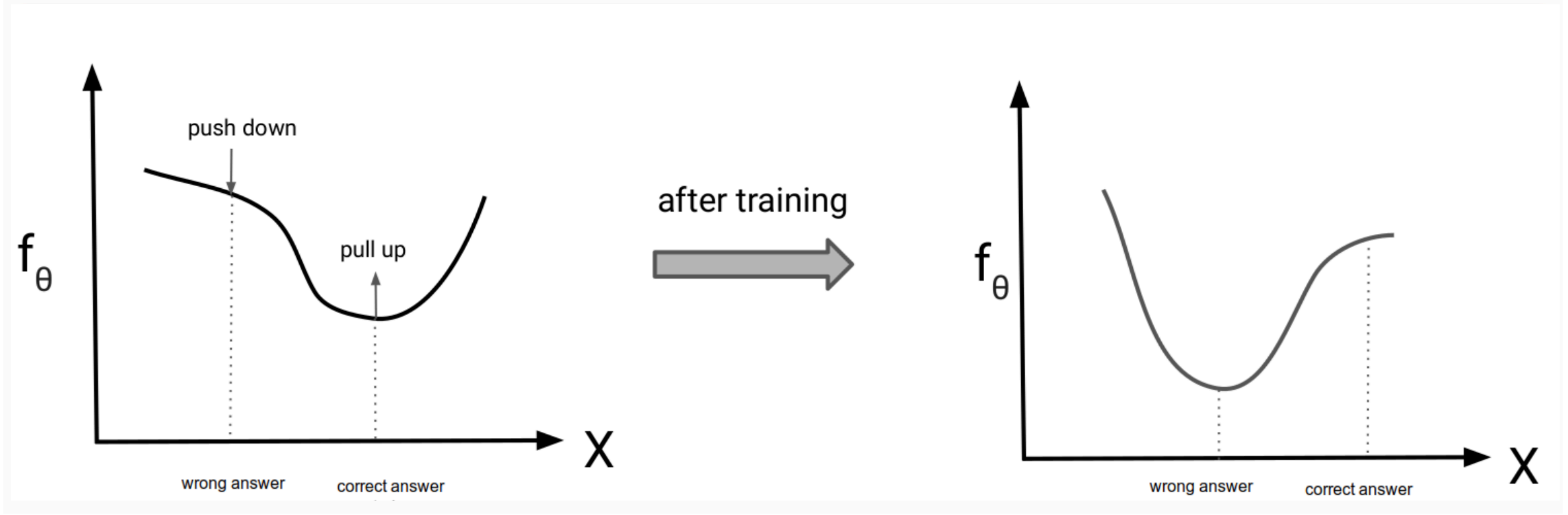
Energy-based models: a recap

- Alternative to likelihood-based models

$$p_\theta(\mathbf{x}) = \frac{\exp(-f_\theta(\mathbf{x}))}{Z_\theta}$$

$$Z_\theta = \int \exp(-f_\theta(\mathbf{x}))dx$$



Stefano Ermon and Aditya Grover

- We cannot ignore during training Z_θ as it also depends on parameters
- For general functions & architectures f_θ intractable to maximise wrt likelihood due to Z_θ

Score-based models: impressive results

- Energy-based models with GAN-like quality in generation, 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)

