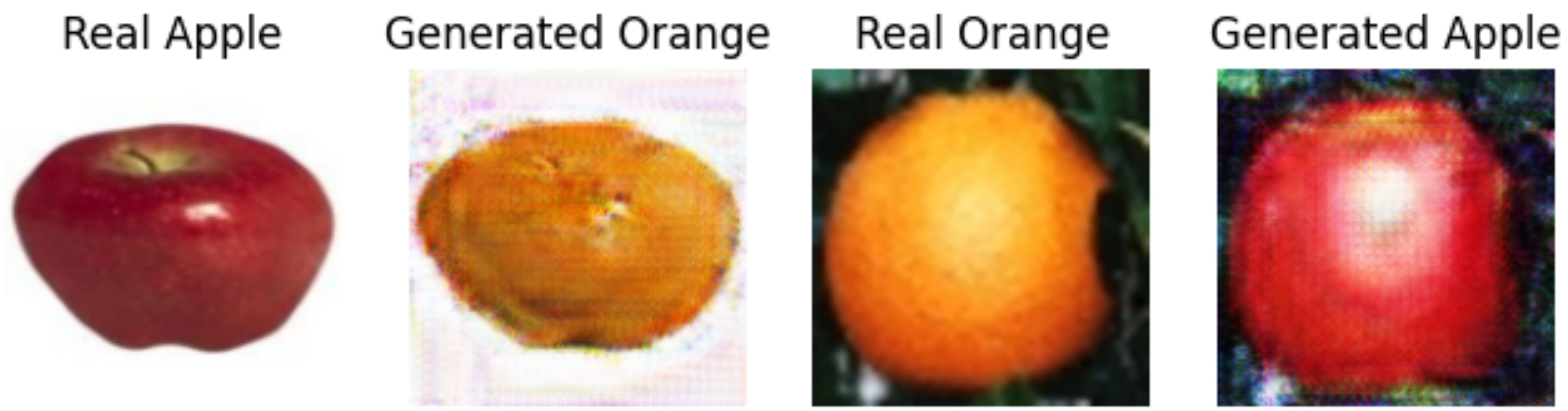
Lecture 9: Energy-based models

CS 5788: Introduction to Generative Models

PS2 out

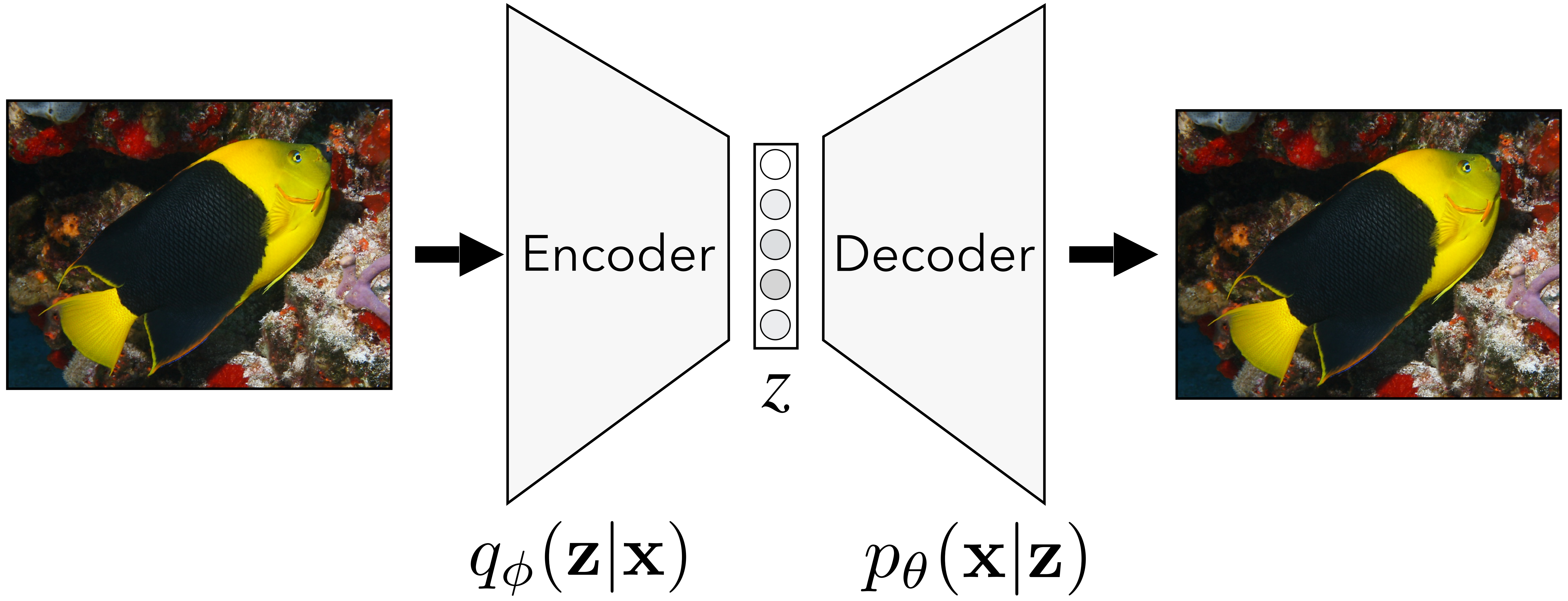
- Variational autoencoders

- CycleGAN



- ~~Normalizing flow~~ (held for PS3)

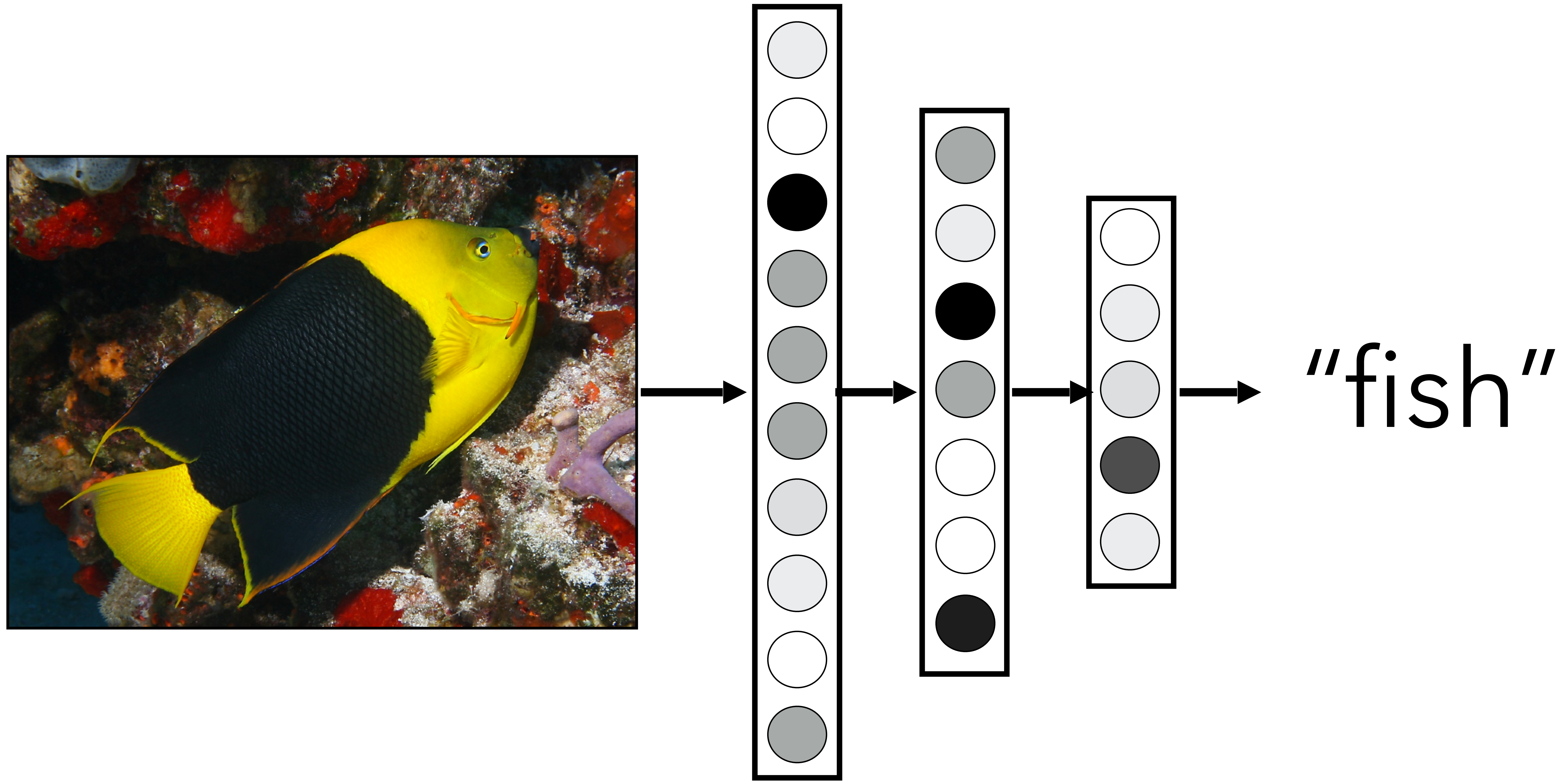
Latent variable model


Encoder
Decoder
z
$q_\phi(\mathbf{z}|\mathbf{x})$
$p_\theta(\mathbf{x}|\mathbf{z})$

