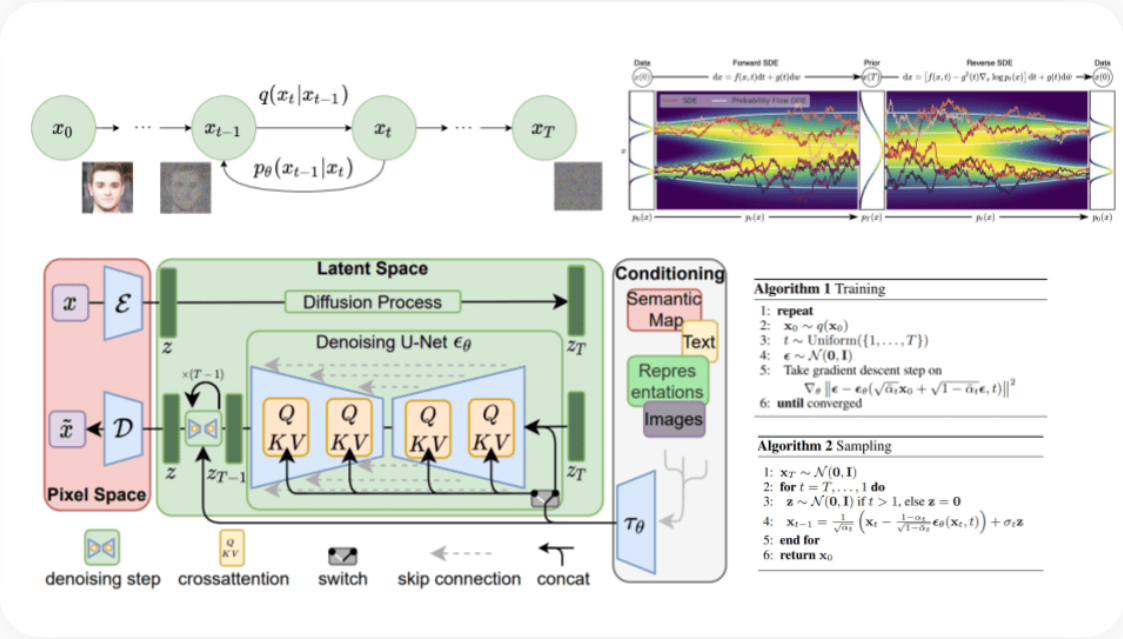
How diffusion models work: the math from scratch

Sergios Karagiannakos , Nikolas Adaloglou on 2022-09-29 · 14 mins

Generative Learning Computer Vision



Diffusion models are a new class of state-of-the-art generative models that generate diverse high-resolution images. They have already attracted a lot of attention after OpenAI, Nvidia and Google managed to train large-

full open-source stable diffusion.

But what is the main principle behind them?

In this blog post, we will dig our way up from the basic principles. There are already a bunch of different diffusion-based architectures. We will focus on the most prominent one, which is the Denoising Diffusion Probabilistic Models (DDPM) as initialized by Sohl-Dickstein et al and then proposed by Ho. et al 2020. Various other approaches will be discussed to a smaller extent such as stable diffusion and score-based models.

> Diffusion models are fundamentally different from all the previous generative methods. Intuitively, they aim to decompose the image generation process (sampling) in many small "denoising" steps.

The intuition behind this is that the model can correct itself over these small steps and gradually produce a good sample. To some extent, this idea of refining the

iterative process makes them slow at sampling, at least compared to GANs.

Diffusion process

The basic idea behind diffusion models is rather simple. They take the input image $\mathbf{x}_0$ and gradually add Gaussian noise to it through a series of T steps. We will call this the forward process. Notably, this is unrelated to the forward pass of a neural network. If you'd like, this part is necessary to generate the targets for our neural network (the image after applying $t < T$ noise steps).

Afterward, a neural network is trained to recover the original data by reversing the noising process. By being able to model the reverse process, we can generate new data. This is the so-called reverse diffusion process or, in general, the sampling process of a generative model.

How? Let’s dive into the math to make it crystal clear.

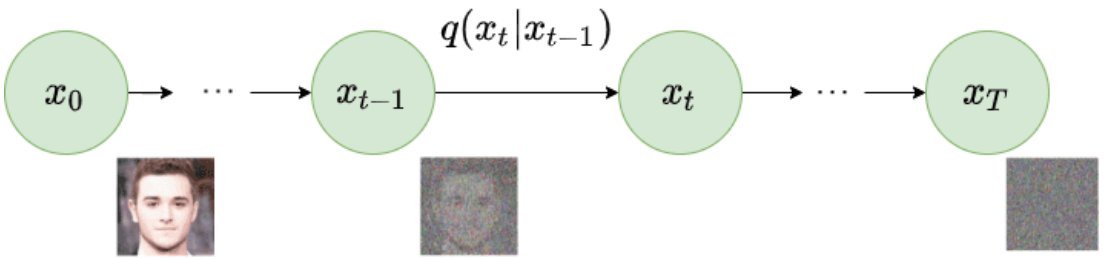
Forward diffusion

continuous feature space. In such a way, they may look similar to variational autoencoders (VAEs).

In practice, they are formulated using a Markov chain of T steps. Here, a Markov chain means that each step only depends on the previous one, which is a mild assumption. Importantly, we are not constrained to using a specific type of neural network, unlike flow-based models.

