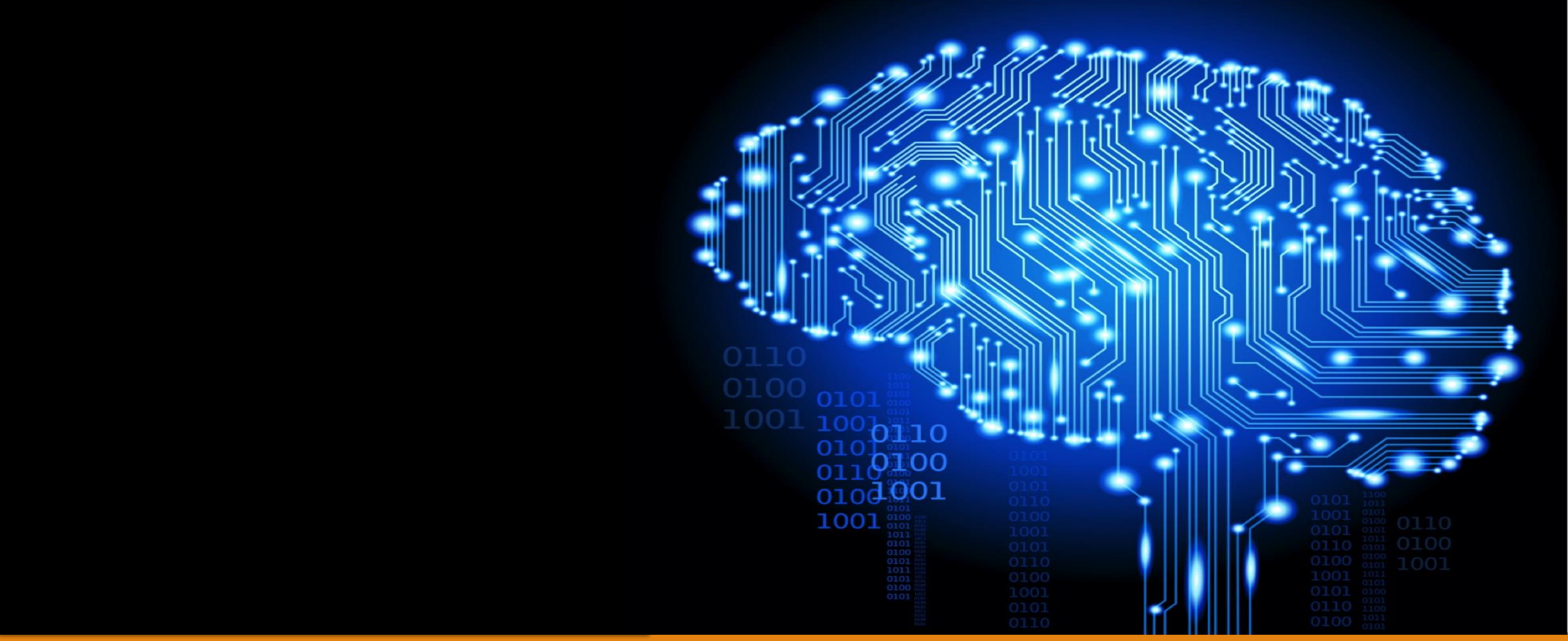


Lecture 8: Deep Generative & Energy Models

Efstratios Gavves

Discriminative models: summary

- So far we have explored discriminative models mainly
- Given an individual input $\boldsymbol{x}$ predict
  - the correct label (classification)
  - the correct score (regression)
- Learning by maximizing the probability of individual classifications/regressions

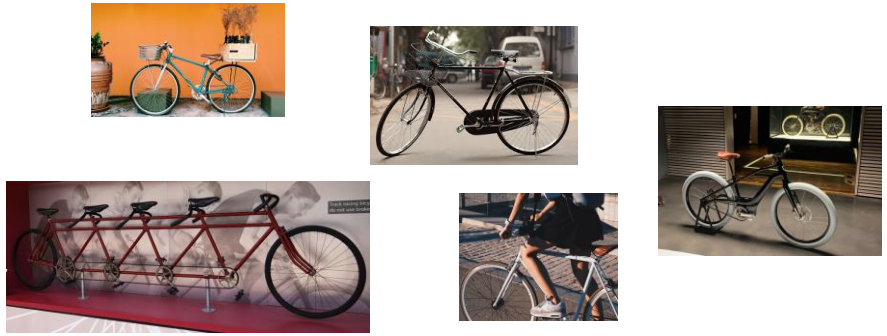
Prediction: "bicycle"

Prediction: "8.7" on IMDb

Generative models: main idea

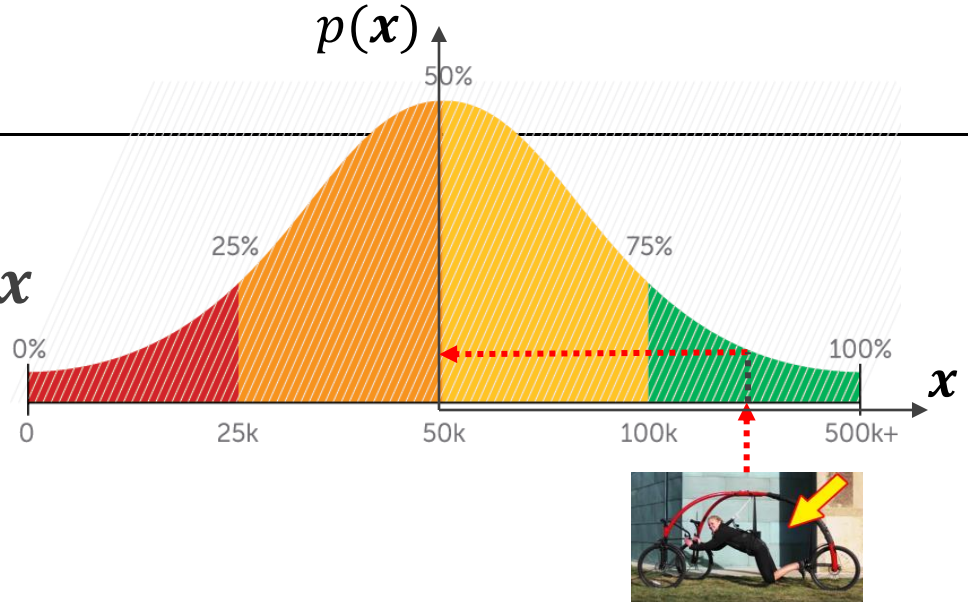
- Discriminative learning does not model data jointly
- Rephrasing: we want to know what is the distribution of data
- For instance: we want to know how likely is $\boldsymbol{x}_a$
  - Or if it is more likely than $\boldsymbol{x}_b$

Our observations

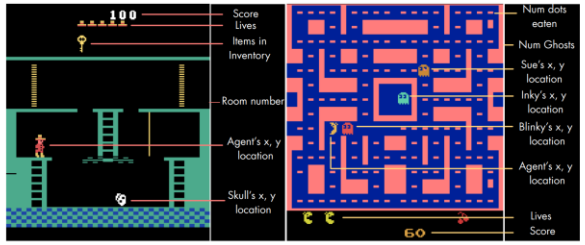


"What are the chances this is a bicycle"?

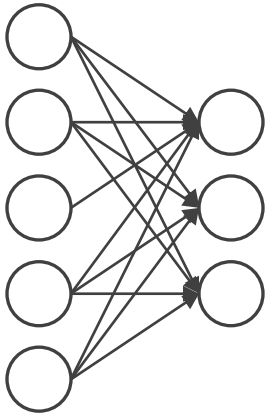
Why/when to learn a distribution?



- Density estimation: estimate the probability of $\boldsymbol{x}$
- Sampling: generate new plausible $\boldsymbol{x}$
  - *E.g.,* model-based reinforcement learning



- Structure/representation learning: learn good features of $\boldsymbol{x}$ <u>unsupervised</u>

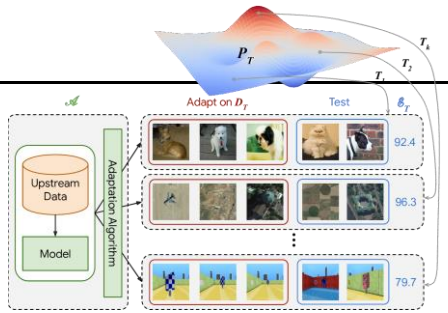


"Pose feature"

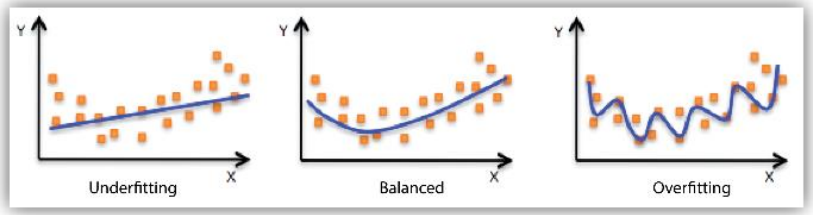
"Color feature"

"…"

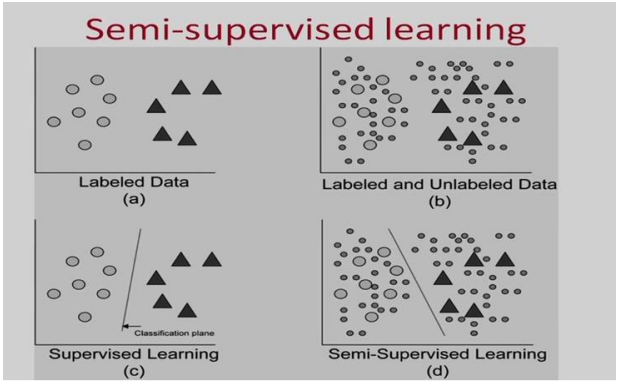
Why/when to learn a distribution?



- Generative models to pretrain for downstream tasks
- Generative models to ensure generalization
  - *E.g.,* model-based reinforcement learning



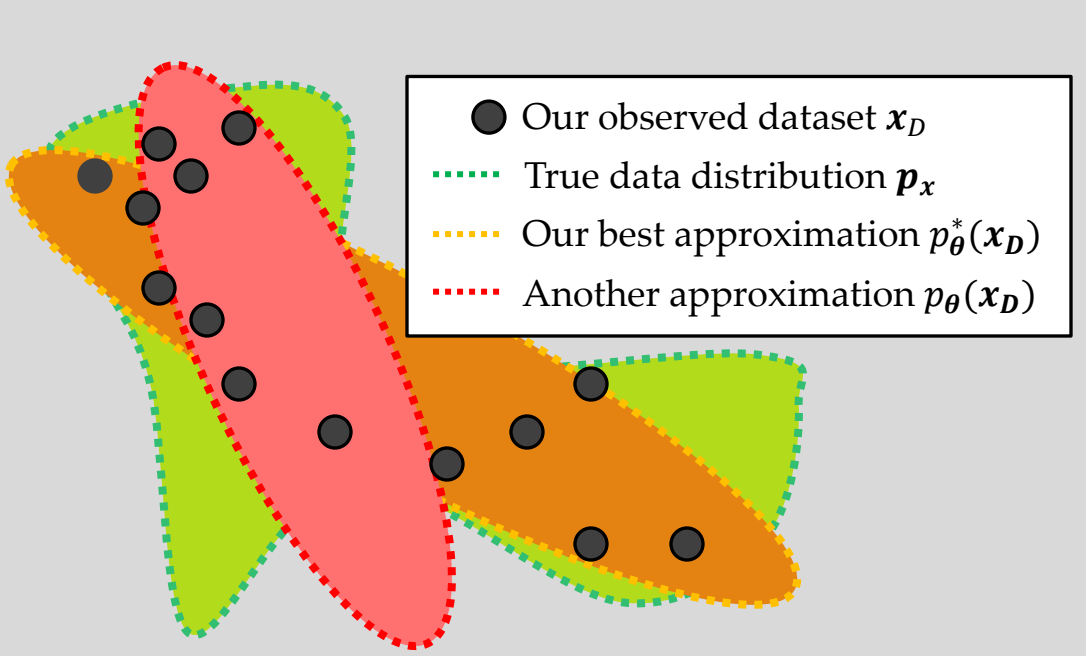
- Semi-supervised learning



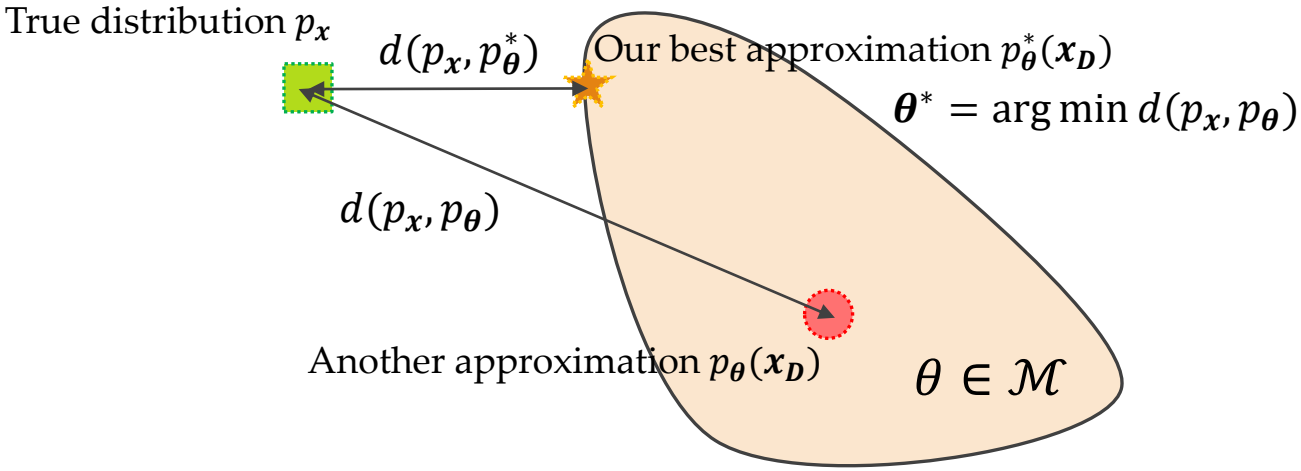
- Simulations

The world as a distribution

In data space: all possible data $\boldsymbol{x}$ (a.k.a. "The world")

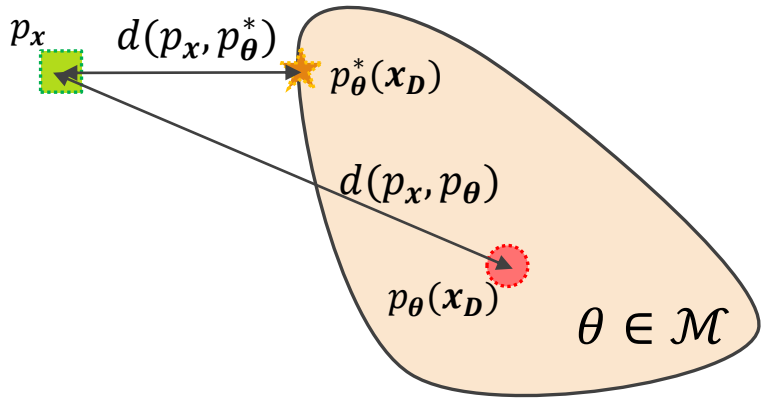


In the distribution space



Generative models: main challenges

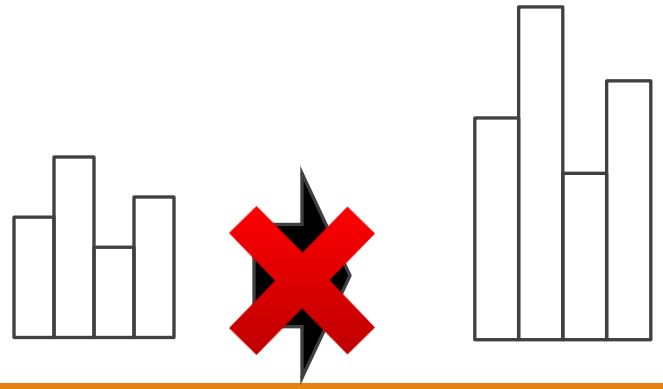
- We are interested in parametric models from a family of models $\mathcal{M}$



- How to pick the right family of models $\mathcal{M}$?
- How to know which θ from $\mathcal{M}$ is a good one?
- How to learn/optimize our models from family $\mathcal{M}$?

