise sampled from a Gaussian distribution, according to some predefined variance scheduling. When applying the forward process to an input data x_0 for a fixed amount of steps T, we gradually lose the original features to end up with *noise* x_T. The larger the T, the closer x_T will be to an isotropic Gaussian distribution [1].

To go in the opposite direction, we have to apply the reverse process $q(x_{t-1}|x_t)$. This process is unknown, and estimating it would be unfeasible. This is where machine learning comes into the picture. For each data sample, we sample a time step $0 < t \leq T$ and a true noise e_t from the standard normal distribution $\mathcal{N}(0, I)$. With these we compute the noised sample x_t, which can be fed into the model, which will try to predict the corresponding noise. By training a neural network on this task for a sufficient amount of epochs, we will eventually learn a model $p_\theta(x_{t-1}|x_t)$ for the reverse process.

Recall that for a large enough T, x_T will approximately belong to an isotropic normal distribution. Using this fact and our previously trained model, we can sample noised data $x_T \sim N(0, I)$ and successively reverse sample by applying $p_\theta(x_{t-1}|x_t)$, gradually building structure from noise until we end up with a generated sample x_0 resembling the training data.

Diffusion models have been shown to possess an ability to generate diverse and high-quality samples. They achieve state-of-the-art performance in generating many types of media; including audio, images, and video.

One of the most widely used applications is art, which has recently become simple for anyone to do, utilizing prompts describing the desired image. So called *text-to-image* models uses variants of diffusion probabilistic models (sometimes together with a prior model for embedding) to generate images conditionally on text prompts [2]. Figure 1 demonstrates such an image, generated by OpenAI's DALL-E.

Another field of application is within computer vision. For example restoring high resolution images from low resolution inputs (super-resolution) or reconstructing damaged regions in an image [2].

In this project, we will take a step back and look at the deeper workings of diffusion models. From the math involved, to the tricks and architectures for training such neural networks. We will also experiment with generating

Figure 1: Image created by DALL-E by prompting "racoons playing poker, digital art" [3].

simpler data in the form of point clouds of various shapes.

2 Method

The following sections will briefly describe the methodology for the various parts of the project.

2.1 1D diffusion process

In the first part of the project, the goal was to implement forward and backward diffusion processes for a known distribution. For simplicity, this was done for the one dimensional case, where $x0 \sim \mathcal{N}(m, p^2)$. The *mean* m was selected as 3.0, while the *standard deviation* p was chosen to be 0.5. For the variance scheduling, a linear schedule with $\beta_1 = 0.02$ and $\beta_T = 0.3$ was chosen. The amount of diffusion time steps was chosen to be $T = 50$.

For the forward process, samples were drawn from the x_0 distribution described above, and iteratively exposed to Gaussian noise T times. Each step here is calculated as in equation 1.

$$q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t) \tag{1}$$

By doing some reparameterization, letting $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, we can compute the forward process at any time t in closed form [1]. The closed formula in equation 2 is naturally much more computationally efficient than iteratively applying equation 1, and was therefore used when many samples were needed; like for approximating the distribution of x_T.

$$q(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t} \cdot x_0, 1 - \bar{\alpha}_t) \tag{2}$$

The reverse process $q(x_{t-1}|x_t)$ is harder to compute since it has to take the entire original distribution into account. However, the reverse conditional probability becomes tractable when conditioned on x_0 [1]. Using Bayes' rule and the previously mentioned reparameterization, we obtain the reverse process as described by equations 3-5.

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t) \tag{3}$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}} \cdot \beta_t}{1-\bar{\alpha}_t} x_0 \tag{4}$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t \tag{5}$$

Now, when reverse sampling, x_0 is of course unknown. However since we in this case know the distribution of x_0, we can use the formulas above with some minor modifications; replace $x0$ in equation 4 with the expected value m, and add the variance p^2 to equation 5. The updated formulas can be seen in equations 6 and 7, and was used to implement the reverse process in this task.

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}} \cdot \beta_t}{1-\bar{\alpha}_t} m \tag{6}$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t + p^2 \tag{7}$$

Looking closer at equations 6 and 7, note what happens when $t = 1$. Since $\bar{\alpha}_0$ is defined as 1, we get

$$\tilde{\mu}_1(x_1, x_0) = 0 \cdot x_1 + 1 \cdot m = m$$

$$\text{and} \quad \tilde{\beta}_1 = 0 \cdot \beta_1 + p^2 = p^2.$$

Therefore, in this known setting, computing the reverse process in *closed form* $q(x0|xt)$ would be equivalent to simply sampling $x0$ from $\mathcal{N}(m, p^2)$.