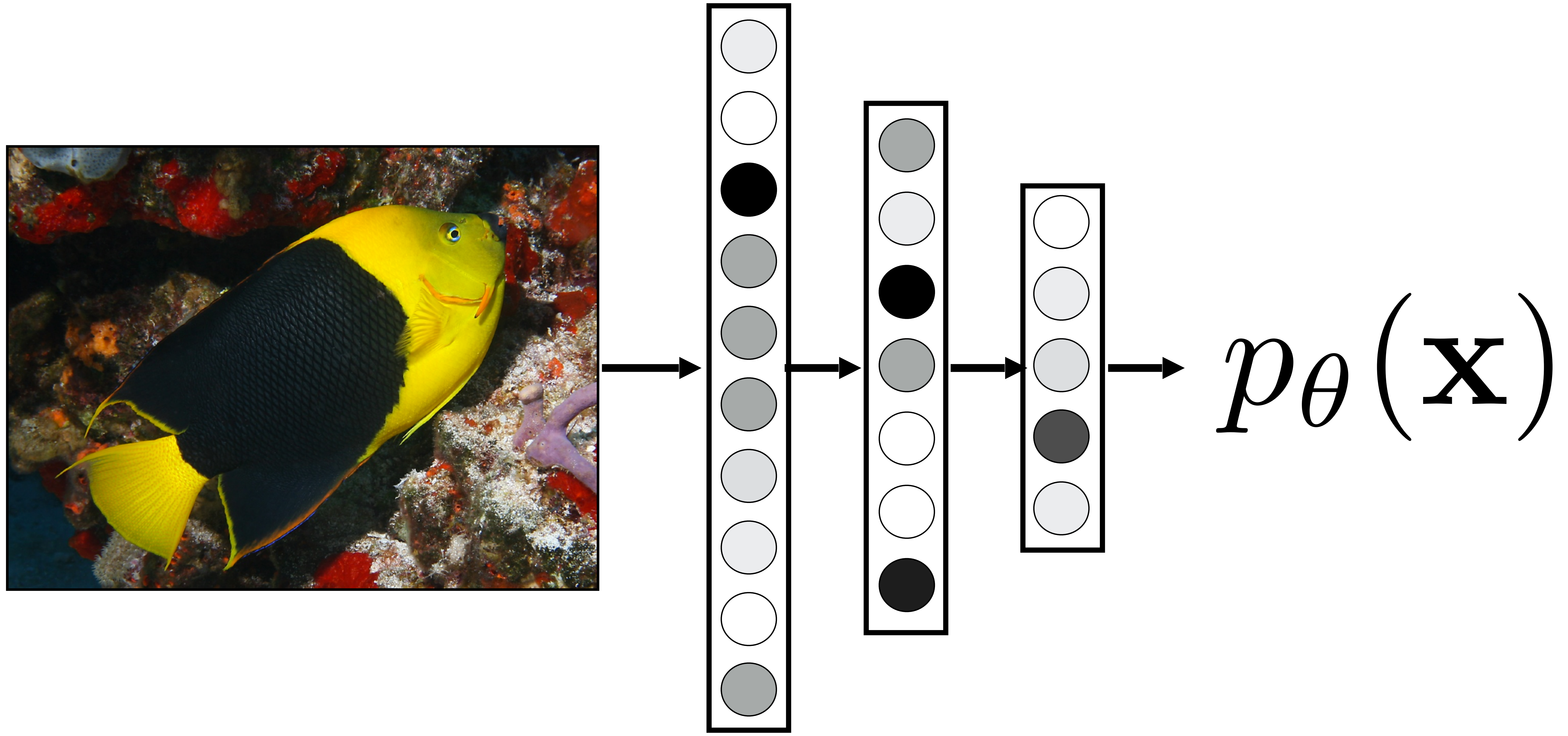
e.g. variational autoencoder

But isn't this a bit of a roundabout way of solving the problem?

Simpler idea: train network to output probabilities?



Simpler idea: train network to output probabilities?



What makes this idea challenging? Needs to sum to 1!

Energy-based model

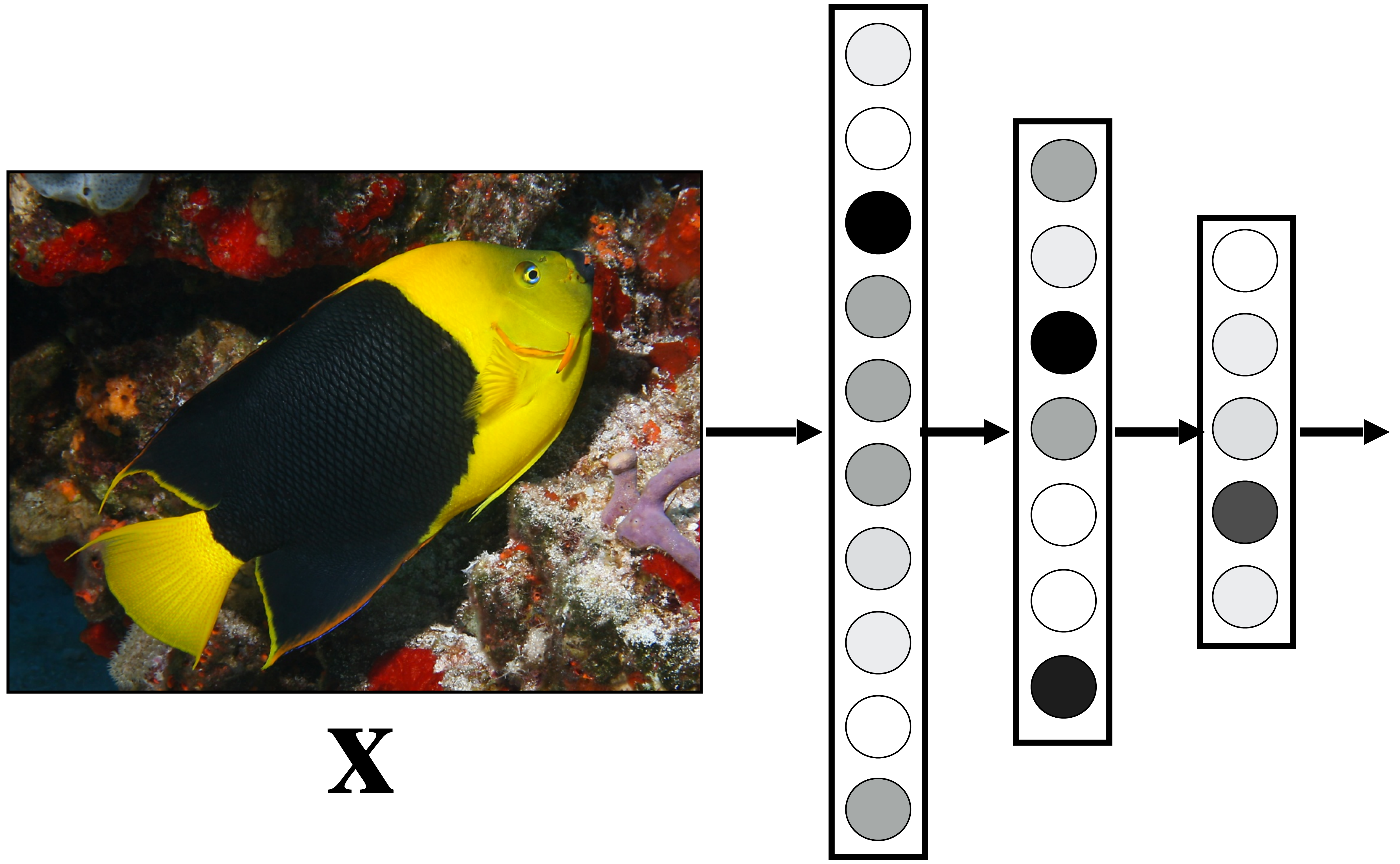
Energy function implemented by neural net, $E_\theta(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$

$$p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z(\theta)}$$

Normalization constant
a.k.a. *partition function*

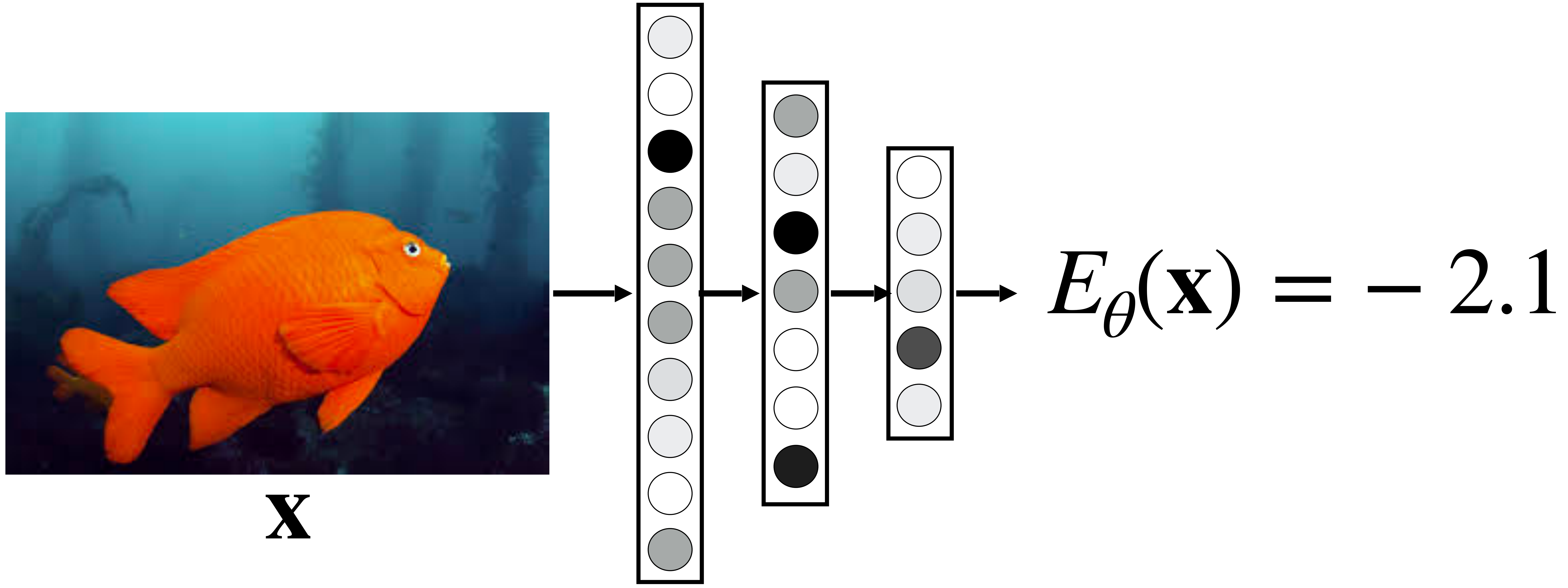
where $Z(\theta) = \int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}$

The energy function must assign energy to data points.

