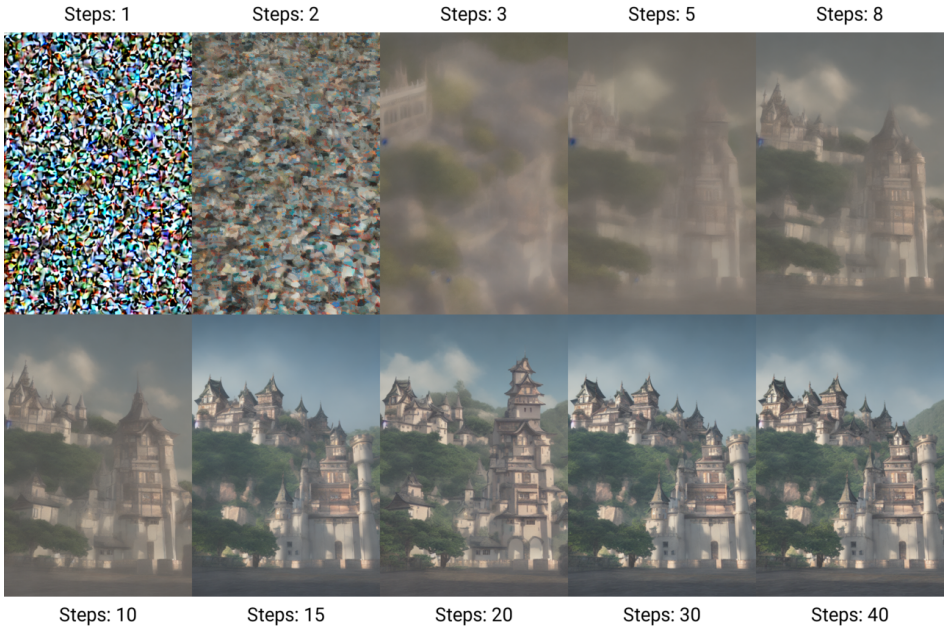
Unit 10 – Diffusion Models

Diffusion models are at the heart of generative A.I., they power media generation (images, video, and audio), 3D model generation, graphic design; and newer versions are being applied to code generation (Gemini Diffusion) and protein folding (AlphaFold 3). The last two applications are interesting because they have traditionally been the domain of transformer-based architectures.

The mathematical foundations that support the training and optimization of diffusion models are fairly involved, but at the center of what makes them work lies a small set of important properties of data in high-dimensional manifolds that can be used to gain a solid understanding of why these models work, and how they achieve the impressive feats of content generation that they are capable of.

One simple way to understand what a diffusion model is expected to do is illustrated below in the context of image generation:



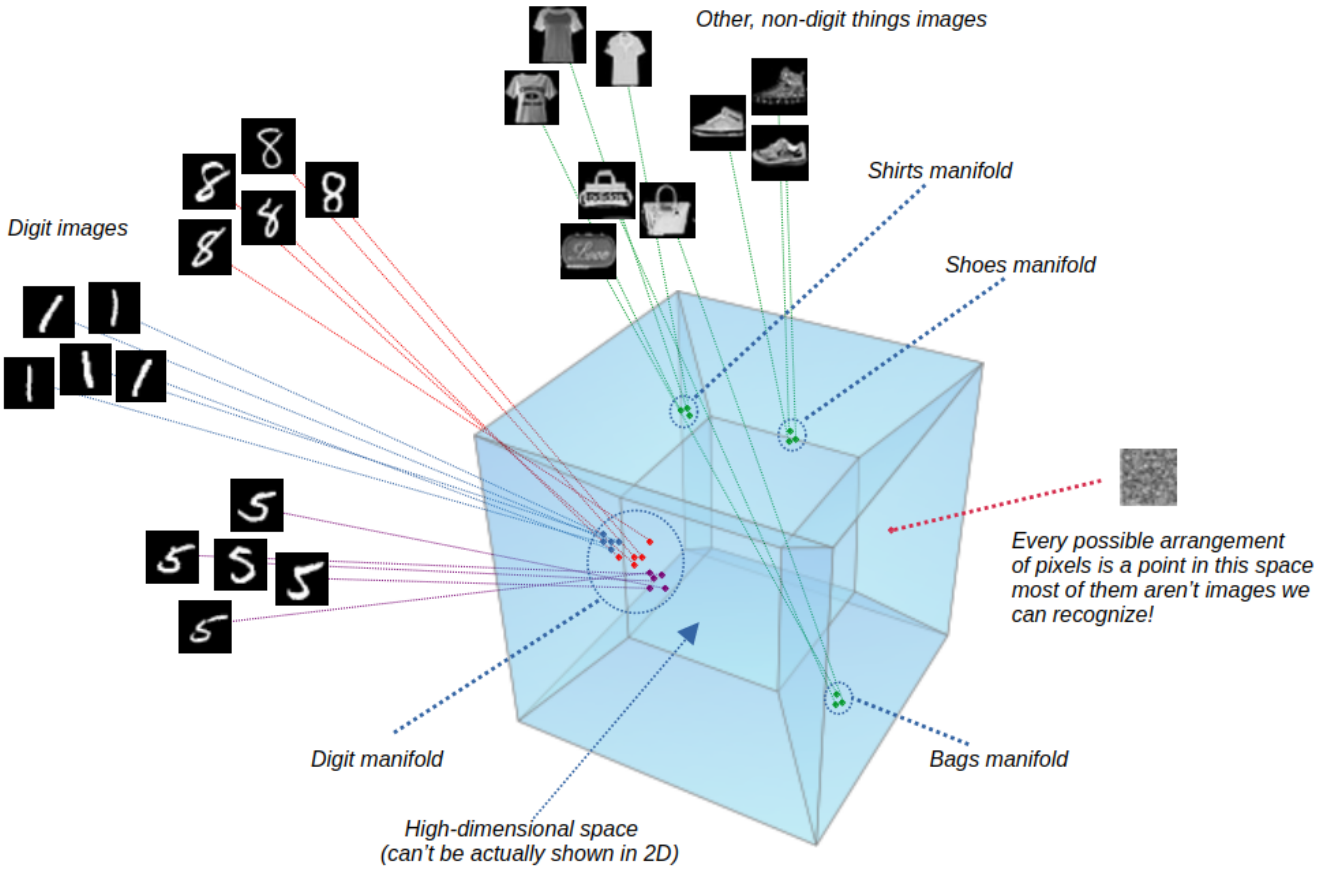
(Image: Berlinsquare, Wikimedia Commons – CC-SA-4.0)

The diffusion models starts with a ***prompt*** for what the image should contain (in the above case, a castle), an with an initial image of ***Gaussian noise***. The model then refines the image iteratively to produce a result that agrees with the prompt while also ***exhibiting novelty –*** the new image should clearly match the prompt, but should not be a trivial modification of an image from the training set used to build the model.

This may seem like magic – ***how is it possible to go from an image of noise to something that has such clear and specific structure?*** – studying this question is the key for understanding how diffusion models work, or indeed, understanding that they can and should be expected to work! Let's see how this is possible with a simple, low-dimensional example.

Data sets and manifolds

A commonly held belief in deep learning is that ***training data*** tends to form a ***manifold*** within the high-dimensional space that contains it. Here's an example – consider the space of ***handwritten digit images*** of size ***28x28*** (standard for many hand-written digit recognition applications). Each image is a point in a space with ***784*** dimensions (one per pixel).



The space of all possible images is ***huge*** and contains every possible combinations of pixel values that can occur in a 28x28 image. This includes ***images of every possibly thing one can show in a 28x28 image***, as well as lots of images with unrecognizable content, and even completely random arrangements of pixels.

But, the images of digits are ***not distributed randomly and uniformly*** within this high-dimensional space. Instead, notice that the images of digit '1' are fairly similar to each other, the images of digit '8' are fairly similar to each other, and so on. This means that images of the digit '1' will be at locations within the 784-dimensional space that are fairly close to one another, and the spatial arrangement of these images is not random – it has ***structure*** that ***provides information about how images of '1' vary*** within this high dimensional space. Similarly, digits '5' and '8' have images that occupy some region of the high dimensional space, the regions they occupy have structure corresponding to the variations in the images for each class.