Given a data-point $\mathbf{x}_0$ sampled from the real data distribution $q(x)$ ($\mathbf{x}_0 \sim q(x)$), one can define a forward diffusion process by adding noise. Specifically, at each step of the Markov chain we add Gaussian noise with variance β_t to $\mathbf{x}_{t-1}$, producing a new latent variable $\mathbf{x}_t$ with distribution $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. This diffusion process can be formulated as follows:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \boldsymbol{\Sigma}_t = \beta_t\mathbf{I})$$



Since we are in the multi-dimensional scenario $\mathbf{I}$ is the identity matrix, indicating that each dimension has the same standard deviation β_t. Note that $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is still a normal distribution, defined by the mean $\boldsymbol{\mu}$ and the variance $\boldsymbol{\Sigma}$ where $\boldsymbol{\mu}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1}$ and $\boldsymbol{\Sigma}_t = \beta_t\mathbf{I}$. $\boldsymbol{\Sigma}$ will always be a diagonal matrix of variances (here β_t)

Thus, we can go in a closed form from the input data $\mathbf{x}_0$ to $\mathbf{x}_T$ in a tractable way. Mathematically, this is the posterior probability and is defined as:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

The symbol : in $q(\mathbf{x}_{1:T})$ states that we apply q repeatedly from timestep 1 to T. It's also called trajectory.

So far, so good? Well, nah! For timestep $t = 500 < T$ we need to apply q 500 times in order to sample $\mathbf{x}_t$. Can't we really do better?

The reparametrization trick provides a magic remedy to this.

If we define $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$ where $\boldsymbol{\epsilon}_0, ..., \epsilon_{t-2}, \epsilon_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, one can use the reparameterization trick in a recursive manner to prove that:

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{1 - \beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}\mathbf{x}_{t-2} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon_0}
\end{aligned}
$$

Note: Since all timestep have the same Gaussian noise we will only use the symbol $\boldsymbol{\epsilon}$ from now on.

Thus to produce a sample $\mathbf{x}_t$ we can use the following distribution:

$$
\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})
$$

Since β_t is a hyperparameter, we can precompute α_t and $\bar{\alpha}_t$ for all timesteps. This means that we sample noise at any timestep t and get $\mathbf{x}_t$ in one go. Hence, we can sample our latent variable $\mathbf{x}_t$ at any arbitrary timestep.

Variance schedule

The variance parameter β_t can be fixed to a constant or chosen as a schedule over the T timesteps. In fact, one can define a variance schedule, which can be linear, quadratic, cosine etc. The original DDPM authors utilized a linear schedule increasing from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$. Nichol et al. 2021 showed that employing a cosine schedule works even better.

Latent samples from linear (top) and cosine (bottom) schedules respectively.
Source: Nichol & Dhariwal 2021

Reverse diffusion

As $T \to \infty$, the latent x_T is nearly an isotropic Gaussian distribution. Therefore if we manage to learn the reverse distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we can sample x_T from $\mathcal{N}(0, \mathbf{I})$, run the reverse process and acquire a sample from $q(x_0)$,

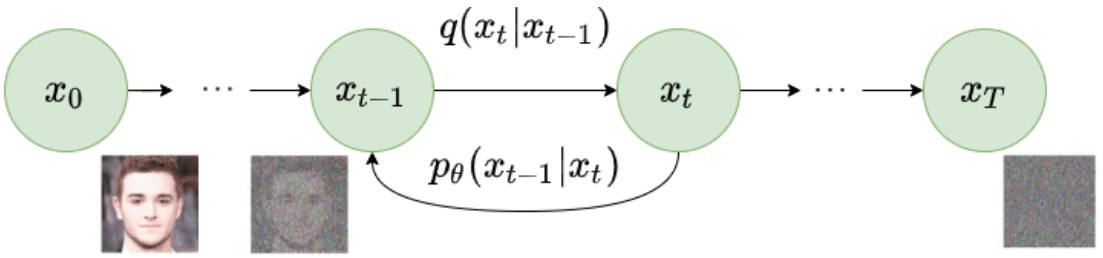
The question is how we can model the reverse diffusion process.

Approximating the reverse process with a neural network

In practical terms, we don't know $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. It's intractable since statistical estimates of $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ require computations involving the data distribution.

Instead, we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ with a parameterized model p_θ (e.g. a neural network). Since $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ will also be Gaussian, for small enough β_t, we can choose p_θ to be Gaussian and just parameterize the mean and variance:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$



Reverse diffusion process. Image modified by Ho et al. 2020

distribution:

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

By additionally conditioning the model on timestep t, it will learn to predict the Gaussian parameters (meaning the mean $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ and the covariance matrix $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$) for each timestep.

But how do we train such a model?

Training a diffusion model

If we take a step back, we can notice that the combination of q and p is very similar to a variational autoencoder (VAE). Thus, we can train it by optimizing the negative log-likelihood of the training data. After a series of calculations, which we won't analyze here, we can write the evidence lower bound (ELBO) as follows:

$$\sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))]$$

$$= L_0 - L_T - \sum_{t=2}^{T} L_{t-1}$$

Let's analyze these terms:

1. The $\mathbb{E}_{q(x_1|x_0)}[log p_\theta(\mathbf{x}_0|\mathbf{x}_1)]$ term can been as a reconstruction term, similar to the one in the ELBO of a variational autoencoder. In Ho et al 2020 , this term is learned using a separate decoder.

