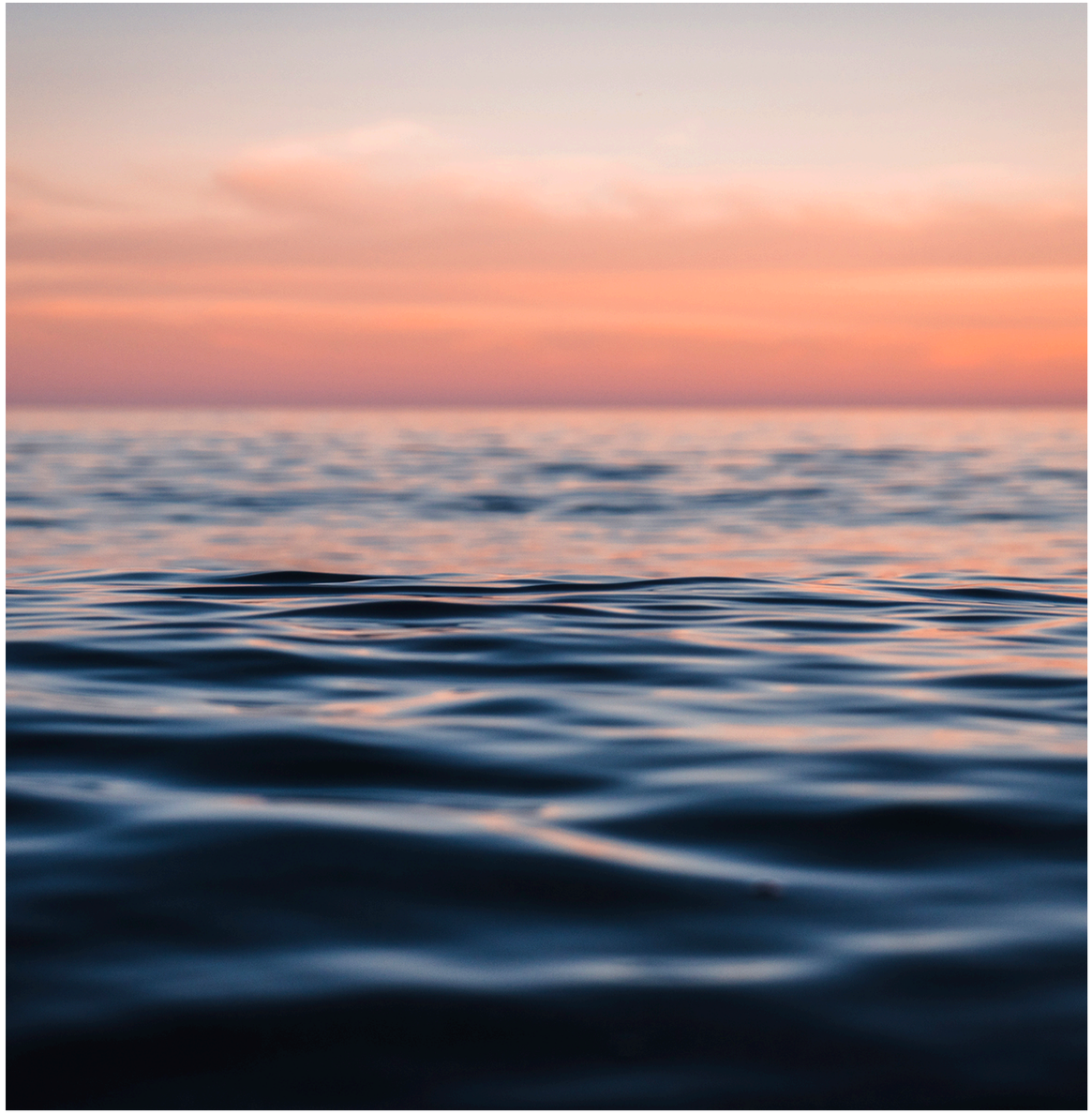
Score-matching & Diffusion Generative Models

Efstratios Gavves

Overview

Introduction to score-matching
Noise conditional score networks
Score-based generation via SDEs
Conditional generation
Diffusion models



https://yang-song.github.io/blog/2021/score/

Tractability v. Flexibility

- In generative modelling there are two opposing forces: tractability and flexibility
- Tractable models are usually analytically computable, thus easy to evaluate and fit
- But they are usually not flexible enough to learn the true data structure
- Flexible models can fit arbitrary structures in data
- But they are usually expensive to evaluate, fit, or sample from
- Diffusion/score-matching models are both tractable and flexible

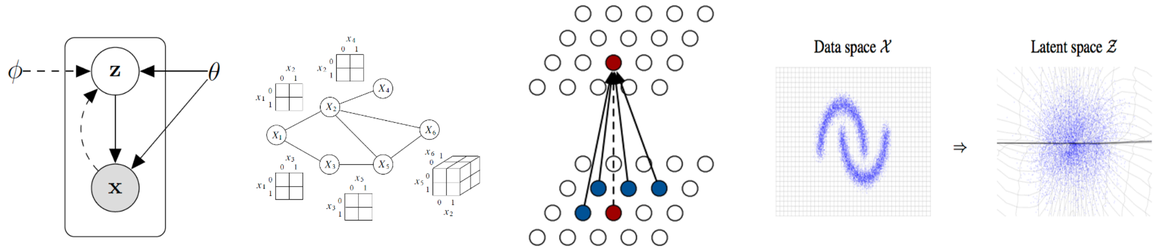
Overview of generative models

Likelihood-based generative models

Implicit generative models

Likelihood-based generative models

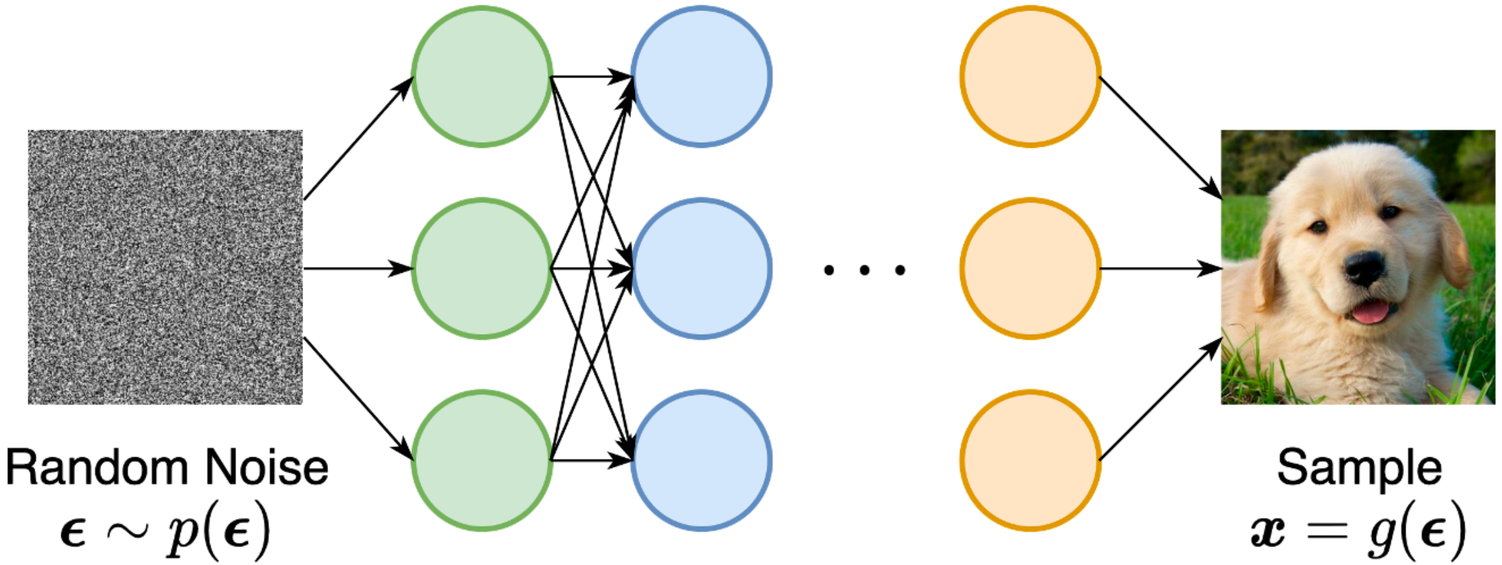
- Typically make strong assumptions to ensure tractability of likelihood
  - specifically of the normalising constant $Z(x)$ in $p(x) = \dfrac{\tilde{p}(x)}{Z(x)}$
- For instance, VAEs assume a tractable variational approximation
- Autoregressive models require causal convolutions
- Normalizing Flows require invertibility in the network architecture



