Introduction to Diffusion Models

This post provides an introduction to *diffusion models*, which are a type of machine learning model whose purpose—as in general unsupervised learning tasks—is to generate samples from a high-dimensional probability distribution. The difficulty of this task is that the probability distribution in question is not known explicitly: we are only given some number of *samples* from this distribution, and for practical purposes we are mostly in the regime of vast *undersampling*, meaning that the number of samples we are given from the distribution does not come close to populating the entirety of the vast space in which these samples are distributed. From these samples—the training data—our task is to learn to generate *new* samples which resemble aspects of the dataset on which we have trained (by virtue of being drawn from the same underlying distribution), but are not simply copies of the training data.

Diffusion models have found widespread and practical use in image and text generation. Here we provide a few minimal examples of how to construct and train a diffusion model, and we discuss the principles by which the diffusion model operates. The explanations are geared towards someone with a physics background, though we also try to make a connection with the statistics/computer science terminology which is widespread.

1 Diffusion models in one dimension

We start with the simplest incarnation of the task at hand: consider a scalar random variable of which we are given a number of samples—our *training dataset*—but whose distribution we do not know explicitly. Our task is to generate new samples from its parent distribution. However, we do *not* want to simply memorize the samples we have been provided, and spit them back with a few small variations. Ideally, we want to develop some *generalized* notion of the distribution from which they are drawn, and then draw samples from this generalized distribution. For a scalar random variable, one might suggest the following: plot a histogram of the samples we are given, and fit some function to this histogram: then use this fit as the probability density function of the distribution, and sample from this density. Although this seems reasonable for one-dimensional data, for high-dimensional data it is not clear how to generalize this strategy, and furthermore it would appear to require an amount of training data that is exponential in the dimension of the data to populate the space in which the data are distributed. One does not typically have this large of a training dataset. In practice it turns out not to be necessary: even with huge undersampling in the space of training data, empirically good generative performance can be achieved, which hints at underlying low-dimensionality in most real-world data distributions of interest.

Diffusion models aim to sample from a desired distribution, say $p_0(x_0)$, by considering a flow—in the space of distributions, parametrized by some "time" t—from $p_0(x_0)$ to some more tractable distribution, $p_T(x_T)$, from which we know how to sample. If we can reverse this flow, then we can turn samples from $p_T(x_T)$ into samples from $p_0(x_0)$.

1.1 Forward process

To be concrete, the flow being considered is that generated by a stochastic process which gradually erodes the structure in $p_0(x)$. Consider x drawn from the distribution $p_0(x)$. If we run dynamics on x given by

$$\dot{x} = -x + \sqrt{2D}\eta(t) \tag{1}$$

where $\eta(t)$ is white noise with correlations $\langle \eta(t)\eta(t')\rangle = \delta(t-t')$ (with angle brackets denoting an average over realizations of the noise), then the distribution of x is gradually converted from $p_0(x)$ to a gaussian distribution with variance D; this is because Equation 1 describes diffusion in a quadratic potential (with linear force pushing x back to the origin), and the equilibrium distribution of these dynamics is a Boltzmann distribution, which is a gaussian since the potential is quadratic.

It is useful to write Equation 1 in terms of *distributions* rather than explicit dynamics of a single samples: this is the shift from a Langevin description of the dynamics to a Fokker Planck description, which describes the deterministic time evolution of the distribution of x as opposed to the stochastic time evolution of a single sample of x. Defining $p_t(x)$ as the probability distribution of x at some time t, the Fokker Planck equation associated with Equation 1 is

$$\partial_t p_t(x) = \partial_x[x p_t(x)] + D\partial_{xx} p_t(x). \tag{2}$$

Note in particular that the steady state solution of this Fokker Planck equation (where $\partial_t p_t = 0$) is in fact a gaussian:

$$p_{t\to\infty}(x) \sim e^{-x^2/2D}. \tag{3}$$

If we want to use a diffusion model to sample from $p_0(x)$, we need to time-reverse the dynamics in a *distributional* sense. Then we can sample from $p_{t\to\infty}(x)$ and reverse the flow to get a distribution $p_0(x)$. Note that Equation 2 has an advection term (inside the single spatial derivative) and a diffusion term (inside the double spatial derivative). However, we can write it purely as an advection equation (denoting $p_t(x)$ simply as p for ease of notation) by rearranging terms as