2. $D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))$ shows how close $\mathbf{x}_T$ is to the standard Gaussian. Note that the entire term has no trainable parameters so it's ignored during training.

3. The third term $\sum_{t=2}^{T} L_{t-1}$, also referred as L_t, formulate the difference between the desired denoising steps $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$ and the approximated ones $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$.

It is evident that through the ELBO, maximizing the likelihood boils down to learning the denoising steps L_t.

intractable Sohl-Dickstein et al illustrated that by additionally conditioning on $\mathbf{x}_0$ makes it tractable.

Intuitively, a painter (our generative model) needs a reference image ($\mathbf{x}_0$) to slowly draw (reverse diffusion step $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$) an image. Thus, we can take a small step backwards, meaning from noise to generate an image, if and only if we have $\mathbf{x}_0$ as a reference.

In other words, we can sample $\mathbf{x}_t$ at noise level t conditioned on $\mathbf{x}_0$. Since $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$, we can prove that:

$$
\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}) \\
\tilde{\beta}_t &= \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t \\
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x_0} + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t
\end{aligned}
$$

Note that α_t and $\bar{\alpha}_t$ depend only on β_t, so they can be precomputed.

as we already saw in the reparameterization trick, we can represent $\mathbf{x}_0$ as

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon})),$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

By combining the last two equations, each timestep will now have a mean $\tilde{\boldsymbol{\mu}}_t$ (our target) that only depends on $\mathbf{x}_t$:

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon})\Big)$$

Therefore we can use a neural network $\epsilon_\theta(\mathbf{x}_t, t)$ to approximate $\boldsymbol{\epsilon}$ and consequently the mean:

$$\tilde{\boldsymbol{\mu}}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big)$$

Thus, the loss function (the denoising term in the ELBO) can be expressed as:

$$\begin{aligned} L_t &= \mathbb{E}_{\mathbf{x}_0,t,\boldsymbol{\epsilon}}\Big[\frac{1}{2||\boldsymbol{\Sigma}_\theta(x_t,t)||_2^2}||\tilde{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_\theta(\mathbf{x}_t,t)||_2^2\Big] \\ &= \mathbb{E}_{\mathbf{x}_0,t,\boldsymbol{\epsilon}}\Big[\frac{\beta_t^2}{2\alpha_t(1-\bar{\alpha}_t)||\boldsymbol{\Sigma}_\theta||_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{a}_t}\mathbf{x}_0 + \sqrt{1-\bar{a}_t}\boldsymbol{\epsilon}, t)||^2\Big] \end{aligned}$$

noise ϵ at each timestep t.

Ho et.al 2020 made a few simplifications to the actual loss term as they ignore a weighting term. The simplified version outperforms the full objective:

$$L_t^{\text{simple}} = \mathbb{E}_{\mathbf{x}_0,t,\boldsymbol{\epsilon}}\left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{a}_t}\mathbf{x}_0 + \sqrt{1-\bar{a}_t}\boldsymbol{\epsilon}, t)||^2\right]$$

The authors found that optimizing the above objective works better than optimizing the original ELBO. The proof for both equations can be found in this excellent post by Lillian Weng or in Luo et al. 2022.

Additionally, Ho et. al 2020 decide to keep the variance fixed and have the network learn only the mean. This was later improved by Nichol et al. 2021, who decide to let the network learn the covariance matrix ($\boldsymbol{\Sigma}$) as well (by modifying L_t^{simple}), achieving better results.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

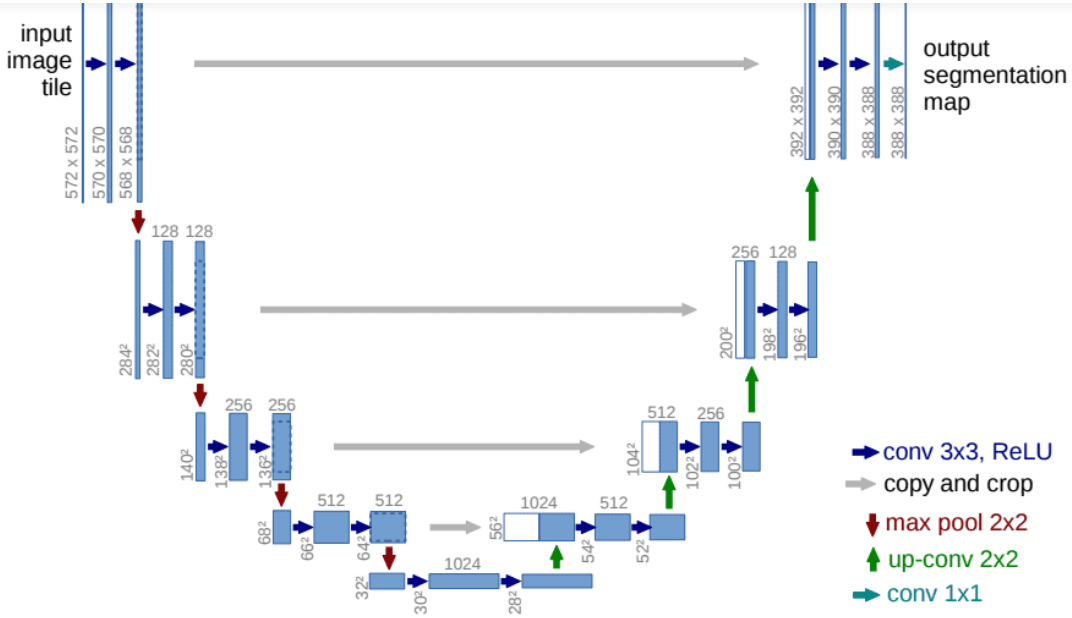
Training and sampling algorithms of DDPMs. Source: Ho et al. 2020

One thing that we haven't mentioned so far is what the model's architecture looks like. Notice that the model's input and output should be of the same size.