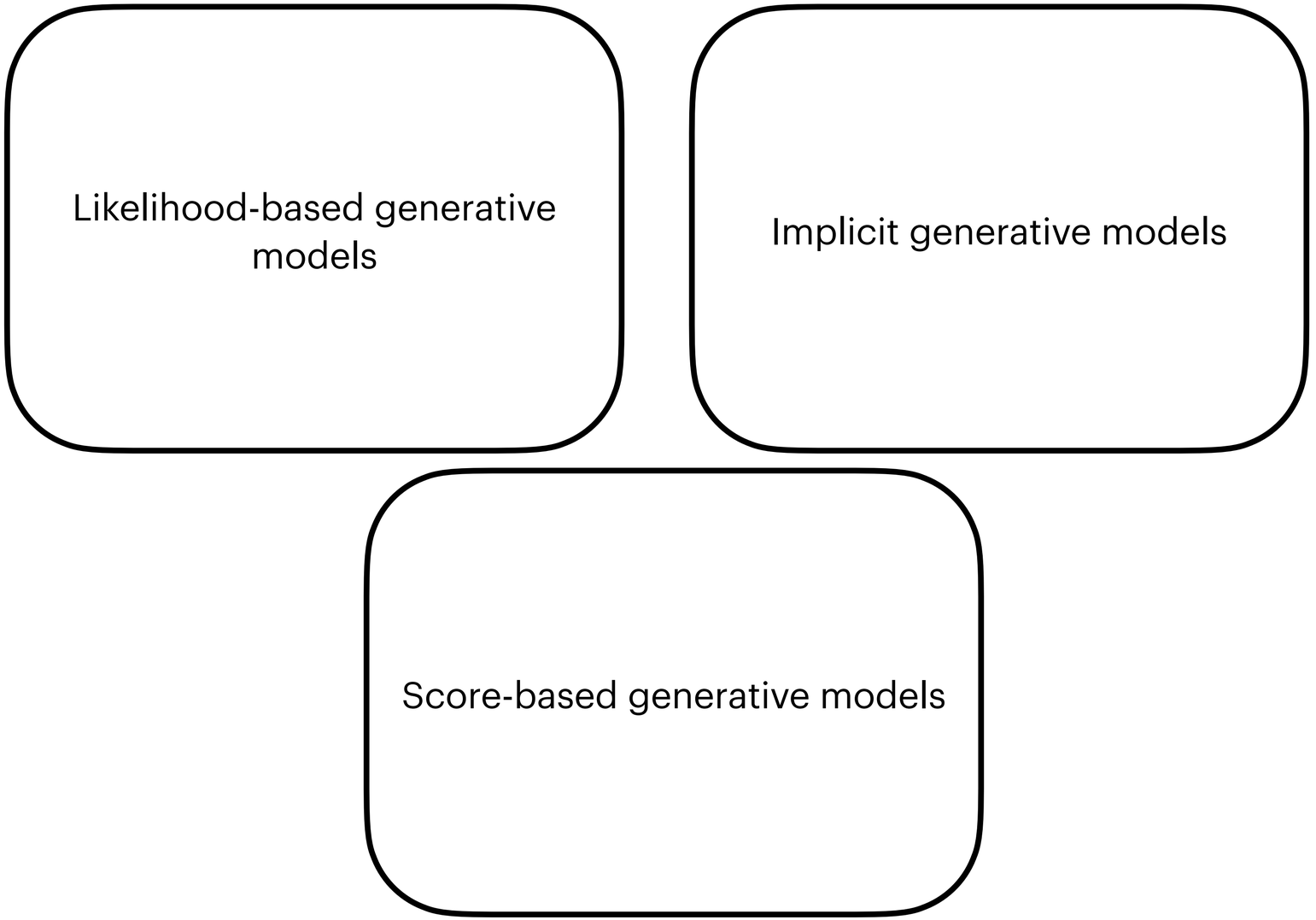
Bayesian networks (e.g., VAEs) | MRF | Autoregressive models | Flow models

Implicit generative models

- Adversarial training for implicit generative models is very unstable
- Adversarial training leads often to mode collapse and reduced sampling variance
- Implicit generative models cannot compute likelihood of a sample, they just sample



Overview of generative models



Tractability v. Flexibility

- In generative modelling there are two opposing forces: tractability and flexibility
- Tractable models are usually analytically computable, thus easy to evaluate and fit
- But they are usually not flexible enough to learn the true data structure
- Flexible models can fit arbitrary structures in data
- But they are usually expensive to evaluate, fit, or sample from
- Diffusion/score-matching models are both tractable and flexible

Energy-based models: a recap

- Alternative to likelihood-based models is energy-based models $f_\theta(x)$ with likelihood

$$p_\theta(x) = \frac{\exp(-f_\theta(x))}{Z_\theta}, Z_\theta = \int \exp(-f_\theta(x))dx$$

- For general functions (and network architectures) f_θ, it is intractable to maximising likelihood due to the normalising constant

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i)$$

Score-based models: impressive results

- GAN-like quality and better, while having the advantages of explicit probabilistic models
  - Explicit likelihood computation
  - Representation learning
- State-of-the-art results in generation, audio synthesis, shape generation, etc

Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021 (outstanding paper award)

Score-based generative models

- Score-based models we do not need a tractable normalising constant
- Instead, we can rely on *score matching*
-

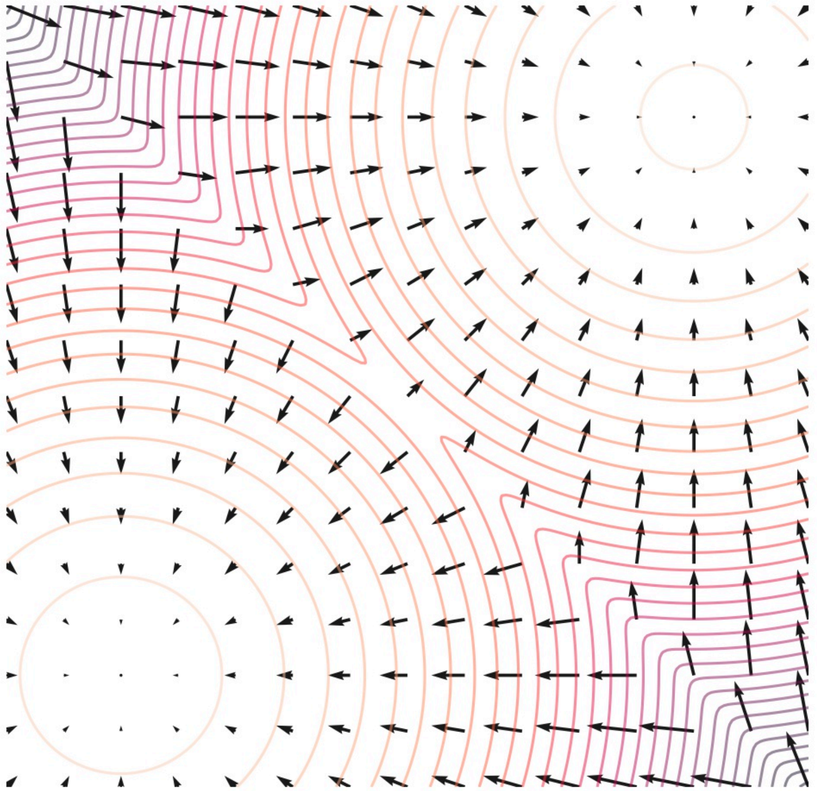
Score function

- The (Stein) score function is the gradient of the log-probability of a distribution w.r.t. to the input