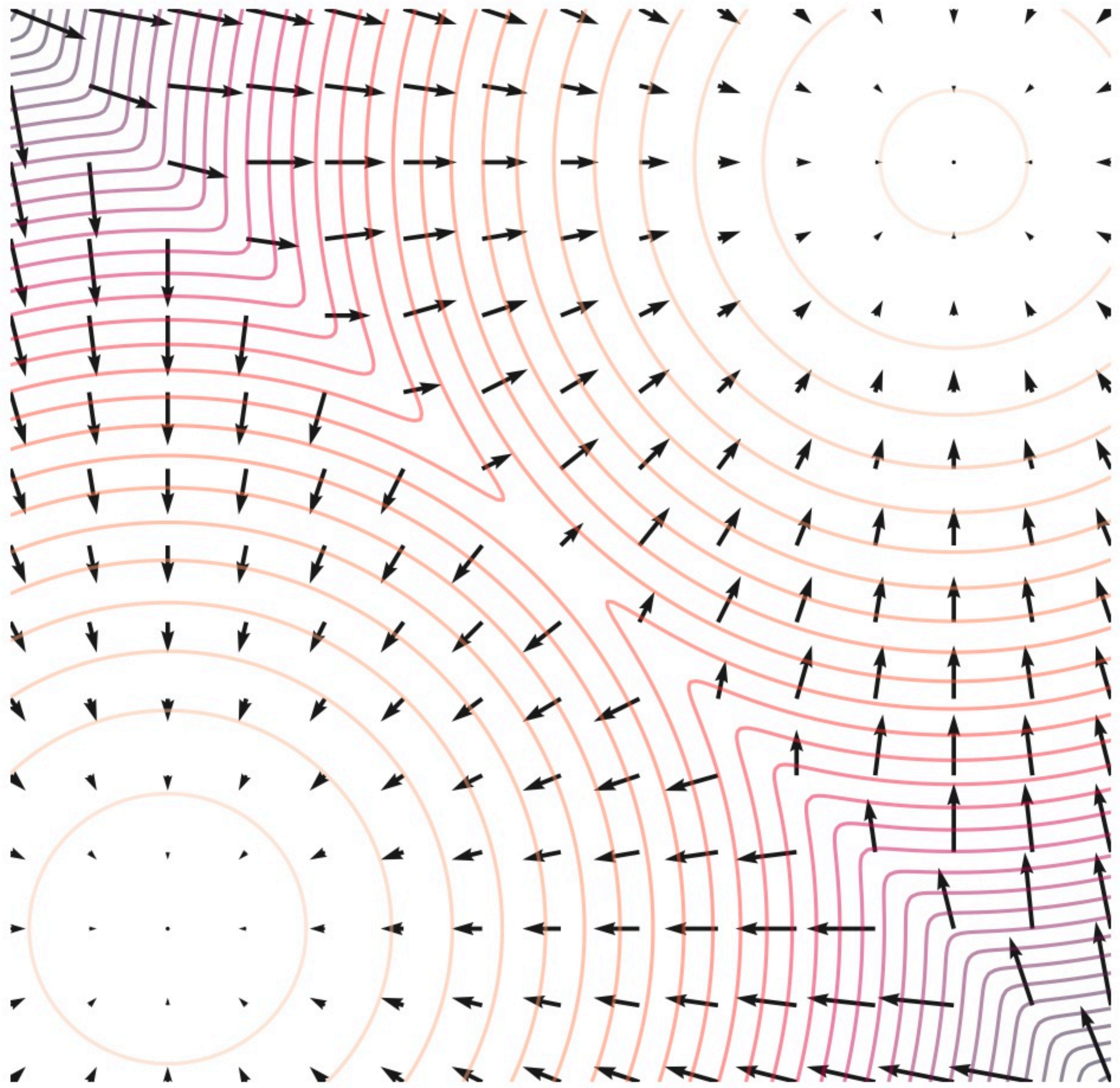
Score function

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

$$\nabla_{\mathbf{x}} \log p(\mathbf{x})$$

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

$$s_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$


The score function of a mixture of two Gaussians

Score-based generative models

- The score function does not depend on the normalising constant Z_θ

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

- Thus we do not care if Z_θ is tractable or not
- What to optimise since score-based model outputs a vector representing gradients?
- E.g., to minimise the Fisher divergence ← optimal gradient/ground truth data score

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

Y. Song, S. Ermon, Generative Modeling by Estimating Gradients of the Data Distribution. NeurIPS 2019

Score-matching

- It can be shown[*] that optimising $\mathbb{E}_{p(\mathbf{x})} \| \nabla_{\mathbf{x}} \log p(\mathbf{x}) - s_\theta(\mathbf{x}) \|_2^2$ 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

- No dependence on ground truth score gradients
- Still, the trace of the Jacobian is too expensive for large networks and approximations are needed

[*] Estimation of Non-Normalized Statistical Models by Score Matching, Hyvarinen, 2005
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 $q_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}}\,|\,\mathbf{x})p_{data}(\mathbf{x})d\mathbf{x}$

- 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

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

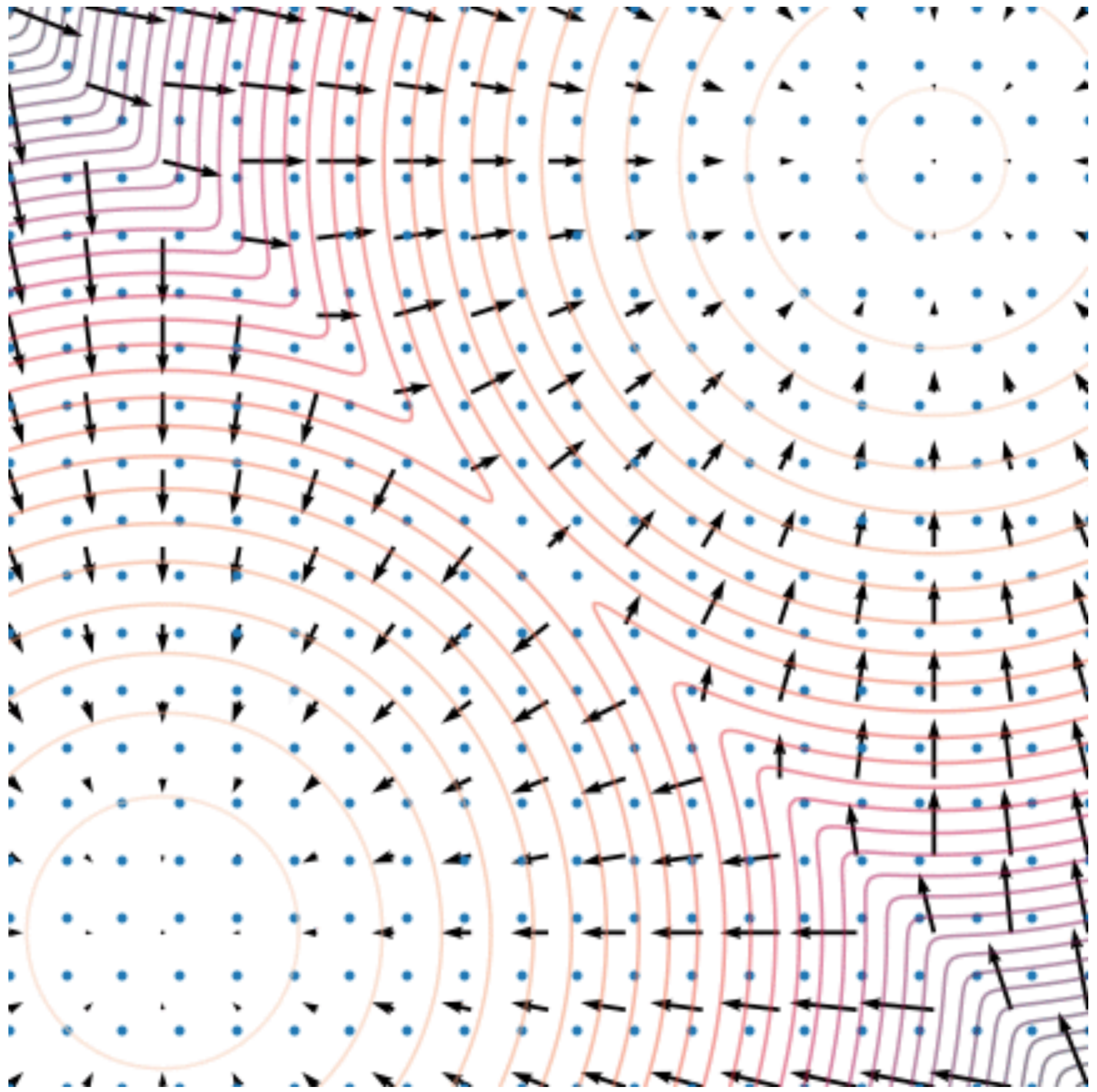
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t, \;\; t = 0, \ldots, K, \;\; \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- At $t = 0$ we sample from an arbitrary prior distribution $\mathbf{x}_0 \sim \pi(\mathbf{x})$