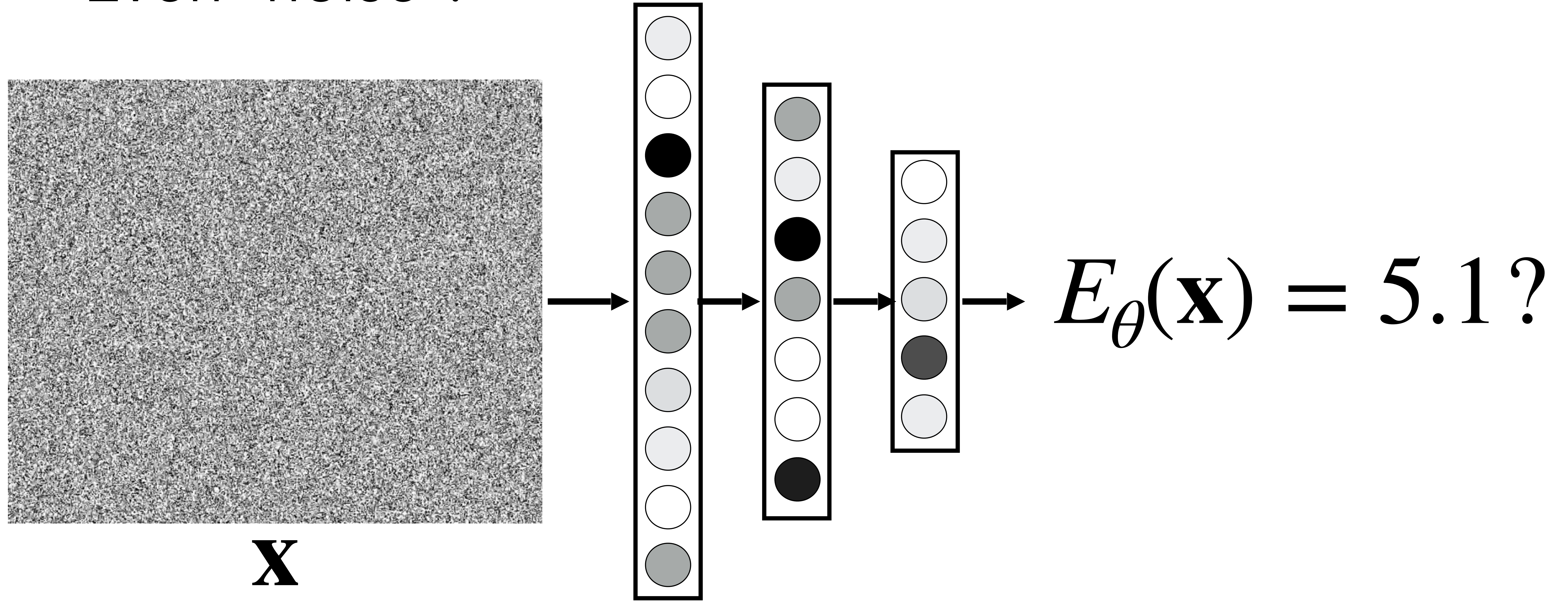


$$E_\theta(\mathbf{x}) = 0.1$$

The energy function must assign energy to data points.



The energy function must assign energy to data points.



When computing $Z(\theta)$, we need to integrate over the full input space $\mathbb{R}^d$, which makes it challenging.

Normalizing constant

$$Z(\theta) = \int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}$$

To compute $Z(\theta)$, we would need to integrate over the full input space $\mathbb{R}^d$! This would be very expensive.

What is (and isn't) tractable to compute?

$$p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z(\theta)}$$

$E_\theta(\mathbf{x}) \longrightarrow$ Yes, if E_θ is a neural net.

$Z(\theta) \longrightarrow$ No! Requires integrating over huge space.

$p_\theta(\mathbf{x}) \longrightarrow$ Only up to a constant factor (i.e., $Z(\theta)$)

What is (and isn't) tractable to compute?

Ratios? $$\frac{p_\theta(\mathbf{x}_1)}{p_\theta(\mathbf{x}_2)} = \frac{\frac{\exp(-E_\theta(\mathbf{x}_1))}{Z(\theta)}}{\frac{\exp(-E_\theta(\mathbf{x}_2))}{Z(\theta)}} = \exp(E_\theta(\mathbf{x}_2) - E_\theta(\mathbf{x}_1))$$

For example: how much more likely one example is than another?

Derivatives

Score function:

$$\nabla_{\mathbf{x}} \log(p_\theta(\mathbf{x})) = \nabla_{\mathbf{x}} \log\left(\frac{1}{Z(\theta)}\exp(-E_\theta(\mathbf{x}))\right)$$

$$= \nabla_{\mathbf{x}}\left[-\log(Z(\theta)) - E_\theta(\mathbf{x})\right] = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x})$$

Can use automatic differentiation if E_θ is a neural net.

Gradient of log likelihood:

$$\nabla_\theta \log(p_\theta(\mathbf{x})) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z(\theta)$$

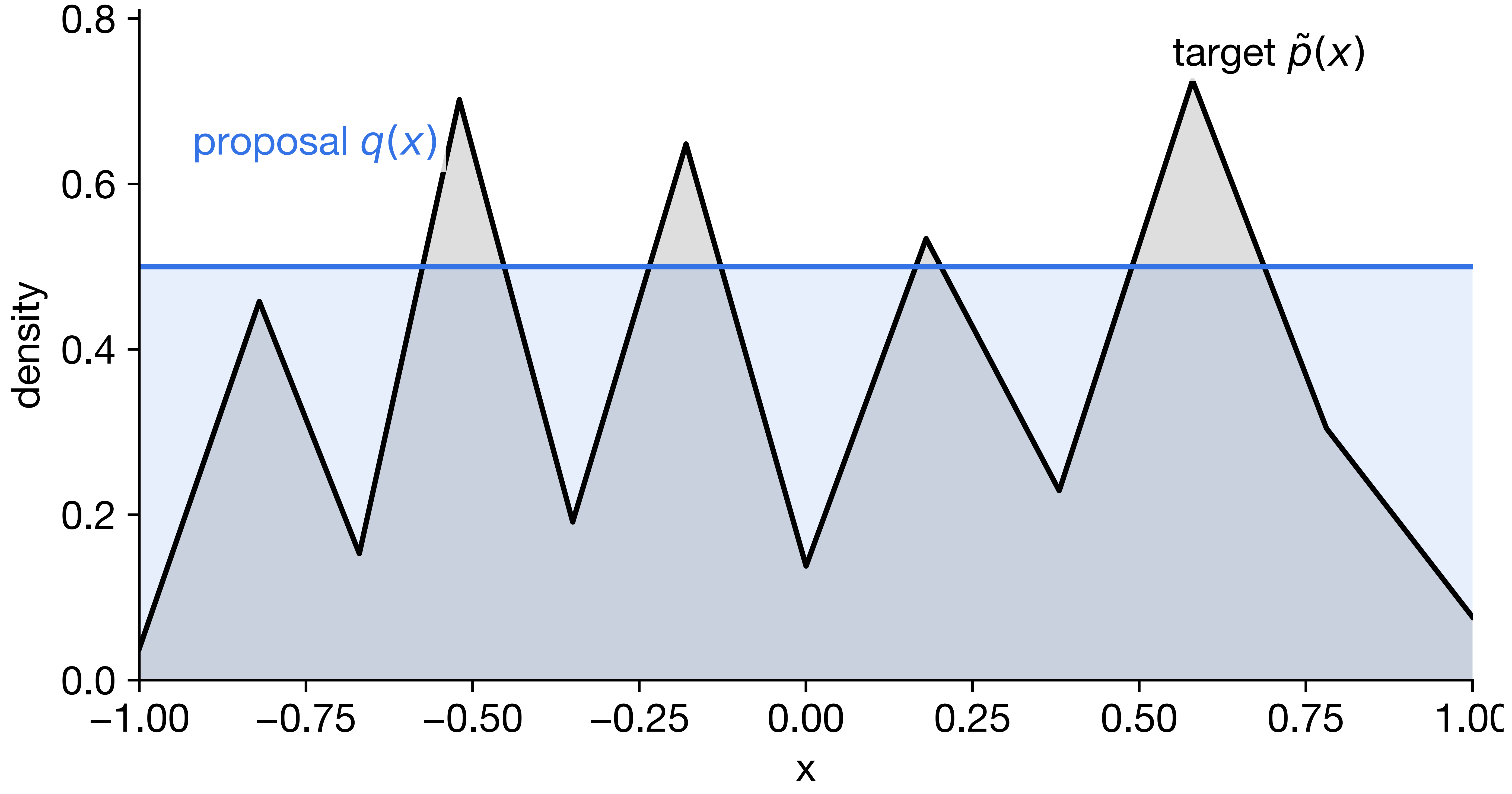
First term? Again, can use autodiff.
Second term? We'll come back to this…

Sampling from an EBM

- How do we draw a **random sample** from $p_\theta(\mathbf{x})$?
- Use *Monte Carlo* sampling methods
- To give you a feel for how these methods work, we'll go through a simple example: *rejection sampling*.

Example: Rejection sampling

Use a simple distribution q to propose samples, and accept them if they pass a test.



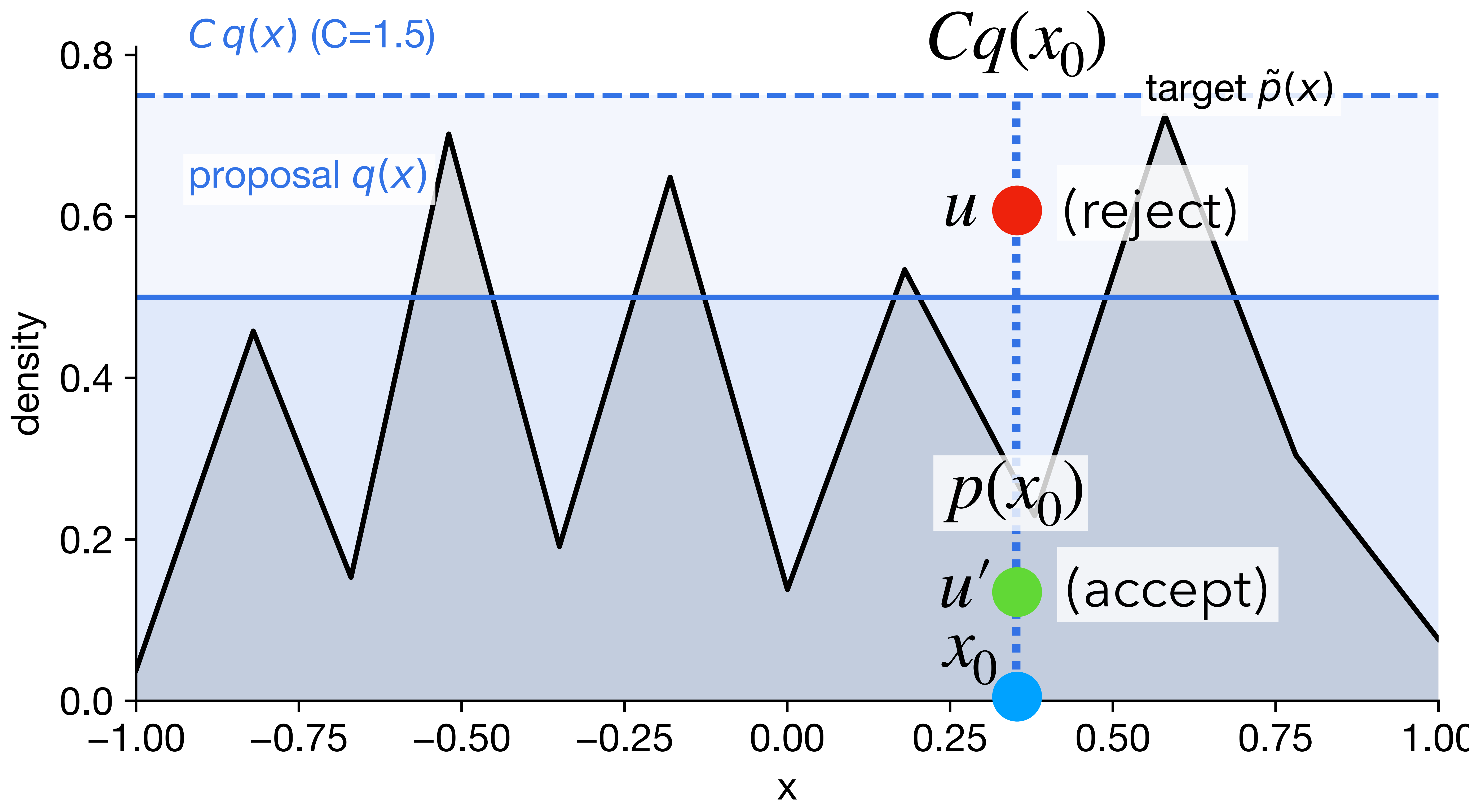
Want to sample from:

$$p(x) = \tilde{p}(x)/Z_p$$

$$\text{where } Z_p = \int_x \tilde{p}(x)dx$$

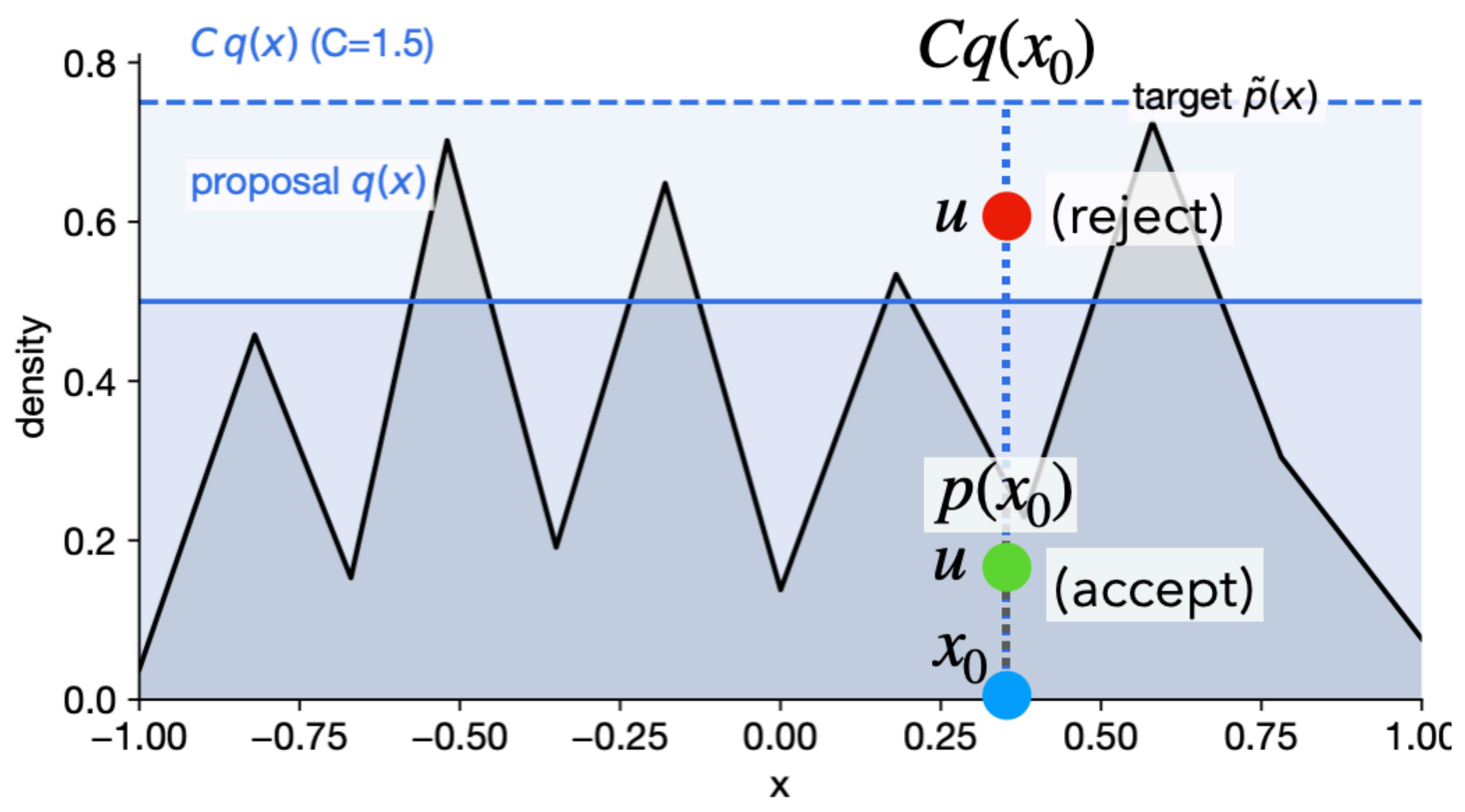
We'll use a *proposal distribution* $q(x)$ such that: $\tilde{p}(x) \leq Cq(x)$ for some constant C.

Example: Rejection sampling



1. Sample $x_0 \sim q(x)$ from proposal distribution.
2. Sample $u \sim U(0,\ Cq(x_0))$.
3. If $u > p(x_0)$, then reject the sample. Otherwise, we'll *accept* x_0 as our sample.

Example: Rejection sampling



Why does this work?

$$P(x \text{ proposed \& accepted}) = q(x_0 = x)P(\text{accept} \mid x_0 = x)$$

$$= q(x)\frac{\tilde{p}(x)}{Cq(x)} = \frac{\tilde{p}(x)}{C}$$

$$P(\text{accepted}) = \int_x P(x \text{ proposed \& accepted})dx$$

$$= \int_x \frac{\tilde{p}(x)}{C}dx = \frac{Z_p}{C}$$

$$P(x \mid \text{accepted}) = \frac{P(x \text{ proposed \& accepted})}{P(\text{accepted})} = \frac{\tilde{p}(x)}{C}\frac{C}{Z_p} = p(x)$$

Problem: inefficient! If C is too high, it will rarely accept.

Sampling using Langevin dynamics

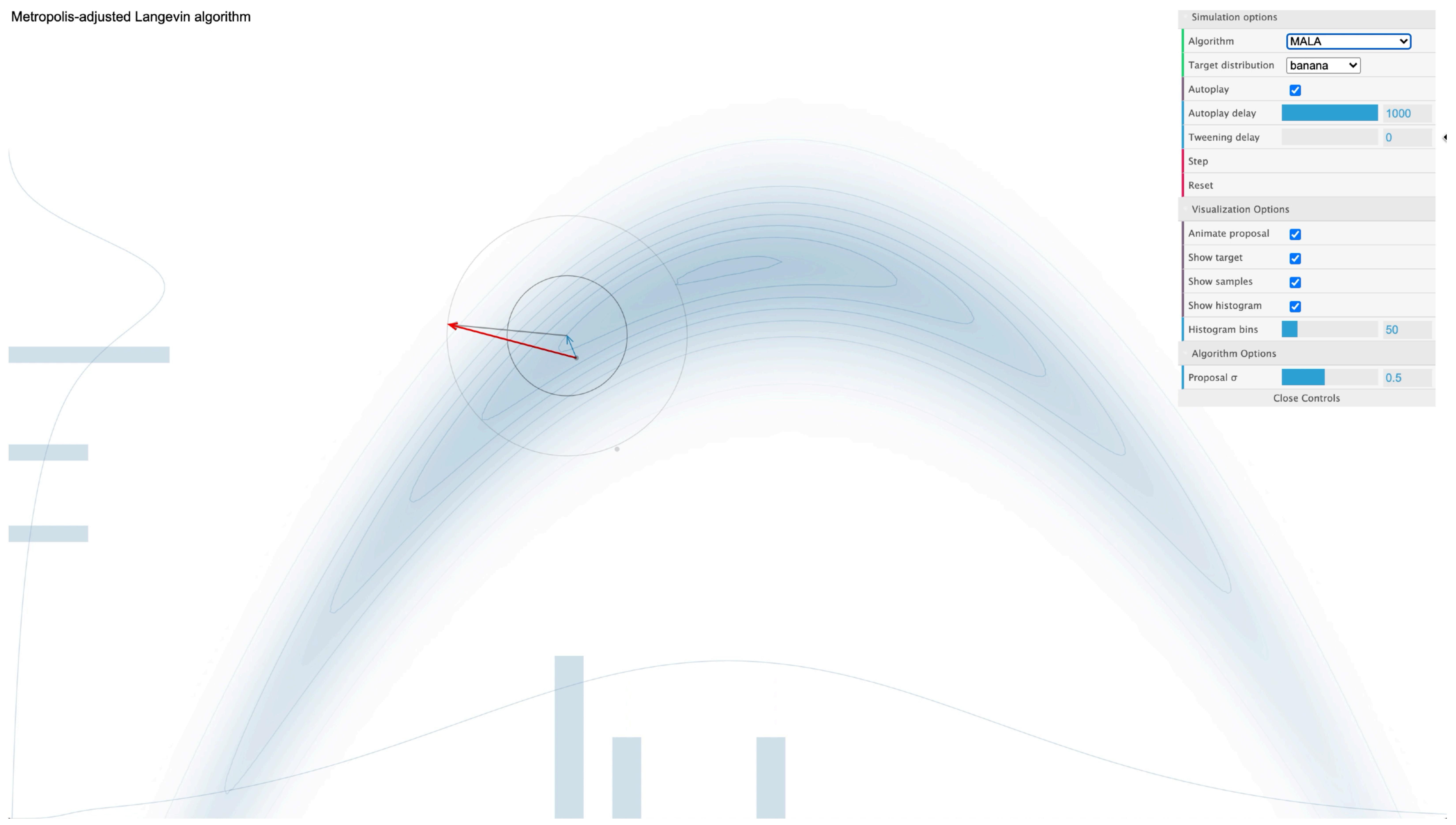
- A Monte Carlo method that exploits gradients.
- It is a *Markov Chain Monte Carlo* (MCMC) method. It keeps a state of previous proposals and transitions to new ones (we won't cover the details of MCMC in this class, but you can see the textbook for details).
- Start with a random noise example $\mathbf{x}_0 \sim U(0,1)$
- Update the random noise by taking a gradient step and adding noise.