$$\nabla_x \log p(x)$$

- A model $s_\theta(x)$, which models the score function explicitly, is a score-based model

$$s_\theta(x) \approx \nabla_x \log p(x)$$



The score function of a mixture of two Gaussians

Score-based generative models

- Score-based models we do not need a tractable normalising constant

$$s_\theta(x) = \nabla_x \log p(x)$$
$$= -\nabla_x f_\theta(x) - \underbrace{\nabla_x \log Z_\theta}_{=0} = -\nabla_x f_\theta(x)$$

- But, a score-based model is literally set to output a vector that represents gradient
- We could minimise the Fisher divergence

$$\mathbb{E}_{p(x)} \| \nabla_x \log p(x) - s_\theta(x) \|_2^2$$

- But we do not know the "optimal gradient"/"ground truth data score"
- How do we train and backdrop? What do we optimize?

Score matching

- It can be shown* that optimising $\mathbb{E}_{p(x)} \| \nabla_x \log p(x) - s_\theta(x) \|_2^2$ is equivalent to

$$\mathbb{E}_{p_{data}(\mathbf{x})} \left[\mathrm{tr}\left(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}) \right) + \frac{1}{2} \| s_\theta(\mathbf{x}) \|_2^2 \right]$$

  up to some regularity conditions

- Still, the trace of the Jacobian is too expensive for large networks and approximations are needed

* Song et al., Sliced score matching: A scalable approach to density and score estimation, UAI 2019

Denoising score matching

- Denoising score matching works well for small level of noise

$$\frac{1}{2}\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{data}(\mathbf{x})}\left[\left\|s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}}\,|\,\mathbf{x})\right\|_2^2\right]$$

where the data $\mathbf{x}$ is corrupted to $\tilde{\mathbf{x}}$ as $q_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}}\,|\,\mathbf{x})p_{data}(\mathbf{x})\mathrm{d}\mathbf{x}$

- First sample a training example from the training set
- Then add noise to it from a pre-specified distribution
- You can repeat the process and average with Monte Carlo simulation (or do it once)

* Vincent, A connection between score matching and denoising auto encoders, Neural Computation, 2011

Sliced score matching

- Slided score matching, which uses random projections to approximate the trace

$$\mathbb{E}_{p(\mathbf{v})}\mathbb{E}_{p_{data}}\left[\mathbf{v}^\top \nabla_{\mathbf{x}} s_\theta(\mathbf{x})\mathbf{v} + \frac{1}{2}\|s_\theta\|_2^2\right]$$

where $p(\mathbf{v})$ is a simple distribution of random vectors like multivariate Gaussian

- First sample a few vectors $\mathbf{v}$ that define the random projections
- Then compute $\mathbf{v}^\top \nabla_{\mathbf{x}} s_\theta(\mathbf{x})\mathbf{v}$ using forward-mode auto-differentiation
- Works on the original, unperturbed data distribution
- But it requires 4x the compute due to the extra auto-differentiation

* Song et al., Sliced score matching: A scalable approach to density and score estimation, UAI 2019

Score matching: advantages

- We can train with score matching directly with SGD like maximising log-likelihood
- We have no constraints on the form of $f_\theta(x)$ as we do not require $s_\theta(x)$ to be the score function of a normalised distribution
- We just compare our neural network output with the ground-truth data score
- The only requirement is that $s_\theta(x)$ is a vector valued function with the same input and output dimensionality

Sampling using Langevin dynamics

- During training we do not involve an explicit “sampling” mechanism
- After training the score-based model, we can sample with Langevin dynamics
- Langevin dynamics are an MCMC procedure to sample from distribution $p(x)$ using only the score function $\nabla_x \log p(x)$

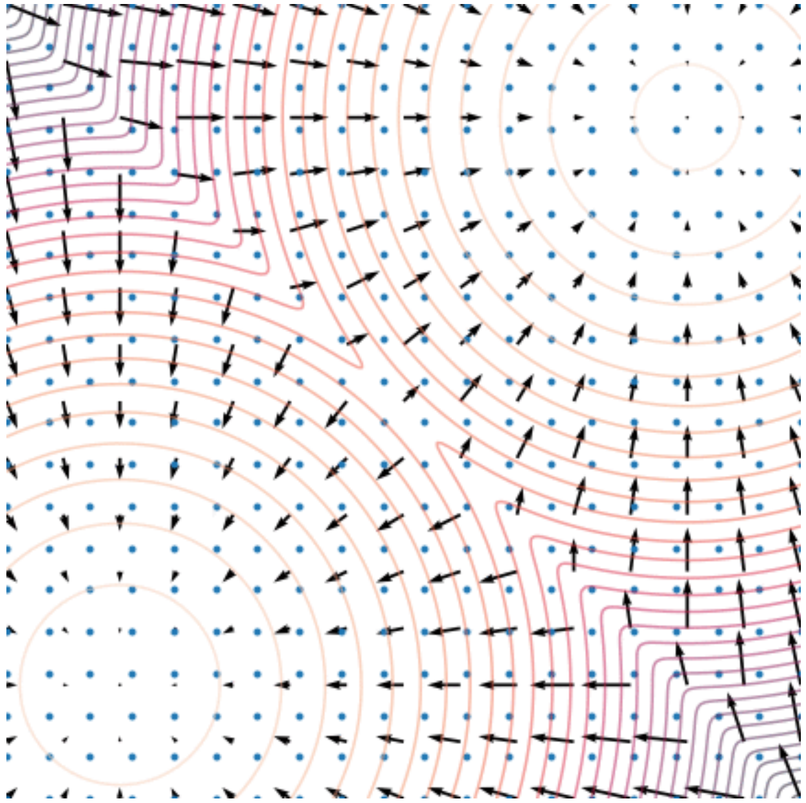
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t, \quad t = 0, \ldots, K, \quad \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- Where for $t = 0$ we sample from an arbitrary prior distribution $\mathbf{x}_0 \sim \pi(\mathbf{x})$
- And is a sample from a standard Gaussian

Sampling using Langevin dynamics

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t, t = 0, \ldots, K$$

- For $\epsilon \to 0$ and $K \to \infty$ we sample from $p(\mathbf{x})$ (under conditions)
- Importantly, this is an iterative sampling procedure for which we only need to score function
- So, we can produce samples by iteratively computing $\mathbf{x}_{t+1}$ via score function $s_\theta(x) \approx \nabla_x \log p(x)$



Langevin Dynamics

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t, \quad t = 0, \ldots, K, \quad \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- Originally developed to model molecular dynamics
- You can think of Langevin dynamics as something similar to stochastic gradient descent, only now we do not necessarily optimise for parameters
- Given your current position $\mathbf{x}_t$ we move to the direction of the gradient ∇ of the score function (log-likehood function) $\log p(\mathbf{x}_t)$, corrupted with some noise $\mathbf{z}_t$, scaled by ϵ (like 'learning rate') annealed over time
- A very nice work making the connection to Bayesian Learning*