Properties for modelling distributions

- We want to learn distributions $p_{\boldsymbol{\theta}}(\boldsymbol{x})$
- Our model must therefore have the following properties
  - Non-negativity: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0 \; \forall \; \boldsymbol{x}$
  - Probabilities of all events must sum up to 1: $\int_{\boldsymbol{x}} p_{\boldsymbol{\theta}}(\boldsymbol{x}) \, d\boldsymbol{x} = 1$
- Summing up to 1 (normalization) makes sure predictions improve relatively
  - Model cannot trivially get better scores by predicting higher numbers
  - The pie remains the same → model forced to make non-trivial improvements



Parameterizing models for distributions: non-negativity

- Our model must therefore have the following properties
  - Non-negativity: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0 \forall\, \boldsymbol{x}$
  - Probabilities of all events must sum up to 1: $\int_{\boldsymbol{x}} p_{\boldsymbol{\theta}}(\boldsymbol{x})\, d\boldsymbol{x} = 1$
- Easy to obtain non-negativity
  - Consider: $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}}^2(\boldsymbol{x})$ where $f_{\boldsymbol{\theta}}$ is a neural network
  - Or $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$
  - But they do not sum up to 1

Energy-based models for distributions

- Normalize by the total volume of the function

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{\text{volume}(g_{\boldsymbol{\theta}})} g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{\int_{\boldsymbol{x}} g_{\boldsymbol{\theta}}(\boldsymbol{x}) d\boldsymbol{x}} g_{\boldsymbol{\theta}}(\boldsymbol{x})$$

  - In simple words, equivalent to normalizing $[3, 1, 4]$ as $\frac{1}{3+1+4}[3, 1, 4]$

- Examples
  - $g_{\boldsymbol{\theta}=(\boldsymbol{\mu},\boldsymbol{\sigma})}(\boldsymbol{x}) = \exp(-(\boldsymbol{x}-\boldsymbol{\mu})^2/2\boldsymbol{\sigma}^2) \Rightarrow \text{Volume}(g_{\boldsymbol{\theta}}) = \sqrt{2\pi\boldsymbol{\sigma}^2} \Rightarrow$ Gaussian
  - $g_{\boldsymbol{\theta}=\boldsymbol{\lambda}}(\boldsymbol{x}) = \exp(-\lambda\boldsymbol{x}) \Rightarrow \text{Volume}(g_{\boldsymbol{\theta}}) = \frac{1}{\lambda} \Rightarrow$ Exponential

- Must find convenient $g_{\boldsymbol{\theta}}$ to be able to compute the integral analytically
  - Otherwise we cannot make sure of valid probabilities

Why is learning a distribution hard?

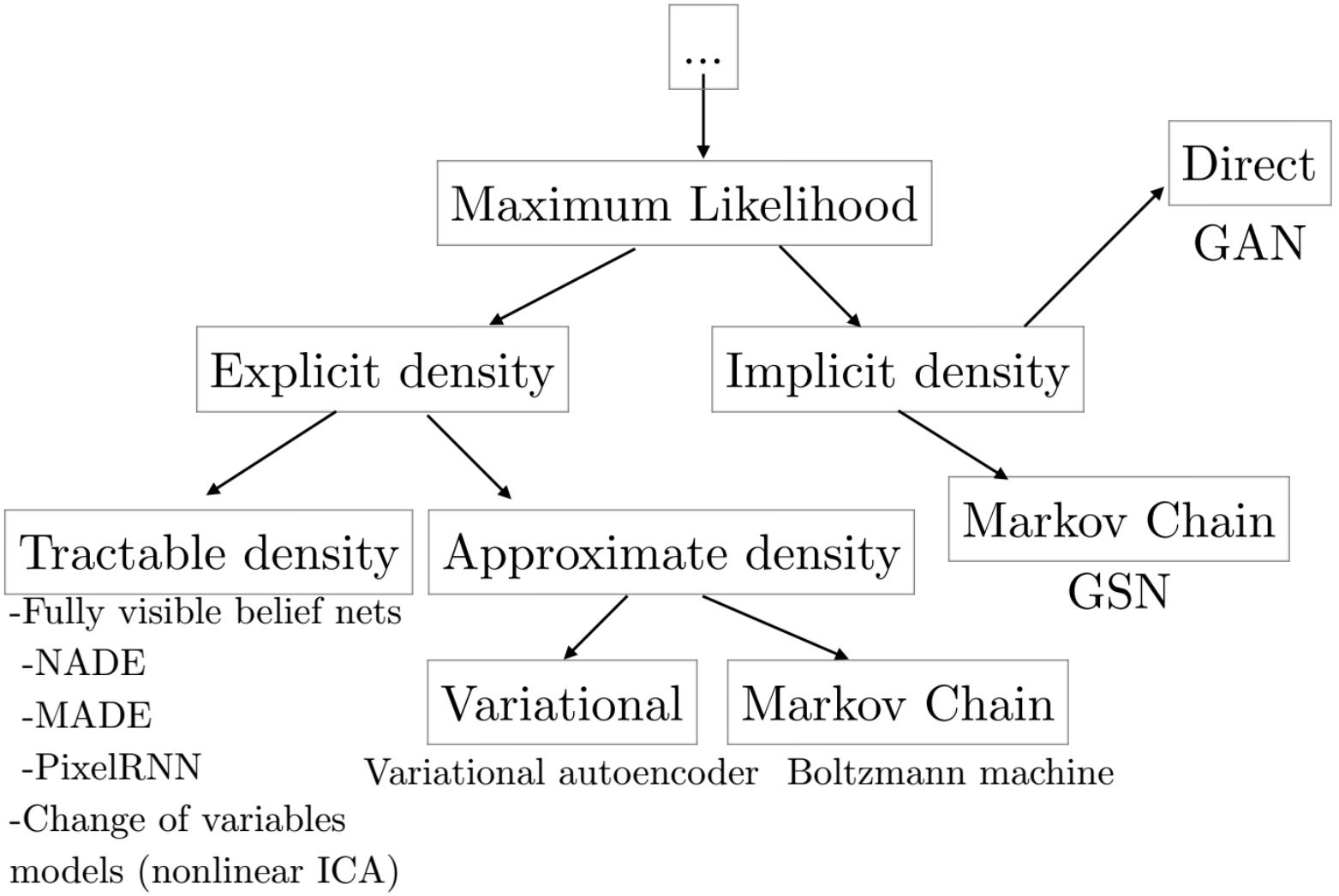
- The integrals mean that learning distributions becomes harder with scale
- Think of 300x400 color images with $[0, 256)$ color range
  - The number of possible images $\boldsymbol{x}$ is $256^{3 \cdot 300 \cdot 400}$
  - In principle must assign a probability to all of them

- While easy to *define* a family of models, we got a $\int_{\boldsymbol{x}} g_{\boldsymbol{\theta}}(\boldsymbol{x}) d\boldsymbol{x}$
  - Not always easy how to sample (needed for evaluating)
  - Not always easy how to optimize (needed for training)
  - Not always data efficient (long training times)
  - Not always sample efficient (many samples needed for accuracy)

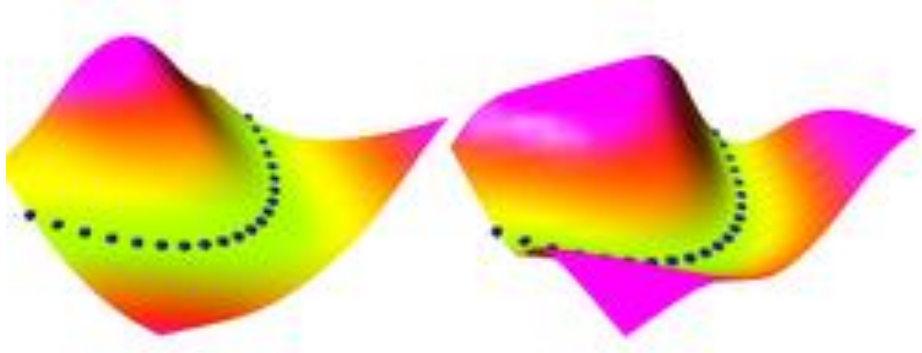
Why/when not to learn a distribution?

- *"One should solve the [classification] problem directly and never solve a more general [and harder] problem as an intermediate step."*
  V. Vapnik, father of SVMs.
- Generative models to be preferred
  - when probabilities are important
  - when you got no human annotations and want to learn features
  - when you want to generalize to (many) downstream tasks
  - when the answer to your question is not: "more data"
- If you have a very specific classification task and lots of data
  - no need to make things complicated

A map of generative models



Energy-based models



Energy-based models for distributions

- Distribution as: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{\int_{\boldsymbol{x}} g_{\boldsymbol{\theta}}(x)dx} g_{\boldsymbol{\theta}}(\boldsymbol{x})$
- $p_{\boldsymbol{\theta}}$ as known probability distributions (Gaussian, exp.) can be restrictive
  - Maybe I want to encode domain knowledge of how variables interact
- We can also define an energy function and divide by its volume

$$g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x})) \Rightarrow p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

Energy-based models for distributions

$$g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x})) \Rightarrow p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})}\exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

- $-f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the energy function
- Partition function is the hard bit

$$Z(\boldsymbol{\theta}) = \int_{\boldsymbol{x}} \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}))\, d\boldsymbol{x}$$

  - Note the multi-dimensional integral due to $\boldsymbol{x}$

Why exponential?

- Why $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ and not $g_{\boldsymbol{\theta}}(\boldsymbol{x}) = f^2_{\boldsymbol{\theta}}(\boldsymbol{x})$?

- Couples well with maximum likelihood and natural logarithms
- Many existing distributions are exponential-based
- They arise often in statistical physics → Good inspiration

Advantages & disadvantages

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

- Very flexible in defining our energy function
- Sampling from $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ can be very hard
  - The CDF introduces another integral
- Evaluating and optimizing likelihood can be hard ⇒ Learning is hard
  - Must be able to compute the partition function
- In vanilla case no latent variables ⇒ no representation learning
  - Latent variables can be added though

Ratio of likelihoods

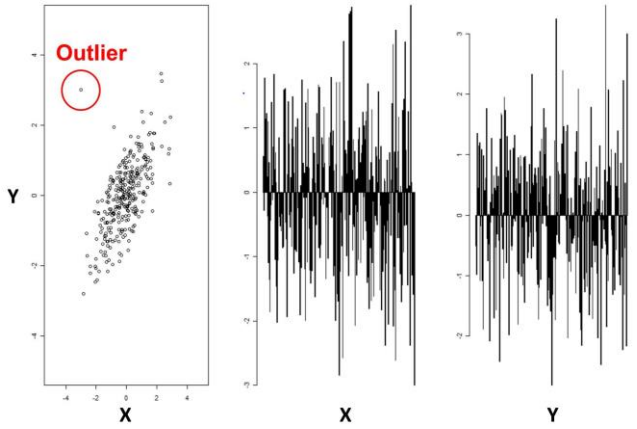
- The partition function is often very hard to compute analytically
- But if we have pairs of inputs

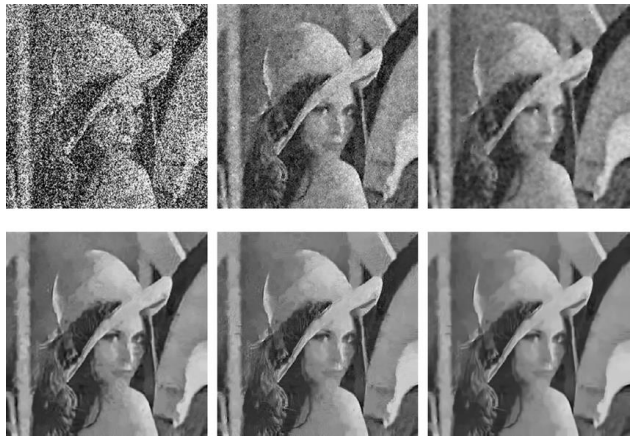
$$\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x_a})}{p_{\boldsymbol{\theta}}(\boldsymbol{x_b})} = \exp(f_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - f_{\boldsymbol{\theta}}(\boldsymbol{x}_b))$$

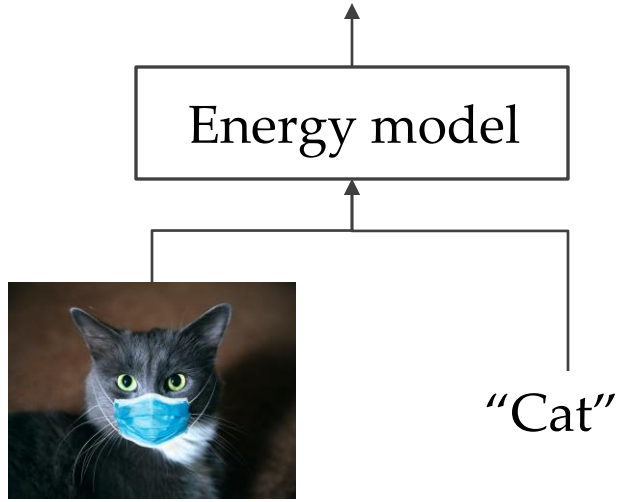
- No partition function anymore

Applications

- Given trained model
  - Anomaly detection
  - Denoising & restoration
  - Classification