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 $\epsilon \rightarrow 0$ and $K \rightarrow \infty$ we sample from $p(\mathbf{x})$
- Iterative sampling process relying only on score function
- Sample $\mathbf{x}_{t+1}$ iteratively via $s_\theta(x) \approx \nabla_x \log p(x)$



Langevin Dynamics

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

- Originally developed to model molecular dynamics
- Similar to noised-up SGD, but not for optimising parameters
- Given current $\mathbf{x}_t$ move towards more likely densities ($\nabla_{\mathbf{x}}$) of $\log p(\mathbf{x}_t)$, yet corrupted with noise $\mathbf{z}_t$ for randomness, and scaled by annealed ϵ (like 'learning rate')
- A very nice work making the connection to Bayesian Learning*

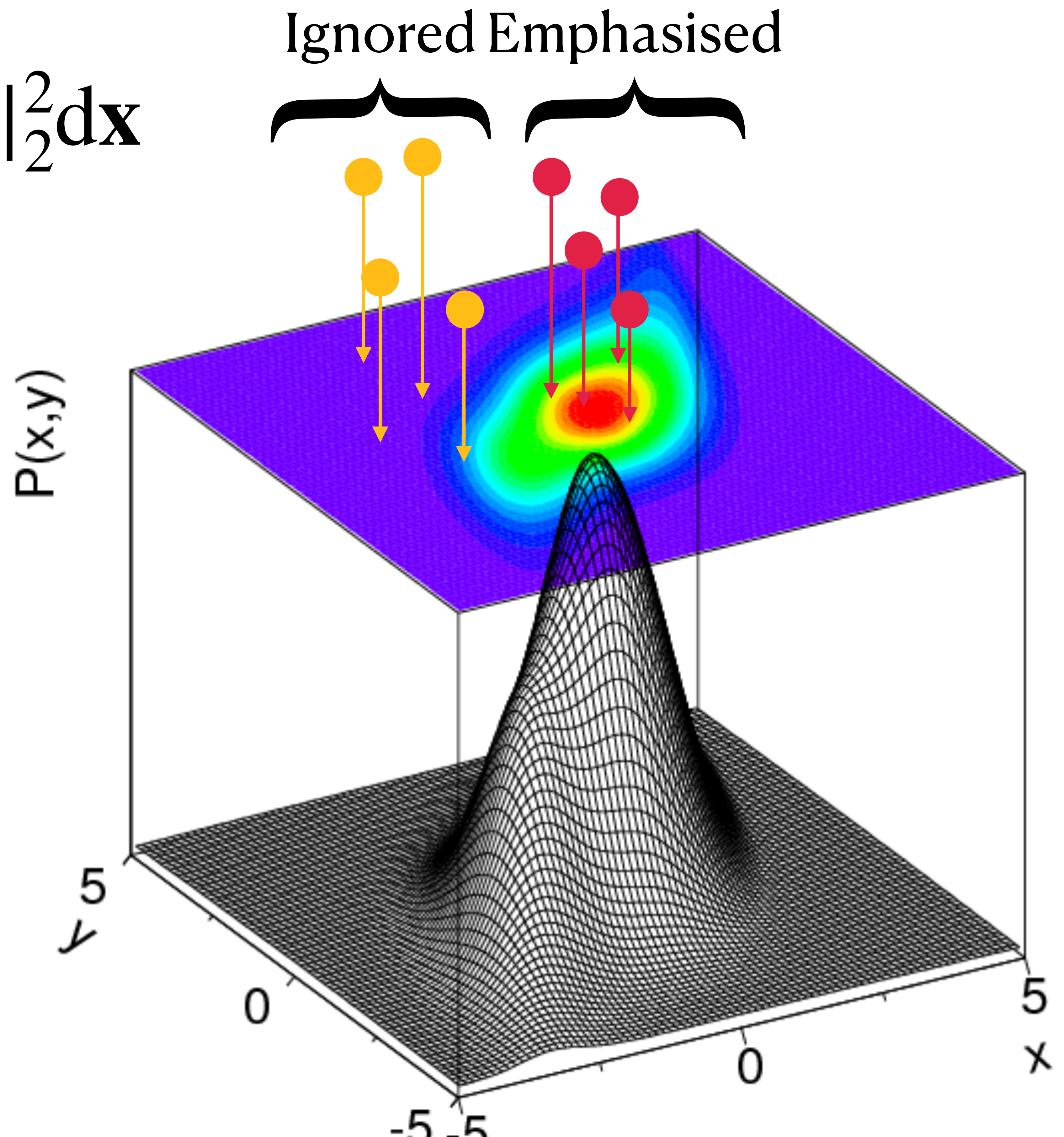
'Bayesian Learning via Stochastic Gradient Langevin Dynamics', M. Welling, Y. W. Teh