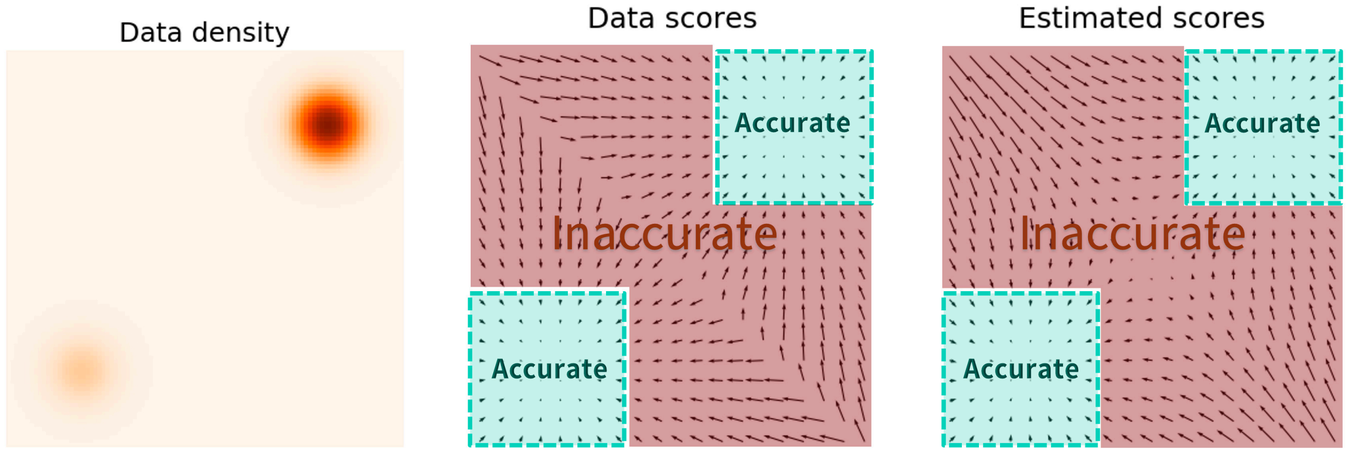
'Bayesian Learning via Stochastic Gradient Langevin Dynamics', M. Welling, Y. W. Teh

Low data density regions

- Minimising Fisher divergence means placing more emphasis where $p(\mathbf{x})$ is high

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

- Even harder in high-dimensional spaces that are mostly empty
- The Monte Carlo sample estimates will not be accurate enough



Slow mixing of Langevin dynamics

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t, \quad t = 0, \ldots, K, \quad \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- When the true density has two (or multiple) modes separated by a low-density region, it is hard for Langevin dynamics to visit them in a reasonable time
- That makes sense: the 'jumps' local around current location of score function and the added noise is unlikely to be large enough to push to far

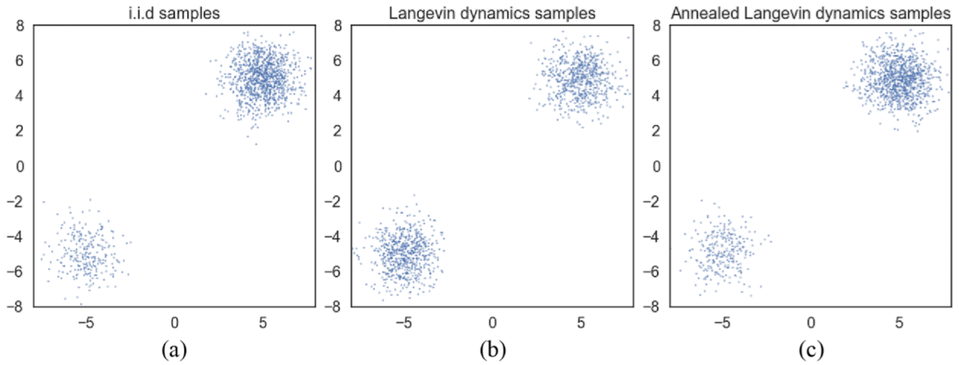
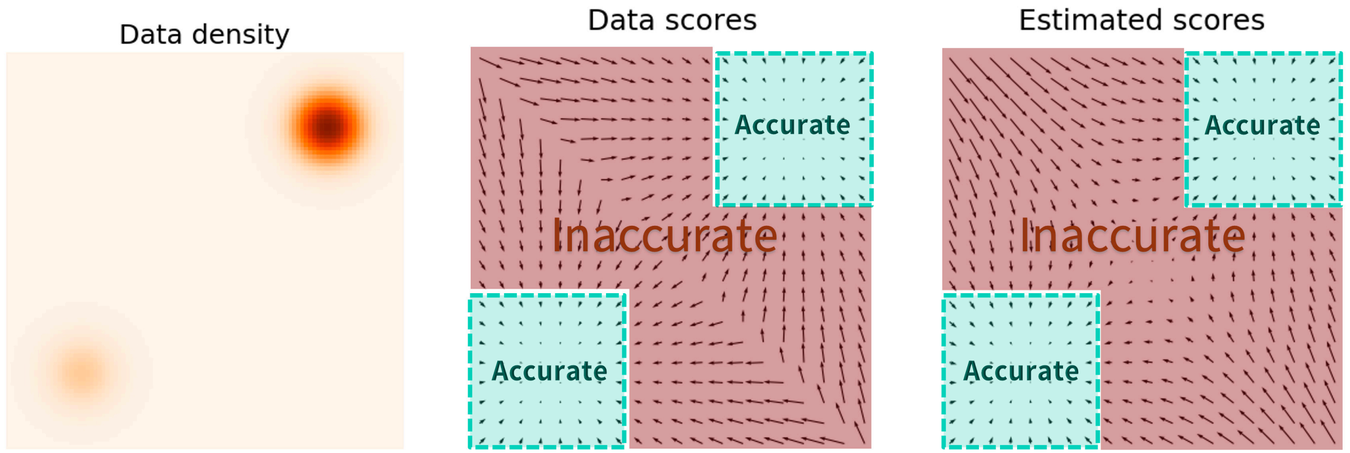


Figure 3: Samples from a mixture of Gaussian with different methods. (a) Exact sampling. (b) Sampling using Langevin dynamics with the exact scores. (c) Sampling using annealed Langevin dynamics with the exact scores. Clearly Langevin dynamics estimate the relative weights between the two modes incorrectly, while annealed Langevin dynamics recover the relative weights faithfully.

From 'Generative Modelling by Estimating Gradients of the Data Distribution', by Song and Ermon

Naive score-based ignores low-density regions

- In the naive case of training score-based methods we have innaccurate score function estimation
- And we have slow mixing of Langevin dynamics
- As a result, the Langevin chain will start from a low density region and get stuck