$$\partial_t p = \partial_x[(x + D\partial_x \log p)p] = \partial_x[(x + DF(x,t))p], \tag{4}$$

where we have defined a time-dependent force field $F(x,t) = \partial_x \log p_t(x)$. If we had a black box that could calculate $F(x,t)$ given some x and t, then we could construct a Langevin process which time-reverses the Fokker-Planck equation. Such a Langevin process might look like

$$\dot{x} = x + 2DF(x,t) + \sqrt{2D}\eta(t), \tag{5}$$

but there is more then one option here: in particular one could even write down a totally deterministic reverse process (for a stochastic initial condition sampled from $p_{\to\infty}(x)$) given by

$$\dot{x} = x + DF(x,t). \tag{6}$$

Both these processes (and in fact others of the form $\dot{x} = x + 2D\alpha F(x,t) + \sqrt{2D(2\alpha-1)}\eta(t)$ for any $\alpha > 1/2$) reverse the distributional dynamics of $p_t(x)$ and therefore allow us to sample from $p_0(x)$.

1.2 The score function

The previously-described reversal of the Fokker-Planck equation assumes that we can calculate $F(x,t)$. However, $F(x,t)$, as seen above, depends on $p_t(x)$ and therefore depends $p_0(x)$, which we crucially *do not know*. The key insight that makes diffusion models possible is that we can use the forward process to estimate $F(x,t)$, namely by fitting a neural net to the data from our forward process. In particular, we want to train our neural network to convert input (x,t) into output $\partial_x \log p_t(x)$. Noting that

$$p_t(x_t) = \int p_0(x_0) q(x_t, t|x_0, 0) dx_0, \tag{7}$$

where $q(x_t, t|x_0, 0)$ is the transition kernel: the likelihood of transition from x_0 at time 0 to x_t at time t, we can explicitly calculate—from properties of the Ornstein-Uhlenbeck process—that

$$F(x_t,t) = \partial_{x_t} p_t(x_t) = -\frac{x_t - \mathrm{E}[x_0|x_t]e^{-t}}{D(1-e^{-2t})}, \tag{8}$$

where we have been explicit about the arguments of our probability density functions, denoting them with an underscore corresponding to the time at which their distribution is defined.

Equation 8 shows that in order to train our neural net, we can provide it with input-output pairs where the input is (x_t,t) and the output is $F(x_t,t)$. Noting that the score function contains the term $\mathrm{E}[x_0|x_t]$, we might wonder how to compute this term given that, since our variables live on the real line, each (x_t,t) pair will occur only once. As a result, $\mathrm{E}[x_0|x_t]$ is simply the x_0 (the initial condition) for the forward trajectory that produced x_t at time t. Although we might worry that this results in the neural net just memorizing x_0 for each (x_t,t) pair, smoothness of the function we are fitting means that the training procedure actually does the averaging for us, and the neural network—given enough data—learns the average value of x_0 conditioned on x_t.

1.3 Reverse process

After having learned the score function by gradient descent on our loss function of choice, we can run the reverse process to generate samples from our initial distribution of interest. When running the reverse process, we start with an initial condition drawn from the gaussian distribution that we know to be close to the final state of the forward process, and integrate Equation 5 in time, for the same amount of time that we ran the forward process for. Each sample from the gaussian distribution, run through the reversed dynamics, results in a sample from our distribution of interest.

In the following section we will concretely implement this process for a simple one-dimensional distribution, to see how it works in practice.

1.4 Concrete implementation

Figure 1 shows the "true distribution" that we will use in our toy example of a diffusion model. In Figure 2 we show 100 realizations of the forward process, wherein we take samples from the true process and use them as initial conditions for the Ornstein-Uhlenbeck forward process. Using the data from this forward process, we can train a neural net to predict $F(x_t, t)$ from (x_t, t), where $F(x, t)$ is given by Equation 8. The results of this process for various timepoints over the course of the forward process are shown in Figure 3.