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\epsilon^2}{2} \nabla_{\mathbf{x}} \log(p_\theta(\mathbf{x}_t)) + \epsilon \mathbf{z}, \quad \mathbf{z} \sim N(0, I)$$

where α controls the step size. Recall: $\nabla_{\mathbf{x}} \log(p_\theta(\mathbf{x}_t)) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x})$

- If there were no random noise, what would happen?

Langevin MCMC



Metropolis-adjusted Langevin algorithm

Simulation options

- Algorithm: MALA
- Target distribution: banana
- Autoplay: ✓
- Autoplay delay: 1000
- Tweening delay: 0
- Step
- Reset

Visualization Options

- Animate proposal: ✓
- Show target: ✓
- Show samples: ✓
- Show histogram: ✓
- Histogram bins: 50

Algorithm Options

- Proposal σ: 0.5

Close Controls

Source: https://chi-feng.github.io/mcmc-demo/app.html?algorithm=MALA&target=banana

Training an EBM

Maximum likelihood

Want to maximize:

$$\mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\log p_\theta(\mathbf{x})\right] \approx \frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{\exp(-E_\theta(\mathbf{x}_i))}{Z(\theta)}\right)$$

What if we use gradient ascent? Recall:

$$\nabla_\theta \log(p_\theta(\mathbf{x})) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z(\theta)$$

Maximum likelihood

$$\nabla_\theta \log(p_\theta(\mathbf{x})) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z(\theta)$$

Problem: The normalizing constant depends on θ, and it's hard to compute it (or its derivative) directly.

$$\log Z(\theta) = \log \int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x} \qquad \nabla_\theta \log Z(\theta) = ?$$

Computing the gradient

$$\nabla_\theta \log Z(\theta) = \nabla_\theta \log \int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x} \;= \frac{\nabla_\theta \int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}}{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}}$$ (chain rule)

$$= \frac{\int_{\mathbf{x}} \nabla_\theta \exp(-E_\theta(\mathbf{x}))d\mathbf{x}}{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}} = \frac{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))(-\nabla_\theta E_\theta(\mathbf{x}))d\mathbf{x}}{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}}$$ (chain rule)

$$= \int_{\mathbf{x}} \frac{1}{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))d\mathbf{x}} \exp(-E_\theta(\mathbf{x}))(-\nabla_\theta E_\theta(\mathbf{x}))d\mathbf{x}$$

$$= \int_{\mathbf{x}} \frac{1}{Z(\theta)} \exp(-E_\theta(\mathbf{x}))(-\nabla_\theta E_\theta(\mathbf{x}))d\mathbf{x} \;= \int_{\mathbf{x}} p_\theta(\mathbf{x})(-\nabla_\theta E_\theta(\mathbf{x}))d\mathbf{x}$$ (definition of EBM)

$$= \mathbb{E}_{x\sim p_\theta(\mathbf{x})}\left[-\nabla_\theta E_\theta(\mathbf{x})\right]$$ (rewrite as an expected value)

Maximum likelihood estimation via sampling

Through algebraic manipulation, we found:

$$\nabla_\theta \log Z(\theta) = \mathbb{E}_{\mathbf{x}\sim p_\theta}\left[-\nabla_\theta E_\theta(\mathbf{x})\right] \approx -\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(\mathbf{x}_i)$$

→ Sample a batch of N examples, and average the gradient!

This is convenient, since we already know how to draw samples.

$\nabla_\theta \log Z(\theta) \approx -\nabla_\theta E_\theta(\tilde{\mathbf{x}})$ where $\tilde{\mathbf{x}}$ is an Monte Carlo sample

Maximum likelihood estimation via sampling

Putting these equations together:

$$\nabla_\theta \log(p_\theta(\mathbf{x})) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z(\theta)$$

$\nabla_\theta \log Z(\theta) \approx -\nabla_\theta E_\theta(\tilde{\mathbf{x}})$ where $\tilde{\mathbf{x}}$ is a Monte Carlo sample

Learning rule:

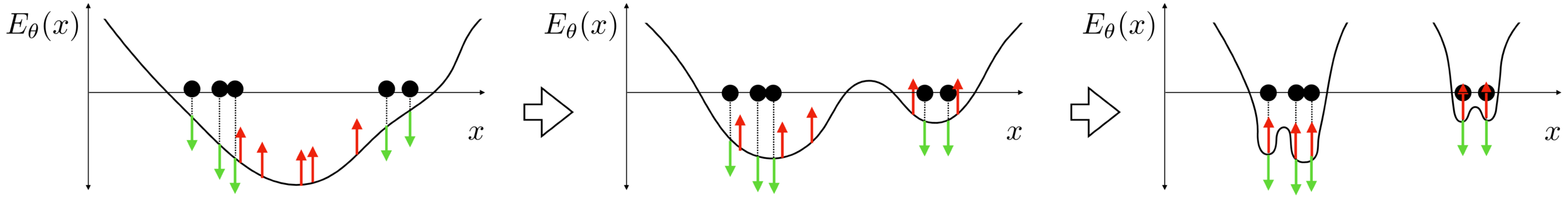
$$\nabla_\theta \log(p_\theta(\mathbf{x})) \approx -\nabla_\theta E_\theta(\mathbf{x}) + \nabla_\theta E_\theta(\tilde{\mathbf{x}})$$

Contrastive divergence

Learning rule:

$$\nabla_\theta \log(p_\theta(\mathbf{x})) \approx -\nabla_\theta E_\theta(\mathbf{x}) + \nabla_\theta E_\theta(\tilde{\mathbf{x}})$$

Contrastive divergence [Hinton 2002]: instead of sampling $\tilde{\mathbf{x}}$ from scratch, which is slow, initialize the the sampler using our training example, $\mathbf{x}$. Typically only run the chain for a few iterations.



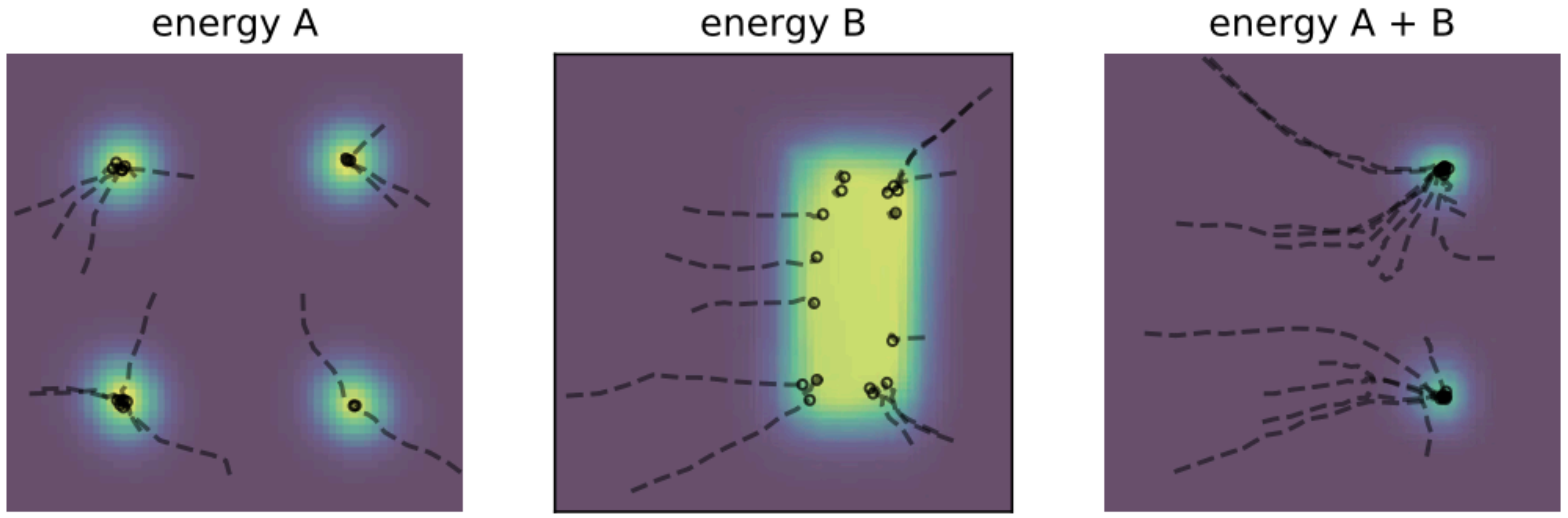
Each iteration, decrease energy for dataset examples, increase energy for samples from the model.

Figure source: Torralba et al. FCV book and [Hinton, 2002]

Sampling from an EBM

Using extended version of CD [Nijkamp et al., 2019]

Combining multiple EBMs together

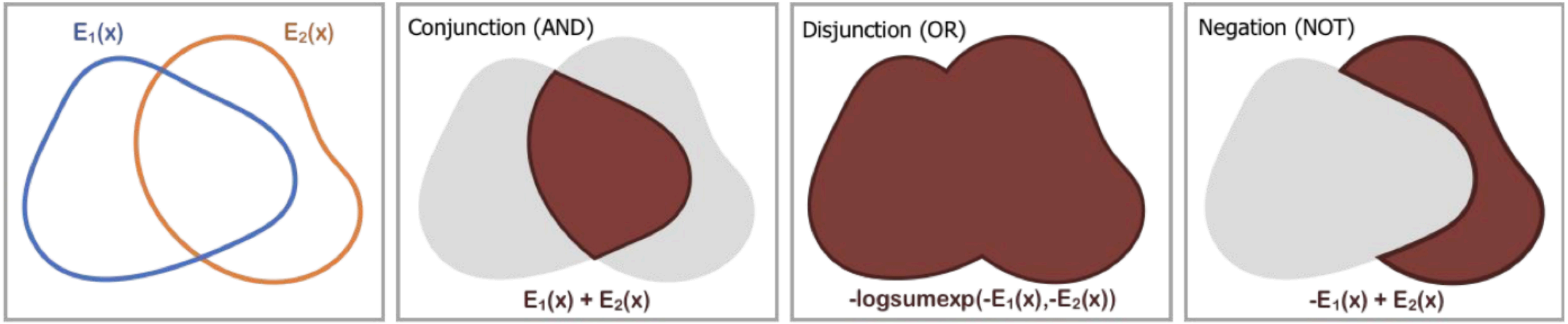


Product of experts: $$p_{12}(\mathbf{x}) = \frac{1}{Z_{12}} p_1(\mathbf{x}) p_2(\mathbf{x})$$

Implemented as two EBMs: $$E_{12}(\mathbf{x}) = E_{\theta_1}(\mathbf{x}) + E_{\theta_2}(\mathbf{x})$$