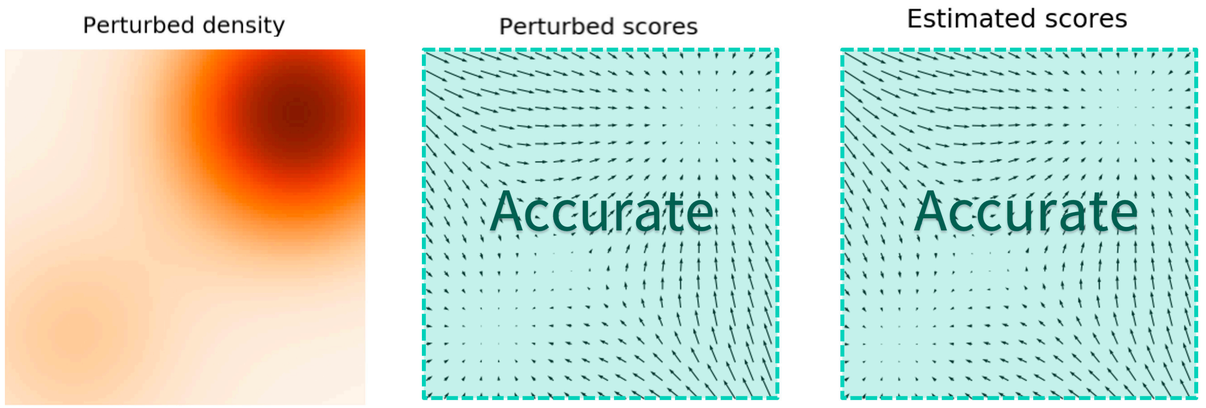


Noise perturbations

- Perturb data with noise ← Noised up data fill up the “empty” space
- Too much noise will over-corrupt the data, however, so caution is needed
- Add noise from $\mathcal{N}(0,\sigma_t)$ with more and more variance: $\sigma_1 < \sigma_2 < \ldots < \sigma_L$, specifically by marginalising out the noise variable

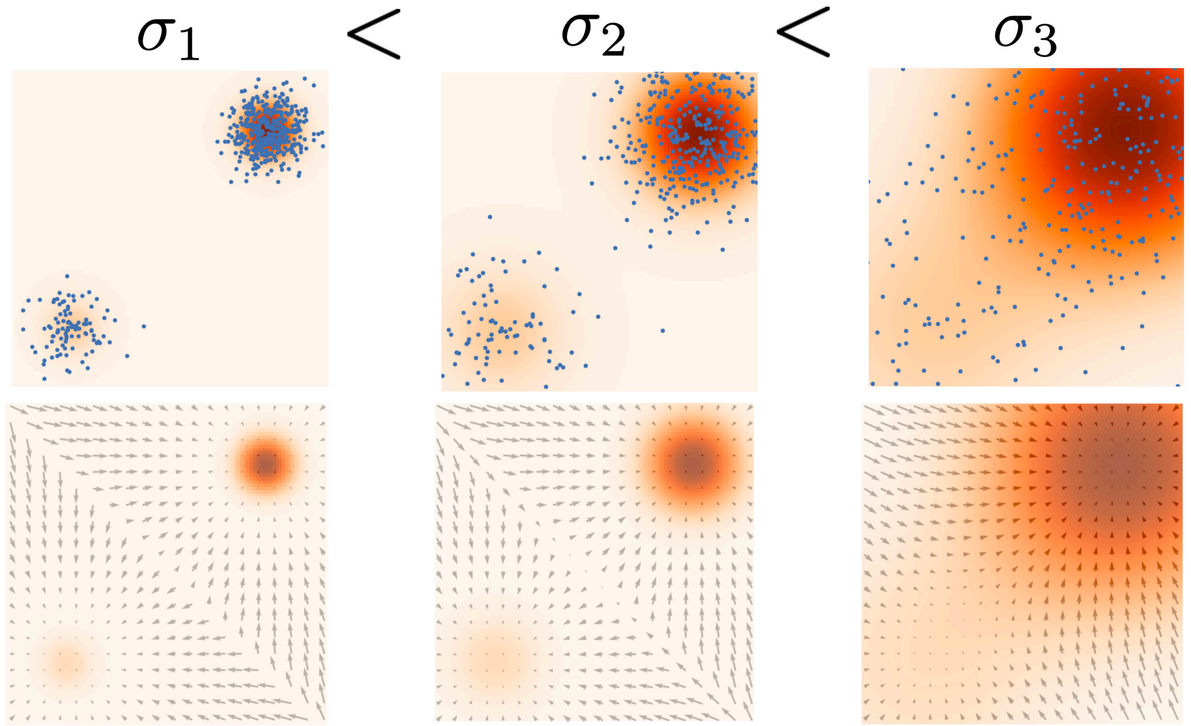
$$p_{\sigma_t}(\mathbf{x}) = \int p(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y} = \int p(\mathbf{y})p(\mathbf{x}\,|\,\mathbf{y})\mathrm{d}\mathbf{y}$$

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



Noise-conditional Score-based Models

- Learn the score-matching function on the perturbed data points



Multiple scales of Gaussian noise to perturb data (above) so that to learn the respective score-matching function (below).

Noise-Conditional Score-based Models

- The final objective is a weighted sum of Fisher divergences

$$\sum_t \lambda(t)\mathbb{E}_{p_{\sigma_t}(\mathbf{x})}\left[\|\nabla_{\mathbf{x}}\log p_{\sigma_t}(\mathbf{x}) - s_\theta(\mathbf{x}, t])\|\right]$$

where $\lambda(t)$ is a weighting function, typical choice $\lambda(t) = \sigma_t^2$

Noising-up real images

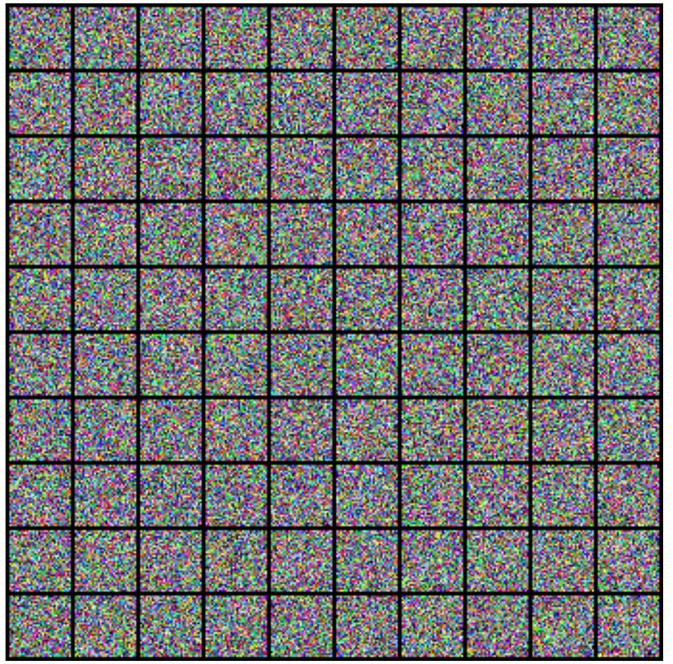
Annealed Langeving Dynamics

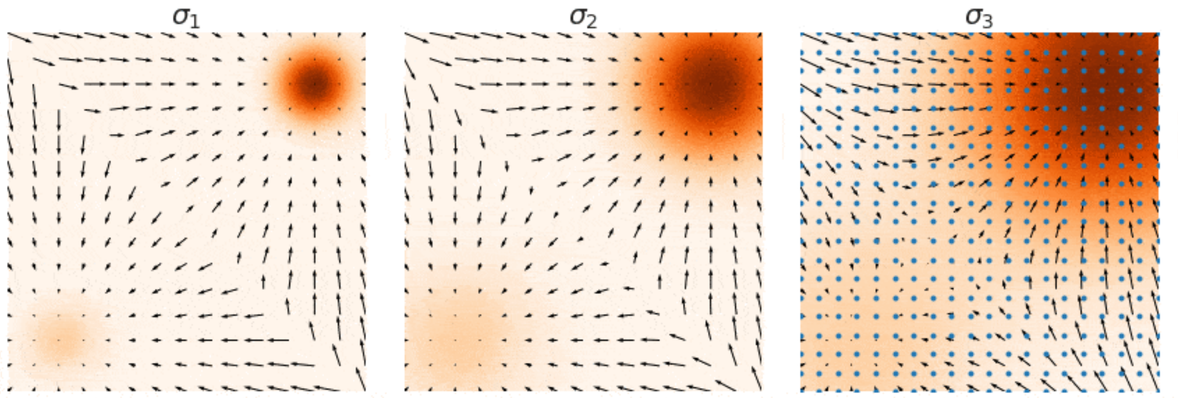
- Like before, but we start sampling from larger noise, which we gradually decrease