2.2 Modelling reverse processes with Neural Networks

In the second part of the project, sets of 2D points for three geometrical shapes were given; a circle, a square, and a dinosaur head. The goal was to learn three different models to reverse sample each of the shapes from noise.

Most of the math involved have already been presented in section 2.1. However to implement this, x_0 has to be represented as

$$x_0 = \frac{1}{\sqrt{\bar{a}_t}}(x_t - \sqrt{1-\bar{a}_t}\epsilon_t)$$

where $\epsilon_t \sim \mathcal{N}(0,1)$ is the noise at time step t. With this, we can reformulate equation 4 as equation 8.

$$\tilde{\mu}_t = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t) \tag{8}$$

Each model were trained on predicting this noise ϵ_t, given a noised sample x_t and time step t. To achieve this, for each batch (subset of points) in the training data, a time step t and a (true) noise ϵ_t (for both the x and y coordinates) were sampled. All points were then transformed using a vectorized function based on equation 2, yielding a tensor of 2D points x_t and a 1D tensor with corresponding time steps. These are then fed into a neural network, which predict the corresponding noise ϵ_θ for each point in the batch. The loss is then computed as the *mean squared error* between the true noise ϵ and predicted noise ϵ_θ, which is used to update the model weights through backpropagation.

For the neural network architecture, four fully connected layers were used (with 128, 256, 64, and 2 neurons respectively). ReLU activations were used for the first three layers, while the final (output) layer has no activation since the noise ϵ_t is expected to be continuous. Also, the inputs x_t and t were embedded using a sinusoidal embedding layer (with embedding size 32), before being flattened and fed into the fully connected layers.

All three models were trained using the Adam optimizer with a learning rate of 10^{-3}. Adam is an adaptive optimizer and generally works well for loss minimization. All models also used the same linear scheduling as in the original diffusion model by Ho et al.; with $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$ [4]. The amount of diffusion time steps was set to T=200. The circle and square models were trained using a batch size of 32 samples, for 100 epochs. Since the dinohead model had significantly more data (about 10 times the amount of points) to be trained on, it used a batch size of 128, and was trained for 300 epochs.

During interference, reverse sampling was performed by first sampling 1000 points from Gaussian noise $x_T \sim \mathcal{N}(0,1)$, and then successively using corresponding model to predict the noise for each time step; applying equations 3, 5 and 8 until t=0 and the reverse sampled points x_0 are an approximation of the original data.

2.3 Generating new data

For the advanced part of the project, the goal was to generate (to some extent) new data. A dataset of 1024 geometrical stars was given, each containing 1024 points.

Unlike the task in section 2.2 were each point were processed separately, the challenge here is that all points within the same star has to be considered together. This is important since there are many stars, and training the model like in 2.2 would result in a reverse process just representing some noisy average of all samples. The approach in this problem was therefore to introduce some common, permutation-invariant information for all points belonging to the same sample (star).

The training loop was updated to process each star within a batch separately (even if gradient descent is still performed with respect to the whole batch). For each star, we sample a *common* time step t and *individual* noise for each point; applying the same procedures as previously to obtain the diffused points. Then we compute the *mean* and *variance* (for both x and y) over the noised points. This information is then fed into the network in addition to x_t and t.

The neural network architecture was slightly altered to be more complex; using an embedding size of 128, and fully connected layers with 1024, 512, 128, respectively

2 neurons. Of course, the input layer was also altered to allow for the four new variables ($mean_x$, $mean_y$, var_x, var_y). Again linear variance scheduling was used, but now with T=200; trained for 300 epochs using a batch size of 32. Like before, Adam with learning rate 10^{-3} was used as optimizer.

The data generation is very similar to the one described in section 2.2, with the difference that the mean and variance (over all 1000 sampled points) are computed and used to predict the noise for each time step.

3 Results

The following sections will present the experimental results for each part of the project.

3.1 Forward and backward trajectories

Figure 2 demonstrates some forward trajectories computed according to what is described in section 2.1. By computing the forward trajectories of 10^6 sampled points x_0, the distribution of x_T was found to be approximately equal to the standard normal distribution. This is illustrated by the histogram in figure 3.