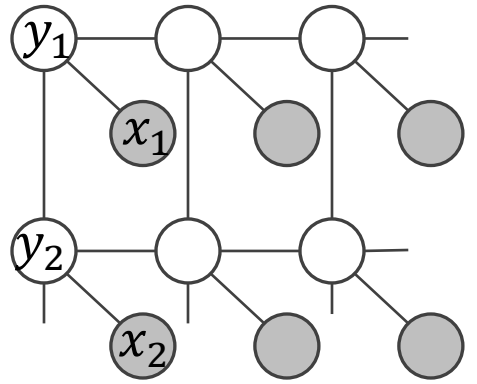




Examples of energy models

- Ising model

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{x}) = \frac{1}{Z} \exp\left(\sum_i \psi_i\left(x_i, y_i\right) + \sum_{i,j \in E} \psi_{ij}\left(y_i, y_j\right)\right)$$

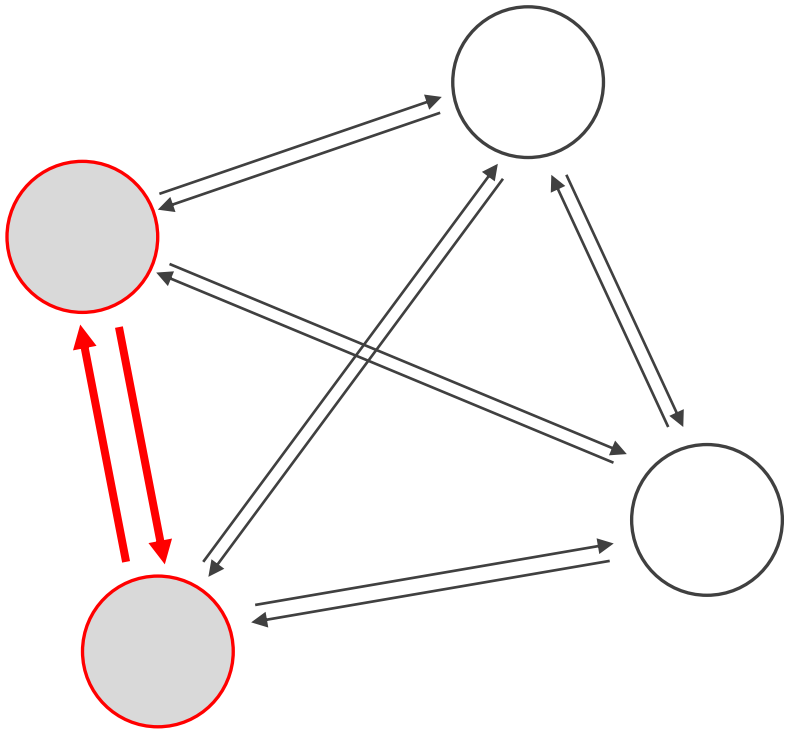


- Product of experts (similar to AND)

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\omega})} q_\theta(\boldsymbol{x}) r_\varphi(\boldsymbol{x}) s_\omega(\boldsymbol{x})$$

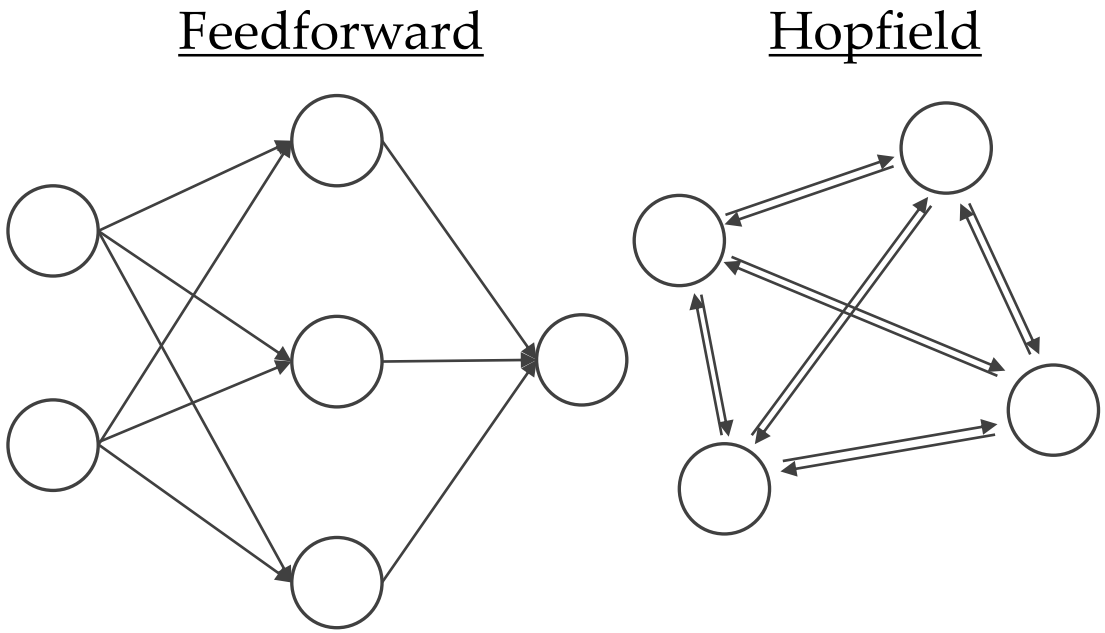
- Hopfield networks
- Boltzmann machines
- Deep belief networks

Hopfield networks



Hopfield vs feedforward networks

- Feedforward networks have connections that make up for acyclic graphs
- Feedback networks are networks that are not feedforward
- Hopfield networks:
  - Fully connected feedback networks
  - Symmetric weights, no self-connections
  - Associative (Hebbian) learning
- No separation of hidden vs visible
  - Neurons (nodes) update themselves
  - Based on all other neurons



Information Theory, Inference, and Learning Algorithms, D. MacKey

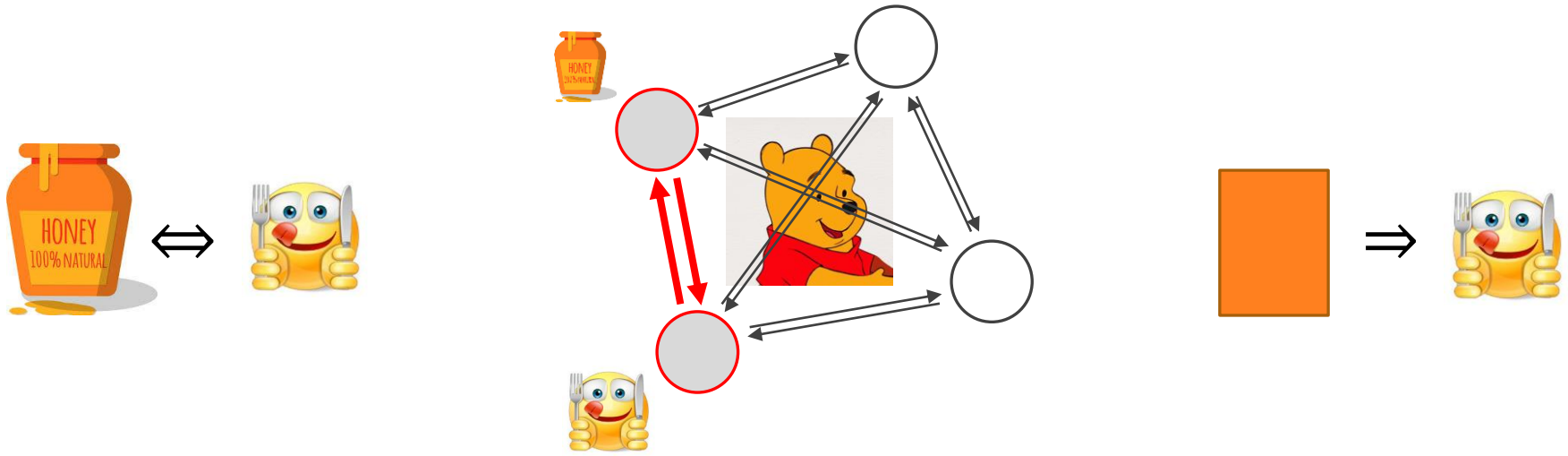
Hebbian learning

- Positively correlated neurons reinforce each other's weights

$$\frac{dw_{ij}}{dt} \propto \text{correlation}\left(x_i, x_j\right)$$

- Associative memories ⇔ No supervision ⇔ Pattern completion

Time t | Winnie the Hopfield | Time $t+1$



Hopfield network

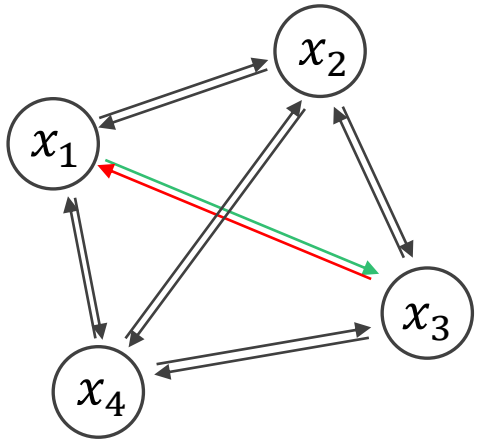
- Binary Hopfield defines neuron states given neuron activation a

$$x_i = h(a_i) = \begin{cases} 1 & a_i \geq 0 \\ -1 & a_i < 0 \end{cases}$$

- Continuous Hopfield defines neuron states given neuron activation a

$$x_i = \tanh(a_i)$$

- Note the feedback connection!
  - Neuron x_1 influences x_3, but x_3 influences x_1 back



- Who influences whom first?
  - Either synchronous updates: $a_i = \sum_j w_{ij} x_j$
  - Or asynchronous updates: one neuron at a time (fixed or random order)

Hopfield memory

- Network updates $x_i \in \{-1, 1\}$ till convergence to a stable state
  - Recurrent inference cycles
  - Not 'single propagation'
- Stable means x_i does not flip states no more

(a)

moscow------russia
lima----------peru
london-----england
tokyo--------japan
edinburgh-scotland
ottawa------canada
oslo--------norway
stockholm---sweden
paris-------france

(b)
```
moscow---::::::::::  ⟹  moscow------russia
::::::::::--canada   ⟹  ottawa------canada
```

(c)
```
otowa-------canada   ⟹  ottawa------canada
egindurrh-sxotland   ⟹  edinburgh-scotland
```

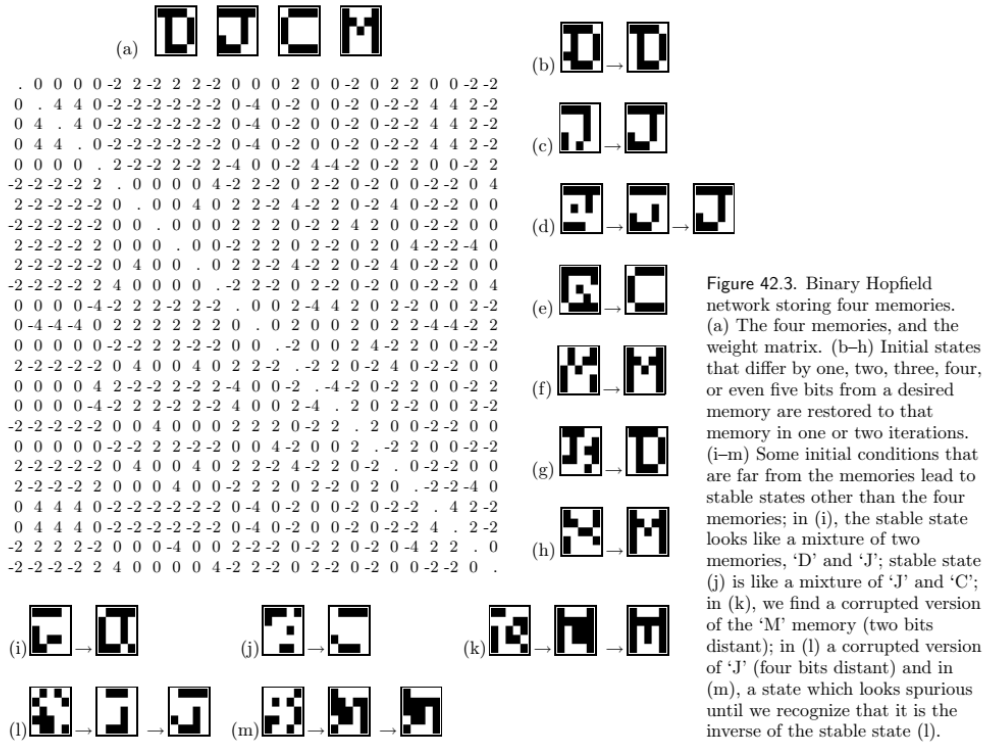


Figure 42.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to stable states other than the four memories; in (i), the stable state looks like a mixture of two memories, 'D' and 'J'; stable state (j) is like a mixture of 'J' and 'C'; in (k), we find a corrupted version of the 'M' memory (two bits distant); in (l) a corrupted version of 'J' (four bits distant) and in (m), a state which looks spurious until we recognize that it is the inverse of the stable state (l).

