

# CENG501 – Deep Learning

Week 13

Spring 2026

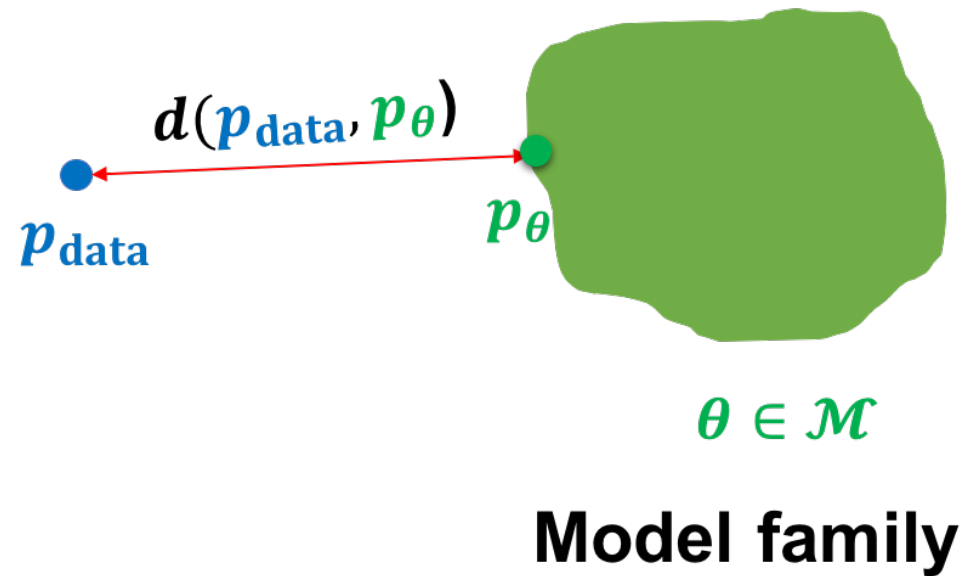
Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on CENG501

# Generative Modeling

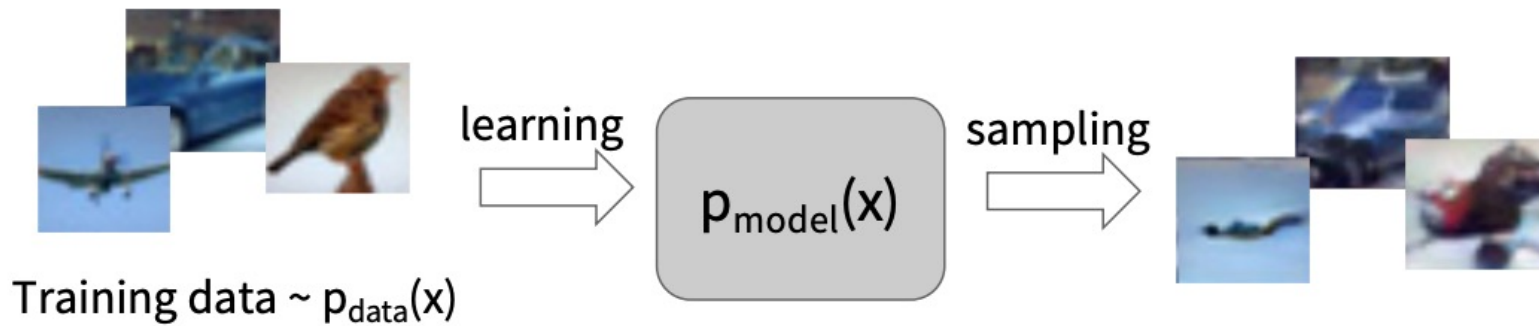
- Learning the probability distribution of data



$$p_{\theta} \equiv p_{\text{model}}$$

# Generative Modeling

- Learning the probability distribution of data



Formulate as density estimation problems:

- Explicit density estimation: explicitly define and solve for  $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from  $p_{\text{model}}(x)$  without explicitly defining it.