To this end, Ho et al. employed a U-Net. If you are unfamiliar with U-Nets, feel free to check out our past article on the major U-Net architectures. In a few words, a U-Net is a symmetric architecture with input and output of the same spatial size that uses skip connections between encoder and decoder blocks of corresponding feature dimension. Usually, the input image is first downsampled and then upsampled until reaching its initial size.

In the original implementation of DDPMs, the U-Net consists of Wide ResNet blocks, group normalization as well as self-attention blocks.

The diffusion timestep t is specified by adding a sinusoidal position embedding into each residual block. For more details, feel free to visit the official GitHub repository. For a detailed implementation of the diffusion model, check out this awesome post by Hugging Face.



The U-Net architecture. Source: Ronneberger et al.

Conditional Image Generation: Guided Diffusion

A crucial aspect of image generation is conditioning the sampling process to manipulate the generated samples. Here, this is also referred to as guided diffusion.

There have even been methods that incorporate image embeddings into the diffusion in order to "guide" the generation. Mathematically, guidance refers to conditioning a prior data distribution $p(\mathbf{x})$ with a

To turn a diffusion model p_θ into a conditional diffusion model, we can add conditioning information y at each diffusion step.

$$p_\theta(\mathbf{x}_{0:T}|y) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, y)$$

The fact that the conditioning is being seen at each timestep may be a good justification for the excellent samples from a text prompt.

In general, guided diffusion models aim to learn $\nabla \log p_\theta(\mathbf{x}_t|y)$. So using the Bayes rule, we can write:

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) &= \nabla_{\mathbf{x}_t} \log(\frac{p_\theta(y|\mathbf{x}_t)p_\theta(\mathbf{x}_t)}{p_\theta(y)}) \\ &= \nabla_{\mathbf{x}_t} log p_\theta(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} log(p_\theta(y|\mathbf{x}_t))\end{aligned}$$

$p_\theta(y)$ is removed since the gradient operator $\nabla_{\mathbf{x}_t}$ refers only to $\mathbf{x}_t$, so no gradient for y. Moreover remember that $\log(ab) = \log(a) + \log(b)$.

And by adding a guidance scalar term s, we have:

$$\nabla \log p_\theta(\mathbf{x}_t|y) = \nabla \log p_\theta(\mathbf{x}_t) + s \cdot \nabla \log(p_\theta(y|\mathbf{x}_t))$$

present two family of methods aiming at injecting label information.

Classifier guidance

Sohl-Dickstein et al. and later Dhariwal and Nichol showed that we can use a second model, a classifier $f_\phi(y|\mathbf{x}_t, t)$, to guide the diffusion toward the target class y during training. To achieve that, we can train a classifier $f_\phi(y|\mathbf{x}_t, t)$ on the noisy image $\mathbf{x}_t$ to predict its class y. Then we can use the gradients $\nabla \log(f_\phi(y|\mathbf{x}_t))$ to guide the diffusion. How?

We can build a class-conditional diffusion model with mean $\mu_\theta(\mathbf{x}_t|y)$ and variance $\mathbf{\Sigma}_\theta(\mathbf{x}_t|y)$.

Since $p_\theta \sim \mathcal{N}(\mu_\theta, \Sigma_\theta)$, we can show using the guidance formulation from the previous section that the mean is perturbed by the gradients of $\log f_\phi(y|\mathbf{x}_t)$ of class y, resulting in:

$$\hat{\mu}(\mathbf{x}_t|y) = \mu_\theta(\mathbf{x}_t|y) + s \cdot \mathbf{\Sigma}_\theta(\mathbf{x}_t|y)\nabla_{\mathbf{x}_t} log f_\phi(y|\mathbf{x}_t, t)$$

In the famous GLIDE paper by Nichol et al, the authors expanded on this idea and use CLIP embeddings to

SIMILAR ARTICLES

Generative Learning

The theory behind Latent Variable Models: formulating a Variational Autoencoder

Deepfakes: Face synthesis with GANs and Autoencoders

GANs in computer vision - semantic image synthesis and learning a

produces an image and text embeddings $g(\mathbf{x}_t)$ and $h(c)$, respectively, wherein c is the text caption.

Therefore, we can perturb the gradients with their dot product:

$$\hat{\mu}(\mathbf{x}_t|c) = \mu(\mathbf{x}_t|c) + s \cdot \mathbf{\Sigma}_\theta(\mathbf{x}_t|c)\nabla_{\mathbf{x}_t} g(\mathbf{x}_t) \cdot h(c)$$

As a result, they manage to "steer" the generation process toward a user-defined text caption.

Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma\nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

Algorithm of classifier guided diffusion sampling. Source: Dhariwal & Nichol 2021

Classifier-free guidance

Using the same formulation as before we can define a classifier-free guided diffusion model as:

$$\nabla \log p(\mathbf{x}_t|y) = s \cdot \nabla log(p(\mathbf{x}_t|y)) + (1-s) \cdot \nabla log p(\mathbf{x}_t)$$

a separate classifier, the authors trained a conditional diffusion model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|y)$ together with an unconditional model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|0)$. In fact, they use the exact same neural network. During training, they randomly set the class y to 0, so that the model is exposed to both the conditional and unconditional setup:

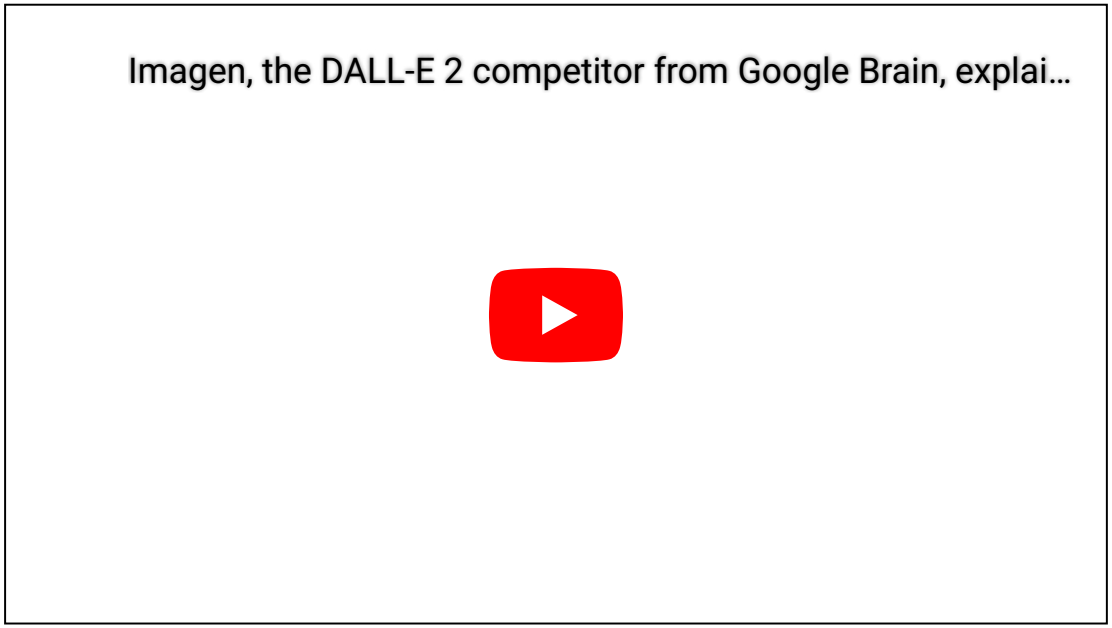
$$\begin{aligned}\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t|y) &= s\cdot\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|y) + (1-s)\cdot\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|0)\\ &= \boldsymbol{\epsilon}_\theta(\mathbf{x}_t|0) + s\cdot(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t|0))\end{aligned}$$

> Note that this can also be used to "inject" text embeddings as we showed in classifier guidance.

This admittedly "weird" process has two major advantages:

- It uses only a single model to guide the diffusion.
- It simplifies guidance when conditioning on information that is difficult to predict with a classifier (such as text embeddings).

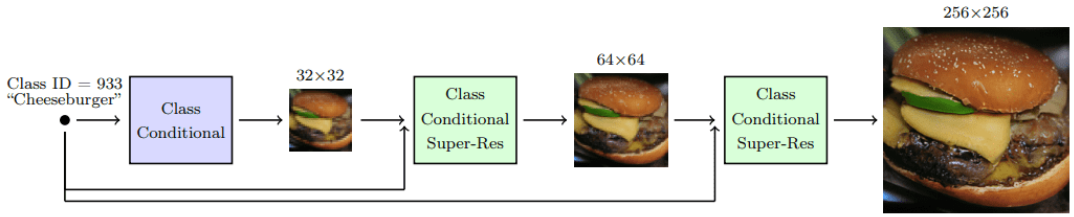
contributor to generating samples with strong image-text alignment. For more info on the approach of Imagen check out this video from AI Coffee Break with Letitia:



Scaling up diffusion models

You might be asking what is the problem with these models. Well, it's computationally very expensive to scale these U-nets into high-resolution images. This brings us to two methods for scaling up diffusion models to higher resolutions: cascade diffusion models and latent diffusion models.

Ho et al. 2021 introduced cascade diffusion models in an effort to produce high-fidelity images. A cascade diffusion model consists of a pipeline of many sequential diffusion models that generate images of increasing resolution. Each model generates a sample with superior quality than the previous one by successively upsampling the image and adding higher resolution details. To generate an image, we sample sequentially from each diffusion model.



Cascade diffusion model pipeline. Source: Ho & Saharia et al.

To acquire good results with cascaded architectures, strong data augmentations on the input of each super-resolution model are crucial. Why? Because it alleviates compounding error from the previous cascaded models, as well as due to a train-test mismatch.

It was found that gaussian blurring is a critical transformation toward achieving high fidelity. They refer to this technique as conditioning augmentation.