Energy function

- Hopfield networks minimize the quadratic energy function

$$f_\theta(\boldsymbol{x}) = \sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i$$

- Lyapunov functions are functions that
  - Decreases under the dynamical evolution of the system
  - Bounded below
- Lyapunov functions converge to fixed points
- The Hopfield energy is a Lyapunov function
  - Provided asynchronous updates
  - Provided symmetric weights

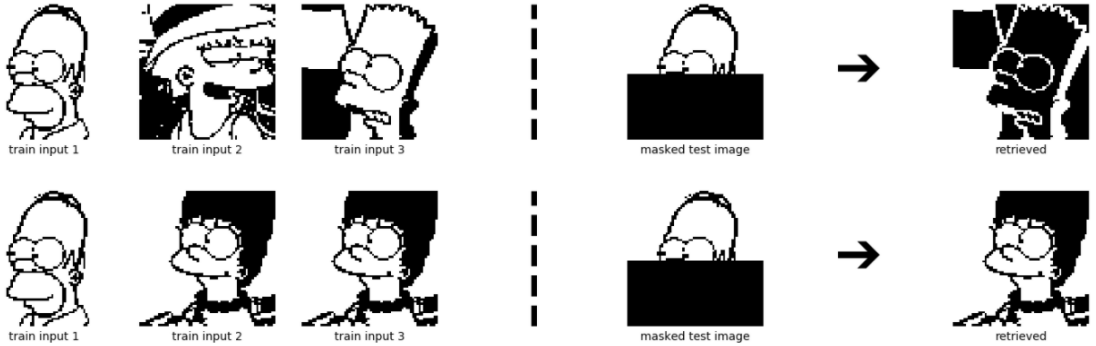
Learning algorithm

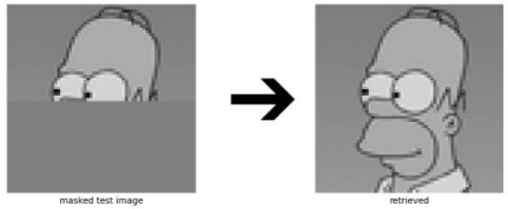
```
w = x' * x ;          # initialize the weights using Hebb rule

for l = 1:L           # loop L times

        for i=1:I             #
          w(i,i) = 0 ;        #  ensure the self-weights are zero.
        end                   #

        a  = x * w      ;     # compute all activations
        y  = sigmoid(a) ;     # compute all outputs
        e  = t - y      ;     # compute all errors
        gw = x' * e     ;     # compute the gradients
        gw = gw + gw'   ;     # symmetrize gradients

        w  = w + eta * ( gw - alpha * w ) ;  # make step

endfor
```

Hopfield networks is all you need

- Retrieving from stored memory patterns
- Update rule as in the attention mechanism in transformer networks

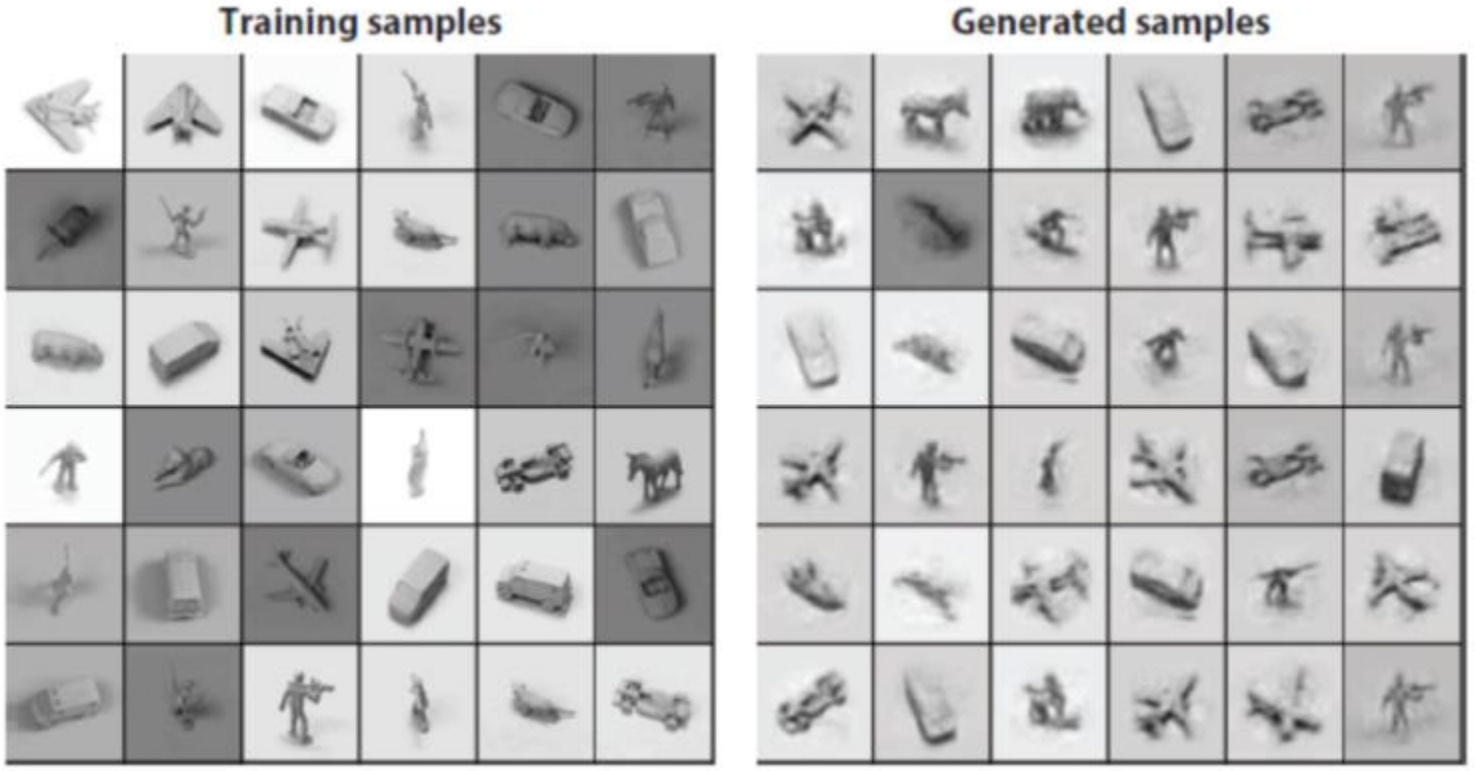
$$\boldsymbol{W} = \sum_{i=1}^{3} \boldsymbol{x}_i \boldsymbol{x}_i^T \,, \qquad \boldsymbol{x}_i \in \{-1, 1\}^d \,. \tag{6}$$





Ramsauer et al., 2020

Boltzmann machines



From Hopfield to Boltzmann

- Hopfield networks minimize the quadratic energy function

$$E = -f_\theta(\boldsymbol{x}) = -\left(\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i\right)$$

- Boltzmann machines are stochastic Hopfield networks
- In Boltzmann machines the neuron response on activation a_i is

$$x_i = \begin{cases} +1 \text{ with probability } 1/1 + \exp(-2a_i) \\ -1 \qquad \text{otherwise} \end{cases}$$

- Gibbs sampling for pdf $p(\boldsymbol{x}) = \frac{1}{Z}\exp(\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W} \boldsymbol{x})$

Restricted Boltzmann machines

- Boltzmann machines are too parameter heavy
  - For $\boldsymbol{x}$ with $256 \times 256 = 65536$ the $\boldsymbol{W}$ has 4.2 billion parameters
- Boltzmann machines learn no features
- Instead, add bottleneck latents $\boldsymbol{v}$

$$E = -f_\theta(\boldsymbol{x}) = -\left(\sum_{i,j} w_{ij} x_i v_j + \sum_i b_i x_i + \sum_j c_j v_j\right)$$

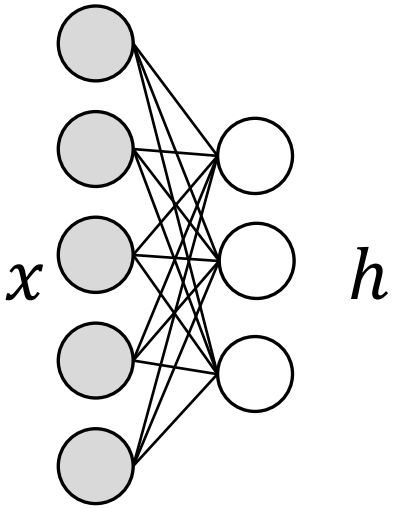
  - x_i and v_j are still binary variables in the original model
  - The quadratic term captures correlations
  - The unary terms capture priors: how likely is a (latent) pixel to be +1 or -1

Restricted Boltzmann Machines

- Energy function: $E(\boldsymbol{x}) = -\boldsymbol{x}^T \boldsymbol{W} \boldsymbol{v} - \boldsymbol{b}^T \boldsymbol{x} - \boldsymbol{c}^T \boldsymbol{v}$

$$p(\boldsymbol{x}) = \frac{1}{Z} \sum_{\boldsymbol{v}} \exp(-E(\boldsymbol{x}, \boldsymbol{v}))$$

  - Not in the form $\propto \exp(\mathbf{x})/\mathrm{Z}$ because of the $\sum$



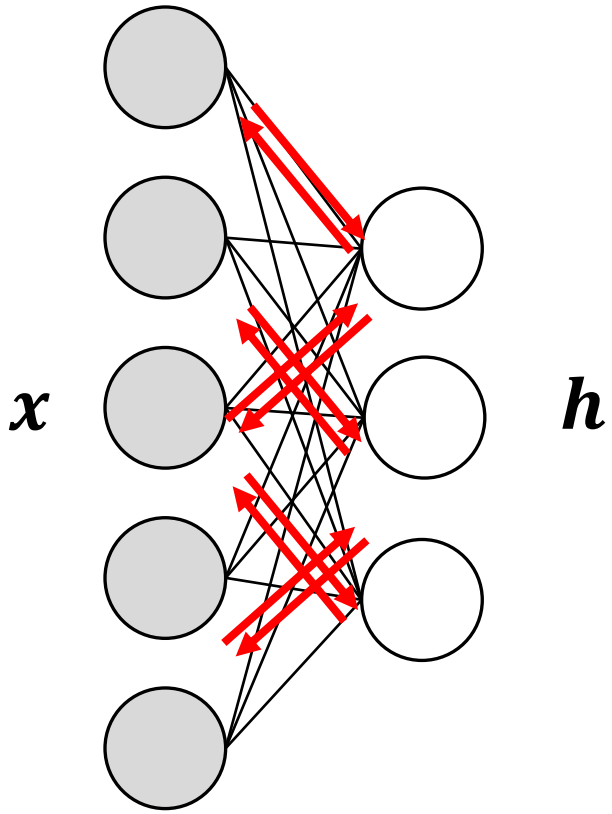
- Free energy function: $F(\boldsymbol{x}) = -\boldsymbol{b}^T \boldsymbol{x} - \sum_i \log \sum_{v_i} \exp(v_i (c_i + \boldsymbol{W}_i \boldsymbol{x}))$

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp(-F(\boldsymbol{x}))$$

$$Z = \sum_{\boldsymbol{x}} \exp(-F(\boldsymbol{x}))$$

Restricted Boltzmann Machines

- The $F(\boldsymbol{x})$ defines a bipartite graph with undirected connections
  - Information flows forward and backward



Restricted Boltzmann Machines

- The hidden variables v_j are independent conditioned on the visible variables

$$p(\boldsymbol{v}|\boldsymbol{x}) = \prod_j p\left(v_j|\boldsymbol{x}, \boldsymbol{\theta}\right)$$

- The visible variables x_i are independent conditioned on the hidden variables

$$p(\boldsymbol{x}|\boldsymbol{v}) = \prod_i p(x_i|\boldsymbol{v}, \boldsymbol{\theta})$$

Training RBM conditional probabilities

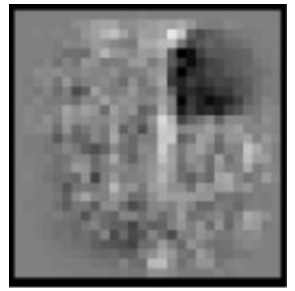
- The conditional probabilities are defined as sigmoids

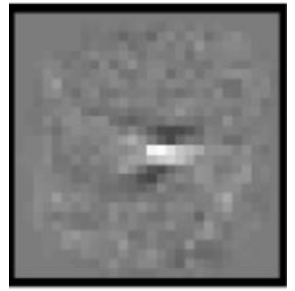
$$p(v_j|\boldsymbol{x},\boldsymbol{\theta}) = \sigma(\boldsymbol{W}_{.j}\boldsymbol{x} + b_j)$$
$$p(x_i|\boldsymbol{v},\boldsymbol{\theta}) = \sigma(\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{W}_{i.} + c_i)$$

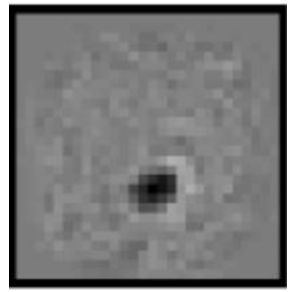
- Since RBMs are bidirectional ⇒ "Loop" between visible and latent

$$\boldsymbol{v}^{(1)} \sim \sigma(\boldsymbol{W}_{.j}\boldsymbol{x}^{(0)} + b_j) \Rightarrow$$
$$\boldsymbol{x}^{(1)} \sim \sigma(\boldsymbol{W}_{.j}\boldsymbol{v}^{(2)} + b_j) \Rightarrow$$
$$\boldsymbol{v}^{(2)} \sim \sigma(\boldsymbol{W}_{.j}\boldsymbol{x}^{(1)} + b_j) \Rightarrow \ \ldots$$

Latent activations







Training any energy model

- Maximizing log-likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{\mathrm{N}} \sum_{n} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n) = \mathbb{E}_{p_0}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]$$

- The expectation w.r.t. a pdf is equivalent to
  - sampling from the pdf and
  - then taking the average

$$\mathbb{E}_{\boldsymbol{x}\sim p_0}[\log p(\boldsymbol{x}|\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{x}\sim p_0}[-E_{\boldsymbol{\theta}}(\boldsymbol{x})] - \log Z(\boldsymbol{\theta})$$

  - where $\log Z(\boldsymbol{\theta}) = \log \sum_{x'} \exp(-E_{\boldsymbol{\theta}}(\boldsymbol{x}'))$
  - and $p_0(\boldsymbol{x})$ is the data distribution

Taking gradients of any energy model

$$\frac{d}{d\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = -\frac{d}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{d}{d\boldsymbol{\theta}}\log Z(\boldsymbol{\theta}) =$$

$$= -\frac{d}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{1}{Z(\boldsymbol{\theta})}\frac{d}{d\boldsymbol{\theta}}Z(\boldsymbol{\theta})$$

$$= -\frac{d}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x}) - \sum_{\boldsymbol{x'}}\frac{1}{Z(\boldsymbol{\theta})}\exp(-E_{\boldsymbol{\theta}}[\boldsymbol{x'}])\left(-\frac{d}{d\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x'})\right)$$

$$= -\frac{d}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x}) + \sum_{\boldsymbol{x'}}p_{\theta}(\boldsymbol{x'})\frac{d}{d\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x'})$$

$$= -\frac{d}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x'}\sim p_{\boldsymbol{\theta}}}\left[\frac{d}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x'})\right]$$