Previously on CENG501

# Taxonomy of Generative Models

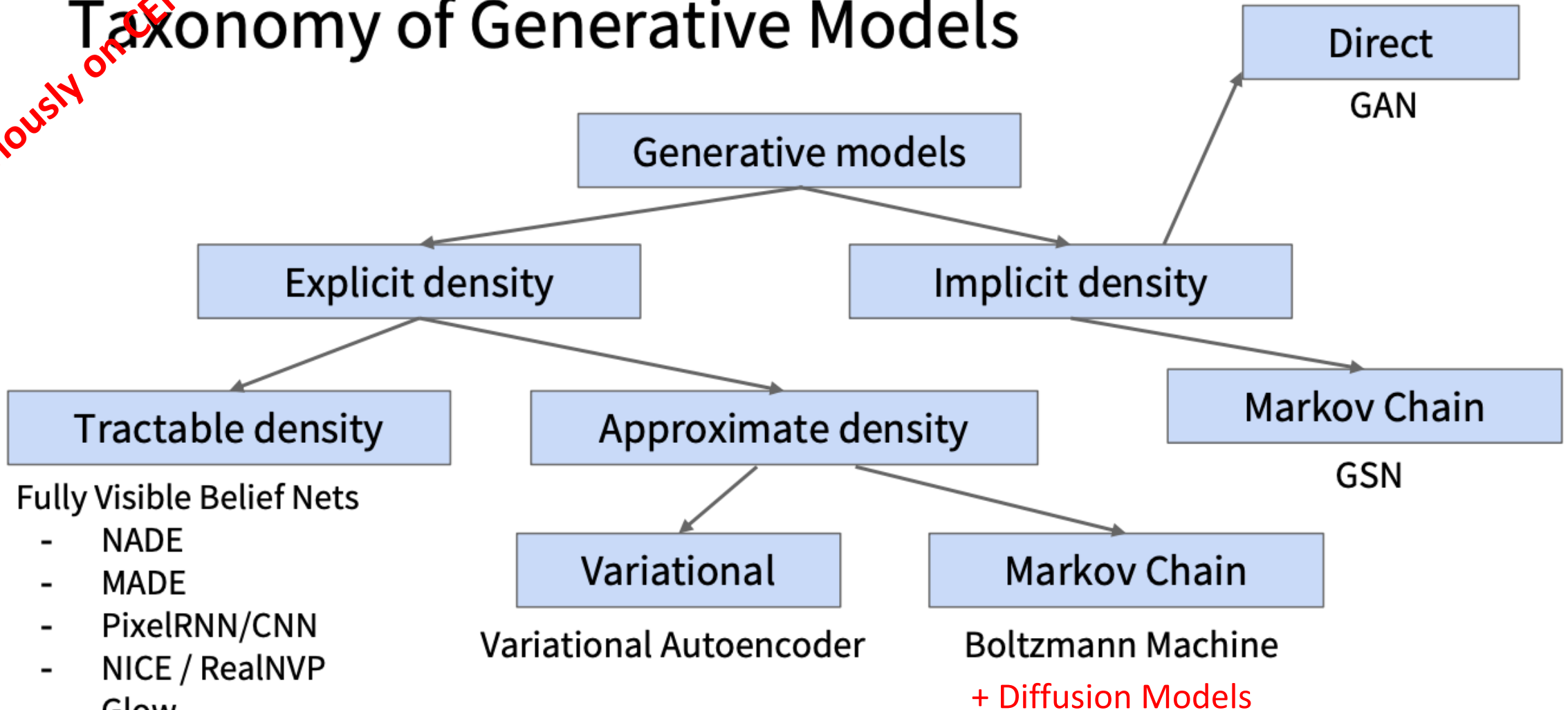


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Fully visible belief network (FVBN)

Explicit density model

$$p(\mathbf{x}) = p(x_1, x_2, \dots, x_n)$$

↑  
Likelihood of  
image  $\mathbf{x}$

↑  
Joint likelihood of each  
pixel in the image



Previously on CENG501

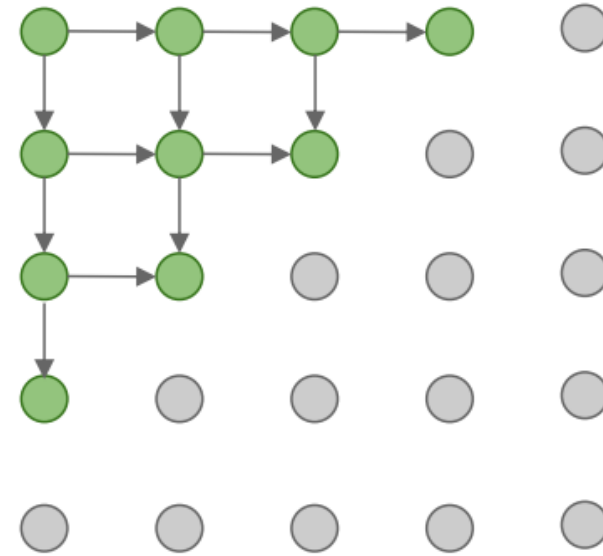
# PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!