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
  end for
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$





Practical tips

- Pick σ_t in geometric progression where σ_L is comparable to max distance between samples in the training set
- L is typical in the order of hundreds or thousands
- Parameterize the score-based model with a U-Net with skip connections
- At test time use exponential moving averages on the weights

Score-based models with SDEs

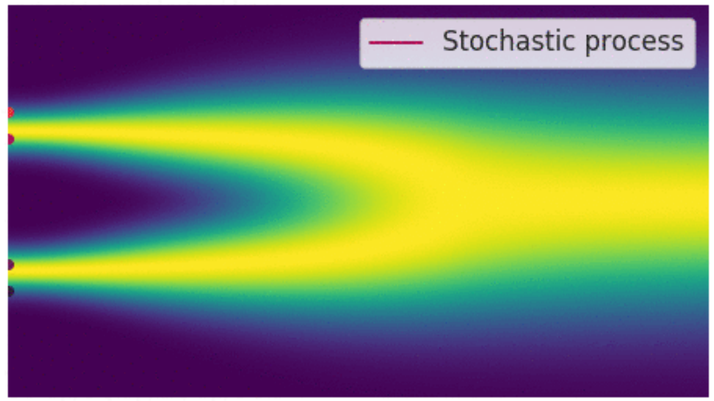
- Adding noise is important, but why ‘hardcode’?
- By generalising with infinite noise scales, we can
  - get higher quality samples
  - exact log-likelihood computation
  - controllable generation with inverse problem solving
- A stochastic process defines a process of generating infinite noise scales

Stochastic processes via SDEs

- A stochastic process can be defined in terms of (solution of) a stochastic differential equation

$$\mathrm{d}\mathbf{x} = f(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

- The change in our random variable is governed a function of the variable itself and time (drift coefficient) plus stochastic perturbation (noise) whose scale is a function of time (diffusion coefficient)



- $f(\,\cdot\,, t) : \mathbb{R}^d \to \mathbb{R}^d, g \in \mathbb{R}$, $\mathbf{w}$ is Brownian motion, and $\mathrm{d}\mathbf{w}$ is infinitesimal white noise

Solutions to SDEs

$$d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)d\mathbf{w}$$

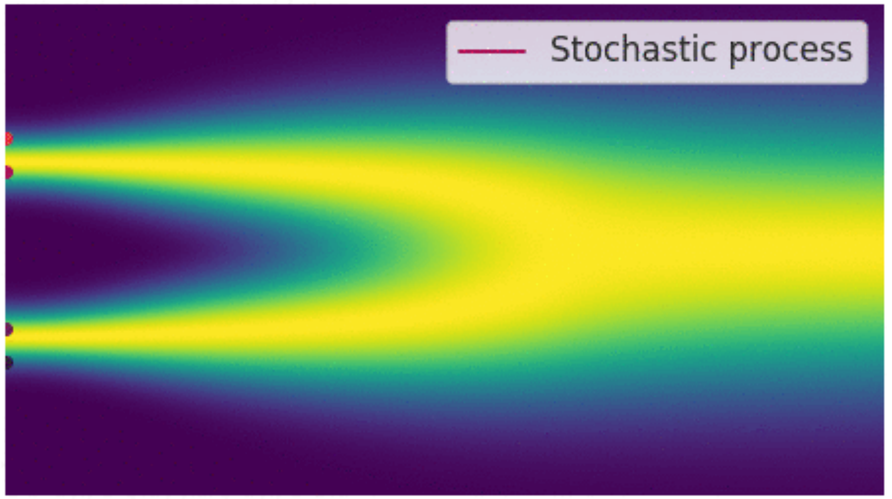
- Solutions to the SDEs are stochastic random variables $\{\mathbf{x}(t)\}_{t\in[0,T]}$
- These random variables are stochastic trajectories over time
- The probability density of $\mathbf{x}(t)$ is $p_t(\mathbf{x})$ (analogous to $p_{\sigma_i}(\mathbf{x})$ for the discrete case)
- $p_0(\mathbf{x})$ means the distribution in the data space, i.e., $p_0(\mathbf{x}) = p(\mathbf{x})$
- $p_T(\mathbf{x})$ is the distribution after all the noising up for period T until we end up to our prior distribution for our data generation process, i.e., $p_T(\mathbf{x}) = \pi(\mathbf{x})$

Perturbing data with noise from SDEs

- This SDE is the generalisation of the finite scaling $\sigma_0, \ldots, \sigma_L$
- Earlier we were perturbing according to a geometric progression of scales
- Now, we perturb with noise controlled by the SDE
- We select manually which SDE to model the process with
- If we were to select $\mathrm{d}\mathbf{x} = e^t \mathrm{d}\mathbf{w}$, we would add Gaussian noise $\mathrm{d}\mathbf{w}$ with a scale e^t that grows exponentially with time

From data to SDE noise

- Let's 'imagine' how the process works
- We can always start from any image sample $\mathbf{x}$ from $p_{data}(x)$
- ... and gradually add noise until it is a sample standard Gaussian distribution $\pi(\mathbf{x})$
- The point is, can we learn to do the reverse?

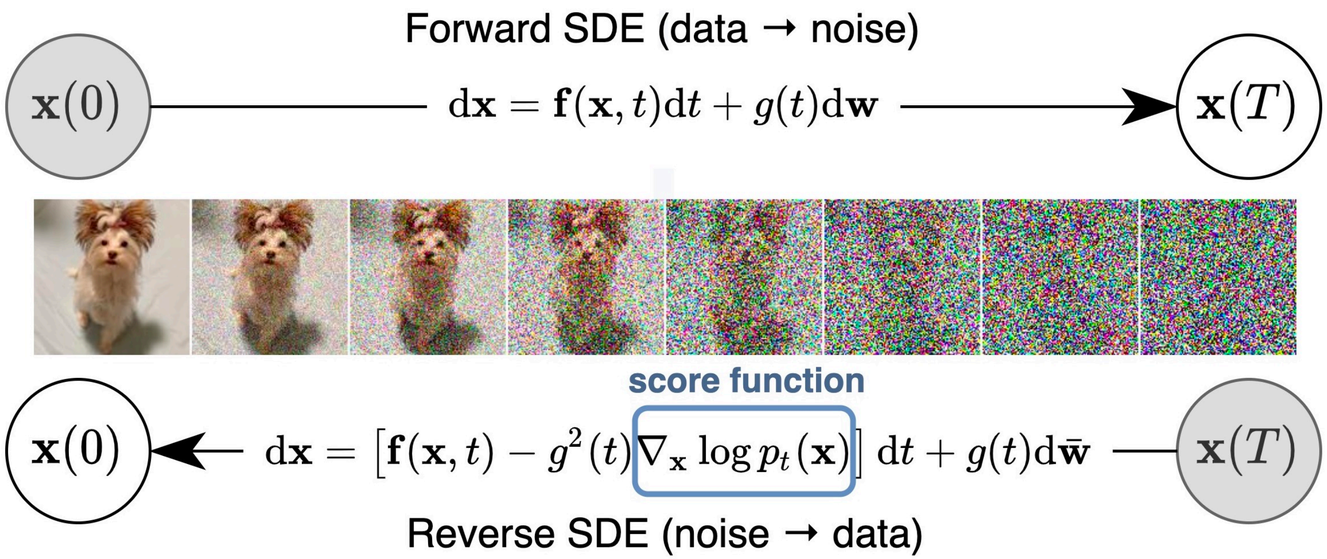


From reverse SDE noise to data

- For any SDE there is a reverse SDE, which corresponds to the reverse trajectories