These spatial arrangements of ***points*** corresponding to a particular ***class*** of image forms what we call a ***manifold***. This is a ***subspace*** typically of lower dimensionality (e.g. in the example above, a subspace with d << 784) that is smooth at the smallest scales – Regularities in the images belonging to

a particular class produce a ***subspace*** that exhibits structure that is specific to the class and that changes smoothly as we move over the space spanned by all the members of the class. To make this clear in your mind, pick a specific image of a '5', find the point in the 784-dimensional space corresponding to this image, and then consider where you would expect to find the point for an image that is ***identical except for 1 pixel, whose value has been changed by 1 gray level*** (out of 256 possible values).

The point is that it is possible to ***smoothly interpolate*** between nearby data points on the ***manifold***, and the intermediate points will correspond to images that look like ***members of the same class***. Each image class will have its own complicated manifold (some of these may intersect in interesting ways). And each manifold, no matter the size of the class and how complex the objects in the class may be, will ***occupy a region of the high dimensional space that is locally smooth*** and that ***occupies a much smaller volume*** than is spanned by the entire high-dimensional volume in which the data resides.

Machine learning methods take advantage of the ***structure*** and ***smoothness*** of manifolds. In particular, we have seen that ***deep neural networks*** are particularly good at approximating complex real-valued functions. So it should not be surprising that they can learn to approximate the structure of complicated manifolds even in high-dimensional spaces such as those occupied by images.

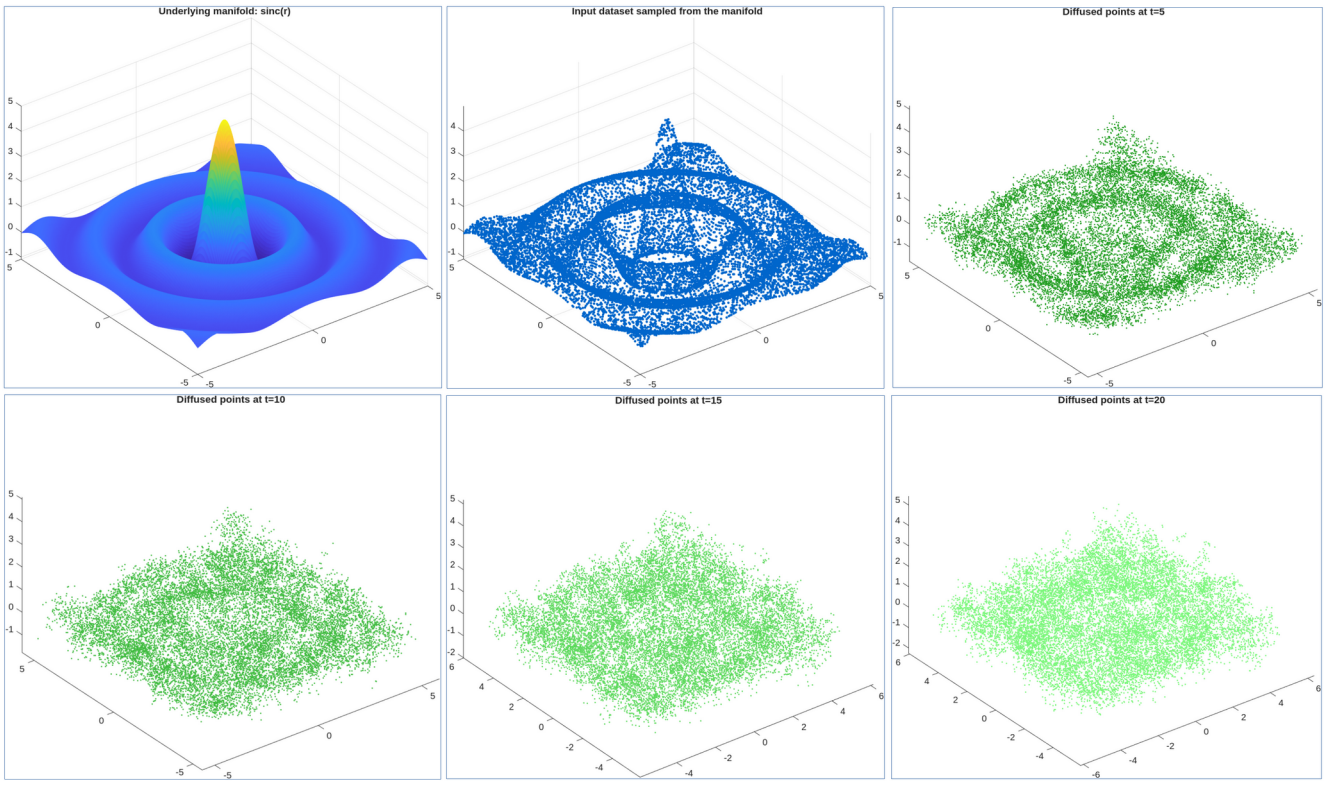
From manifolds to noise – the forward diffusion process