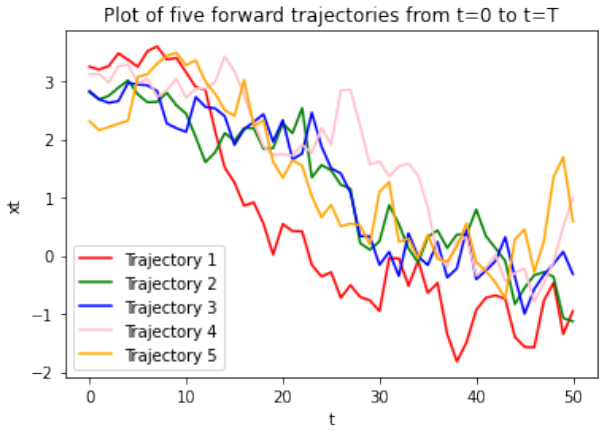

Figure 2: Five different 1D forwards trajectories, using linear variance scheduling, T=50 and $x_0 \sim \mathcal{N}(3, 0.5^2)$.

Five backward trajectories, and the approximated distribution of the reverse sampled x_0 are found in figures 4

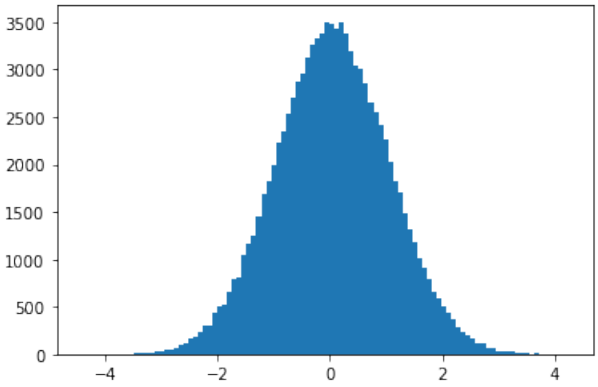

Figure 3: The distribution of x_T, approximated from 10^6 forward trajectories.

respectively 5. The mean and standard deviation of the reverse sampled $x0$'s are approximately equal to the selected ones (m=3 and p=0.5).

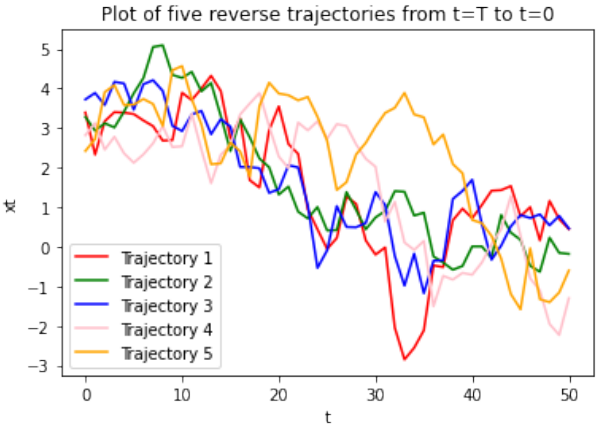

Figure 4: Five different 1D backward trajectories, using linear variance scheduling, T=50 and $x_T \sim \mathcal{N}(0, 1)$.

3.2 Reverse sampled 2D shapes

Figures 6, 7 and 8 demonstrates the results of the models, trained on the circle, square respectively dinosaur head datasets as described in section 2.2. The figures show results obtained at different stages of the training pro-

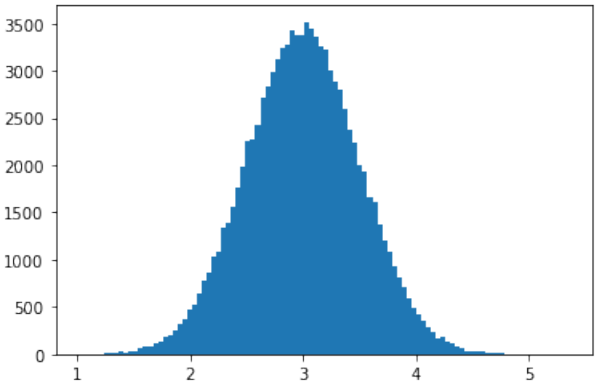

Figure 5: The distribution of x_0, approximated from 10^6 backward trajectories.

cess, including the fully trained models. An interesting observation made is the generation quality with respect to the training loss; although loss stops decreasing, training loss stops decreasing long before generation quality stops improving.