# PixelCNN

[van der Oord et al. 2016]

Previously on CENG501

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN (can parallelize convolutions since context region values known from training images)

Generation is still slow:  
For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

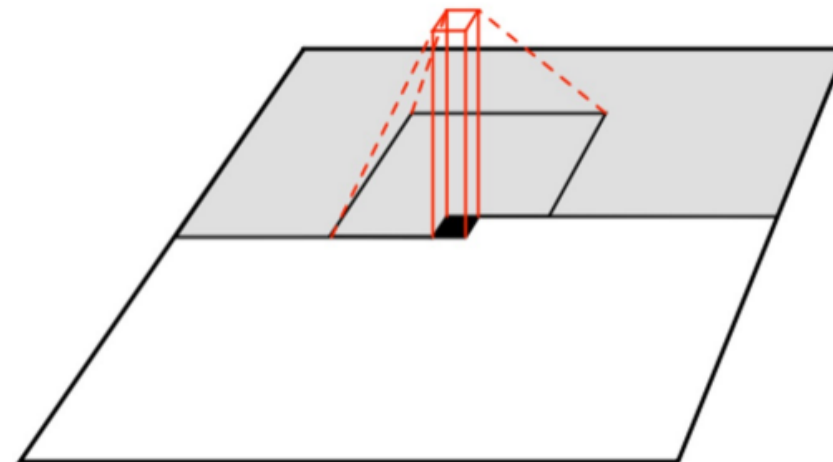


Figure copyright van der Oord et al., 2016. Reproduced with permission.

# PixelRNN and PixelCNN

## Pros:

- Can explicitly compute likelihood  $p(x)$
- Easy to optimize
- Good samples

## Con:

- Sequential generation => slow

## Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc...

## See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

Previously in CEN 501

# Autoregressive Models vs Variational Autoencoders

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent  $z$ :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

No dependencies among pixels, can generate all pixels at the same time!

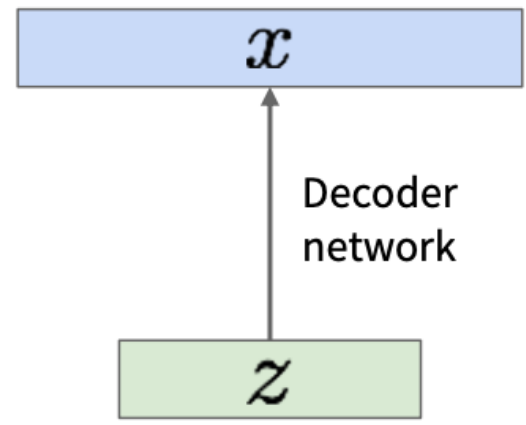
Cannot optimize directly, derive and optimize lower bound on likelihood instead

Why latent  $z$ ?

Previously on CENG501

# Variational Autoencoders (VAEs)

Sample from true conditional  
 $p_{\theta^*}(x | z^{(i)})$



Sample from true prior  
 $z^{(i)} \sim p_{\theta^*}(z)$

We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How to train the model?

Learn model parameters to maximize likelihood of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Q: What is the problem with this?

Intractable!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders (VAEs)

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Posterior density also intractable:  $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

Solution: In addition to modeling  $p_{\theta}(x|z)$ , learn  $q_{\phi}(z|x)$  that approximates the true posterior  $p_{\theta}(z|x)$ .

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data  $x$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Previously on CENG501

# Variational Autoencoders (VAEs)

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} [\log p_{\theta}(x^{(i)})] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

Decoder:  
reconstruct  
the input data

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

Encoder:  
make approximate  
posterior distribution  
close to prior

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take  
gradient of and optimize! ( $p_{\theta}(x|z)$  differentiable,  
KL term differentiable)