We can then proceed to run the reverse process with our learned score function. Figure 5 shows a comparison between different types of reverse processes (one stochastic and one deterministic) and the true underlying distribution that is known because we are working with a toy example.

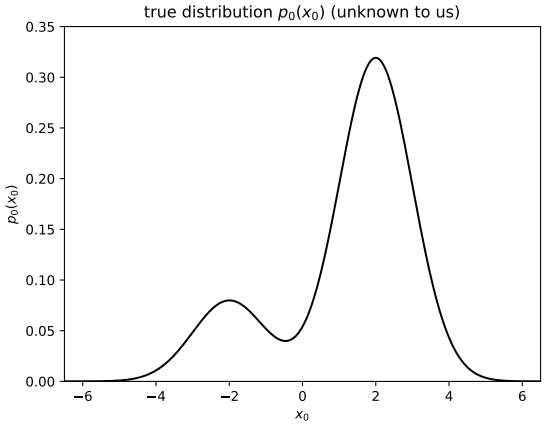

Figure 1: The underlying distribution from which we wish to sample. We do not have explicit access to this distribution, but are granted access to data points sampled from this distribution.

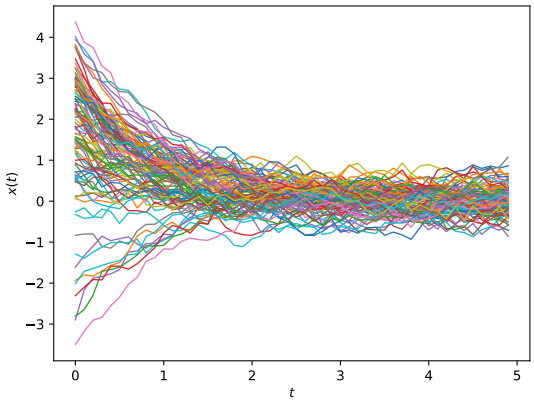

Figure 2: 100 realizations of the forward process. Each realization starts with a sample from $p_0(x_0)$ and adds noise to the sample (along with exponential decay towards zero): as we see this results in the information from the sample being lost and the distribution $p_0(x_0)$ being converted into the stationary gaussian distribution of the forward process. Since we run the forward process only for a finite amount of time, we will never *truly* reach the stationary distribution, but the convergence is exponential and so for practical purposes this does not matter.

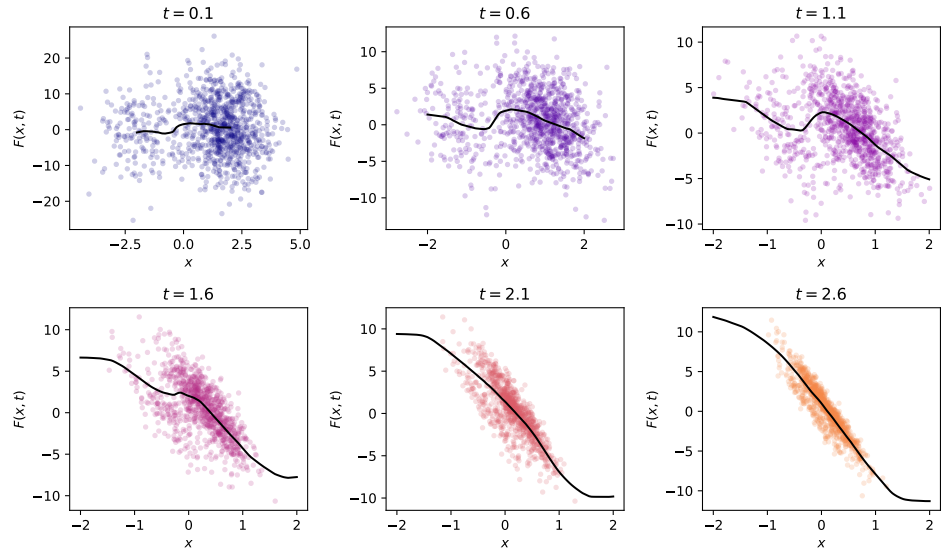

Figure 3: The score function as learned by the neural net. The black curves show the function computed by the neural net after training, and the points show the training data which come from the forward process. The training data for the figures herein are amalgamated from 1000 runs of the forward process.