Remember: $\sum p(x)\, f(x) = \mathbb{E}_{p(x)}[f(x)]$

$$\int_x p(x)f(x)dx = \mathbb{E}_{p(x)}[f(x)]$$

Taking gradients in an RBM

- For an RBM we must integrate out the latent variables

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{\mathrm{N}}\sum_{n} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n) = \frac{1}{\mathrm{N}}\sum_{n} \log \sum_{\boldsymbol{v}} p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{v})$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log \sum_{\boldsymbol{v}} p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{v}) = -\mathbb{E}_{\boldsymbol{v}\sim p_{\boldsymbol{\theta}}(\boldsymbol{v}|\boldsymbol{x}_n)}\left[\frac{d}{d\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{v})\right] + \mathbb{E}_{\boldsymbol{x}', \boldsymbol{v}\sim p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})}\left[\frac{d}{d\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}', \boldsymbol{v})\right]$$

Taking gradients in an RBM

- And since for RBM $E_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x} - \boldsymbol{c}^T \boldsymbol{v}$

$$\frac{d}{d\boldsymbol{W_{ij}}} E_{\boldsymbol{\theta}}(x_i, v_j) = -x_i v_j \Rightarrow$$

$$\frac{d\mathcal{L}}{d\boldsymbol{W_{ij}}} = \mathbb{E}_{\boldsymbol{v} \sim p_{\boldsymbol{\theta}}(\boldsymbol{v}|\boldsymbol{x}_n)}[x_i v_j] - \mathbb{E}_{\boldsymbol{x'}, \boldsymbol{v} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})}[x_i v_j]$$

- Easy: substitute $\boldsymbol{x}_n$ and sum over $\boldsymbol{v}$
- Hard (normalization): sum over all 2^{m+d} combinations of images & latents
  - Intractable due to exponential complexity w.r.t. $m + d$
  - Evaluating and optimizing $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})$ takes a long time
  - If we had only the unnormalized part we would have no problem

Tackling intractability by sampling

- $\mathbb{E}_{\boldsymbol{x}',\boldsymbol{v}\sim p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})}\left[\frac{d}{d\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x}',\boldsymbol{v})\right]$ stands for an expectation
  - One can sample very many $\boldsymbol{x}', \boldsymbol{v}$ from $p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})$
  - Take average instead of computing analytically (Monte Carlo sampling)
- Question: how to even sample from a hard pdf?
  - Markov Chain Monte Carlo with Gibbs sampling
  - Convergence after many rounds

Initialization: Initialize $\mathbf{x}^{(0)} \in \mathcal{R}^D$ and number of samples N

- **for** $i = 0$ to $N-1$ **do**
- $x_1^{(i+1)} \sim p(x_1|x_2^{(i)}, x_3^{(i)}, ..., x_D^{(i)})$
- $x_2^{(i+1)} \sim p(x_2|x_1^{(i+1)}, x_3^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_j^{(i+1)} \sim p(x_j|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_D^{(i+1)} \sim p(x_D|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{D-1}^{(i+1)})$

return $(\{\mathbf{x}^{(i)}\}_{i=0}^{N-1})$

Sampling the normalizing constant

- We can rewrite the gradient as

$$\frac{d}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = -\mathbb{E}_0 \left[\frac{d}{\partial \boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}) \right] + \mathbb{E}_\infty \left[\frac{d}{\partial \boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}') \right]$$

  - $\mathbb{E}_0 \equiv E_{\boldsymbol{x} \sim p_0}$ means sampling from training data and average gradients
  - $\mathbb{E}_\infty \equiv E_{\boldsymbol{x}, \boldsymbol{v} \sim p_{\boldsymbol{\theta}}}$ means sampling from the model and average gradients
- Unfortunately, MCMC can be very slow → 2nd source of intractability

Ergo, contrastive diverge learning

- To motivate contrastive divergence, we revisit maximum likelihood learning

$$\mathrm{KL}(p_0 \parallel p_\infty) = \int p_0 \log p_0 - \int p_0 \log p_\infty \propto - \int p_0 \log \mathrm{p}_\infty$$

- Contrastive divergence minimizes

$$\mathrm{CD}_n = \mathrm{KL}(p_0 \parallel p_\infty) - \mathrm{KL}(p_n \parallel p_\infty)$$

- Updates weights using CD_n gradients instead of ML gradients

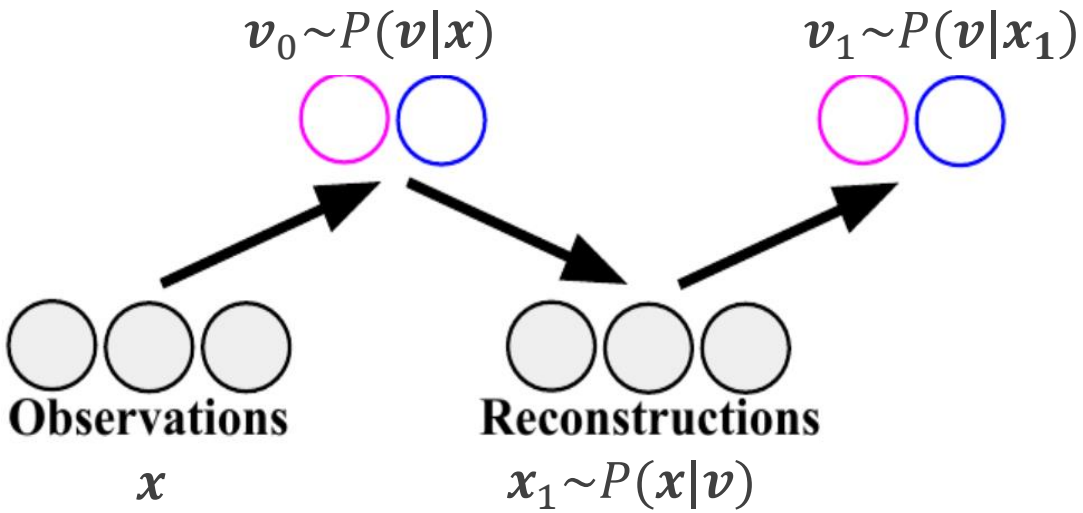
$$\frac{d}{\partial \boldsymbol{\theta}} \mathrm{CD}_n = -\mathbb{E}_0 \left[\frac{d}{\partial \boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}) \right] + \mathbb{E}_n \left[\frac{d}{\partial \boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}') \right] + \frac{d}{\partial \boldsymbol{\theta}} [\dots]$$

  - where $\mathbb{E}_n$ is computed by sampling after n steps in the Markov Chain
  - The last term is small and can be ignored

Hinton, *Training Products of Experts by Minimizing Contrastive Divergence*, Neural Computation, 2002

Contrastive diverge learning: intuition

- Make sure after n sampling step not far from data distribution
  - Usually, one step only (n=1) is enough
  - Something similar to 'minimizing reconstruction error'
- Because of conditional independence of $\boldsymbol{x}|\boldsymbol{v}$ and $\boldsymbol{v}|\boldsymbol{x}$ → parallel computations
  - Sample a data point $\boldsymbol{x}$
  - Compute the posterior $\boldsymbol{p}(\boldsymbol{v}|\boldsymbol{x})$
  - Take sample of latents $\boldsymbol{v}\sim\boldsymbol{p}(\boldsymbol{v}|\boldsymbol{x})$
  - Compute the conditional $\boldsymbol{p}(\boldsymbol{x}|\boldsymbol{v})$
  - Sample from $\boldsymbol{x}'\sim\boldsymbol{p}(\boldsymbol{x}|\boldsymbol{v})$
  - Minimize difference using $\boldsymbol{x}, \boldsymbol{x}'$

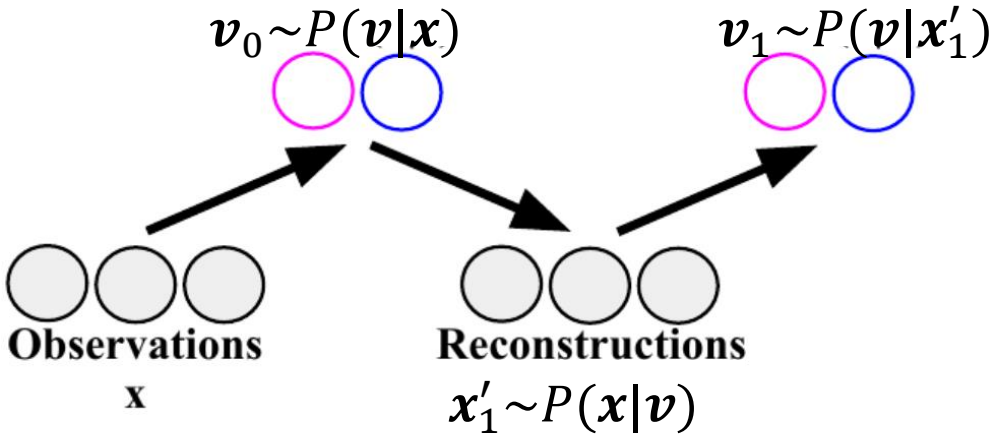


Contrastive divergence for RBMs

- Contrastive divergence approximates gradient by k-steps Gibbs sampler

$$\frac{d}{d\boldsymbol{\theta}} \log p(\boldsymbol{x}_n|\boldsymbol{\theta}) = -\frac{d}{d\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{v}_0) - \frac{d}{d\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}'_k, \boldsymbol{v}_k)$$

- Pushing the nominator up while pushing the denominator down



Hinton, 2002

Carreira-Perpinan and Hinton, 2005

Ermon and Grover, Deep Generative Models

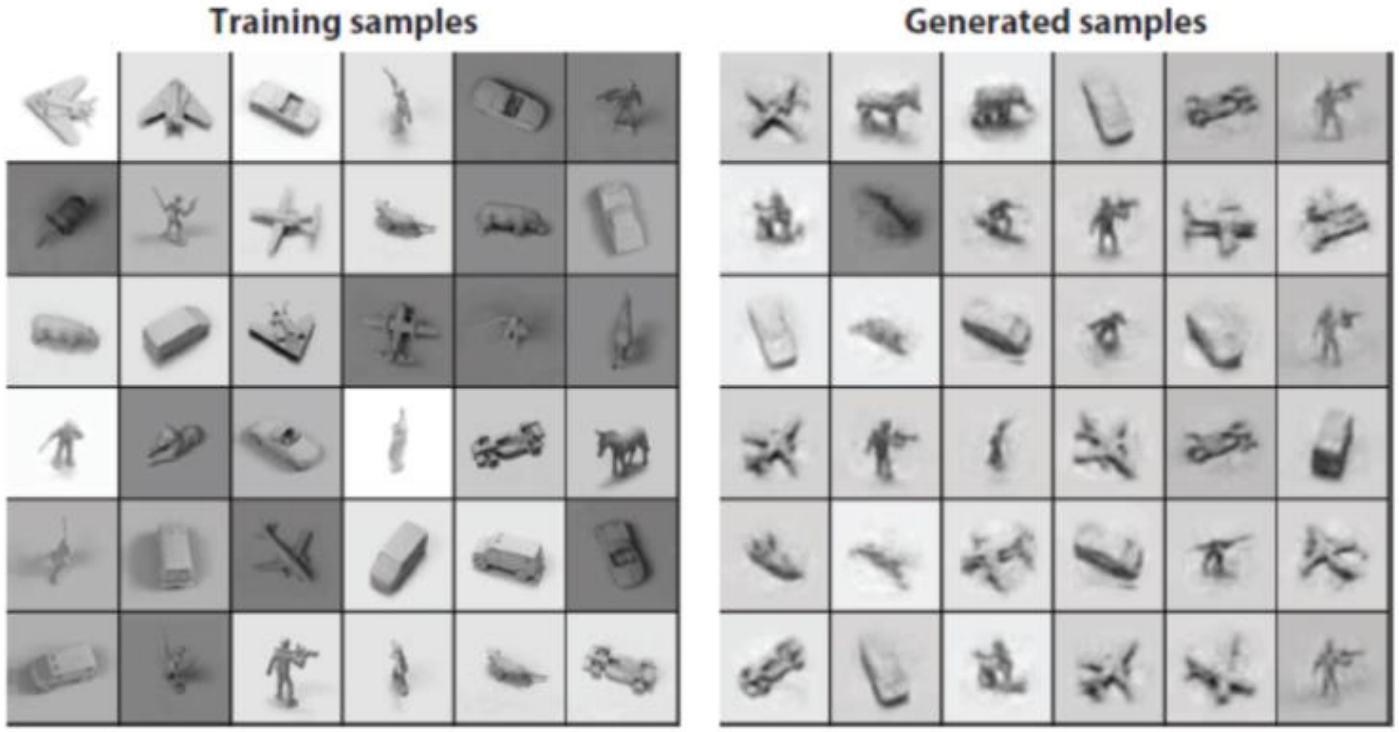
How to sample? Markov Chain Monte Carlo

We want to sample an $\boldsymbol{x}$ from a pdf $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ with MCMC with Gibbs sampler

- Step 1. Initialize $\boldsymbol{x}^0$ randomly
- Step 2. Let $\hat{\boldsymbol{x}} = \boldsymbol{x}^t + \text{noise}$
  - If $f_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}) > f_{\boldsymbol{\theta}}(\boldsymbol{x}^{\mathrm{t}})$, set $\boldsymbol{x}^{t+1} = \hat{\boldsymbol{x}}$
  - Otherwise $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t$ with probability $\frac{p(\hat{\boldsymbol{x}})}{p(\boldsymbol{x}^t)} = \exp(f_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}) - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{\mathrm{t}}))$
- Go to step 2
- Because of the ratio of likelihoods → no $Z(\boldsymbol{\theta})$

Using RBMs

- Some of the first models to show nice generations of images
- Use RBMs to pretrain networks for classification afterward