Previously on CENG501

# Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

$$\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

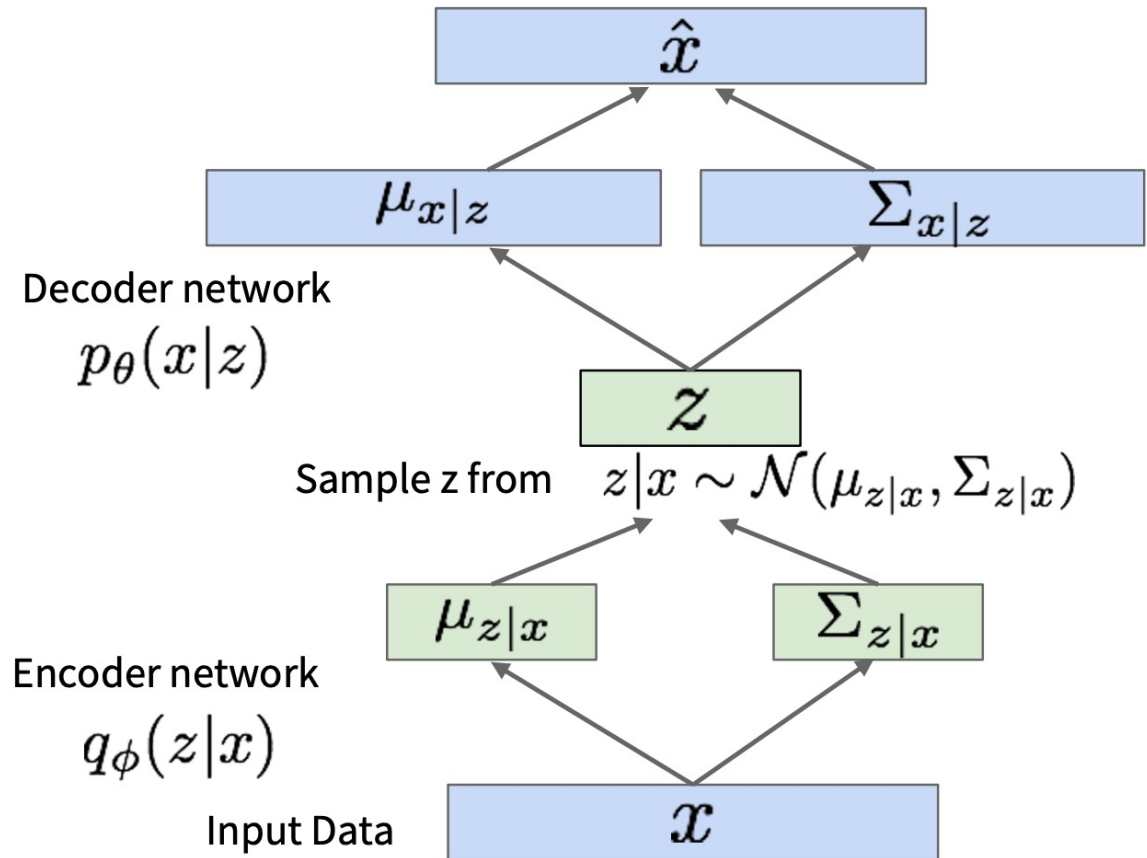
$\mathcal{L}(x^{(i)}, \theta, \phi)$

For every minibatch of input data: compute this forward pass, and then backprop!

Reparameterization trick to make sampling differentiable:

$$\text{Sample } \epsilon \sim \mathcal{N}(0, I)$$

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$



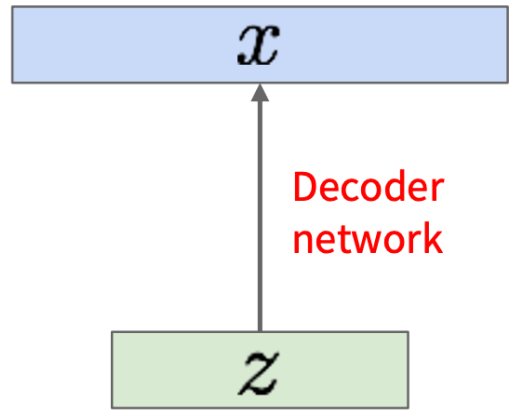
Previously on CENG501

# Variational Autoencoders (VAEs)

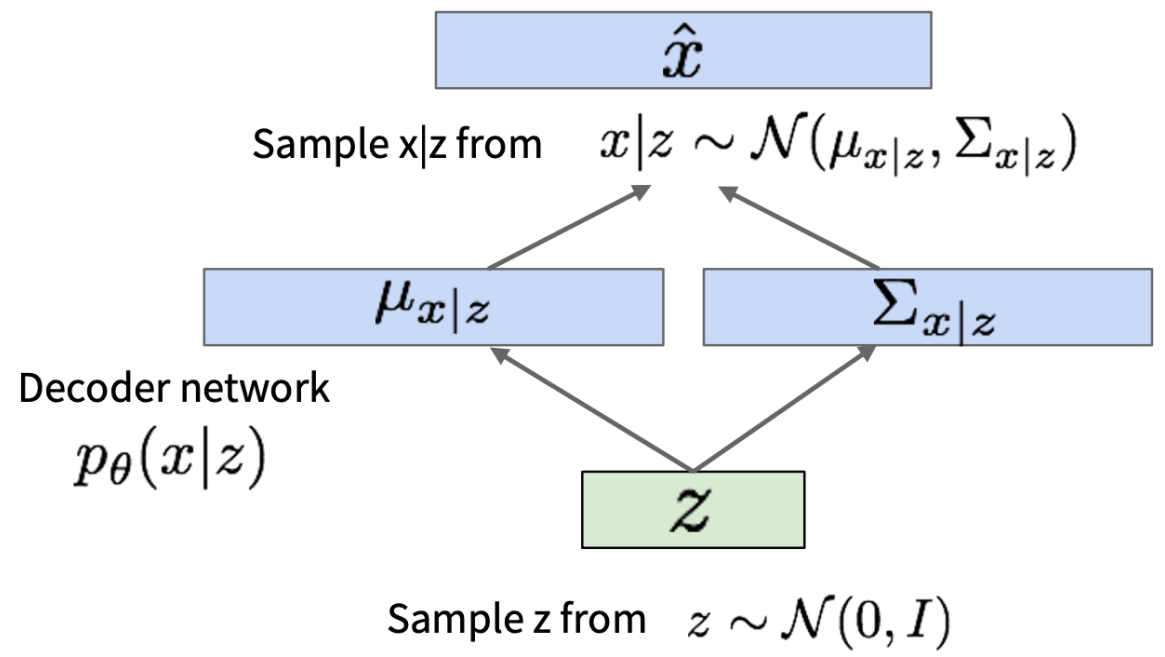
Our assumption about data generation process