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

- That is, for the reverse SDE we need precisely the score function of $p_t(\mathbf{x})$



Learning the reverse SDE

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

- Once we have the neural network approximating the score function
- We start from the prior distribution $\pi(\mathbf{x})$ for an initial sample $\mathbf{x}(T) \sim \pi(\mathbf{x})$
- To solve the reverse SDE, that is obtain all $\mathbf{X}(t),\ \ \forall t \in (T,0]$
- So that our final model p_θ approximates well the true data distribution, $p_\theta \approx p_0$
- If we set $\lambda(t) = g^2(t)$, it can be shown that

$$\mathrm{KL}(p_0(\mathbf{x}) \| p_\theta(\mathbf{x}))) \leq \mathbb{E}_{t\sim\mathcal{U}(0,T)} \mathbb{E}_{x\sim p_t(x)} \Big[\Big\| \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - s_\theta(\mathbf{x}) \Big\|_2^2 \Big] + \mathrm{KL}(p_T(\mathbf{x}) \| \pi(\mathbf{x})))$$

- Assuming perfect score-matching, we can approximate the data distribution as well as we match the prior distribution

Solving the reverse SDE

- Once we have trained the score-matching function, we can solve the reverse SDE from the prior π all the way to our data distribution p_0 to generate new data
- We can use any numerical solver, e.g., the Euler-Maruyama, for a small negative Δt

$$\Delta \mathbf{x} \leftarrow \left[f(\mathbf{x}, t) - g^2(t) s_\theta(\mathbf{x}, t) \right] \Delta t + g(t) \sqrt{|\Delta t|}\, \mathbf{z}_t, \quad \mathbf{z}_t \sim \mathcal{N}(0, I)$$

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}$$

$$t \leftarrow t + \Delta t$$

- With better sampling procedures for SDEs and better architectures, one gets state-of-the-art in generated samples

Time-dependent score-matching

- Train a neural network for score-matching that depends on time

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

where typically $\lambda(t) \propto 1/\mathbb{E}\left[\|\nabla_{\mathbf{x}(t)}\log p(\mathbf{x}(t)\,|\,x(0))\|_2^2\right]$

- Randomly sample time steps
- Then sample data from training set
- Then optimise your score matching approximation

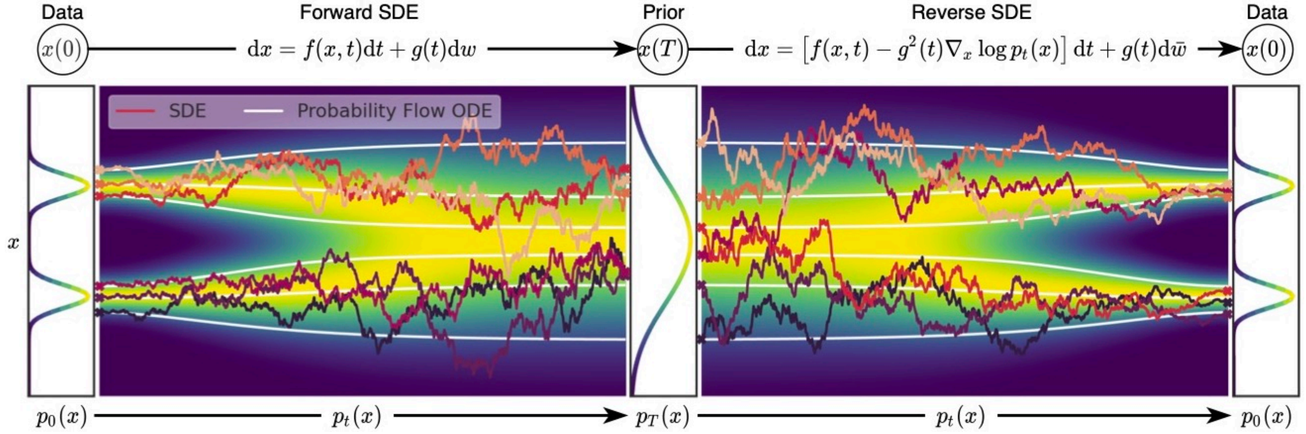
Qualitative examples

Probability flow ODE

- With Langevin MCMC samplers and SDE solvers we can't get exact log-likelihoods
- We can convert the SDE to a corresponding ODE without changing the marginal distributions $\{p_t(\mathbf{x})\}_{t\in[0,T]}$, name the probability flow ODE

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

- Solving the ODE, we can get the exact log-likelihood



Diffusion Probabilistic Models

- Concurrently, another very similar class of models appeared: diffusion models
- Diffusion models also define a forward and reverse diffusion process, where $t = 0$ corresponds to the data distribution, and $t = T$ a unit-Gaussian distribution

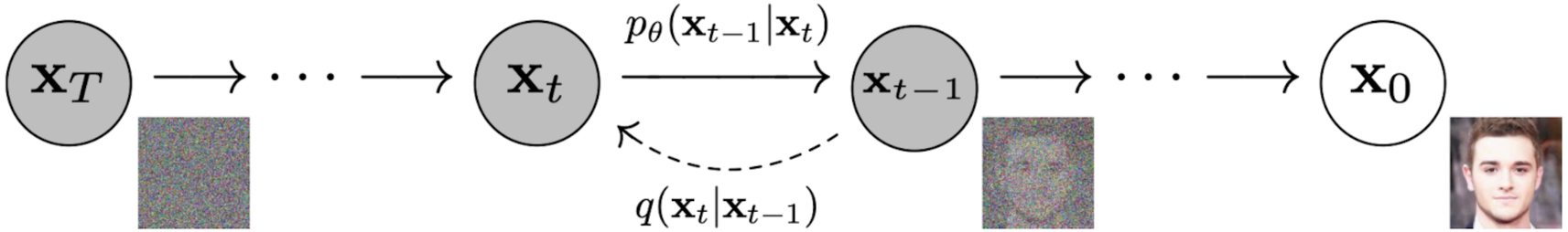


Figure 2: The directed graphical model considered in this work.

Diffusion probabilistic models, Sohl-Dickstein et al., 2015
Denoising diffusion probabilistic models, Ho et al., 2020
Diffusion models beat GANs on image synthesis, 2021
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Forward diffusion process

- In forward diffusion we add small Gaussian noise to our data till it looks like isotropic Gaussian

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}), \quad q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

- We can define the conditional distribution at any time step t w.r.t. step $t = 0$

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{a_t}\mathbf{x}_{t-1} + \sqrt{1-a_t}\mathbf{z}_{t-1} && \text{, where } \mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \ldots \sim \mathcal{N}(0,1) \\
&= \sqrt{a_t a_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-a_t a_{t-1}}\bar{\mathbf{z}}_{t-2} && \text{, where } \bar{\mathbf{z}}_{t-2} \text{ merges two Guassians} \\
&= \ldots \\
&= \sqrt{\bar{a}_t}\mathbf{x}_0 + \sqrt{1-\bar{a}_t}\bar{\mathbf{z}} \\
q(\mathbf{x}_t \mid \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{a}_t}\mathbf{x}_0, \sqrt{1-\bar{a}_t}\mathbf{I})
\end{aligned}$$

- For this we use that when merging two Gaussians, we get another Gaussian with variance $\sigma_1^2 + \sigma_2^2$