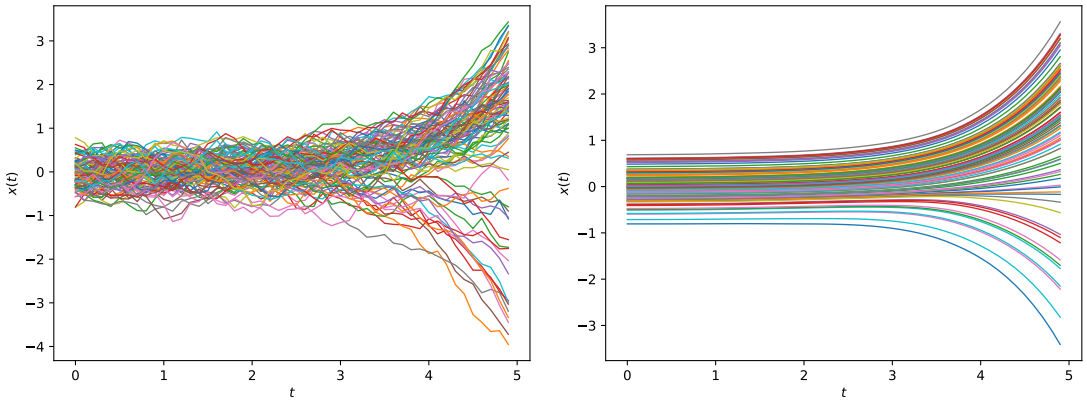

Figure 4: On the left, we show 100 realizations of the reverse process with the learned score function (Equation 5). On the right we show a different reverse process: this one is deterministic (Equation 6) and the stochasticity comes only from the initial condition.

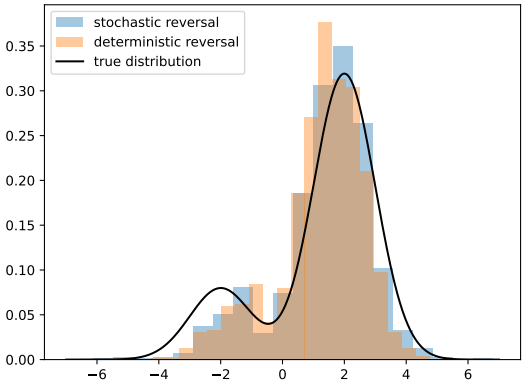

Figure 5: Comparison between the distributions sampled by the diffusion models (both stochastic and deterministic) and the true distribution.

1.5 Score fitting in the variational picture

Here we discuss an alternate formulation of the principles behind diffusion models. Above, we discussed training the diffusion model by fitting a neural net to minimize the loss in the training data, which it is natural to define as the mean squared error between prediction and true value of the score function.

However, one can also frame this error minimization procedure as a likelihood maximization procedure for a gaussian reverse process (just as minimizing the mean squared error for a regression problem can be thought of as likelihood maximization under a gaussian model). One subtlety is that instead of directly maximizing the log likelihood of our training data for a given model architecture, we will deal with a *lower bound* (the evidence lower bound or ELBO) on this log likelihood which is more convenient to work with. Let us denote the likelihood of datapoint x_0 as $p_\theta(x_0)$ (where θ denotes the parameters of the distribution that will be tuned to maximize the likelihood of the training data), and the distribution $q(x_1, \dots x_T|x_0)$ as the likelihood of the forward process assuming values $\{x_1 \dots x_T\}$ conditioned on initial value x_0.

For diffusion models, the ELBO is defined as

$$\text{ELBO} = \log p_\theta(x_0) - D_{KL}[q(x_1 \dots x_T|x_0) \| p_\theta(x_1, \dots x_T|x_0)], \tag{9}$$

where $D_{KL}[q(x)|p(x)] = \sum_x q(x) \log \frac{q(x)}{p(x)}$ is the Kullback-Leibler divergence and is guaranteed to be nonnegative.