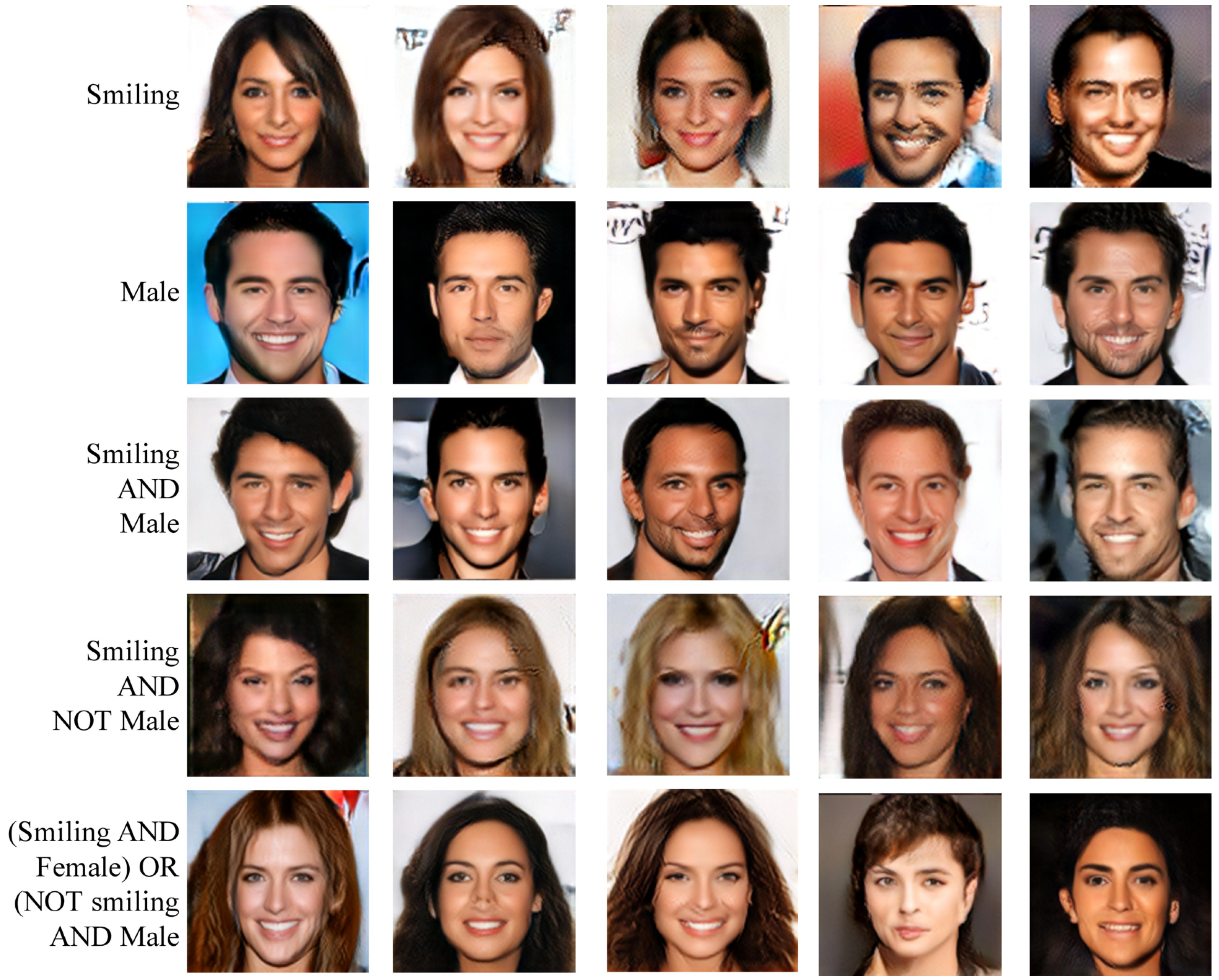
Source: Du et al. via Murphy PMLAT

Combining multiple EBMs together



Source: [Du et al., 2020]

Combining multiple EBMs together



Source: [Du et al., 2020]

Energy-based models

+ Very flexible. Define $E_\theta(\, . \,)$ architecture however you'd like!
+ Easy to estimate (unnormalized) probabilities.
+ Simple idea. No encoder, latent code (unlike VAE)

– Hard to generate samples.
– Hard to train.
– No latent code (which is useful for some applications)

Score-based models

What if we directly estimate the score?

- Can we directly learn $s_\theta(\mathbf{x}) = \nabla_\theta \log(p_\theta(\mathbf{x}))$?
- Don't need to directly model $Z(\theta)$, since $\nabla_{\mathbf{x}} \log(p_\theta(\mathbf{x}_t)) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x})$.
- And we could plug it into Langevin dynamics:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\epsilon^2}{2} \nabla_{\mathbf{x}} \log(p_\theta(\mathbf{x}_t)) + \epsilon \mathbf{z}, \qquad \mathbf{z} \sim N(0, I)$$

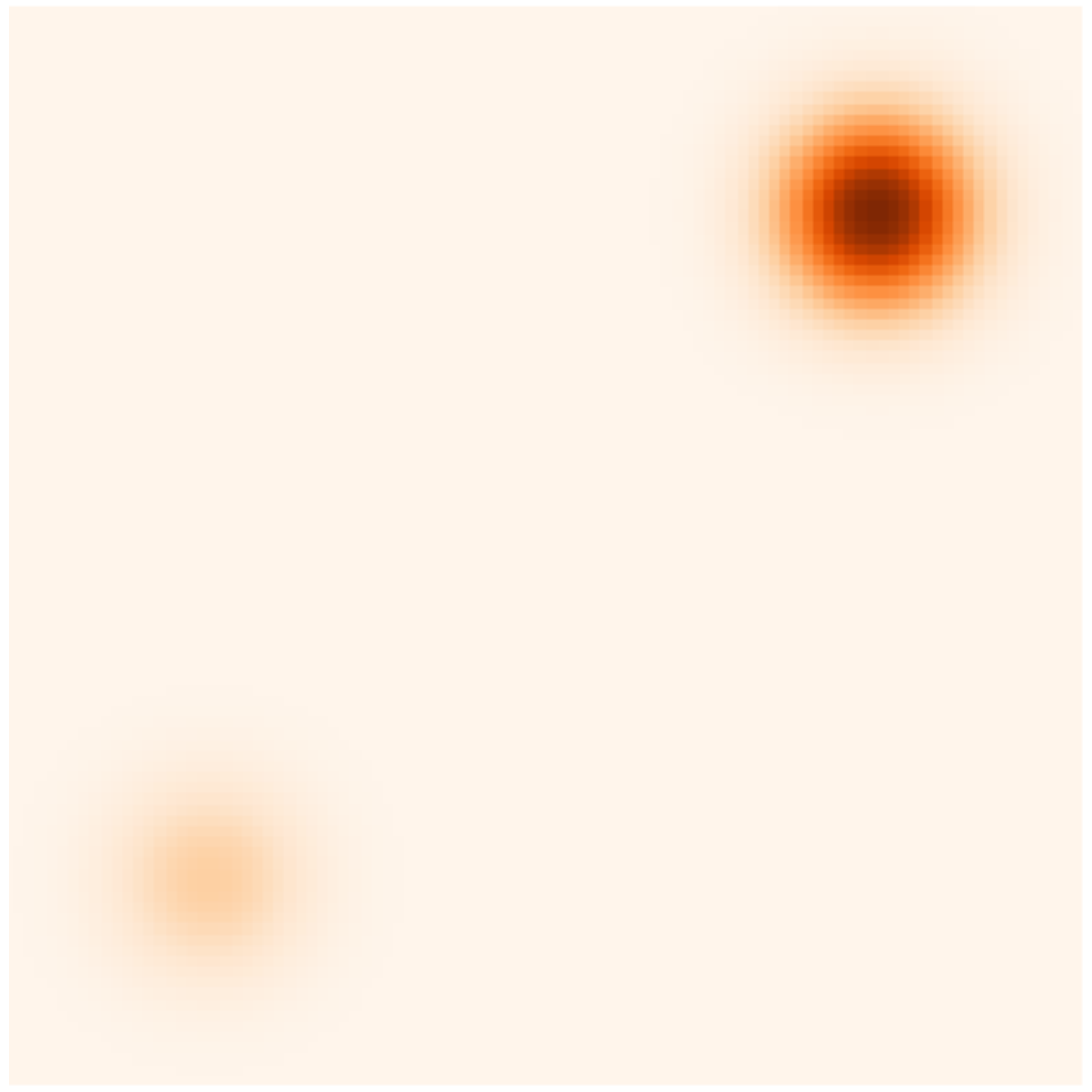
$$\Downarrow$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\epsilon^2}{2} s_\theta(\mathbf{x}) + \epsilon \mathbf{z},$$