The first step for understanding diffusion models is to study the process of ***diffusion*** – so called because it is analogous to the process of ***heat diffusion***. The idea is as follows: Consider a set of points x_i from set K on an ***manifold*** in $\mathbb{R}^n$, we can simulate a diffusion process by repeatedly applying the following procedure:

$$\vec{x}_i^t = \vec{x}_i^{t-1} + \sigma_t \vec{w}_i^t$$

given the current set of points at time ***t-1***, produce a new set by ***adding zero-mean Gaussian noise*** (the noise vector w is drawn from a multi-variate Gaussian distribution with diagonal unit covariance). The amount of noise is controlled by σ_t, and it could be different for each time step, but for simplicity we could also imagine it to be ***small*** and ***constant***. Given the initial set of point at ***t=0*** (which would be sitting on the data's original ***manifold***), the above diffusion process will yield (after suitably many steps) a cloud of points distributed as an ***isotropic Gaussian*** (with diagonal covariance). In other words – the diffusion process progressively removes the structure present in the input's manifold and turns it into a blob in $\mathbb{R}^n$.

This process is called ***forward diffusion*** and is illustrated below for data sampled from a manifold in $\mathbb{R}^3$.



Original manifold, sampled points, and cloud of points after 20 steps of diffusion

You can easily see that the diffusion process muddles and eventually removes information about the original shape and location of the manifold. The final cloud of points at ***t=20*** could have come from an entirely different manifold.

Getting back to the manifold – reversing diffusion

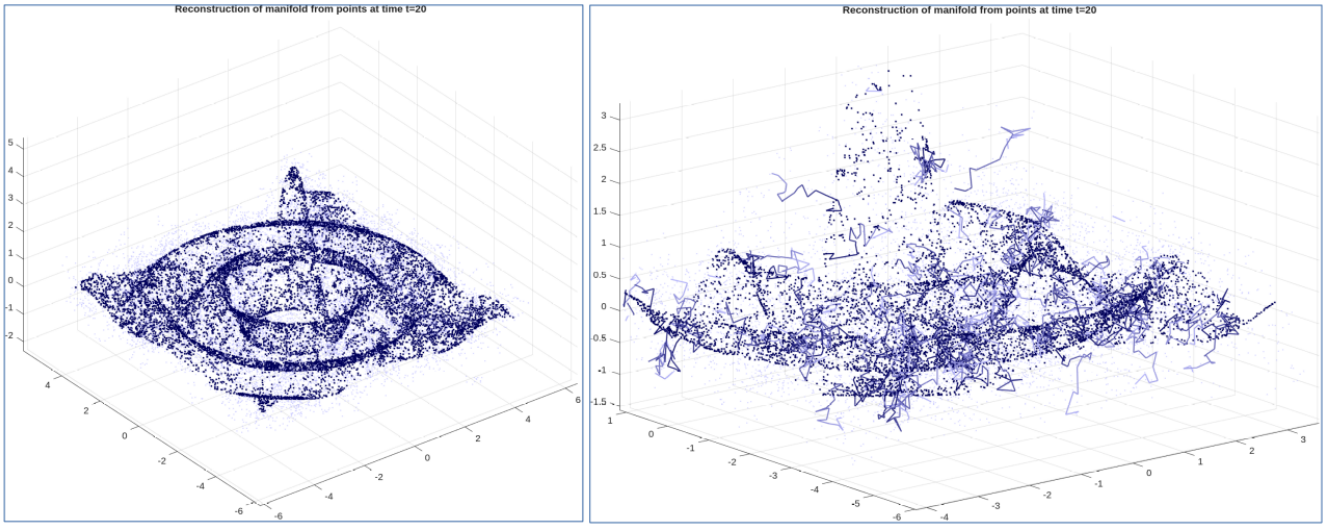
The question now is ***how could we get from a point in the point cloud at t=20*** back to some ***point in the original manifold***. A more general question is ***how could we get from a randomly sampled point in*** $\mathbb{R}^3$ ***back to the manifold.*** Ideally, we would like the final point to be ***different*** from our initial samples (i.e. we want to ***generate*** a ***new*** point on the manifold, something we haven't seen before), and we want to be able to reach ***any point on the manifold***.