1.6 Sidenote on variational inference

Here we see a common trick of variational inference: we have introduced a bunch of latent variables: here $\{x_1 \dots x_T\}$, and we write $p_\theta(x)$ as a marginalization of the joint distribution of p_θ over x and the latent variables. This then allows to derive an estimator for $p_\theta(x)$ since

$$p_\theta(x) = \mathrm{E}_{q(z|x)} \frac{p_\theta(x, z)}{q(z|x)}, \tag{10}$$

suggesting that $\frac{p_\theta(x,z)}{q(z|x)}$ is an unbiased estimator for $p_\theta(x)$. Note that this function is easy to evaluate once we have a generative model for x from the z, since $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$. However, we want an estimator for $\log p_\theta(x)$ (the quantity that we are trying to maximize), so we can put the $\log$ inside the expectation of Equation 10 at the cost of making the equality into an inequality. We now have a lower bound

$$\log p_\theta(x) \geq \mathrm{E}_{q(z|x)} \log \frac{p_\theta(x, z)}{q(z|x)}. \tag{11}$$

We will try to maximize this lower bound which is the ELBO function; note that it can also be written as

$$\mathrm{E}_{q(z|x)} \log \frac{p_\theta(x, z)}{q(z|x)} = \log p_\theta(x) - D_{KL}[q(z|x) \| p_\theta(z|x)]. \tag{12}$$

In the general setting of variational inference, we have not so far said what the distribution $q(z|x)$ is. We can take it to be parameterized by some parameters $q_\phi(z|x)$ and then do the optimization over the parameters ϕ as well

as θ. Then the outcome is not only an optimized $p_\theta(x)$ which is a generative model of the distribution we want, but also what is known as a discriminative model $q_\phi(z|x)$ which approximates $p_\theta(z|x)$. For diffusion models (unlike variational autoencoders), we do not do this optimization of q_ϕ: we just take this distribution to be fixed by the forward stochastic process. The latent variables $\{x_1, \ldots, x_T\}$ are then used to generate samples from the desired distribution.

1.7 Back to diffusion models

The advantage of the ELBO is that it can be written, for a Markovian kernel of the diffusion process, as

$$\mathrm{E}_{q(x_1 \ldots x_T|x_0)} \log \left(\frac{p_\theta(x_0 \ldots x_T)}{q(x_1 \ldots x_T|x_0)} \right) = \mathrm{E}_{q(x_1 \ldots x_T|x_0)} \left[\log p_\theta(x_T) + \sum_t \log \left(\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right) \right]. \tag{13}$$

Note that q describes the forward process and p_θ is the learned transition kernel for the reverse process. In particular this reverse transition kernel is parametrized by

$$p_\theta(x_{t-1}|x_t) \sim \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \tag{14}$$

In order to draw an analogy between this approach and the score-learning picture from the section above, consider again Equation 5: $\dot{x} = x + 2DF(x,t) + \sqrt{2D}\eta(t)$. In discrete time, this looks like $x_{t-1} = (1+\Delta t)x_t + 2D\Delta t F_\theta(x_t, t) + \sqrt{2D\Delta t}\epsilon$. Therefore

$$\mu_\theta(x_t, t) = (1+\Delta t)x_t + 2D\Delta t F_\theta(x_t, t), \quad \Sigma_\theta = \sqrt{2D\Delta t}. \tag{15}$$

If we then write, as is true for gaussian noise, that

$$\log p_\theta(x_{t-1}|x_t) \sim -\frac{1}{2} \frac{(x_{t-1} - [(1+\Delta t)x_t + 2D\Delta t F_\theta(x_t, t)])^2}{2T\Delta t}, \tag{16}$$

we can see that the ELBO is maximized when

$$F_\theta(x_t, t) = \frac{x_{t-1} - (1+\Delta t)x_t}{2D\Delta t} \sim -\frac{x_t - e^{-\Delta t} x_{t-1}}{D(1 - e^{-2\Delta t})}. \tag{17}$$

This expression is reminiscent of the optimization process in the score-based diffusion modeling described above, in which the score function looks like

$$F_\theta(x_t, t) = -\frac{x_t - \mathrm{E}[x_0|x_t]e^{-t}}{D(1 - e^{-2t})}. \tag{18}$$

2 Discrete state spaces

In the previous example the data were real numbers. This fact allowed us to use the Ornstein-Uhlenbeck process as the forward process, since it converges to a continuous gaussian distribution. However, one can also adapt a diffusion model to generate data which live in a discrete state space. For example, we could try to generate strings that come from a particular desired distribution. Here the forward process that takes us from a string to e.g. an i.i.d. random measure on consecutive characters cannot be a gaussian process simply because the state space for characters is not continuous. Equation 8 is therefore no longer valid for learning the score function.