Some experimentation was done on variance scheduling. The constant scheduling provided quite bad results, with the generated dinosaur head being completely unrecognizable (just a circular cloud of points). The linear scheduling instead worked really well for these tasks. A cosine schedule was also tried, which showed promising results (perhaps even better than linear) but was abandoned due to some unknown bug; at the final few reverse sample steps, the points seemed to *collapse*. More research is needed to be sure, but the reason is possibly related to how the model is trained.

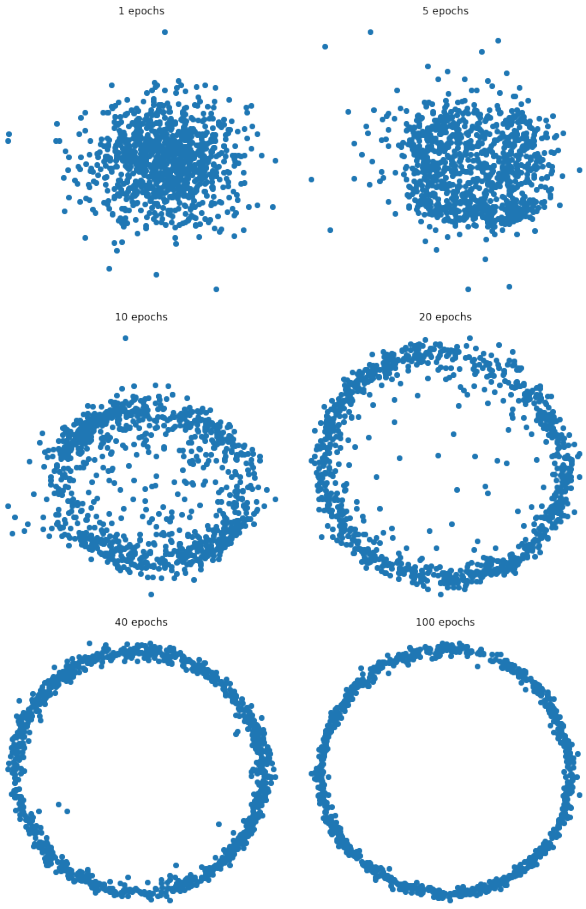

Figure 6: Reverse sampled circles; results after 1, 5, 10, 20, 40, and 100 (*all*) epochs.

3.3 Generated stars

The stars generated using the method described in section 2.3 was found to be somewhat noisy. Although there is room for improvement, the results still shows capabilities of generating varying stars resembling the ones in the training data. Figure 9 demonstrates 5 randomly generated stars.

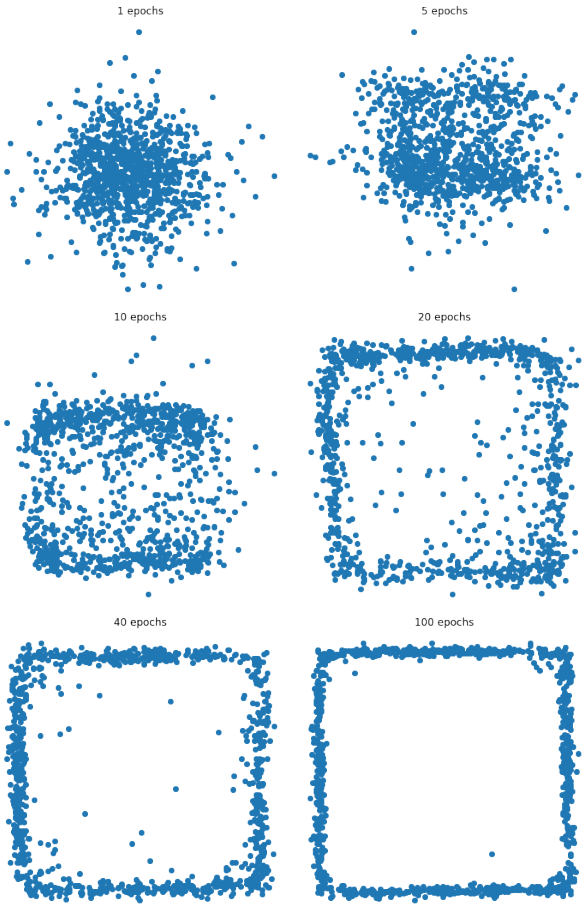

Figure 7: Reverse sampled squares; results after 1, 5, 10, 20, 40, and 100 (*all*) epochs.