Sample from true conditional  $p_{\theta^*}(x | z^{(i)})$

Sample from true prior  $z^{(i)} \sim p_{\theta^*}(z)$



Now given a trained VAE:  
use decoder network & sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Previously on CENG501

# Variational Autoencoders (VAEs)

Diagonal prior on  $z$   
=> independent latent variables

Different dimensions of  $z$  encode interpretable factors of variation

Also good feature representation that can be computed using  $q_\phi(z|x)$ !

Degree of smile

Vary  $z_1$



Vary  $z_2$

Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders (VAEs)

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of  $q(z|x)$ , can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

Previously on CENG501

# Normalizing Flow

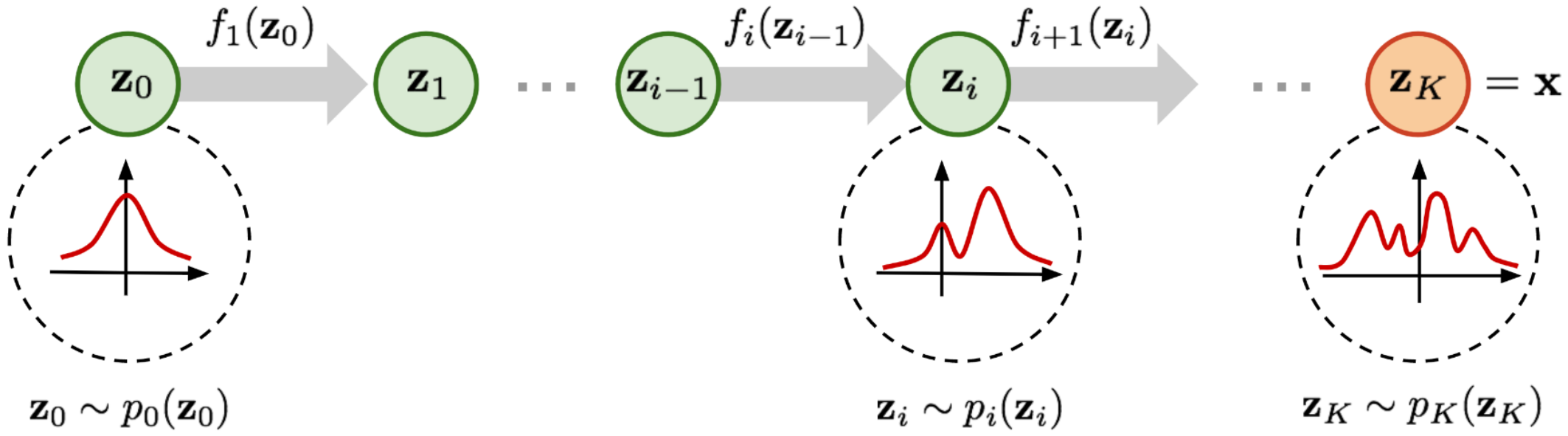


Fig. 2. Illustration of a normalizing flow model, transforming a simple distribution  $p_0(\mathbf{z}_0)$  to a complex one  $p_K(\mathbf{z}_K)$  step by step.