Latent diffusion models are based on a rather simple idea: instead of applying the diffusion process directly on a high-dimensional input, we project the input into a smaller latent space and apply the diffusion there.

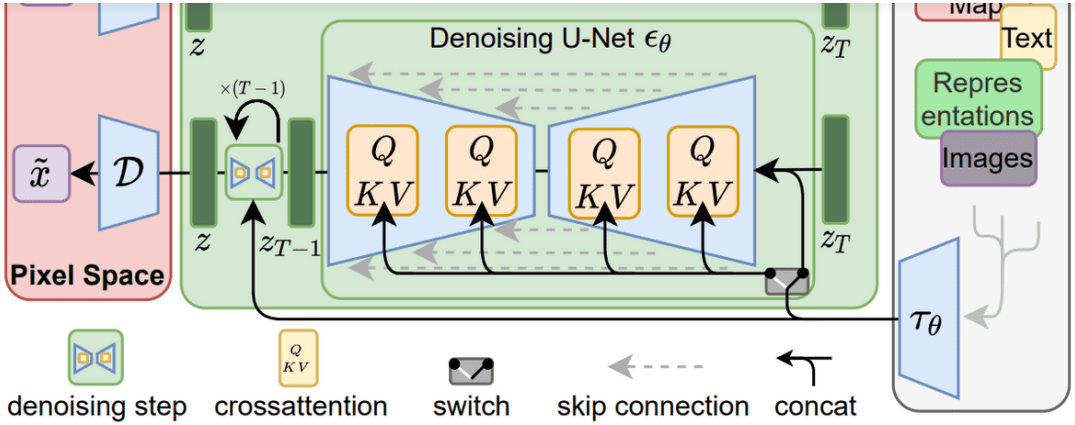
In more detail, Rombach et al. proposed to use an encoder network to encode the input into a latent representation i.e. $\mathbf{z}_t = g(\mathbf{x}_t)$. The intuition behind this decision is to lower the computational demands of training diffusion models by processing the input in a lower dimensional space. Afterward, a standard diffusion model (U-Net) is applied to generate new data, which are upsampled by a decoder network.

If the loss for a typical diffusion model (DM) is formulated as:

$$L_{DM} = \mathbb{E}_{\mathbf{x},t,\boldsymbol{\epsilon}}\Big[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)||^2\Big]$$

then given an encoder $\mathcal{E}$ and a latent representation z, the loss for a latent diffusion model (LDM) is:

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(\mathbf{x}),t,\boldsymbol{\epsilon}}\Big[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t)||^2\Big]$$



Latent diffusion models. Source: Rombach et al

For more information check out this video:

Score-based generative models

that appears to have many similarities with diffusion models. Score-based models tackle generative learning using score matching and Langevin dynamics.

> Score-matching refers to the process of modeling the gradient of the log probability density function, also known as the score function. Langevin dynamics is an iterative process that can draw samples from a distribution using only its score function.

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \frac{\delta}{2}\nabla_{\mathbf{x}} \log p\left(\mathbf{x}_{t-1}\right) + \sqrt{\delta}\boldsymbol{\epsilon}, \quad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where δ is the step size.

Suppose that we have a probability density $p(x)$ and that we define the score function to be $\nabla_x \log p(x)$. We can then train a neural network s_θ to estimate $\nabla_x \log p(x)$ without estimating $p(x)$ first. The training objective can be formulated as follows:

$$\mathbb{E}_{p(\mathbf{x})}[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2] = \int p(\mathbf{x})\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2 \mathrm{d}\mathbf{x}$$

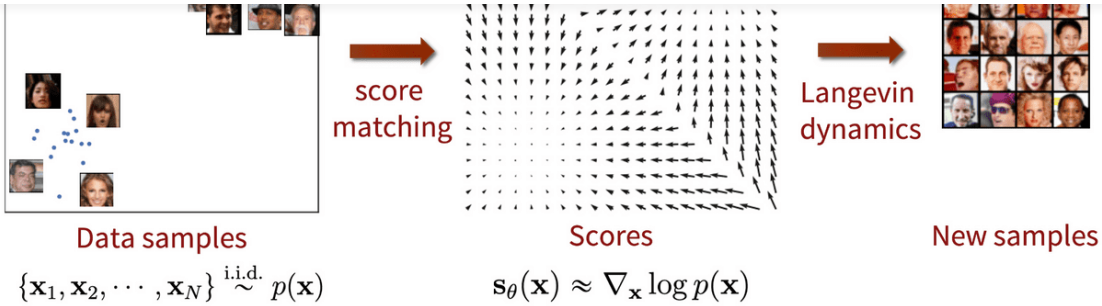
> In case you missed it, guided diffusion models use this formulation of score-based models as they learn directly $\nabla_x \log p(x)$. Of course, they don't rely on Langevin dynamics.

Adding noise to score-based models: Noise Conditional Score Networks (NCSN)

> The problem so far: the estimated score functions are usually inaccurate in low-density regions, where few data points are available. As a result, the quality of data sampled using Langevin dynamics is **not** good.

Their solution was to perturb the data points with noise and train score-based models on the noisy data points instead. As a matter of fact, they used multiple scales of Gaussian noise perturbations.

Thus, adding noise is the key to make both DDPM and score based models work.