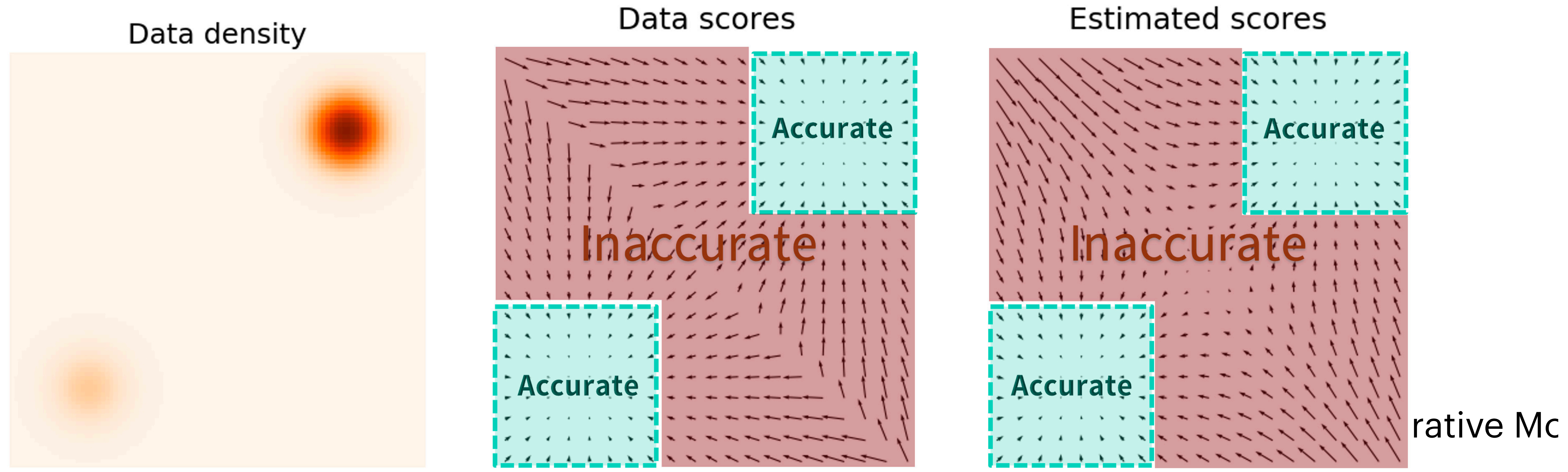
Low data density regions

- Minimising Fisher divergence → emphasising high $p(\mathbf{x})$ regions as they matter most

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

- In high-dimensions harder as space is mostly empty
- Monte Carlo 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 far enough