In the example above, ***because we have clouds of points for each step of the diffusion process*** we could try a very simple process:

```
Repeat until t=0
    Given a point x_k at t=k
        Find nearby points in the point cloud for t=k-1
        (look for any points within a sphere of small radius centered at x_k)
        Compute the weighted mean μ of these points (weighted by inverse distance)
        Sample a new point x_k-1 from a Gaussian distribution with mean μ and small covariance
```

The above process will iteratively move in the direction of the manifold because for any value of ***t***, the point cloud at ***t-1*** is on average ***closer*** to the manifold, which means that averaging over a small region as the value of ***t*** is decreased will move the estimate closer and closer to the original surface from which the starting points were sampled. At the same time, because we are actually ***sampling*** from a ***Gaussian*** distribution, we are generating new points at each time step that we have not seen before in any of the point clouds.

If we apply the above process (with suitable values for the different parameters) we obtain a set of points that clearly shows the shape of the original manifold:



Final points after reversing the diffusion process for every point at ***t=20***. Notice the final points cover the original surface fairly well. The image to the right shows a crop of the manifold and the trajectories of several points from their initial location at t=20 to their final spot close to the original manifold

The take home message is simple: to get back onto the manifold form wherever a point may be, all we need to know is ***what direction moves that point closer to the manifold.*** In the example above we used the point clouds generated at each step of the diffusion process. ***But in practice, for any real application of diffusion, we can't do that***.

The ***manifolds*** for realistic problems are ***too complex***, the spaces these samples occupy are ***too high-dimensional***, the ***amount of training data*** required to learn their structure is ***huge***, and there is no way we could possibly keep around additional data for the diffused samples over possibly hundreds of iterations of the diffusion process. This is where deep learning comes in.

Diffusion models

A ***diffusion model*** is a ***deep neural network*** that has been ***trained to predict***, given an input point x_t and a value σ for the amount of noise used to generate it; what the direction is that would move this point toward the original manifold. The model is applied iteratively to get points that are closer and closer to the manifold until the last iteration yields a ***new point not in the original training set*** but nevertheless one that belongs in the same class – be that a never before seen hand-written digit, or a picture of a cat stealing a roasted turkey from the kitchen table.

The formulation for diffusion discussed below is from the paper by Permenter and Yuan (ICML 2024). Note that this is not the standard or most common formulation, but it is the one most closely related to the geometric intuition described above.