Score-based generative modeling with score matching + Langevin dynamics.
Source: Generative Modeling by Estimating Gradients of the Data Distribution

Mathematically, given the data distribution $p(x)$, we perturb with Gaussian noise $\mathcal{N}(\mathbf{0}, \sigma_i^2 I)$ where $i = 1, 2, \cdots, L$ to obtain a noise-perturbed distribution:

$$p_{\sigma_i}(\mathbf{x}) = \int p(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 I) \mathrm{d}\mathbf{y}$$

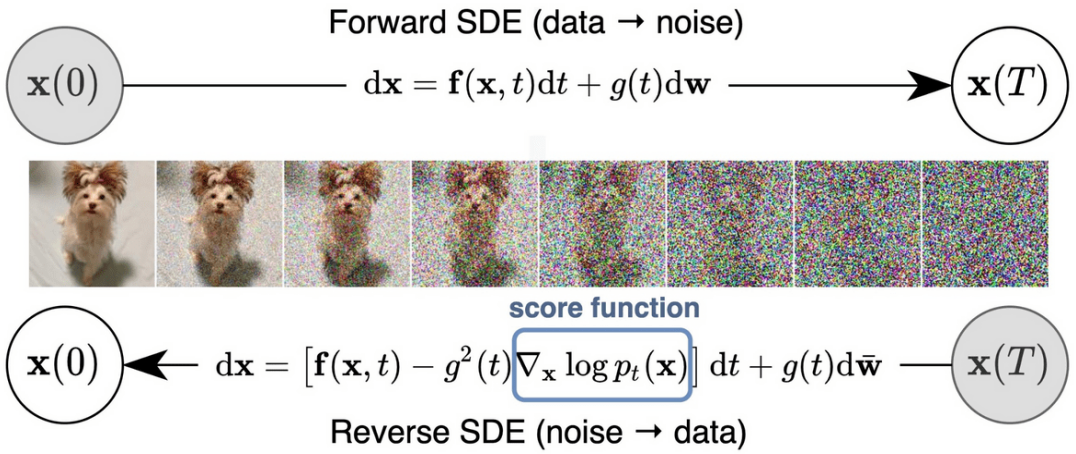
Then we train a network $s_\theta(\mathbf{x}, i)$, known as Noise Conditional Score-Based Network (NCSN) to estimate the score function $\nabla_\mathbf{x} \log d_{\sigma_i}(\mathbf{x})$. The training objective is a weighted sum of Fisher divergences for all noise scales.

$$\sum_{i=1}^{L} \lambda(i) \mathbb{E}_{p_{\sigma_i}(\mathbf{x})}[\|\nabla_\mathbf{x} \log p_{\sigma_i}(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, i)\|_2^2]$$

Score-based generative modeling through stochastic differential equations (SDE)

encapsulate both NSCNs and DDPMs under the same umbrella, they proposed the following:

Instead of perturbing data with a finite number of noise distributions, we use a continuum of distributions that evolve over time according to a diffusion process. This process is modeled by a prescribed stochastic differential equation (SDE) that does not depend on the data and has no trainable parameters. By reversing the process, we can generate new samples.



Score-based generative modeling through stochastic differential equations (SDE). Source: Song et al. 2021

We can define the diffusion process $\{\mathbf{x}(t)\}_{t\in[0,T]}$ as an SDE in the following form:

where $\mathbf{w}$ is the Wiener process (a.k.a., Brownian motion), $\mathbf{f}(\cdot, t)$ is a vector-valued function called the drift coefficient of $\mathbf{x}(t)$, and $g(\cdot)$ is a scalar function known as the diffusion coefficient of $\mathbf{x}(t)$. Note that the SDE typically has a unique strong solution.

> To make sense of why we use an SDE, here is a tip: the SDE is inspired by the Brownian motion, in which a number of particles move randomly inside a medium. This randomness of the particles' motion models the continuous noise perturbations on the data.

After perturbing the original data distribution for a sufficiently long time, the perturbed distribution becomes close to a tractable noise distribution.

To generate new samples, we need to reverse the diffusion process. The SDE was chosen to have a corresponding reverse SDE in closed form:

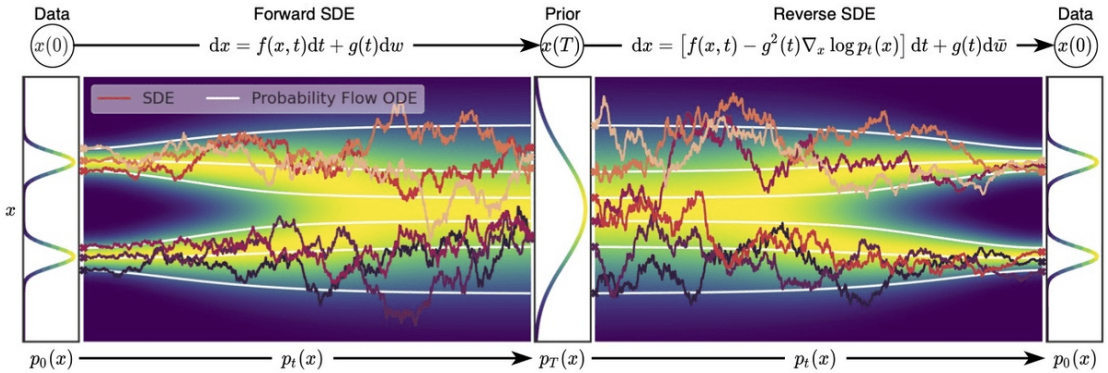
$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

based model $s_\theta(\mathbf{x}, i)$ and Langevin dynamics. The training objective is a continuous combination of Fisher divergences:

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_t(\mathbf{x})}[\lambda(t)\|\nabla_{\mathbf{x}}\log p_t(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t)\|_2^2]$$

where $\mathcal{U}(0, T)$ denotes a uniform distribution over the time interval, and λ is a positive weighting function. Once we have the score function, we can plug it into the reverse SDE and solve it in order to sample $\mathbf{x}(0)$ from the original data distribution $p_0(\mathbf{x})$.