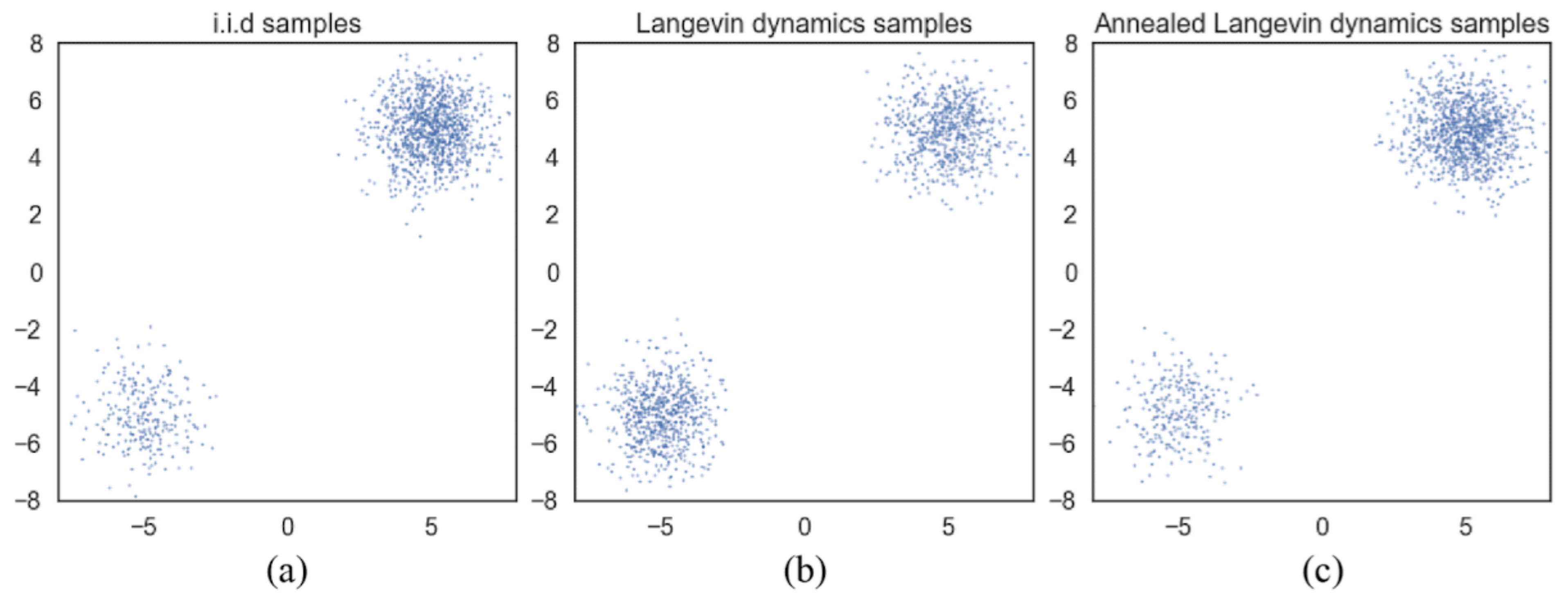
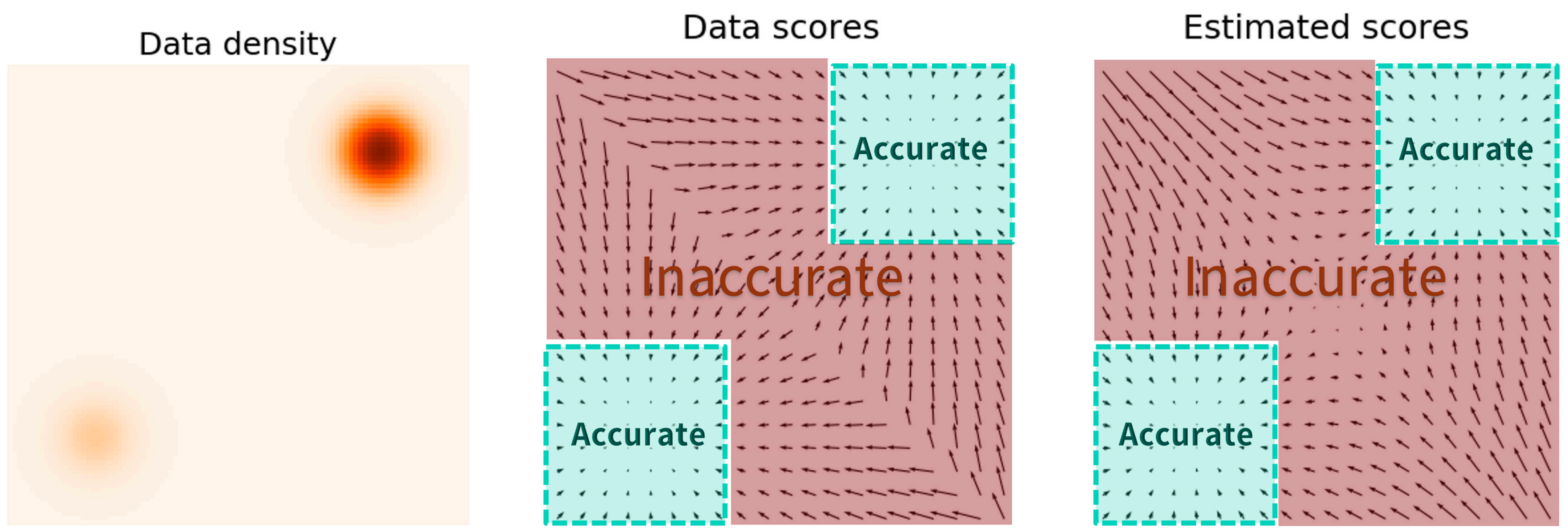


Figure 3: Samples from a mixture of Gaussian with different methods. (a) Exact sampling. (b) Sampling using Langevin dynamics with the exact scores. (c) Sampling using annealed Langevin dynamics with the exact scores. Clearly Langevin dynamics estimate the relative weights between the two modes incorrectly,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

- Naive score-based training leads to inaccurate 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