The input training data consists of points

$$\vec{x}_\sigma = \vec{x}_0 + \sigma\vec{\epsilon}$$

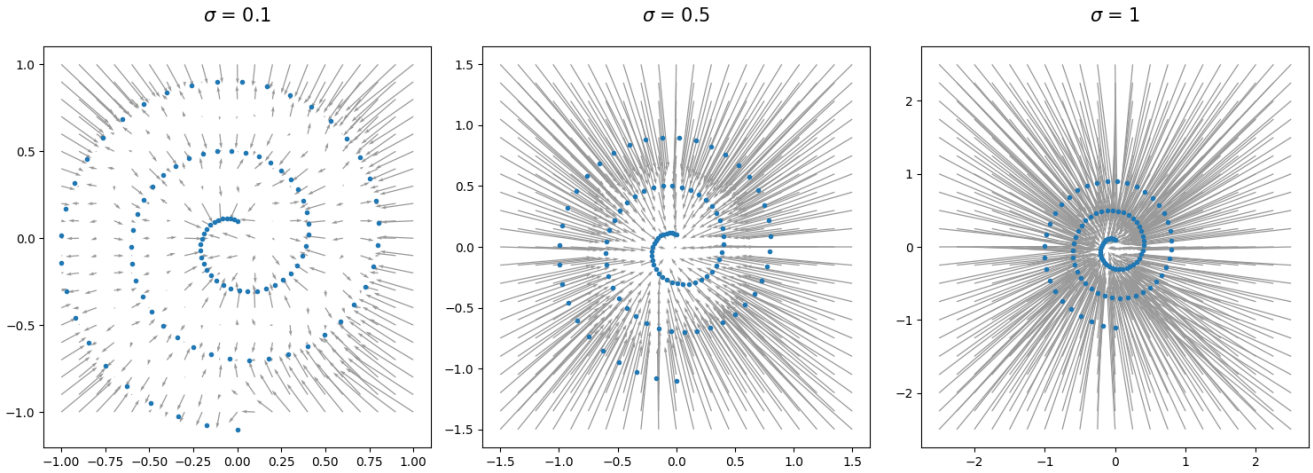
The initial point $\boldsymbol{x_0}$ is sampled from the training set (e.g. images of digits), and the noise vector $\boldsymbol{\varepsilon}$ is sampled from a ***zero-mean multi-variate Gaussian with identity covariance***. The standard deviation $\boldsymbol{\sigma}$ is sampled from within an interval $\boldsymbol{[\sigma_{min}, \ \sigma_{max}]}$. The network is trained with different noise levels because the directions it needs to learn depend on the amount of noise added to produce $\boldsymbol{x_\sigma}$ as will be seen below.

Notice that the network never gets to 'see' the original, clean input data points. This is important – the network can not ***cheat*** by spitting back data from the original training set.

We can train a deep network with parameters θ (this is a standard notation in machine learning to represent, in this case, all the weights that need to be learned): $\epsilon_\theta(\vec{x}, \sigma)$. The network is trained to ***predict the amount of noise*** added to the input points $\boldsymbol{x_0}$ in order to generate the corresponding noisy samples. This can be done by setting the ***loss function*** for minimizing ***expected squared error***:

$$L(\theta) = E||\epsilon_\theta(\vec{x}_0 + \sigma\vec{\epsilon}, \sigma) - \vec{\epsilon}||_2^2$$

The actual details of the network architecture and training process are dependent on the type of input the network has to work with (for example, if the network is going to be used to generate images, an architecture such as UNET may be used, for other problems the architecture will be different). With the appropriate training process, the network learns to predict the direction toward the manifold in a way that is dependent on the position of a point, and the amount of noise added.



A set of training data (blue dots) on a spiral manifold, and predicted directions from random locations in 2D toward the manifold as a function of $\boldsymbol{\sigma}$. Image by Chenyang Yuan (***https://www.chenyang.co/diffusion.html***)

The critical insight from Permenter and Yuan's paper is that the ***denoising process can be thought of as an approximation to projection*** from a point outside the manifold, to points on the manifold.

First, they define ***distance from a point to a set*** as

$$\text{dist}_{\mathcal{K}}(\vec{x}) = min\{||\vec{x} - \vec{x}_0|| : \vec{x}_0 \in \mathcal{K}\}$$