> There are a number of options to solve the reverse SDE which we won't analyze here. Make sure to check the original paper or this excellent blog post by the author.



Summary

Let's do a quick sum-up of the main points we learned in this blogpost:

- Diffusion models work by gradually adding gaussian noise through a series of T steps into the original image, a process known as diffusion.
- To sample new data, we approximate the reverse diffusion process using a neural network.
- The training of the model is based on maximizing the evidence lower bound (ELBO).
- We can condition the diffusion models on image labels or text embeddings in order to "guide" the diffusion process.
- Cascade and Latent diffusion are two approaches to scale up models to high-resolutions.
- Cascade diffusion models are sequential diffusion models that generate images of increasing resolution.

computational efficiency using a variational autoencoder for the up and downsampling.

- Score-based models also apply a sequence of noise perturbations to the original image. But they are trained using score-matching and Langevin dynamics. Nonetheless, they end up in a similar objective.

- The diffusion process can be formulated as an SDE. Solving the reverse SDE allows us to generate new samples.

Finally, for more associations between diffusion models and VAE or AE check out these really nice blogs.

Cite as

```
@article{karagiannakos2022diffusionmodels,
    title   = "Diffusion models: toward state-of-the-ar
    author  = "Karagiannakos, Sergios, Adaloglou, Nikol
    journal = "https://theaisummer.com/",
    year    = "2022",
    howpublished = {https://theaisummer.com/diffusion-m
```

References

[1] Sohl-Dickstein, Jascha, et al. Deep Unsupervised Learning Using Nonequilibrium Thermodynamics. arXiv:1503.03585, arXiv, 18 Nov. 2015

[2] Ho, Jonathan, et al. Denoising Diffusion Probabilistic Models. arXiv:2006.11239, arXiv, 16 Dec. 2020

[3] Nichol, Alex, and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models. arXiv:2102.09672, arXiv, 18 Feb. 2021

[4] Dhariwal, Prafulla, and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. arXiv:2105.05233, arXiv, 1 June 2021

[5] Nichol, Alex, et al. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models. arXiv:2112.10741, arXiv, 8 Mar. 2022

[6] Ho, Jonathan, and Tim Salimans. Classifier-Free Diffusion Guidance. 2021. openreview.net

arXiv, 12 Apr. 2022

[8] Saharia, Chitwan, et al. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. arXiv:2205.11487, arXiv, 23 May 2022

[9] Rombach, Robin, et al. High-Resolution Image Synthesis with Latent Diffusion Models. arXiv:2112.10752, arXiv, 13 Apr. 2022

[10] Ho, Jonathan, et al. Cascaded Diffusion Models for High Fidelity Image Generation. arXiv:2106.15282, arXiv, 17 Dec. 2021

[11] Weng, Lilian. What Are Diffusion Models? 11 July 2021

[12] O'Connor, Ryan. Introduction to Diffusion Models for Machine Learning AssemblyAI Blog, 12 May 2022

[13] Rogge, Niels and Rasul, Kashif. The Annotated Diffusion Model . Hugging Face Blog, 7 June 2022

[14] Das, Ayan. “An Introduction to Diffusion Probabilistic Models.” Ayan Das, 4 Dec. 2021

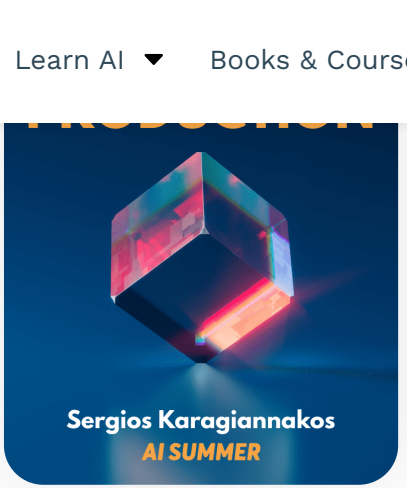
arXiv:1907.05600, arXiv, 10 Oct. 2020

[16] Song, Yang, and Stefano Ermon. Improved Techniques for Training Score-Based Generative Models. arXiv:2006.09011, arXiv, 23 Oct. 2020

[17] Song, Yang, et al. Score-Based Generative Modeling through Stochastic Differential Equations. arXiv:2011.13456, arXiv, 10 Feb. 2021

[18] Song, Yang. Generative Modeling by Estimating Gradients of the Data Distribution, 5 May 2021

[19] Luo, Calvin. Understanding Diffusion Models: A Unified Perspective. 25 Aug. 2022



infrastructure and MLOps using hands-on examples.

Learn more

** Disclosure: Please note that some of the links above might be affiliate links, and at no additional cost to you, we will earn a commission if you decide to make a purchase after clicking through.*