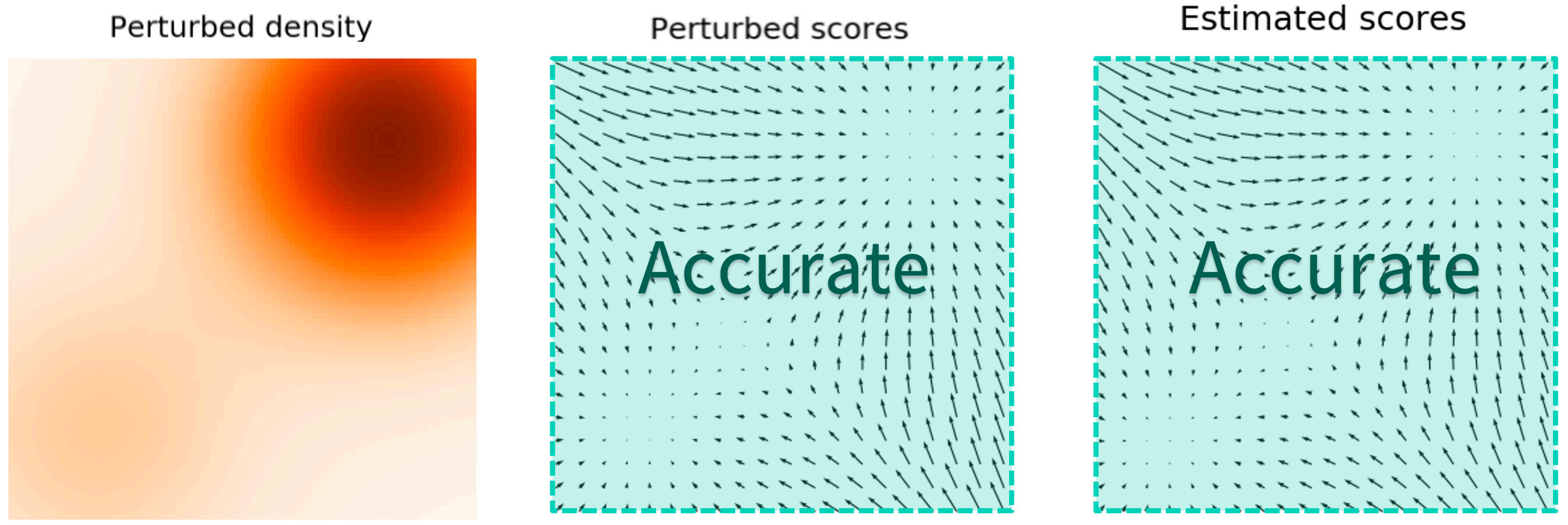


Solution: Perturb with noise (and denoise)

- For noised-up score matching perturb data with noise

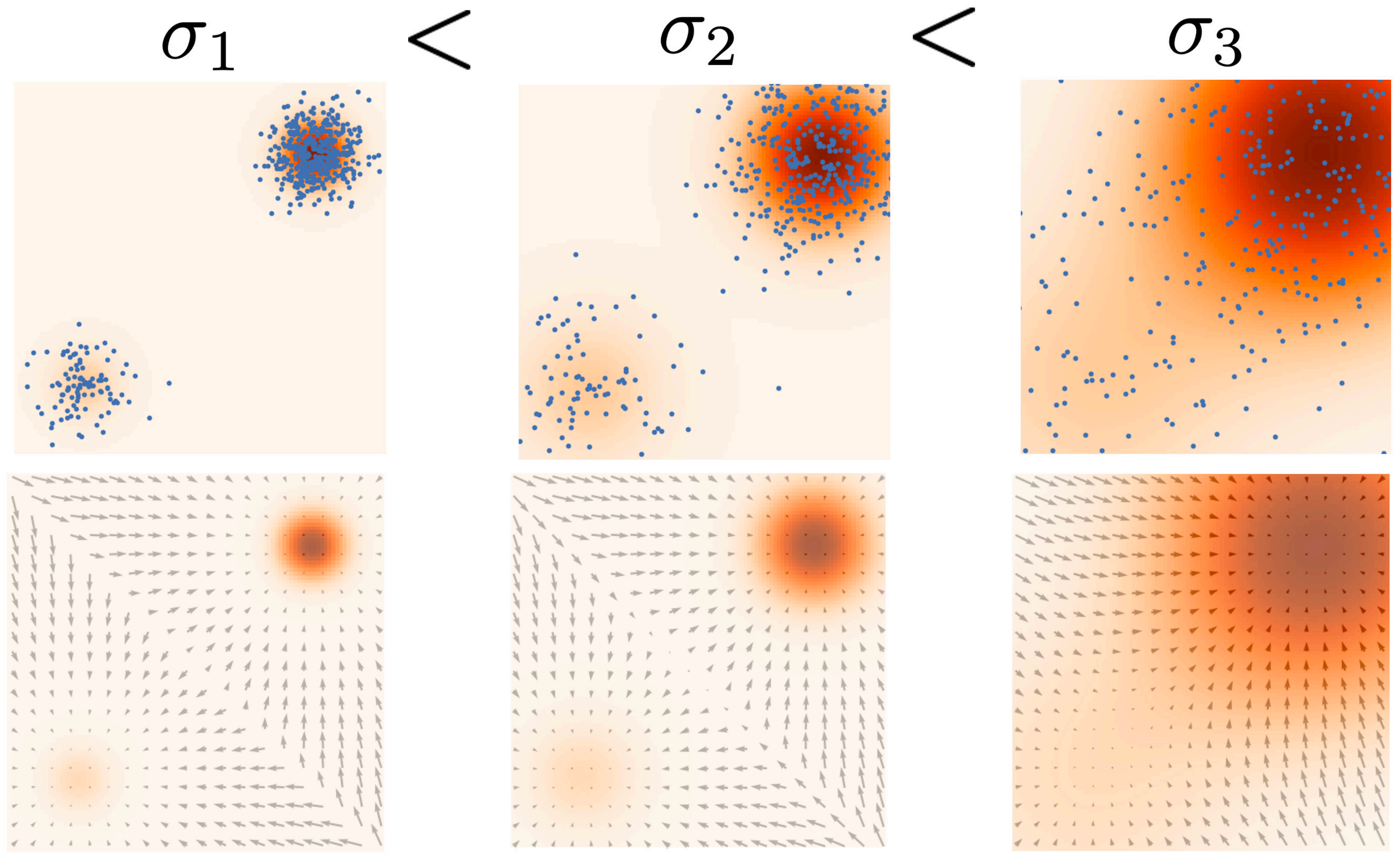
$$q_{\sigma_t}(\tilde{\mathbf{x}}) = \int p(\mathbf{x}) q(\tilde{\mathbf{x}} \mid \mathbf{x}) \mathrm{d}\mathbf{x} = \int p(\mathbf{x}) \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma_t^2 \mathbf{I}) \mathrm{d}\mathbf{x}$$