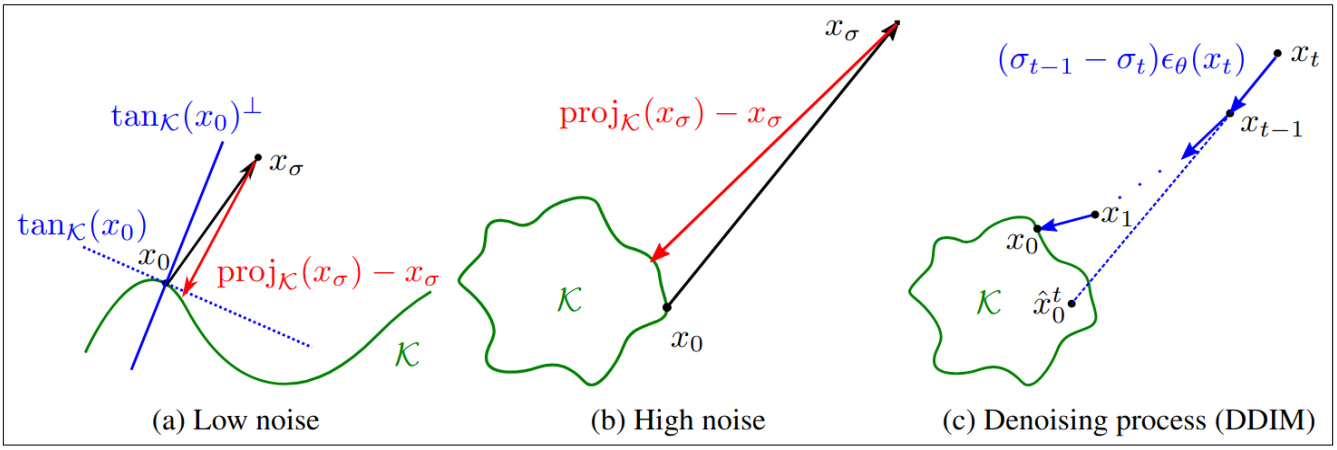
given the distance from a point to the set, the ***projection*** of the point onto the set is defined as

$$\text{proj}_{\mathcal{K}}(\vec{x}) = \{\vec{x}_0 \in \mathcal{K} : \text{dist}_{\mathcal{K}}(\vec{x}) = ||\vec{x} - \vec{x}_0||\}$$

and if the projection is unique (there is a single point x_0 at distance $dist_K(x)$), then the ***gradient of the distance function*** points in the direction of x_0

$$\nabla \frac{1}{2}\text{dist}_{\mathcal{K}}(\vec{x})^2 = \vec{x} - \text{proj}_{\mathcal{K}}(\vec{x})$$

This works because of two key observations illustrated on the figure below



Geometric interpretation of gradient descent as projection onto the manifold. Image from Permenter and Yuan, ICML 2024

The left side of the image shows the situation when σ added is small, in this case, most of the noise is orthogonal to the ***tangent space around*** x_0, so the vector in the direction from x_σ to its ***projection*** onto set ***K*** achieves denoising. The image at the center shows the opposite situation, when the amount of noise is large. The vector toward the ***projection*** of x_σ is still in the direction of denoising.

The right-most image shows the standard denoising process which starts with a large σ and proceeds along the gradient direction in small steps, reducing the noise amount at each step, until convergence somewhere on the manifold. More on this in a moment.

Permenter and Yuan show that when the input data is ***uniformly sampled*** from a finite set ***K***, the ***ideal denoiser*** points toward the ***weighted mean of all points in K***. With weights given by the distance between the initial location and the set. Importantly, the weighted distance depends on the value of σ so that at high noise levels, a larger cloud of points have meaningful contributions toward deciding the

denoising direction. Their conclusion is that the ***ideal denoiser*** is simply ***gradient descent*** over a ***σ-smoothed distance function to the underlying manifold***.

Generating new points on the manifold

Once we have trained the network $\epsilon_\theta(\vec{x}, \sigma)$, we have to consider what the process is for generating a ***new sample*** likely to be part of the ***manifold***. The key insight here is that ***a single call to our diffusion model*** is unlikely to produce good results:

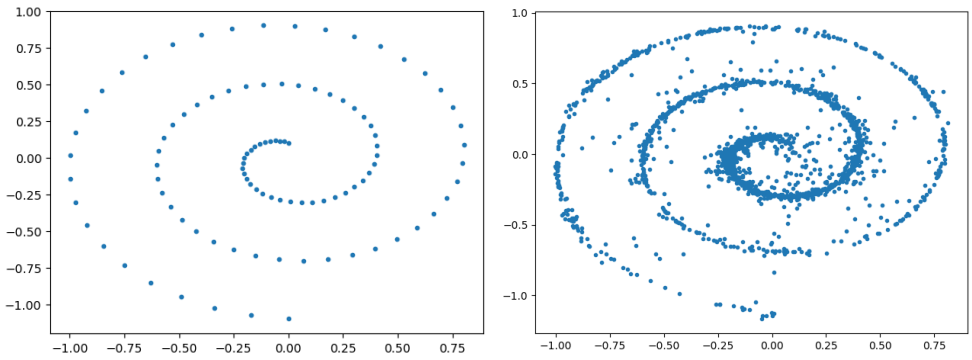
- If we start with a point far away from the manifold (large ***σ***), the diffusion model will produce a move toward the ***mean*** of the points on the manifold. The mean will in all likelihood ***not be on the manifold*** itself so this is not a good solution.