Previously on CENG501

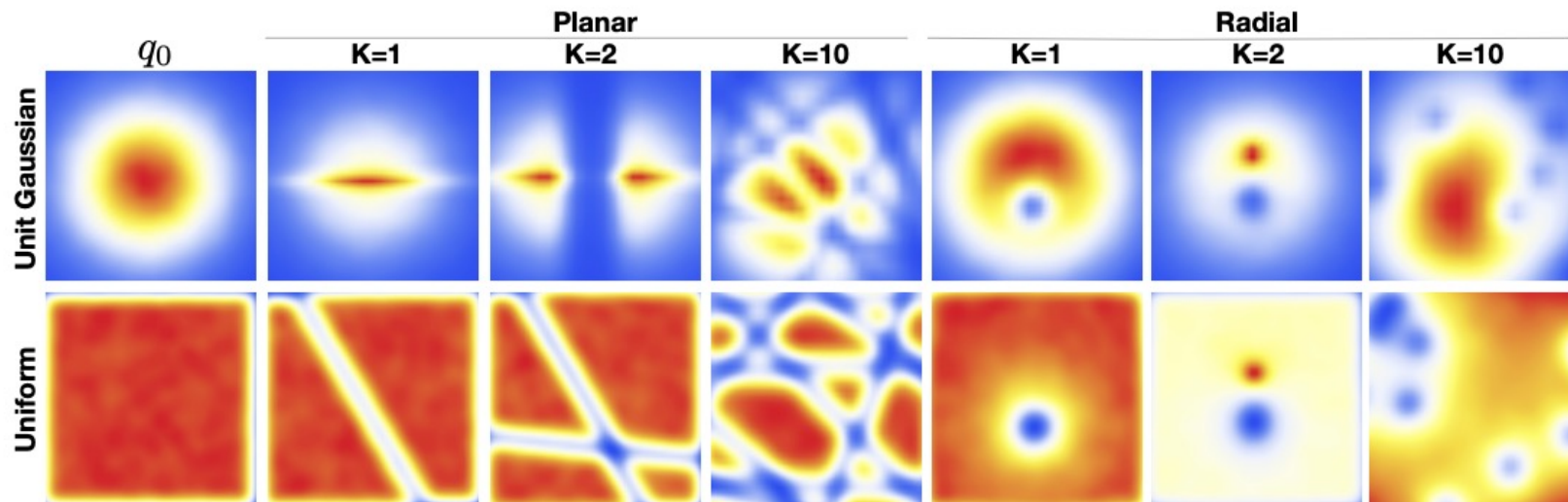


Figure 1. Effect of normalizing flow on two distributions.

Figure: "Variational Inference with Normalizing Flows", 2016.

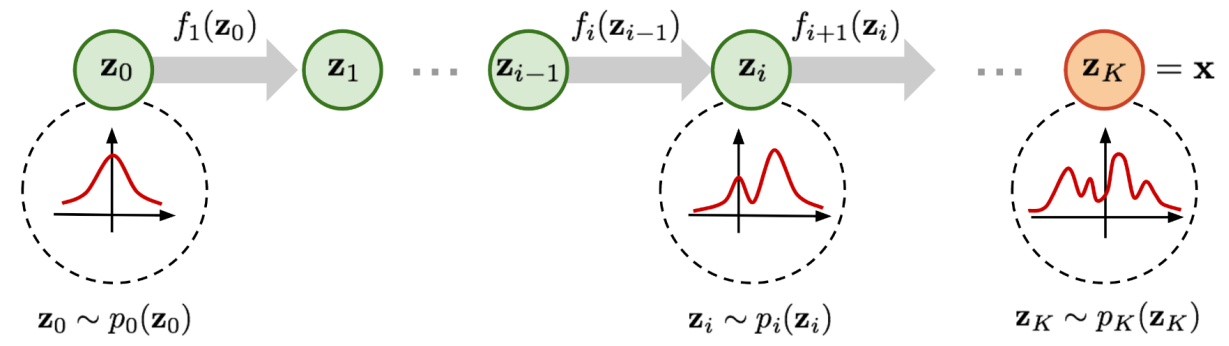
$$\arg \max_{\theta} \sum_{i=0}^N \log(p_{\theta}(x_i))$$

### Pseudo-code

1.  $\mathbf{x} \leftarrow$  Sample a batch

2.  $\mathbf{z}_0 \sim p_{\theta}(\mathbf{z}_0 | \mathbf{x})$

3. loss:  $\log p_{\theta}(\mathbf{x}) = \log p_z(\mathbf{z}_0) + \log \left| \det \left( \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$



# RealNVP (Real-valued Non-Volume Preserving; Dinh et al., 2017)

The **RealNVP** (Real-valued Non-Volume Preserving; Dinh et al., 2017) model implements a normalizing flow by stacking a sequence of invertible bijective transformation functions. In each bijection  $f : \mathbf{x} \mapsto \mathbf{y}$ , known as *affine coupling layer*, the input dimensions are split into two parts:

- The first  $d$  dimensions stay same;
- The second part,  $d + 1$  to  $D$  dimensions, undergo an affine transformation ("scale-and-shift") and both the scale and shift parameters are functions of the first  $d$  dimensions.

$$\begin{aligned}\mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})\end{aligned}$$

where  $s(\cdot)$  and  $t(\cdot)$  are *scale* and *translation* functions and both map  $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$ . The  $\odot$  operation is the element-wise product.