- Noised up data fill up the "empty" space
- Be careful: if too much noise data will be over-corrupted (strong distribution shift)

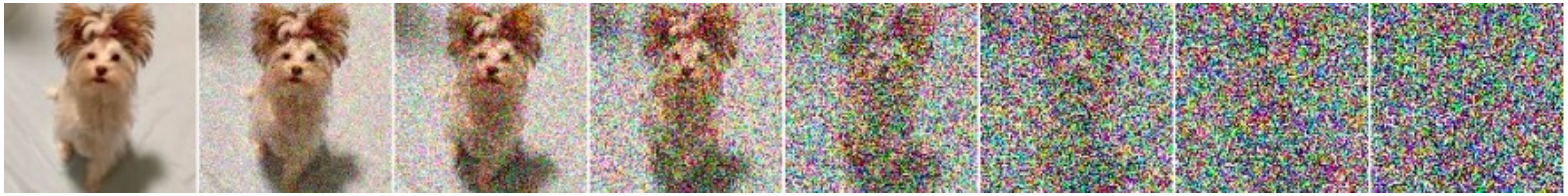


Noise-conditional Score-based Models

- 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

- For a single noise scale and denoising a Gaussian noise auxiliary noise

$$\mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I}) \Rightarrow \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = -(\tilde{\mathbf{x}} - \mathbf{x})/\sigma^2$$

The loss becomes

$$\frac{1}{2} \mathbb{E}_{p_{data}(\mathbf{x})} \mathbb{E}_{\tilde{\mathbf{x}} \mathcal{N}(\mathbf{x}, \sigma^2 \mathbf{I})} \left[\left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}, \sigma) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \right\|_2^2 \right]$$