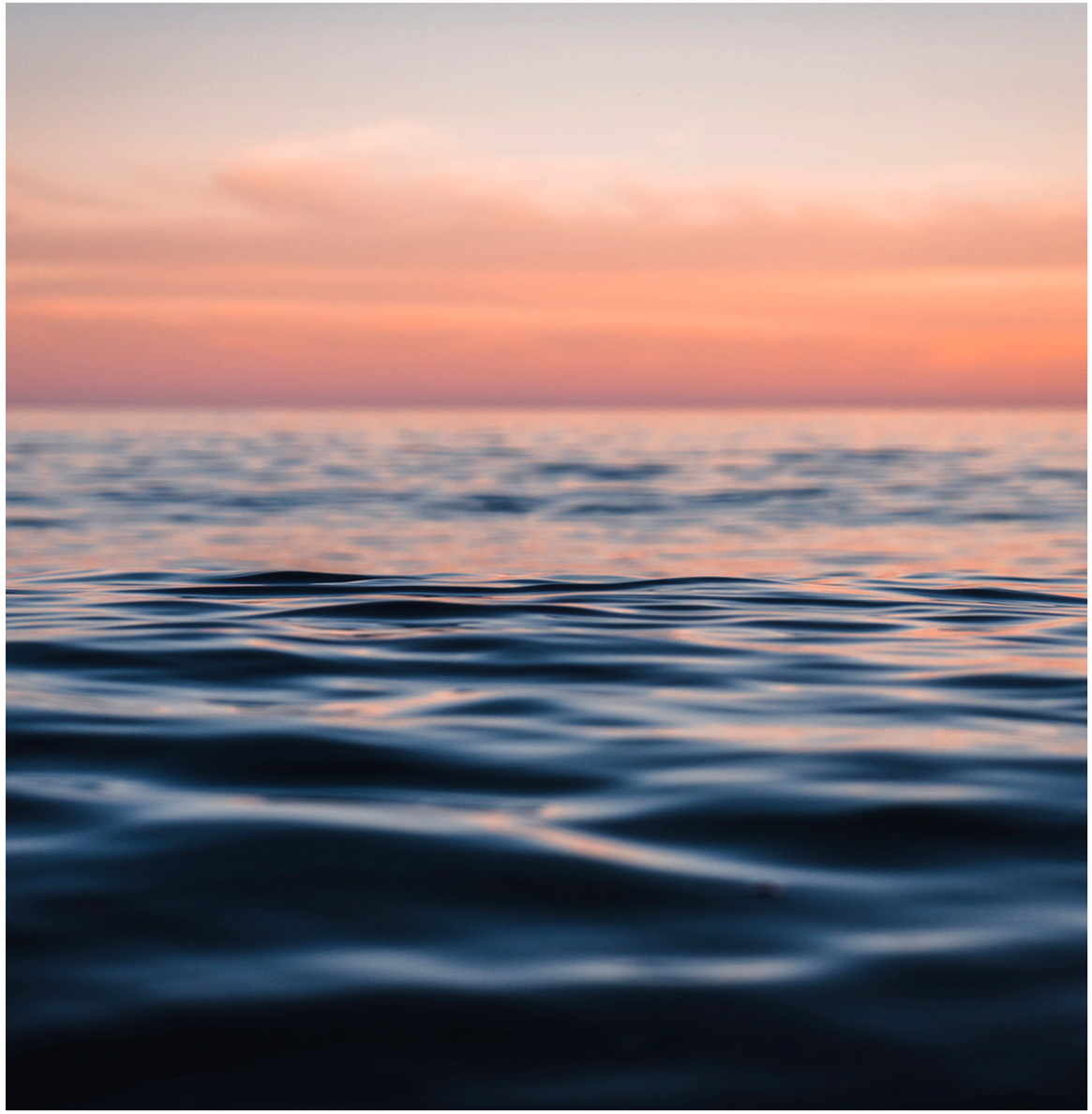


Score-matching & Diffusion Generative Models

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

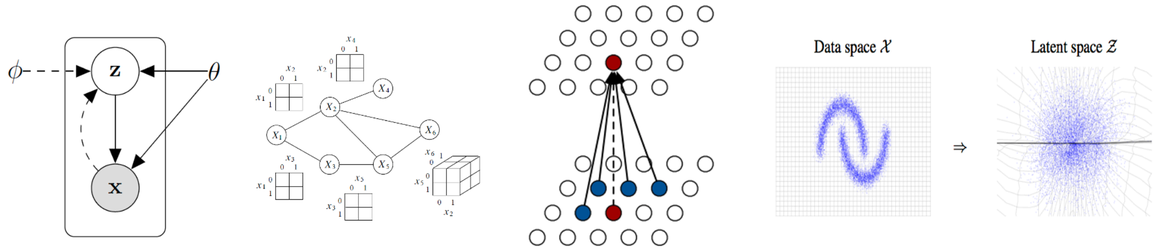
Overview of generative models

Likelihood-based generative models

Implicit generative models

Likelihood-based generative models

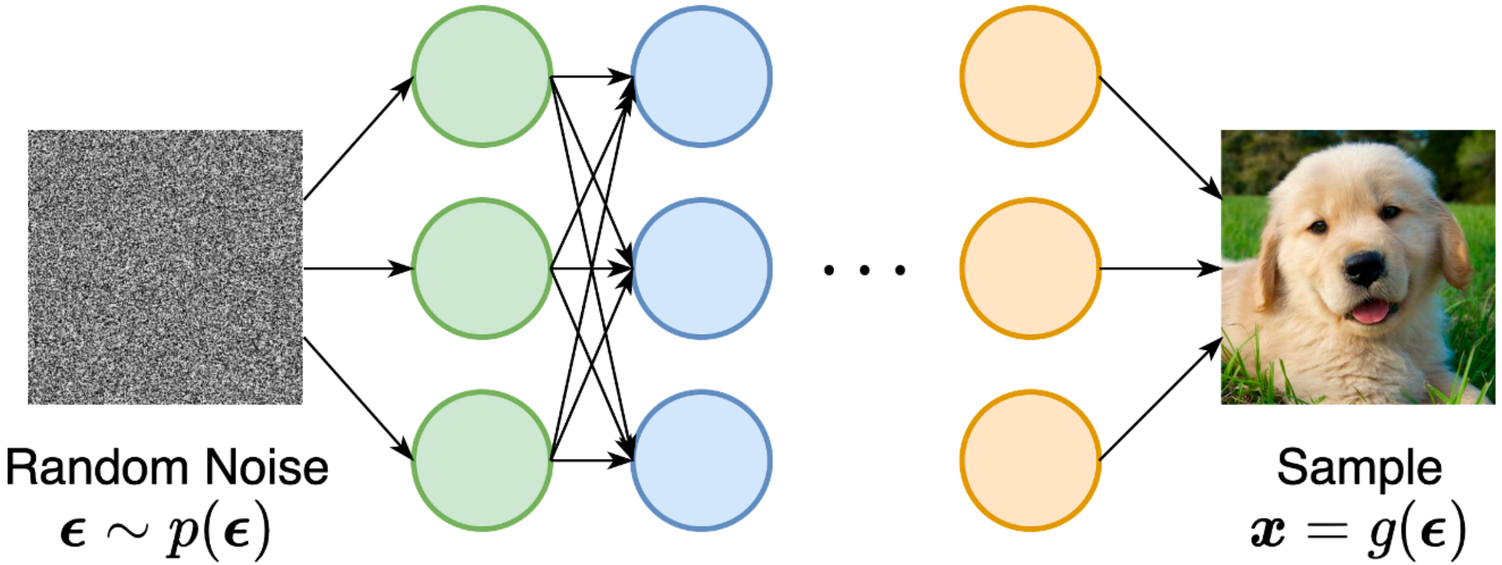
- Typically make strong assumptions to ensure tractability of likelihood
  - specifically of the normalising constant $Z(x)$ in $p(x) = \dfrac{\tilde{p}(x)}{Z(x)}$
- For instance, VAEs assume a tractable variational approximation
- Autoregressive models require causal convolutions
- Normalizing Flows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



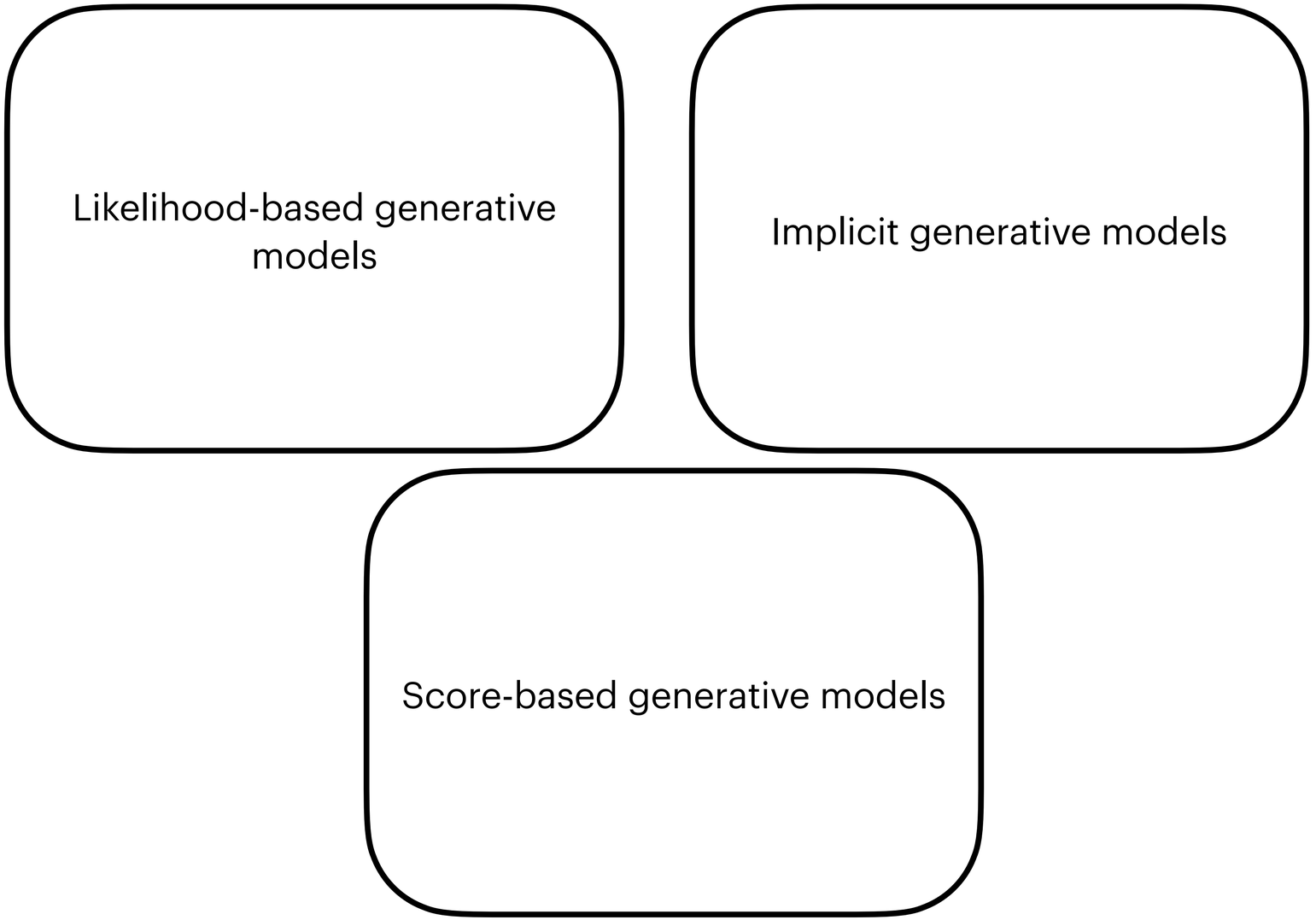
Bayesian networks (e.g., VAEs) | MRF | Autoregressive models | Flow models

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



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*
-

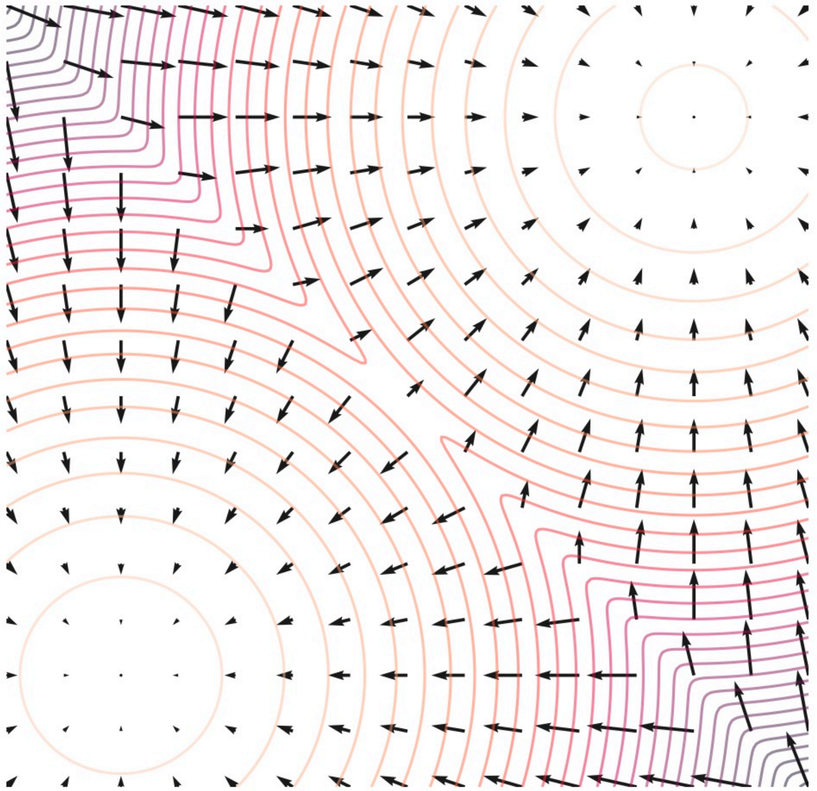
Score function

- 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 $\nabla_x \log p(x)$

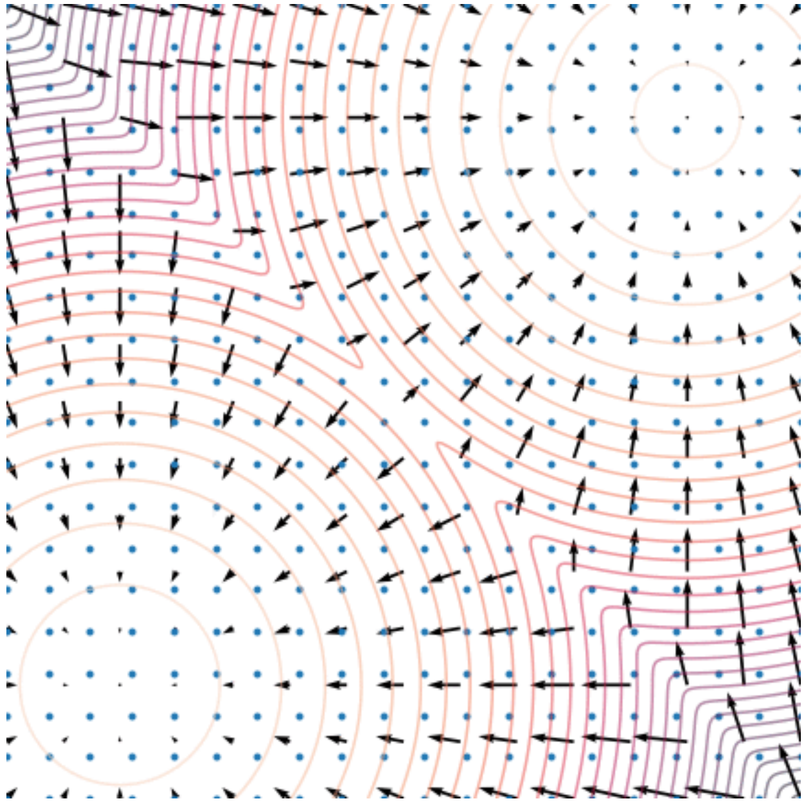
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t, \quad t = 0, \ldots, K, \quad \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- 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*