# Normalizing Flows

Previously on CENG501

- Pros:
  - Successful results in estimating high-dimensional densities
  - Stable training compared to GANs
  - Easier to converge compared to GANs & VAEs
- Cons:
  - Latent space is not lower-dimensional than the input => may not be useful in some applications (e.g., image compression)
  - Fails in estimating the likelihood of out-of-distribution samples
  - Invertibility may not be guaranteed in practice due to numerical imprecision
  - Lower quality generation

# Today

- (Deep) Generative Models
  - Autoregressive models
  - Variational AEs
  - Flow Models
  - Generative Adversarial Networks
  - Energy-based Models
  - Diffusion Models

# Administrative Notes

- Project next steps:
  - Milestones:
    1. Milestone (April 10, midnight):
      - Read & understand the paper
      - Download the datasets
      - Prepare the Readme file excluding the results & conclusion
    2. Milestone (May 4, midnight)
      - The results of the first experiment
    3. Milestone (June 1, midnight)
      - Final report (Readme file)
      - Repo with all code & trained models

# Firindan Sıcak Sıcak

We report an exploratory red-teaming study of autonomous language model-powered agents deployed in a live laboratory environment with persistent memory, email accounts, Discord access, file systems, and shell execution. Over a two-week period, twenty AI researchers interacted with the agents under benign and adversarial conditions. Focusing on failures emerging from the integration of language models with autonomy, tool use, and multi-party communication, we document eleven representative case studies. **Observed behaviors include unauthorized compliance with non-owners, disclosure of sensitive information, execution of destructive system-level actions, denial-of-service conditions, uncontrolled resource consumption, identity spoofing vulnerabilities, cross-agent propagation of unsafe practices, and partial system takeover. In several cases, agents reported task completion while the underlying system state contradicted those reports.** We also report on some of the failed attempts. Our findings establish the existence of security-, privacy-, and governance-relevant vulnerabilities in realistic deployment settings. These behaviors raise unresolved questions regarding accountability, delegated authority, and responsibility for downstream harms, and warrant urgent attention from legal scholars, policymakers, and researchers across disciplines. This report serves as an initial empirical contribution to that broader conversation.

## Agents of Chaos

Natalie Shapira<sup>1</sup> Chris Wendler<sup>1</sup> Avery Yen<sup>1</sup>  
Gabriele Sarti<sup>1</sup> Koyena Pal<sup>1</sup> Olivia Floody<sup>2</sup> Adam Belfki<sup>1</sup>  
Alex Loftus<sup>1</sup> Aditya Ratan Jannali<sup>2</sup> Nikhil Prakash<sup>1</sup> Jasmine Cui<sup>1</sup>  
Giordano Rogers<sup>1</sup> Jannik Brinkmann<sup>1</sup> Can Rager<sup>2</sup> Amir Zur<sup>3</sup> Michael Ripa<sup>1</sup>  
Aruna Sankaranarayanan<sup>8</sup> David Atkinson<sup>1</sup> Rohit Gandikota<sup>1</sup> Jaden Fiotto-Kaufman<sup>1</sup>  
EunJeong Hwang<sup>4,13</sup> Hadas Orgad<sup>5</sup> P Sam Sahil<sup>2</sup> Negev Taglicht<sup>2</sup> Tomer Shabtay<sup>2</sup>  
Atai Ambus<sup>2</sup> Nitay Alon<sup>6,7</sup> Shiri Oron<sup>2</sup> Ayelet Gordon-Tapiero<sup>6</sup> Yotam Kaplan<sup>6</sup>  
Vered Shwartz<sup>4,13</sup> Tamar Rott Shaham<sup>8</sup> Christoph Riedl<sup>1</sup> Reuth Mirsky<sup>9</sup>  
Maarten Sap<sup>10</sup> David Manheim<sup>11,12</sup> Tomer Ullman<sup>5</sup> David Bau<sup>1</sup>

<sup>1</sup> Northeastern University <sup>2</sup> Independent Researcher <sup>3</sup> Stanford University  
<sup>4</sup> University of British Columbia <sup>5</sup> Harvard University <sup>6</sup> Hebrew University  
<sup>7</sup> Max Planck Institute for Biological Cybernetics <sup>8</sup> MIT <sup>9</sup> Tufts University  
<sup>10</sup> Carnegie Mellon University <sup>11</sup> Alter <sup>12</sup> Technion <sup>13</sup> Vector Institute