What is the discrete analog of a score function to be learned? Let us deal in terms of discrete time. Then, under the forward process, with transition matrix elements $q(x_t|x_{t-1})$, the probability distribution on state space evolves as

$$p_t(x_t) = \sum_{x_{t-1}} q(x_t|x_{t-1}) p_{t-1}(x_{t-1}). \tag{19}$$

If we want to reverse this time evolution, we can write the reverse kernel as

$$q(x_{t-1}|x_t) = \frac{q(x_t|x_{t-1}) p_{t-1}(x_{t-1})}{p_t(x_t)}. \tag{20}$$

In this case we can write

$$p_{t-1}(x_{t-1}) = \sum_{x_t} q(x_{t-1}|x_t) p_t(x_t). \tag{21}$$

The log reverse kernel is therefore given by

$$\log q(x_{t-1}|x_t) = \log q(x_t|x_{t-1}) + \log\left(\frac{p_{t-1}(x_{t-1})}{p_t(x_t)}\right). \tag{22}$$

Note that the elements $q(x_t|x_{t-1})$ are known: these are properties of the forward process which we are allowed to choose. The machine learning task is then to learn the function $\log \frac{p_{t-1}(x_{t-1})}{p_t(x_t)}$.

Let us specialize now to the case where the state space is a collection of Ising spins. We will specify a forward process which, at each timestep, flips each spin independently with probability r—the stationary distribution of this forward process is an i.i.d. distribution over spins where each spin is uniformly up or down. From the forward process we can construct a training dataset where the input is (x_t, t) and the desired output is $q(x_{t-1}|x_t)$, which is a delta function at the true output (for the training dataset). Our goal is to learn the reverse transition kernel.

Denoting an Ising spin configuration as x_t, and a particular spin i as $x_{t,i}$, we want to learn a local distribution $q_\theta(x_{t-1,i}|x_t, t)$ for each spin in the lattice, where θ denotes of the parameters of the neural net that we use to parametrize the kernel. When generating new samples, each spin evolves independently under the learned kernel and the correlations between spins come from conditioning on the entire spin configuration.

We might worry that since the model will just see each x_t once (given the enormity of state space) it will just memorize the x_{t-1} associated with this x_t. The success of diffusion models is predicated on this memorization not happening, which corresponds to the model learning to *generalize*. In particular, the model does *not* simply learn that the transition kernel for a spin $x_{t,i}$ at time t is a delta function at $x_{t-1,i}$: it ideally learns general patterns for what state a spin tends to be in, conditioned on the rest of the spin configuration. Given the locality of the Ising model Hamiltonian and the resulting spin configurations that it generates, one could attempt to learn the reverse kernel conditioned not on the entire previous spin configuration, but on some local filter of the previous configuration, which emphasizes the neighborhood of spin i in determining its reverse transition probability. This is precisely how convolution layers are useful for learning the reverse diffusion process: if we used a convolutional neural net to learn the mapping from x_{t-1} to x_t, it would be better able to take advantage of the local structure in this mapping. Though we will not explore the use of convolutional layers in diffusion models here, they are widely used in practice for image generation.

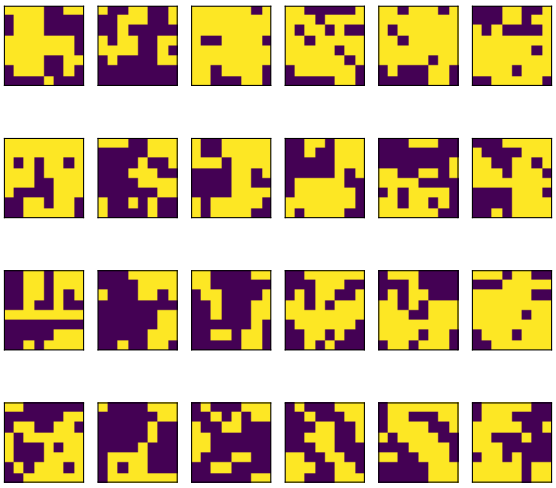

Figure 6: Sample images from an 8×8 Ising model at inverse temperature $\beta = 0.35$, in the paramagnetic phase (compare to critical $\beta_c \approx 0.44$ at which the phase transition occurs. We trained the diffusion model on 5000 such images.