- Conversely, if we start with a point very close to the manifold (small ***σ***), then the starting point will already be fairly similar to one (or a few) of our original training samples, and the network's output will only make it more similar to those. This is also not a good solution since we want to generate novel data points.

The solution is to implement an ***iterative process*** that ***samples directions toward the manifold*** from the ***diffusion model*** as the value of ***σ*** is progressively decreased. The process begins with a large ***σ*** and takes small steps in the direction of the manifold, at each step the ***diffusion model*** produces a denoising direction which is used to find a point closer to the manifold:

$$\vec{x}_{t-1} = \vec{x}_t - (\sigma_t - \sigma_{t-1})\epsilon_\theta(\vec{x}_t, \sigma_t)$$

The particular ***schedule*** used to determine the value of ***σ*** at different steps of the denoising process is important in terms of the implementation, but not needed for understanding the process. Permenter and Yuan show that the above is equivalent to the standard DDIM (Denoising Diffusion Implicit Model) sampling method, currently in used for high-quality image generation.



Input data set (left) and points generated with the sampling process above. Note that the sampling is able to generate novel points in the manifold. Image by Yuan (***https://www.chenyang.co/diffusion.html***)

Sounds simple, what's the trick?

Making diffusion models work for realistic problems is not that simple – we are training a network to approximate an incredibly complex manifold in a very high dimensional space given only noisy

samples of the points within. The details of the network's architecture will be critical for success, the optimization process is not trivial (alternate, common formulations of the diffusion optimization model make use of fairly involved Math in order to set up the learning process and loss function), the amount and quality of the data available for training is essential, and often, some trickery is involved – for instance, it turns out that ***adding a small amount of noise*** to the ***points obtained from the diffusion model*** improves the ability of the model to generate novel samples over the manifold.

All of these are important when implementing and using diffusion models, but not particularly relevant to understanding ***what they are doing*** or ***why they can be expected to work***. The discussion above should have provided a general and solid overview of the diffusion process and how it can be reversed with the aid of a suitably trained deep neural network. But there is a lot more to learn if you want to keep exploring:

- How to combine language models with diffusion models so we can use text prompts to generate content

- How to train models that generate sequences (e.g. video) – consider that we can't just run a diffusion model multiple times and expect the resulting images to form a coherent movie

- How to train and use cross-modal diffusion models (e.g. take as input an image and a text prompt) to generate content

- How to build a model that can generate video ***and*** the corresponding audio

And much more. These are very specialized applications and each would require more time to study than we have in one A.I. course. But now that you understand the basics of deep learning, you can continue your study of whatever particular applications you find most interesting.

But here is a question that I would like to leave in your mind to close the course:

How is it that humans can regularly achieve all of the more advanced tasks our amazing A.I. models are now beginning to perform – without having to learn from gazillions of data points, having access to the computational resources of an underground, nuclear powered data center, or consuming megawatts of power for their daily operation?

This is not an idle question – we have now seen that given enough training data and sufficient computational resources, A.I. models can achieve impressive feats. And yet, we are not surprised when a human is able to come up with a new, beautiful poem, a moving piece of music, an inspiring painting, a mathematical proof no one thought of before, or just came up with the ***transformer*** architecture that powers LLMs. And they just needed some water, air, food, and other humans to talk to (or read from) as they were growing up.

Human intelligence is pretty amazing, and we're probably missing something important about how it works because it doesn't seem to require any of the things without which our modern deep-learning A.I.s can't be trained or operated. Think about that, and ***never underestimate your own intelligence***, no matter how 'smart' A.I. models may appear to become in the future.