Score estimation by training score-based models

Probability density

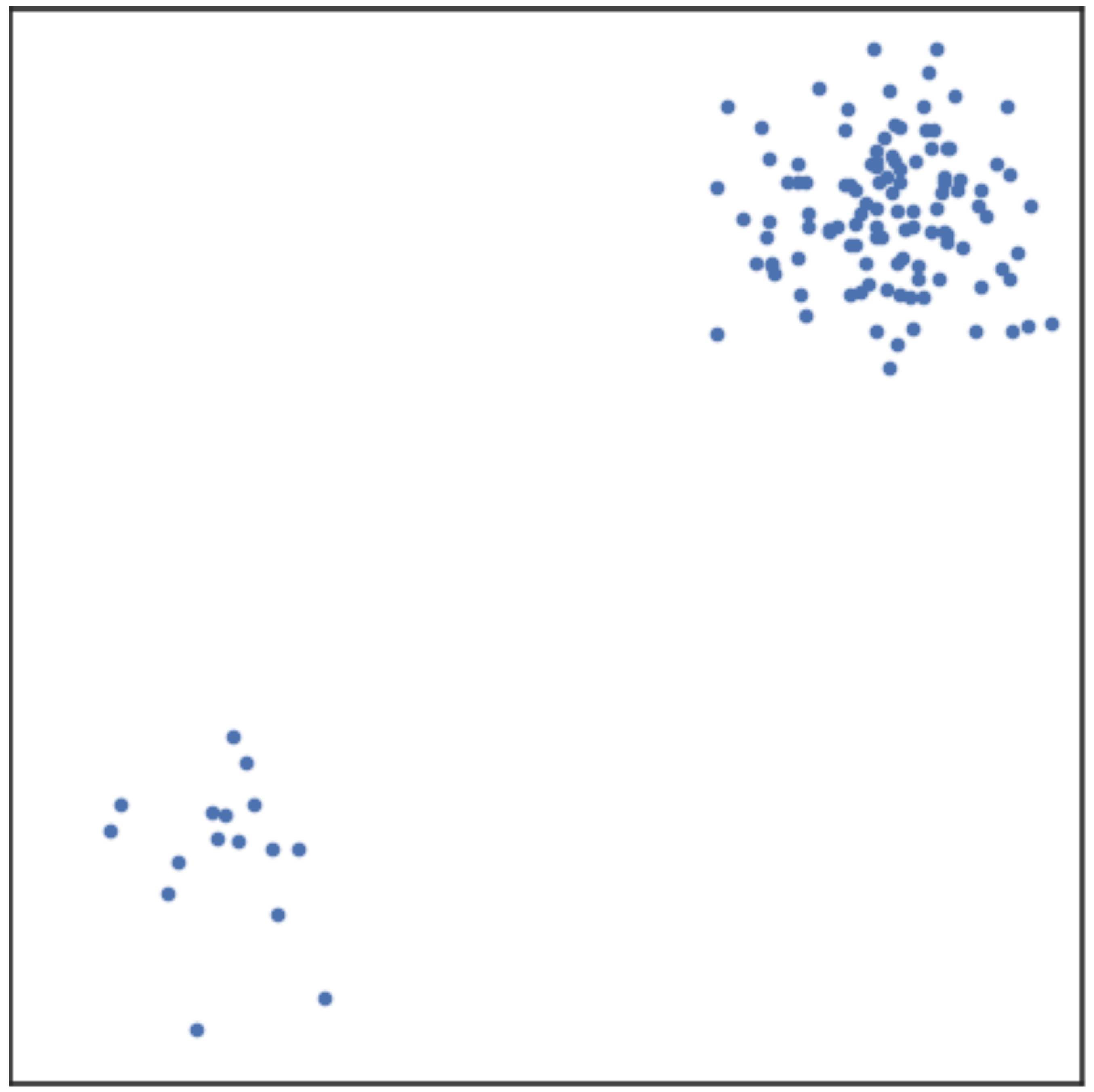
$p_{\text{data}}(\mathbf{x})$



$\rightarrow$

i.i.d. samples

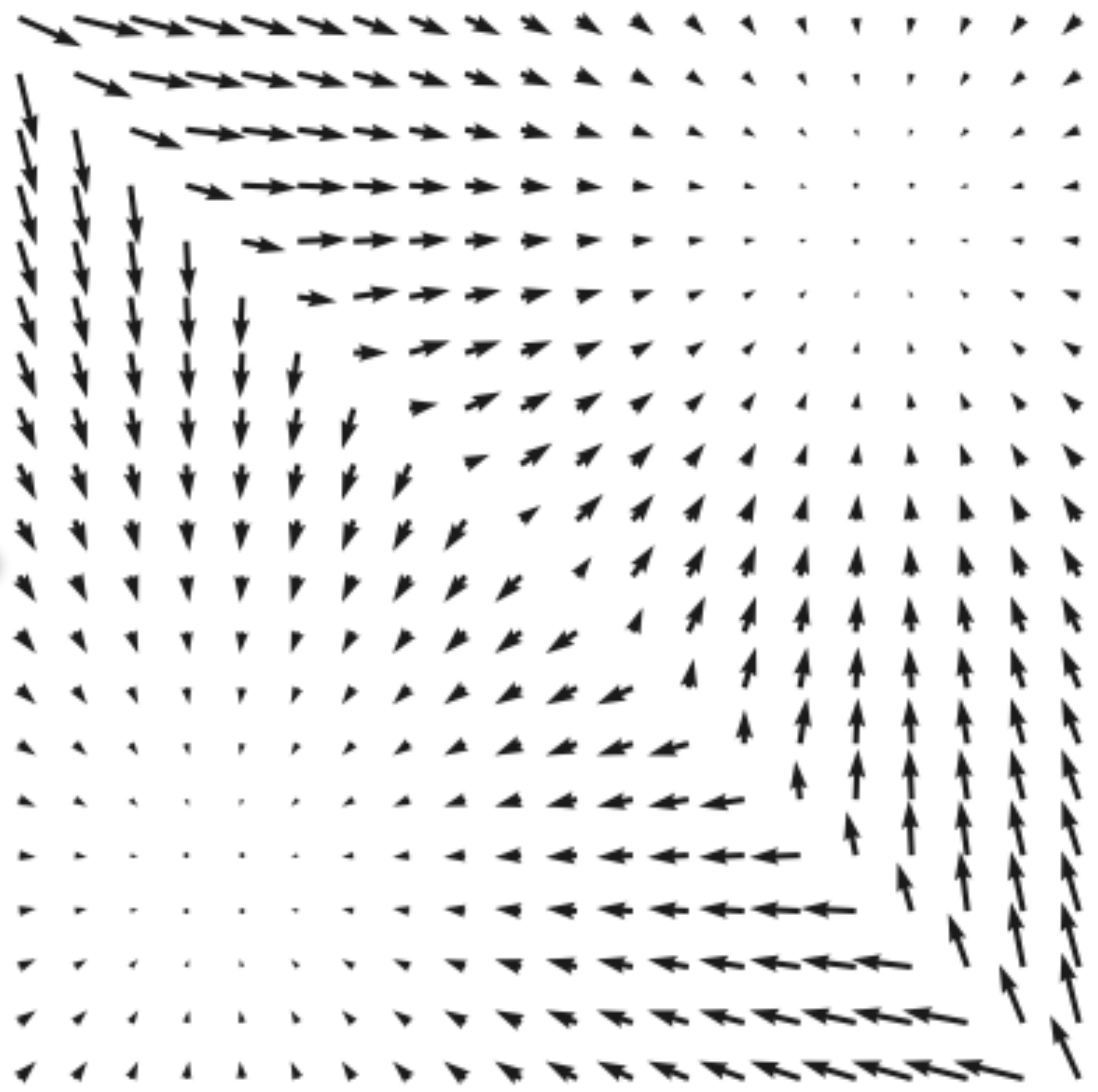
$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$



$\overset{?}{\rightarrow}$

Score function

$\boldsymbol{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$



Slide Source: Stefano Ermon

What if we directly estimate the score?

How does true distribution change when you change $\mathbf{x}$?

Score function for our learned distribution (implemented as $s_\theta(\mathbf{x})$).

Want to minimize *Fisher divergence*:

$$D_F(p_D(\mathbf{x}) \parallel p_\theta(\mathbf{x})) = \mathbb{E}_{\mathbf{x}\sim p_D}\left[\frac{1}{2}\left\| \nabla_{\mathbf{x}} \log p_D(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) \right\|^2\right]$$

where p_D is the true distribution. However, we don't know $\nabla_{\mathbf{x}} \log(p_D(\mathbf{x}))$!

Many approaches to address this (see textbook for details). We'll do a very brief overview of a few approaches, without going into much technical detail.

Using 2nd derivatives

We don't know $\nabla_{\mathbf{x}} \log p_D(\mathbf{x})$. But we can place constraints on 2nd derivatives. You can show [Hyvärinen, 2005]:

$$D_F(p_D(\mathbf{x}) \parallel p_\theta(\mathbf{x})) = \mathbb{E}_{\mathbf{x} \sim p_D}\left[\frac{1}{2} \parallel \nabla_{\mathbf{x}} \log p_D(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) \parallel^2\right]$$

$$= \mathbb{E}_{\mathbf{x} \sim p_D}\left[\frac{1}{2}\sum_{i=1}^{d}\left(\frac{\partial E_\theta(\mathbf{x})}{\partial x_i}\right)^2 - \frac{\partial^2 E_\theta(\mathbf{x})}{\partial x_i^2}\right]$$

after some algebra and calculus identities

$$= \mathbb{E}_{\mathbf{x} \sim p_D}\left[\frac{1}{2}\|s_\theta(\mathbf{x})\|^2 + \mathrm{tr}(\nabla s_\theta(\mathbf{x}))\right]$$

only have to take *1st* derivatives of $s_\theta(\mathbf{x})$

Where do 2nd derivatives come from? It comes from integration by parts and lots of algebra.

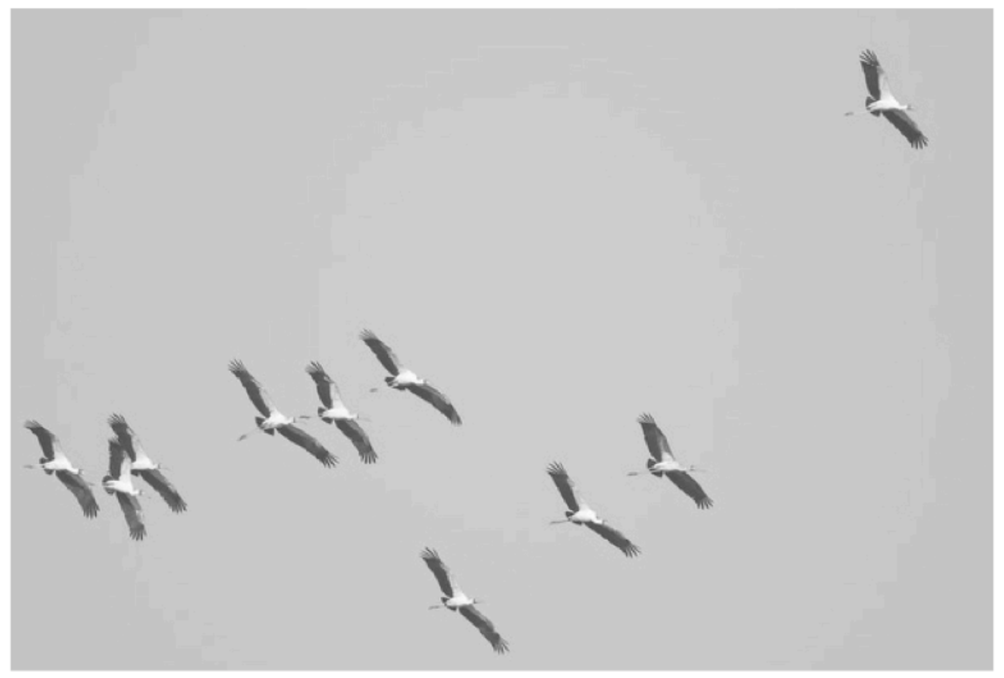
Denoising score matching

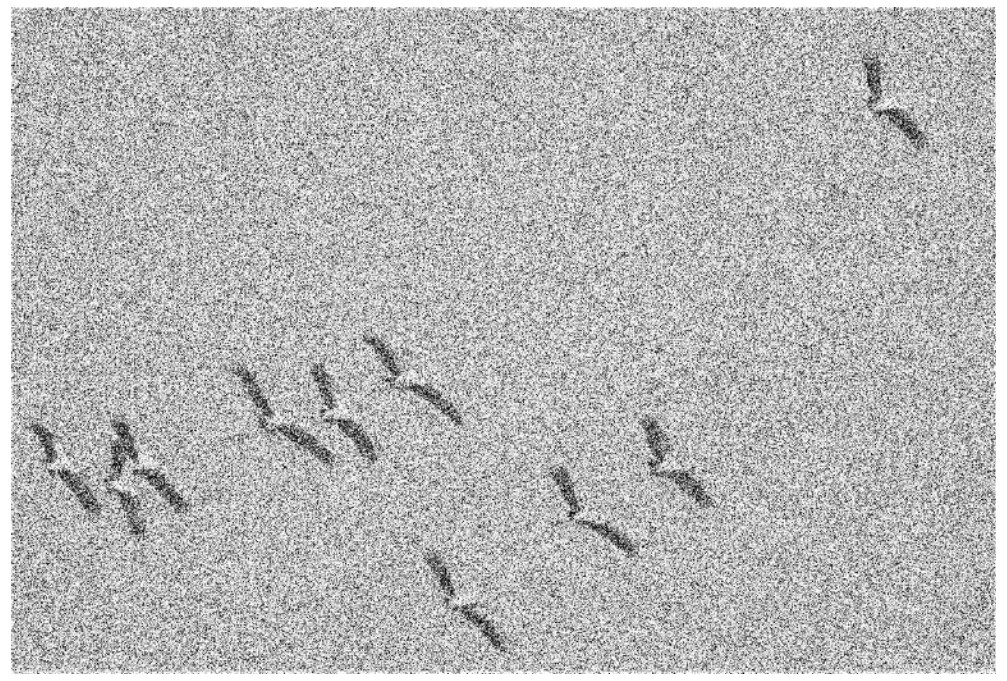
Idea #2: It's easier to use the score function for *noisy* inputs.