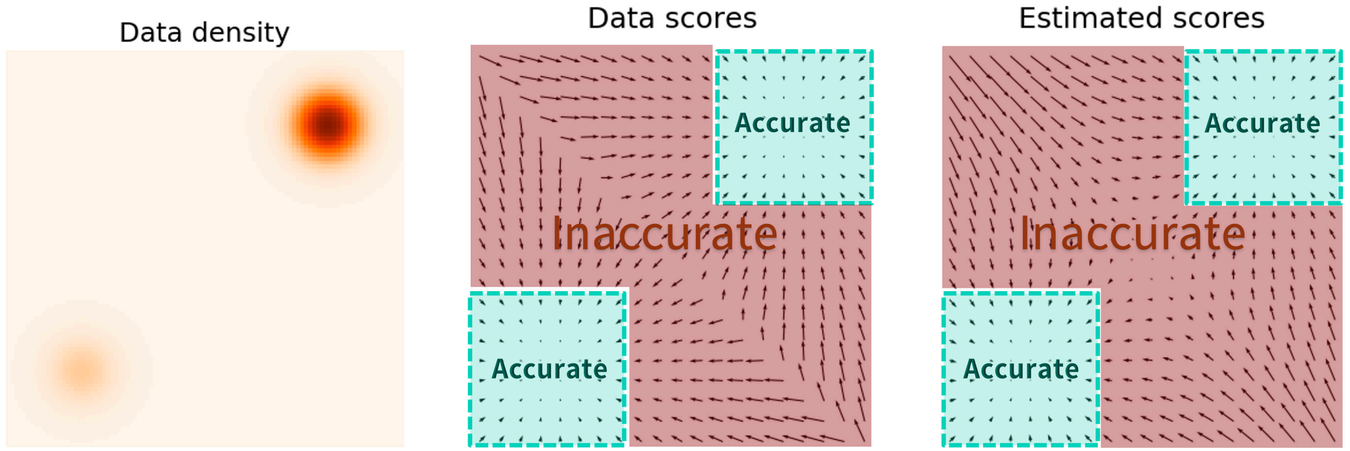
'Bayesian Learning via Stochastic Gradient Langevin Dynamics', M. Welling, Y. W. Teh

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

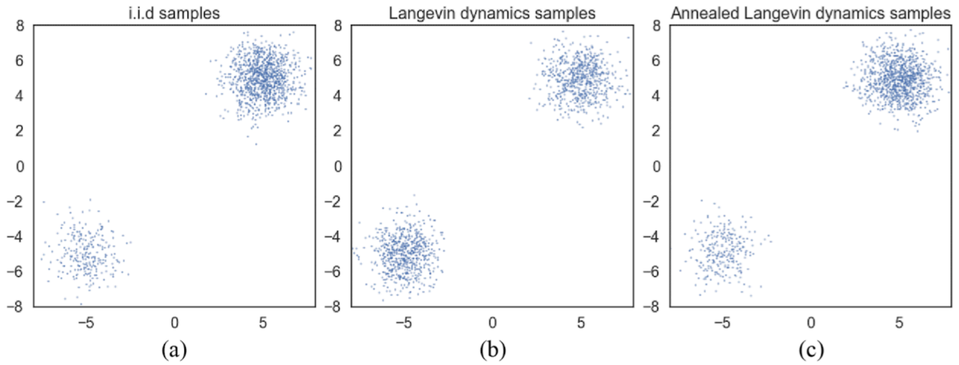
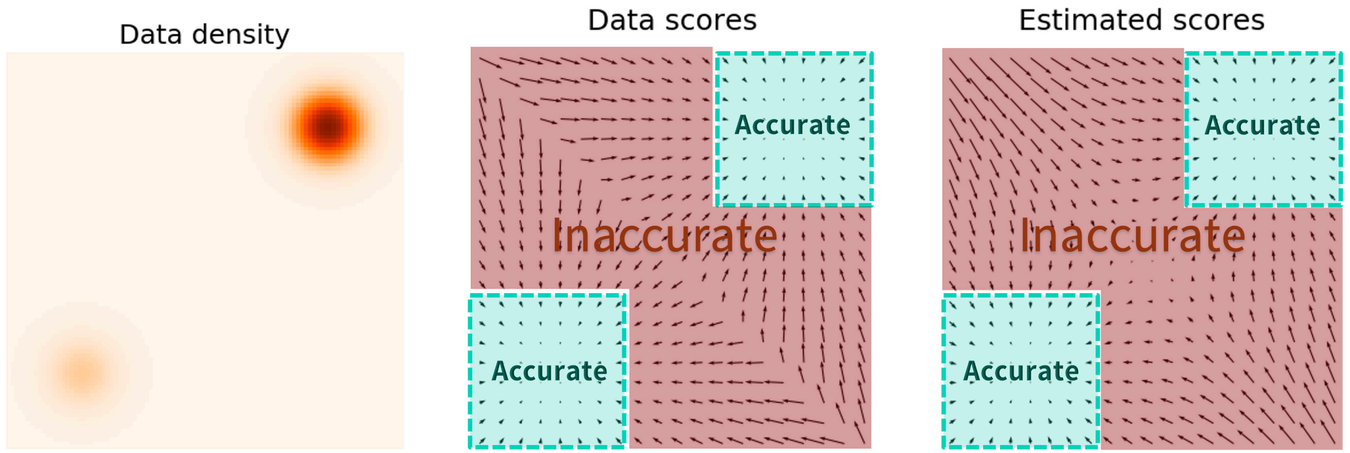


Figure 3: Samples from a mixture of Gaussian with different methods. (a) Exact sampling. (b) Sampling using Langevin dynamics with the exact scores. (c) Sampling using annealed Langevin dynamics with the exact scores. Clearly Langevin dynamics estimate the relative weights between the two modes incorrectly,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

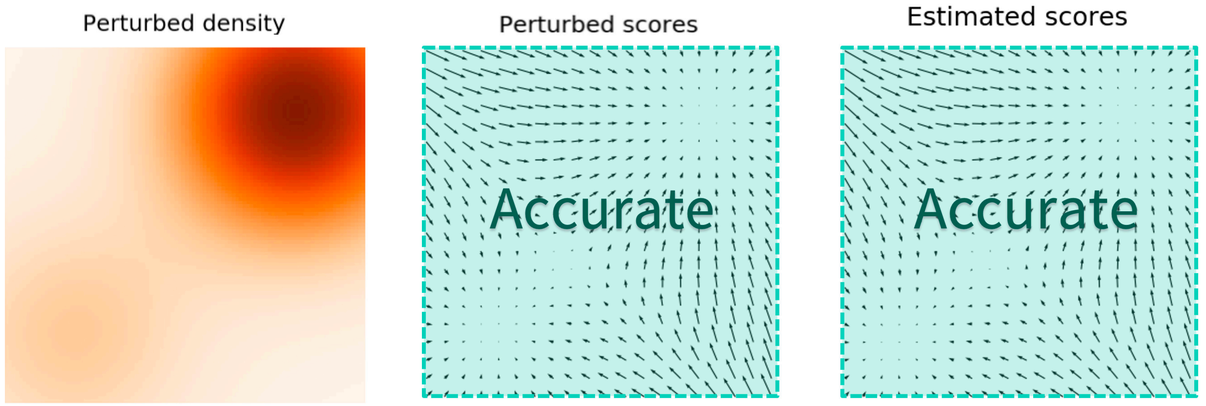


Noise perturbations

- 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

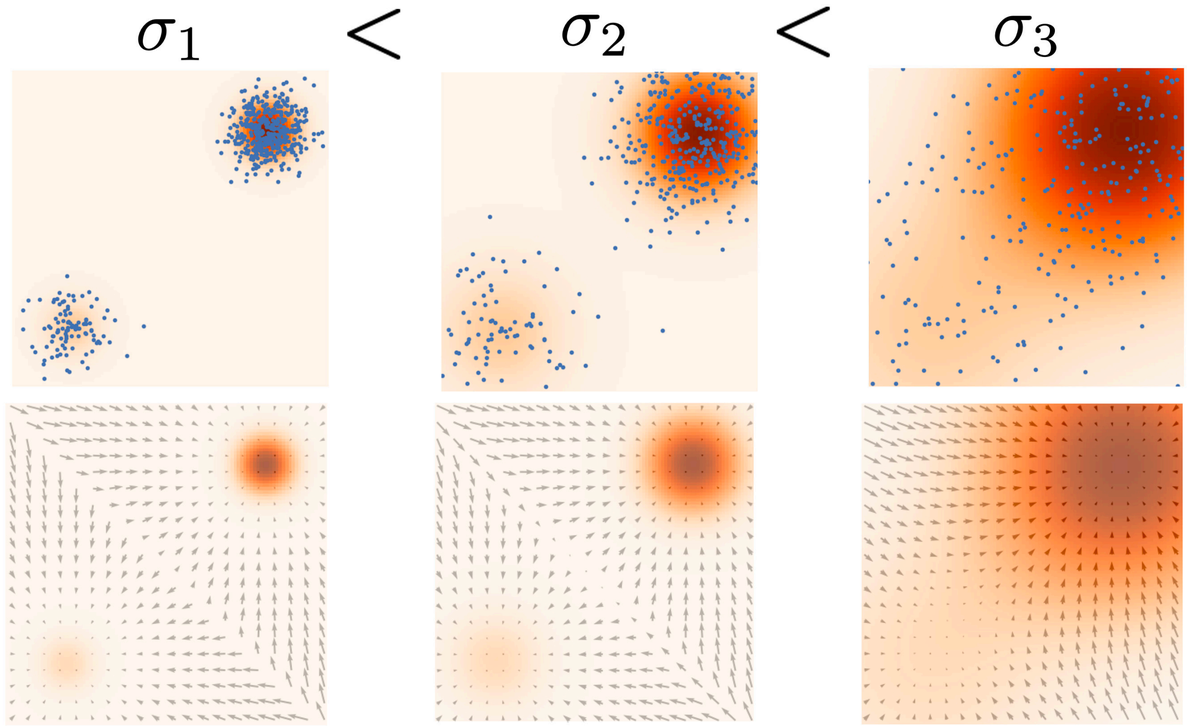
$$p_{\sigma_t}(\mathbf{x}) = \int p(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y} = \int p(\mathbf{y})p(\mathbf{x}\,|\,\mathbf{y})\mathrm{d}\mathbf{y}$$

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



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

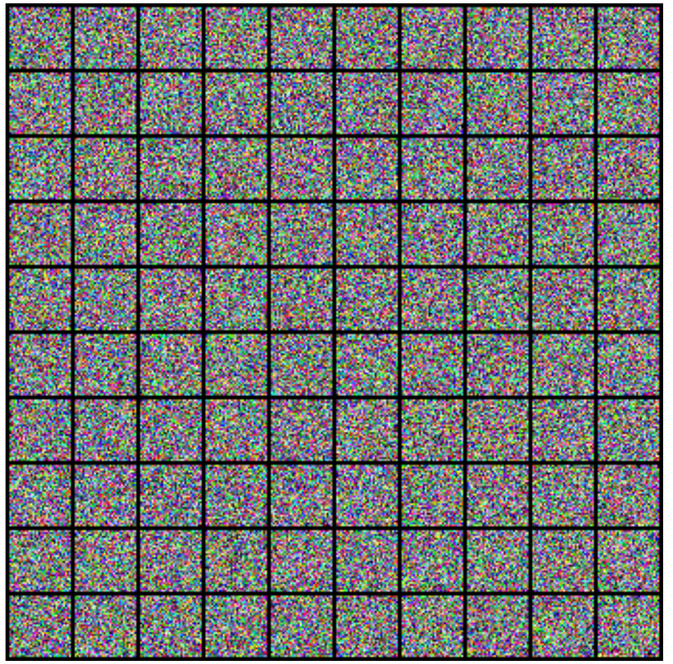
Annealed Langeving Dynamics

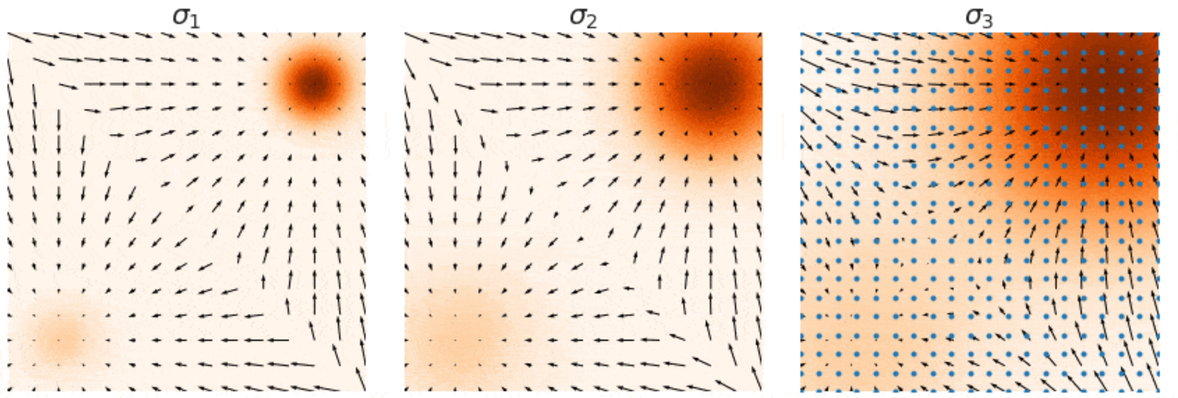
- Like before, but we start sampling from larger noise, which we gradually decrease