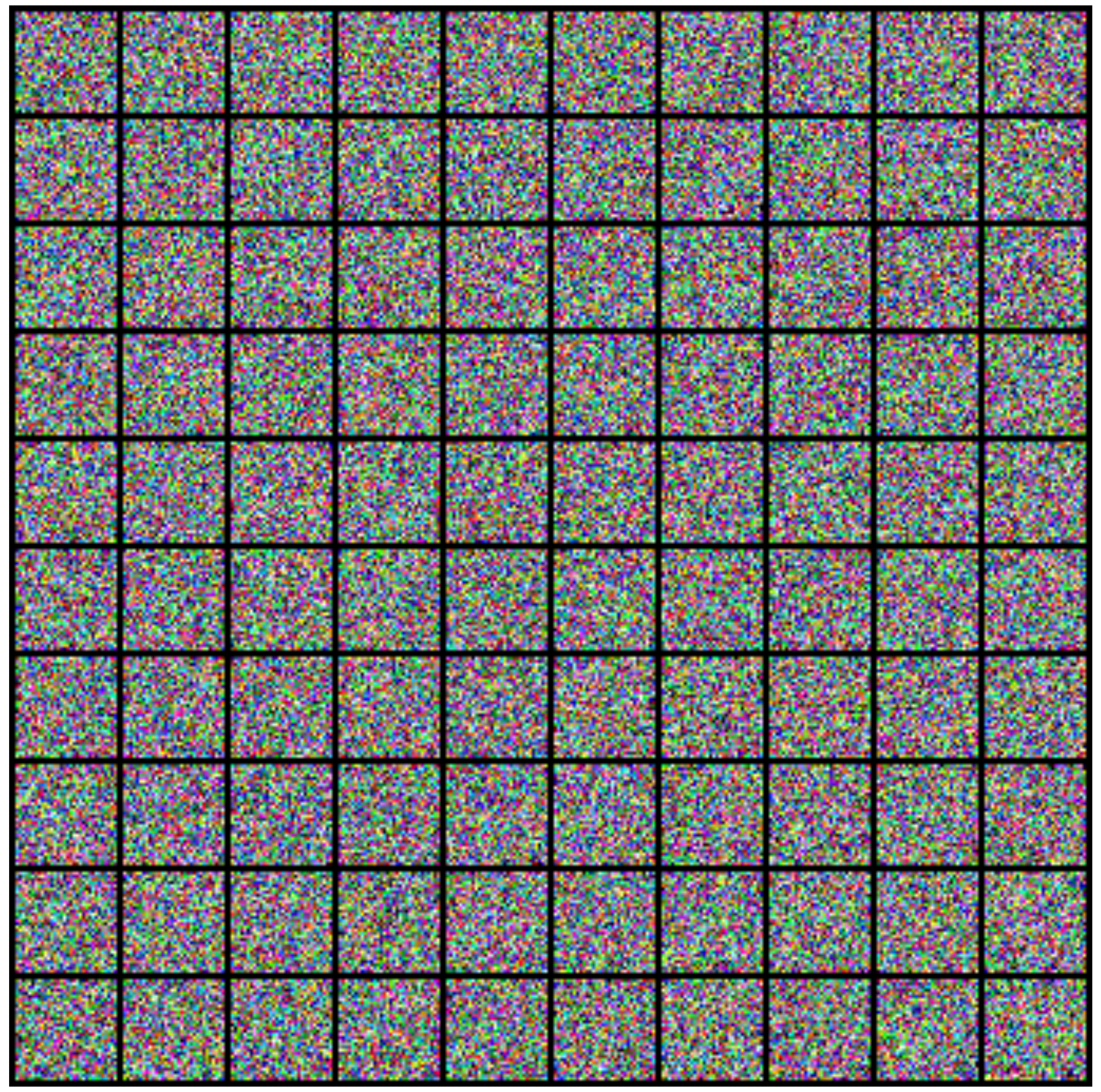
- And for multiple scales

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

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

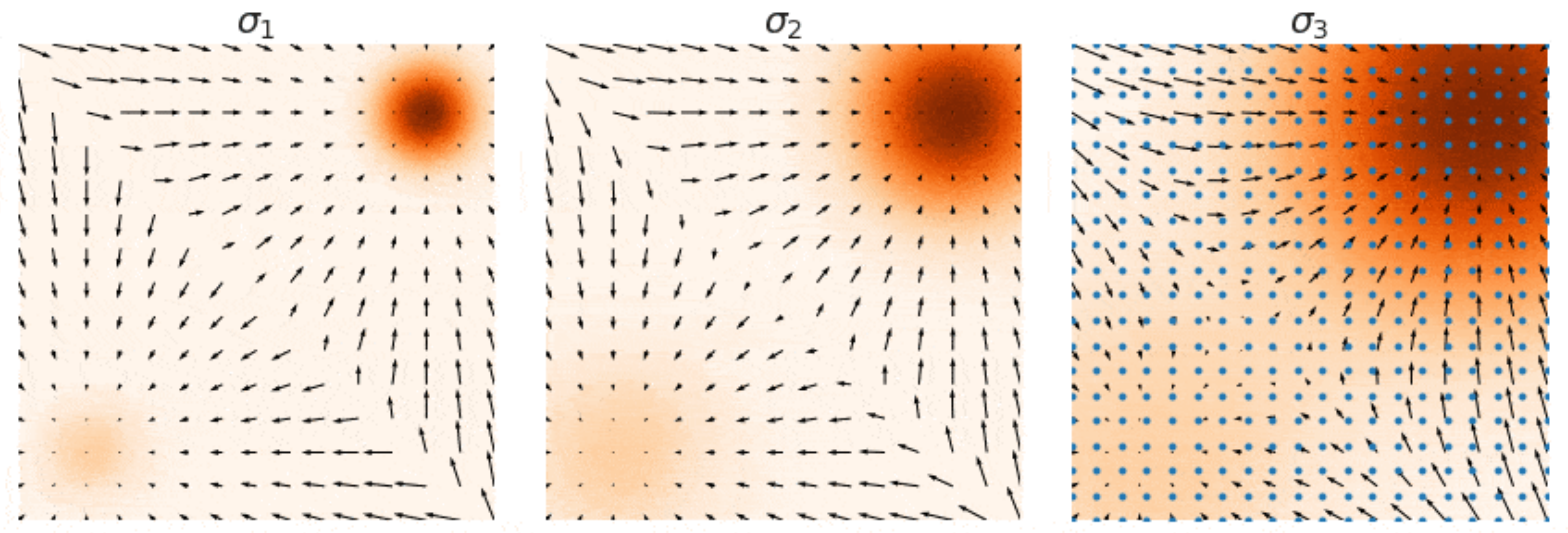
Annealed Langevin Dynamics

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



Algorithm 1 Annealed Langevin dynamics.

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



Practical tips

- Pick σ_t in geometric progression $\frac{\sigma_1}{\sigma_2} = \frac{\sigma_2}{\sigma_3} = \ldots.$
- σ_L should be comparable to max distance between samples in the training set
- L is usually a few hundreds or thousands
- Parameterise the score-based model with a U-Net with skip connections
- At test time use exponential moving averages on the weights