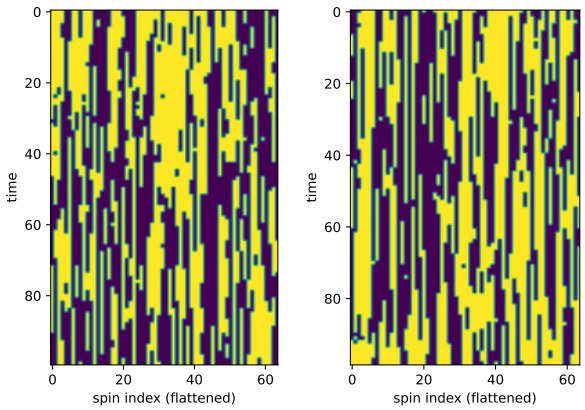

Figure 7: Forward and reverse processes for a single sample from the Ising model. The left image is a forward process which, at each timestep, flips each spin with probability $r = 0.05$ and therefore converges to an i.i.d. uniform distribution on spins. The 2D image is flattened out into a 64 dimensional vector. The right image shows a specific instance of the backward process, with the learned reverse transition kernel guiding an i.i.d. sample to a sample from the generalized distribution learned from the training process.

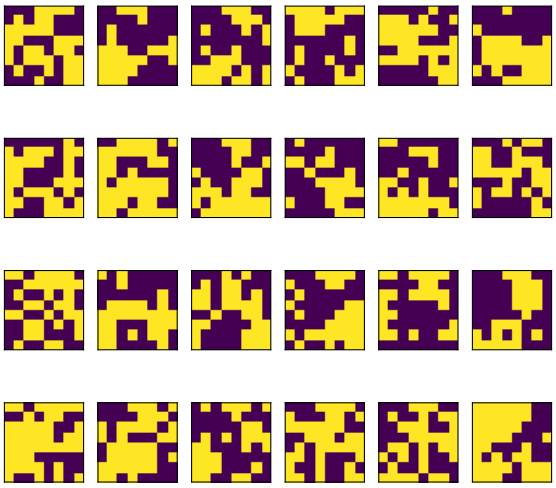

Figure 8: Sample images generated by the diffusion model trained on Ising configurations at inverse temperature $\beta = 0.35$. Some amount of spatial correlations (with up spins and down spins tending to cluster together) appears to have been learned.

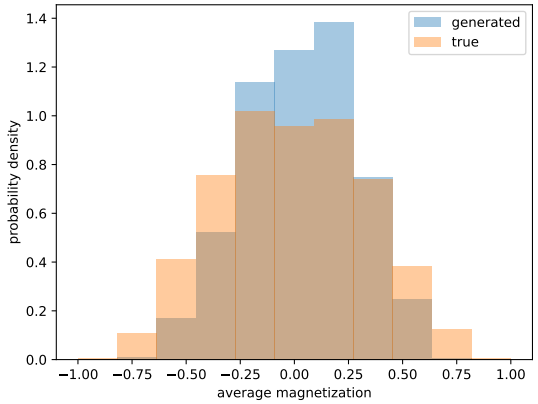

Figure 9: A histogram of the average magnetization $M = \frac{1}{64}\sum_i \sigma_i$ for samples drawn (1) generated by the diffusion model and (2) drawn directly from the underlying Gibbs measure on the Ising model. The approximate agreement shows that the diffusion model has learned something about the distribution from which Ising spin configurations are drawn. Running the same process at different Ising temperatures results in a differently-trained diffusion model that produces samples similar to spin configurations at the temperature, both in the ferromagnetic and paramagnetic phases (not shown).