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

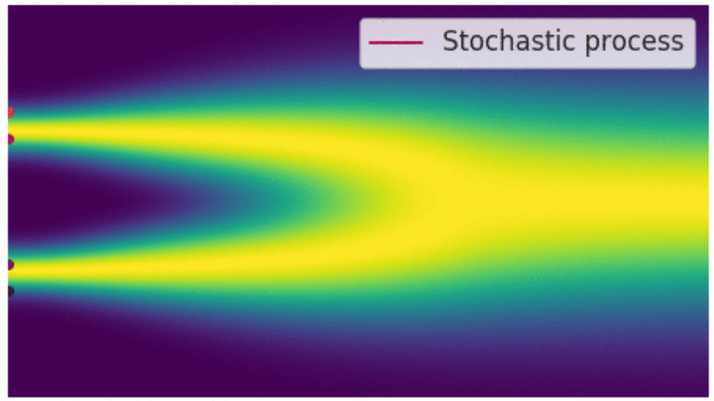
- Adding noise is important, but why ‘hardcode’?
- By generalising with infinite noise scales, we can
  - get higher quality samples
  - exact log-likelihood computation
  - controllable generation with inverse problem solving
- A stochastic process defines a process of generating infinite noise scales

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

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

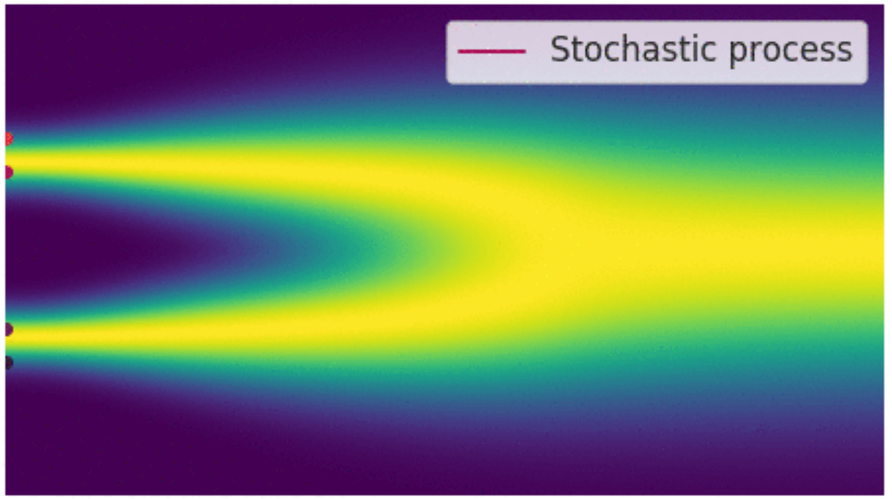
- Solutions to the SDEs are stochastic random variables $\{\mathbf{x}(t)\}_{t\in[0,T]}$
- These random variables are stochastic trajectories over time
- The probability density of $\mathbf{x}(t)$ is $p_t(\mathbf{x})$ (analogous to $p_{\sigma_i}(\mathbf{x})$ for the discrete case)
- $p_0(\mathbf{x})$ 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?

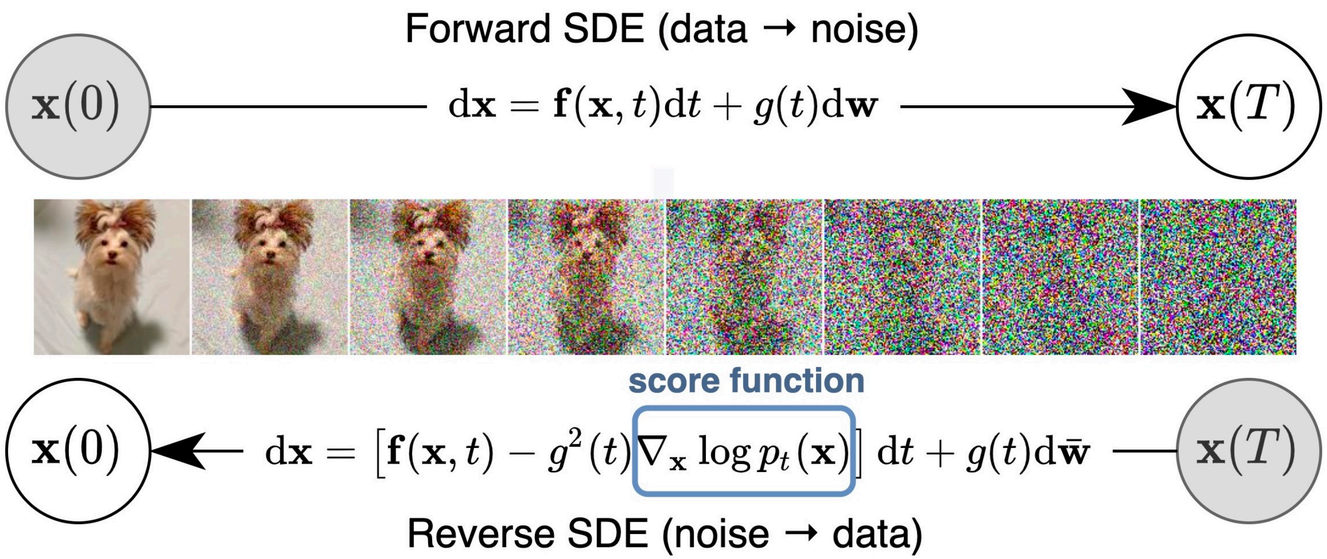


From reverse SDE noise to data

- 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

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

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

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

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

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

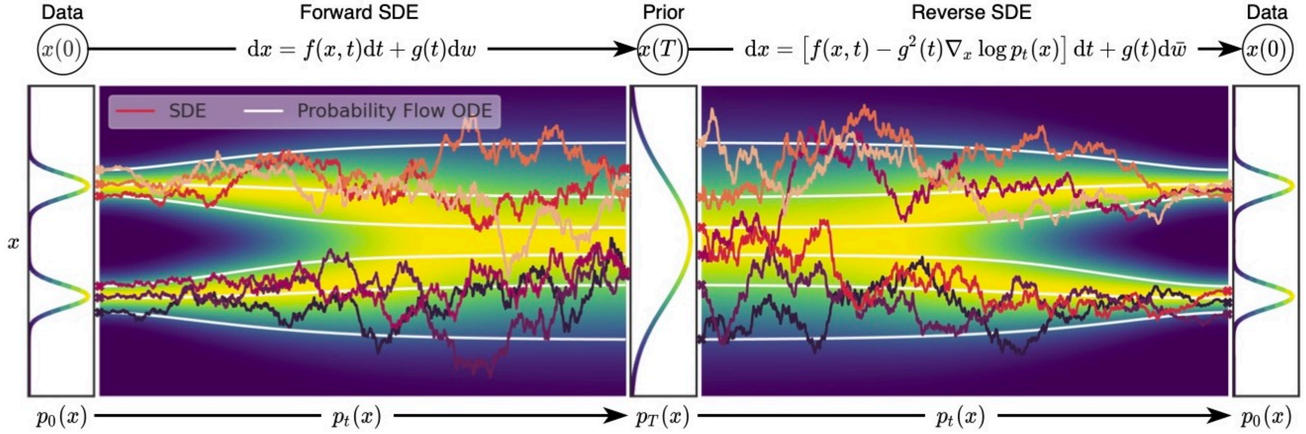
- With better sampling procedures for SDEs and better architectures, one gets state-of-the-art in generated samples

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



Qualitative examples

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

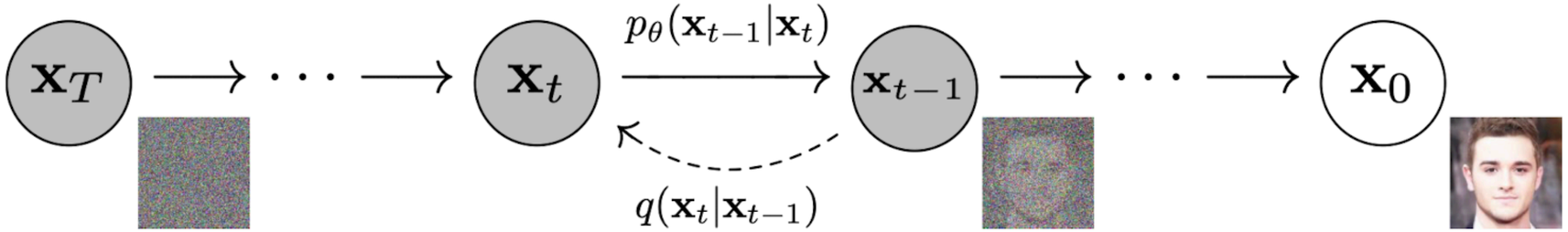


Figure 2: The directed graphical model considered in this work.

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

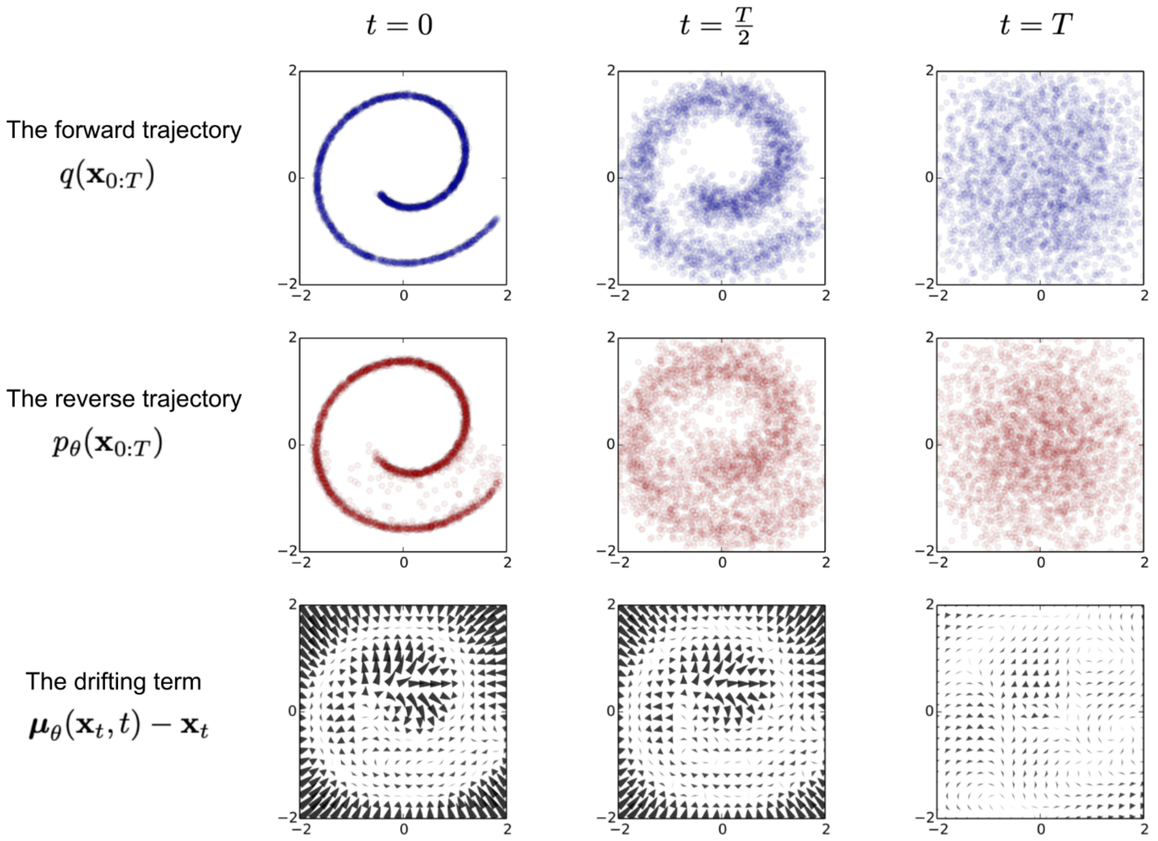
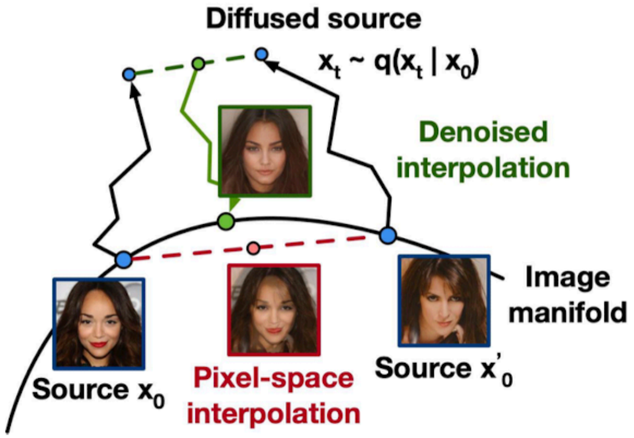


Fig. 3. An example of training a diffusion model for modeling a 2D swiss roll data. (Image source: Sohl-Dickstein et al., 2015)

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

Qualitative results



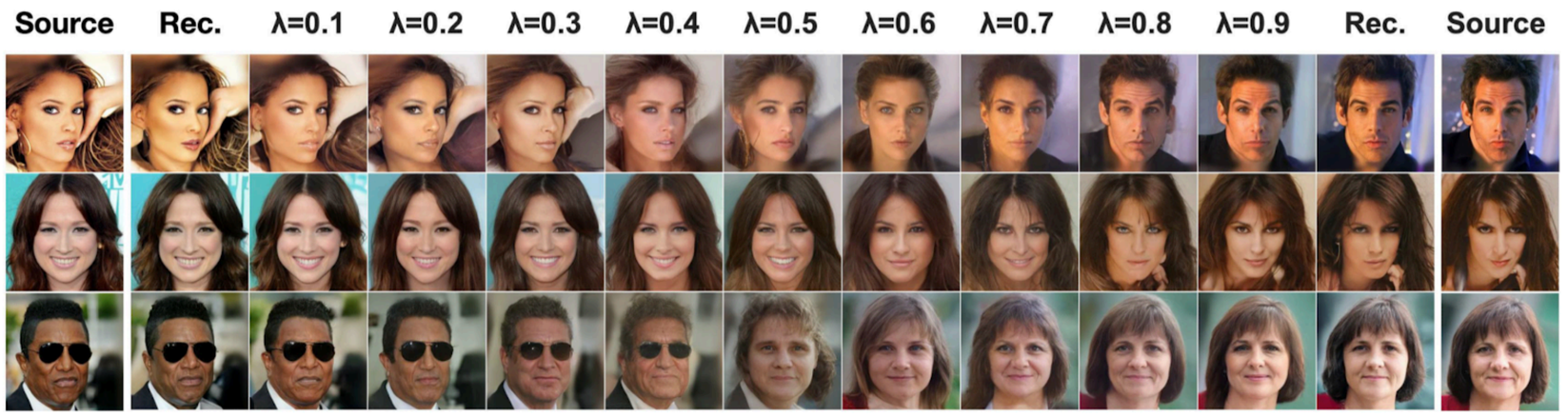


Figure 8: Interpolations of CelebA-HQ 256x256 images with 500 timesteps of diffusion.

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