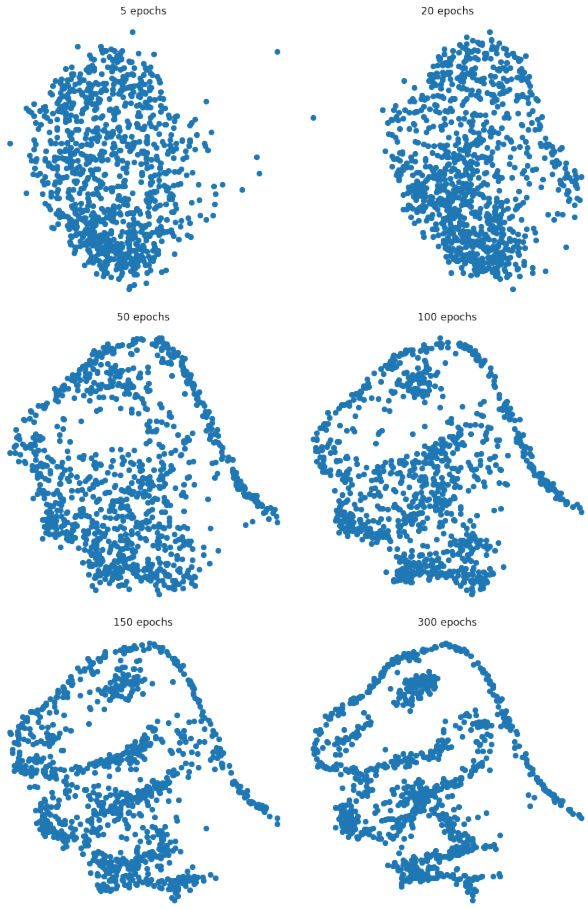

Figure 8: Reverse sampled dinosaur heads; results after 5, 20, 50, 100, 150 and 300 (*all*) epochs.

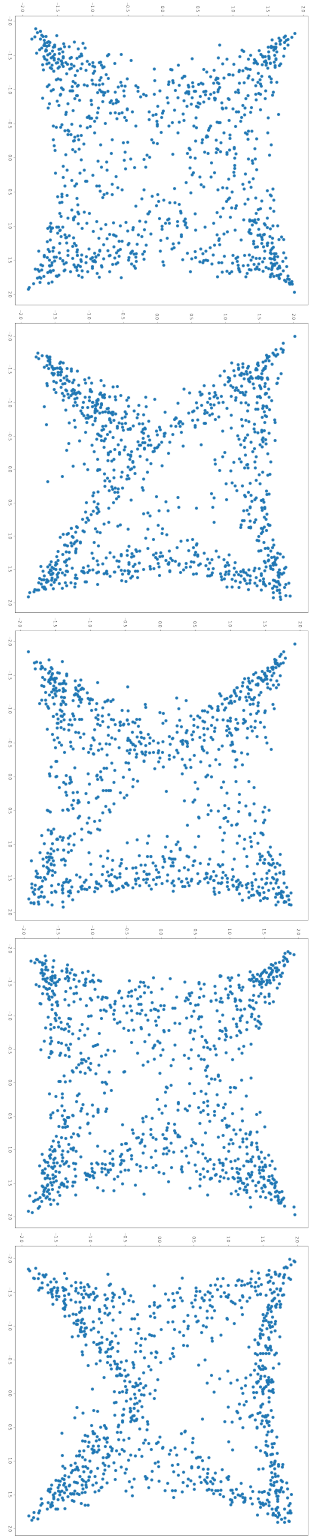

Figure 9: Five randomly generated stars. *Note:* image is flipped 90 degrees to fit page.

References

[1] L. Weng, "What are diffusion models?" *lilianweng.github.io*, Jul 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[2] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," 2024.

[3] M. Metzler, "Waschbär spielt poker, dall-e," 2023, cC BY-SA 4.0 via Wikimedia Commons. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=128548196

[4] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.