3 Generating MNIST digits

We conclude with one more example, this time again for data in a continuous space, but now with high-dimensional image data as opposed to the one-dimensional toy example that we explored earlier. We will try to generate images of handwritten digits using a denoiser trained on the MNIST dataset. The process is the same: treat each 28×28 grayscale image as a vector in 784 dimensions, and run the forward process as an Ornstein-Uhlenbeck process (Equation 1). Then train a neural net to predict the score function (Equation 8), and run the reverse process to generate images.

One interesting feature of the MNIST dataset is that there are 10 natural categories into which the data fall. The empirical distribution from which the training data are drawn therefore consists of 10 clusters of data points in some high-dimensional space, with each cluster being populated by instances of the corresponding hand-drawn digit. When running the reverse process, we do not have control over which digit we end up generating (if any). This is analogous to saying that if we trained a diffusion model on Ising spin configurations in the ferromagnetic phase, we would not have control over whether our generated images had positive or negative magnetization.

Therefore, on the right side of Figure 10 we see that the digits that get generated by the model assume a variety of identities. If we wanted to bias our model to generate, say, images of the digit '3,' we could use a classifier model trained to classify digits and bias the reverse process to take steps where the likelihood of the image being classified as a '3' increases. This is sometimes used in practice to guide the output of generative models toward a desired region of the distribution from which they sample.

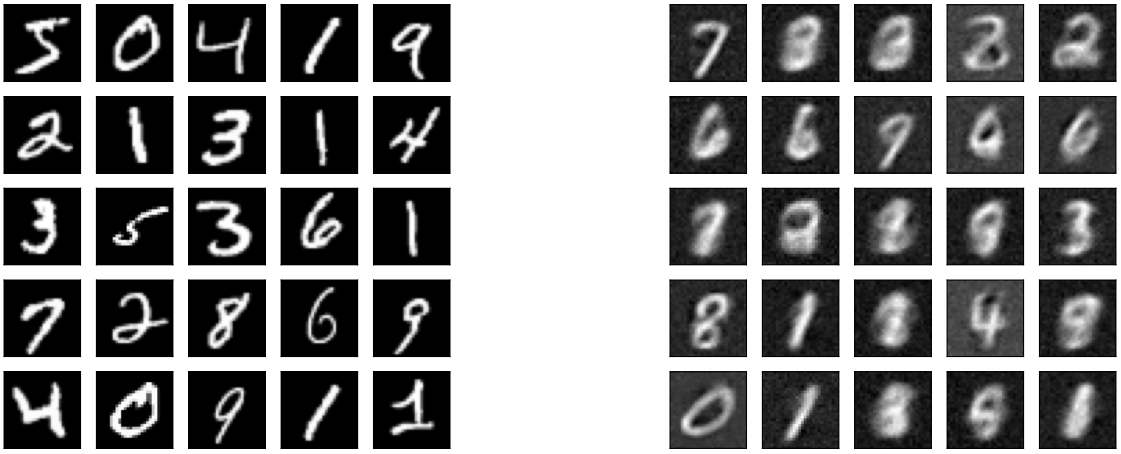

Figure 10: On the left we show a sample of images from the MNIST dataset, on which we trained our diffusion model (we trained on 2000 such samples). On right is a sample of images generated from the trained model. Note that some of the images are not discernible as digits, while others are recognizable.

In summary, we have implemented a few basic diffusion models in both continuous and discrete state spaces, and seen how they function from the perspective of learning a time-dependent force field that interpolates between easily-sampled noise and the data distribution of interest. Exactly why these models tend to work for most datasets of interest remains a mystery, but suggests that there tends to be underlying structure in high dimensional empirical distributions that is not obvious but is nonetheless discoverable by these models. Therefore they can robustly generate samples even when the training data are highly sparse in the space where they live.

4 Useful References

Diffusion Models: A Comprehensive Survey of Methods and Applications (Yang et al)
Generative diffusion in very large dimensions (Biroli, Mezard)
Deep Unsupervised Learning using Nonequilibrium Thermodynamics (Sohl-Dickstein et al)
Denoising Diffusion Probabilistic Models (Ho et al)
Score-based Generative Modeling Through Stochastic Differential Equations (Song et al)