Suppose $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$ Then $q(\tilde{\mathbf{x}}) = \int_{\mathbf{x}} \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 I) p_D(\mathbf{x}) d\mathbf{x}$

When conditioned on an example $\mathbf{x}$, the score function has a simple form:

$$\nabla_{\tilde{\mathbf{x}}} \log q(\tilde{\mathbf{x}} \mid \mathbf{x}) = \nabla_{\tilde{\mathbf{x}}} \log \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 I) = -\frac{(\tilde{\mathbf{x}} - \mathbf{x})}{\sigma^2}$$


$\mathbf{x} \sim p_D(\mathbf{x})$


$\tilde{\mathbf{x}} \sim q(\tilde{\mathbf{x}} \mid \mathbf{x})$

Denoising score matching

Idea: It's easier to use the score function for *noisy* inputs.

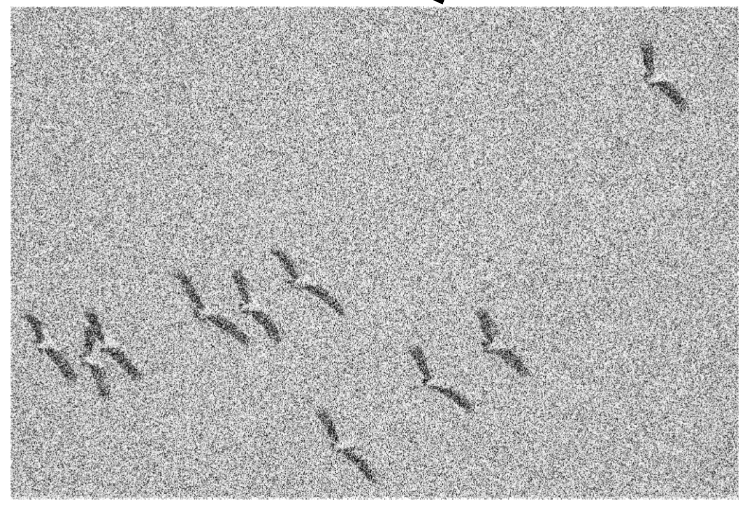
$$\nabla_{\tilde{\mathbf{x}}} \log q(\tilde{\mathbf{x}} \mid \mathbf{x}) = \nabla_{\tilde{\mathbf{x}}} \log \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 I) = -\frac{(\tilde{\mathbf{x}} - \mathbf{x})}{\sigma^2}$$

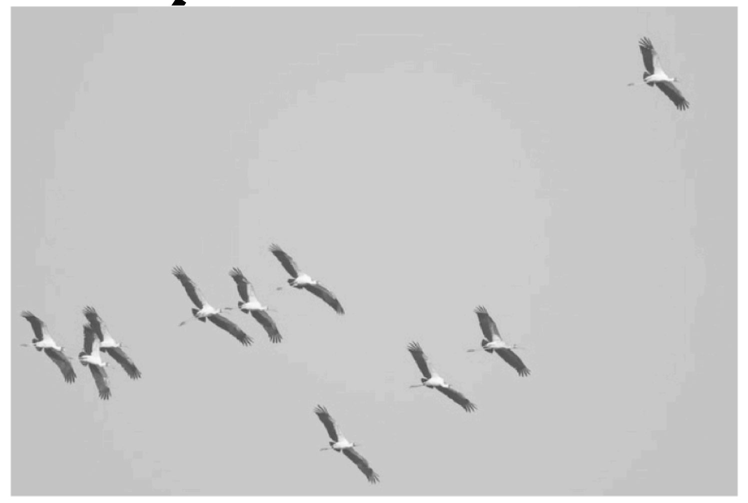
We can take advantage of this to define a simpler loss function (sketch):

$$D_F(q(\tilde{\mathbf{x}}) \parallel p_\theta(\tilde{\mathbf{x}})) = \mathbb{E}_{q(\tilde{\mathbf{x}})}\left[\frac{1}{2} \left\| \nabla_{\tilde{\mathbf{x}}} \log p_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} q(\tilde{\mathbf{x}}) \right\|^2\right]$$

$$= \mathbb{E}_{q(\tilde{\mathbf{x}}|\mathbf{x}) p_D(\mathbf{x})}\left[\frac{1}{2} \left\| \nabla_{\tilde{\mathbf{x}}} \log p_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} q(\tilde{\mathbf{x}} \mid \mathbf{x}) \right\|^2\right] + \text{constant}$$

add noise to each example





Denoising score matching

Idea: It's easier to use the score function for *noisy* inputs.

$$\nabla_{\tilde{\mathbf{x}}} \log q(\tilde{\mathbf{x}} \mid \mathbf{x}) = \nabla_{\tilde{\mathbf{x}}} \log \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 I) = -\frac{(\tilde{\mathbf{x}} - \mathbf{x})}{\sigma^2}$$

We can take advantage of this to rewrite the loss function (sketch):

$$D_F(q(\tilde{\mathbf{x}}) \parallel p_\theta(\tilde{\mathbf{x}})) = \mathbb{E}_{q(\tilde{\mathbf{x}})}\left[\frac{1}{2} \parallel \nabla_{\tilde{\mathbf{x}}} \log p_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} q(\tilde{\mathbf{x}}) \parallel^2\right]$$

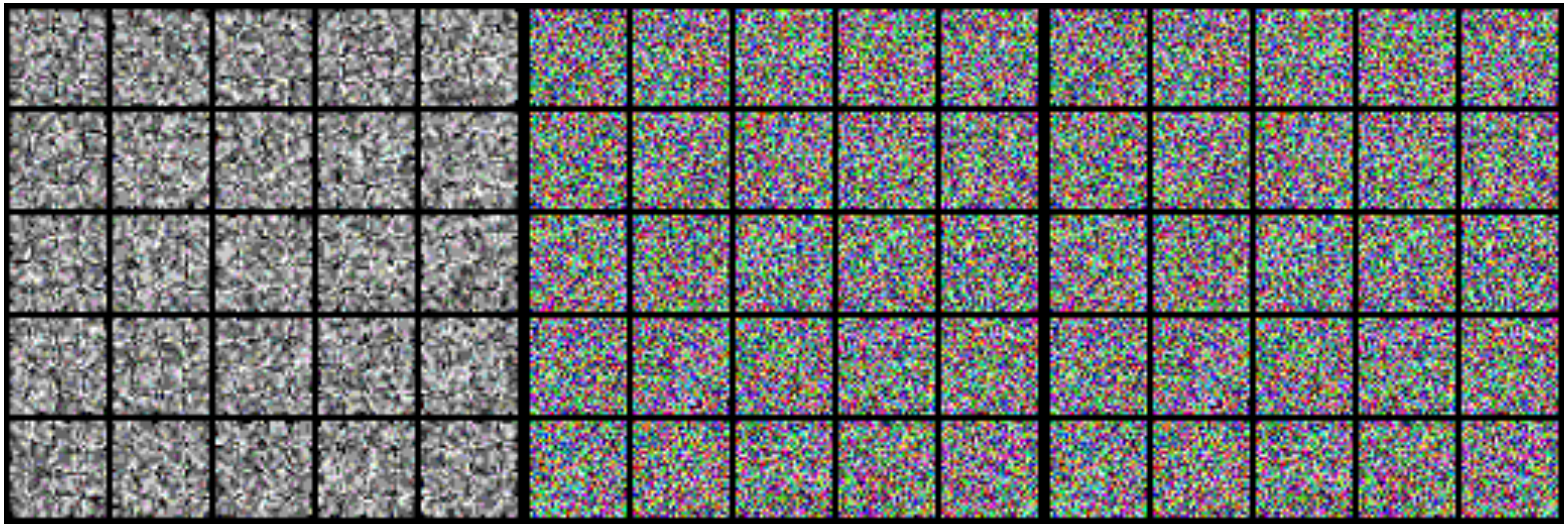
$$= \mathbb{E}_{q(\tilde{\mathbf{x}}|\mathbf{x})p_D(\mathbf{x})}\left[\frac{1}{2} \parallel \nabla_{\tilde{\mathbf{x}}} \log p_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} q(\tilde{\mathbf{x}} \mid \mathbf{x}) \parallel^2\right] + \text{constant}$$

add noise to each example

$$= \mathbb{E}_{q(\tilde{\mathbf{x}}|\mathbf{x})p_D(\mathbf{x})}\left[\frac{1}{2}\left\| s_\theta(\tilde{\mathbf{x}}) - \frac{(\mathbf{x} - \tilde{\mathbf{x}})}{\sigma^2} \right\|^2\right] + \text{constant}$$

plug in formula for score of noisy data

Sampling from a trained score matching model



[Song & Ermon, “Generative Modeling by Estimating Gradients of the Data Distribution”, 2019]

Samples from denoising score matching

Samples from [Song & Ermon 2021]

Score-based models

- Model the score function of an EBM $s_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p(\mathbf{x})$
- Designed for easy sampling
- Avoids modeling the normalizing constant $Z(\theta)$
- Doesn't explicitly model $p_\theta(\mathbf{x})$ or $E_\theta(\mathbf{x})$.
- Many ways to train them, including by denoising.

Next class: diffusion models