Reverse diffusion process

- The reverse diffusion process can be efficiently parameterised to combine with variational inference

$$L_{VLB} = L_T + L_{T-1} + \ldots + L_0$$

$$\text{where } L_T = D_{KL}\big(q(\mathbf{x}_T \,|\, \mathbf{x}_0) \| p_\theta(\mathbf{x}_T)\big)$$

$$L_t = D_{KL}\big(q(\mathbf{x}_t \,|\, \mathbf{x}_{t+1}, \mathbf{x}_0) \| p_\theta(\mathbf{x}_t \,|\, \mathbf{x}_{t+1})\big) \text{ for } 1 \leq t \leq T-1$$

$$L_0 = -\log p_\theta(\mathbf{x}_0 \,|\, \mathbf{x}_1)$$

- Since we have Gaussian distributions the KL terms can be computed in closed form
- L_T does not depend any parameters and it can be dropped
- L_0 depends on the final decoder output

Parameterising L_t

- By smart parameterisation of the intermediate Gaussians, learning boils down to minimising

$$L_t^{\text{simple}} = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{a}_t}\mathbf{x}_0 + \sqrt{1-\bar{a}_t}\boldsymbol{\epsilon}_t, t\right)\right\|_2^2\right]$$

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** $\mathbf{x}_0$

Example trajectories

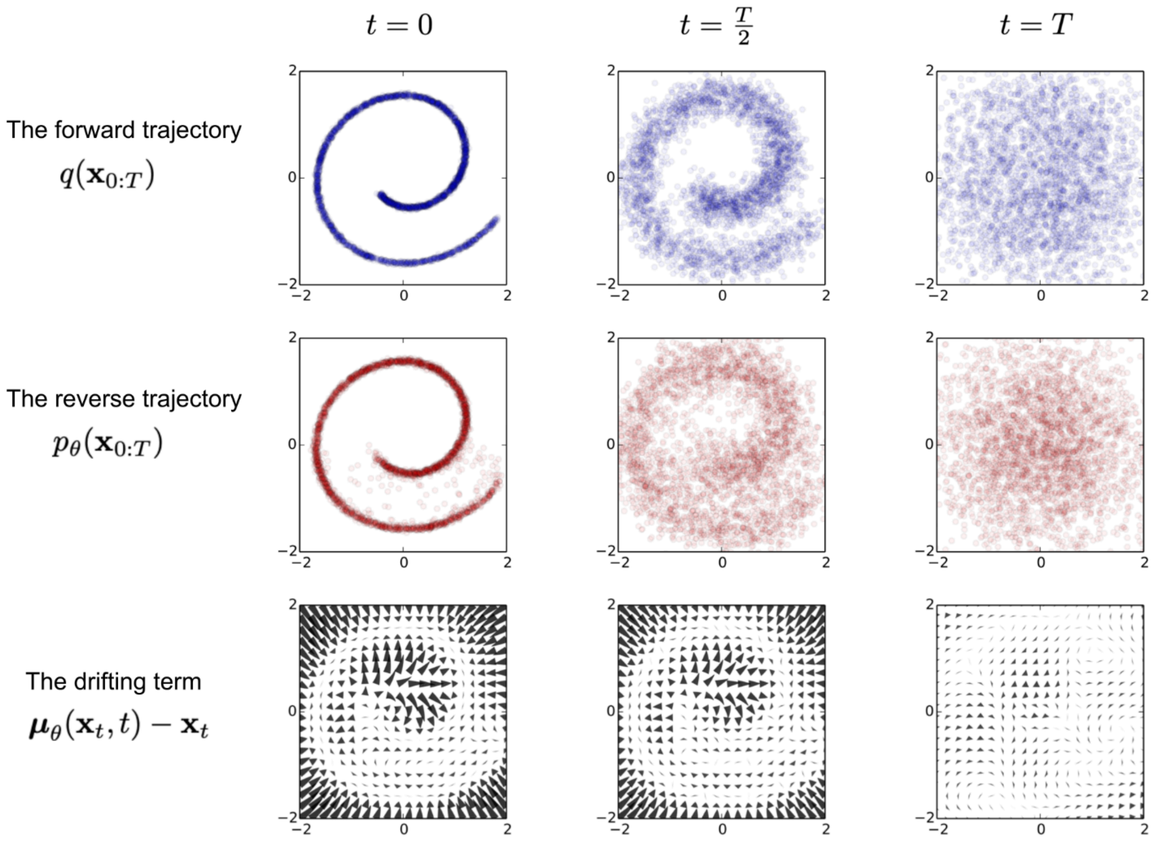
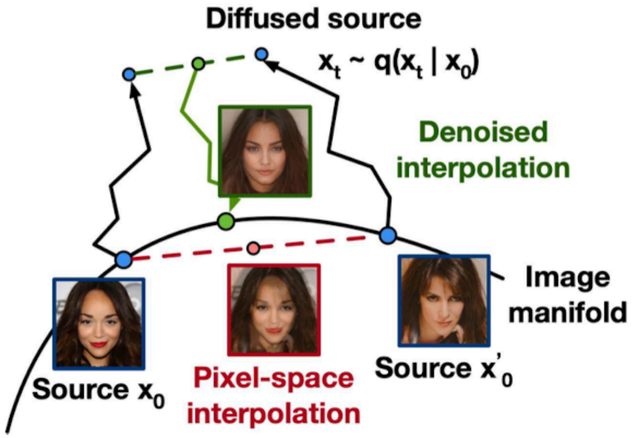


Fig. 3. An example of training a diffusion model for modeling a 2D swiss roll data. (Image source: Sohl-Dickstein et al., 2015)

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Qualitative results



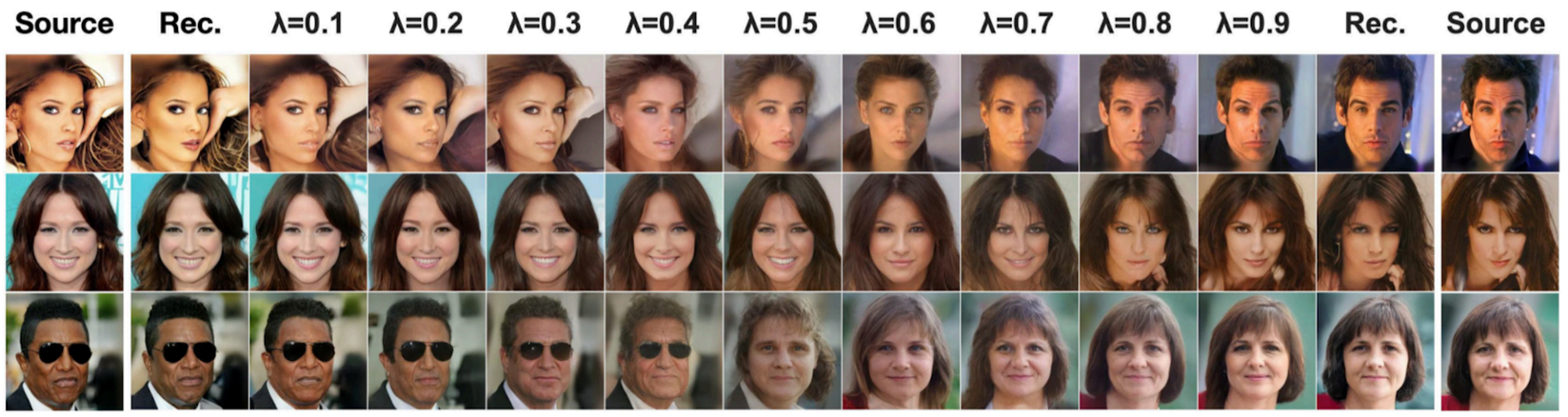


Figure 8: Interpolations of CelebA-HQ 256x256 images with 500 timesteps of diffusion.

Take-home message

- Diffusion/score-matching models are both tractable and flexible
- However, they are still quite slow to sample from compared to GANs
- The reason is that they require very long chains of time steps up to $T = 1{,}000$
- Great opportunities for learning the data structure effectively and efficiently enough
- Promising results in modelling inverse problems