<https://arxiv.org/pdf/2602.20021>

# Firindan Sıcak Sıcak

## Discovering Differences in Strategic Behavior between Humans and LLMs

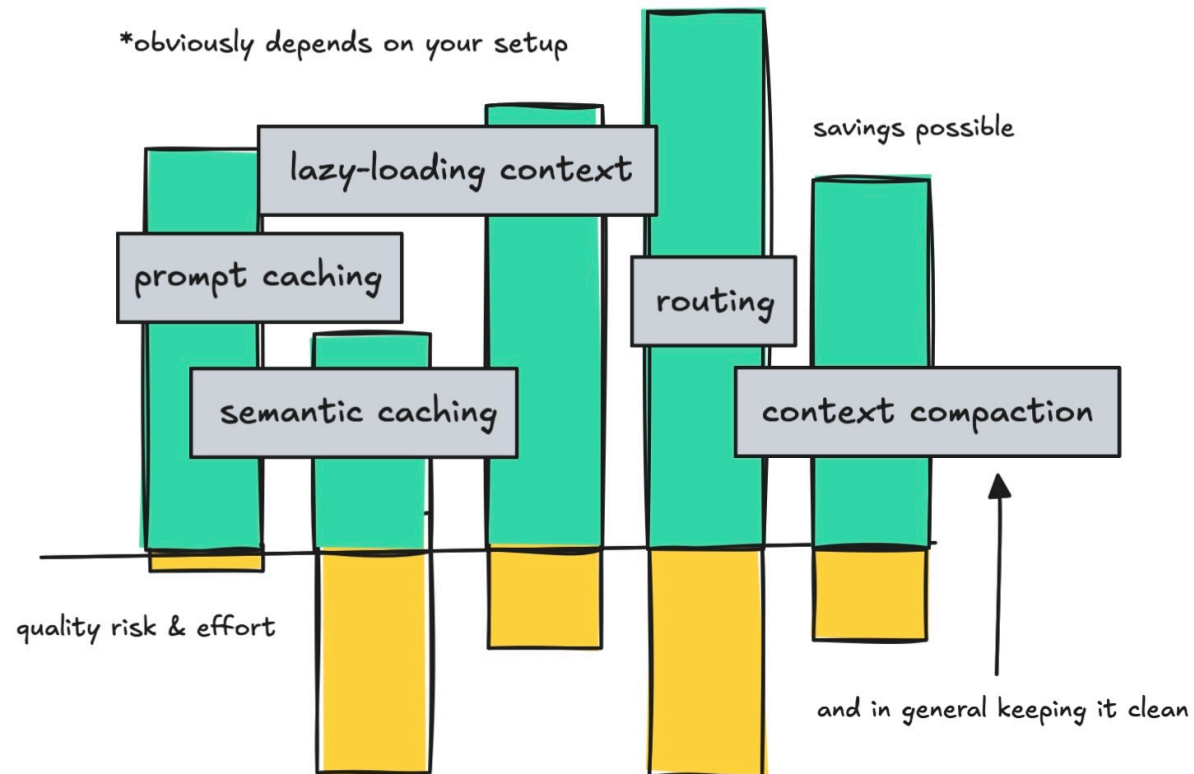
Caroline Wang<sup>1,\*</sup>, Daniel Kasenberg<sup>2</sup>, Kim Stachenfeld<sup>2</sup> and Pablo Samuel Castro<sup>2</sup>

<sup>\*</sup>Work performed as a student researcher, <sup>1</sup>The University of Texas at Austin, <sup>2</sup>Google DeepMind

As Large Language Models (LLMs) are increasingly deployed in social and strategic scenarios, it becomes critical to understand where and why their behavior diverges from that of humans. While behavioral game theory (BGT) provides a framework for analyzing behavior, existing models do not fully capture the idiosyncratic behavior of humans or black-box, non-human agents like LLMs. We employ AlphaEvolve, a cutting-edge program discovery tool, to directly discover *interpretable* models of human and LLM behavior from data, thereby enabling open-ended discovery of structural factors driving human and LLM behavior. Our analysis on iterated rock-paper-scissors reveals that frontier LLMs can be capable of deeper strategic behavior than humans. These results provide a foundation for understanding structural differences driving differences in human and LLM behavior in strategic interactions.

<https://arxiv.org/pdf/2602.10324>

# Firindan Sıcak Sıcak



<https://towardsdatascience.com/agent-ai-how-to-save-on-tokens/>

# Taxonomy of Generative Models

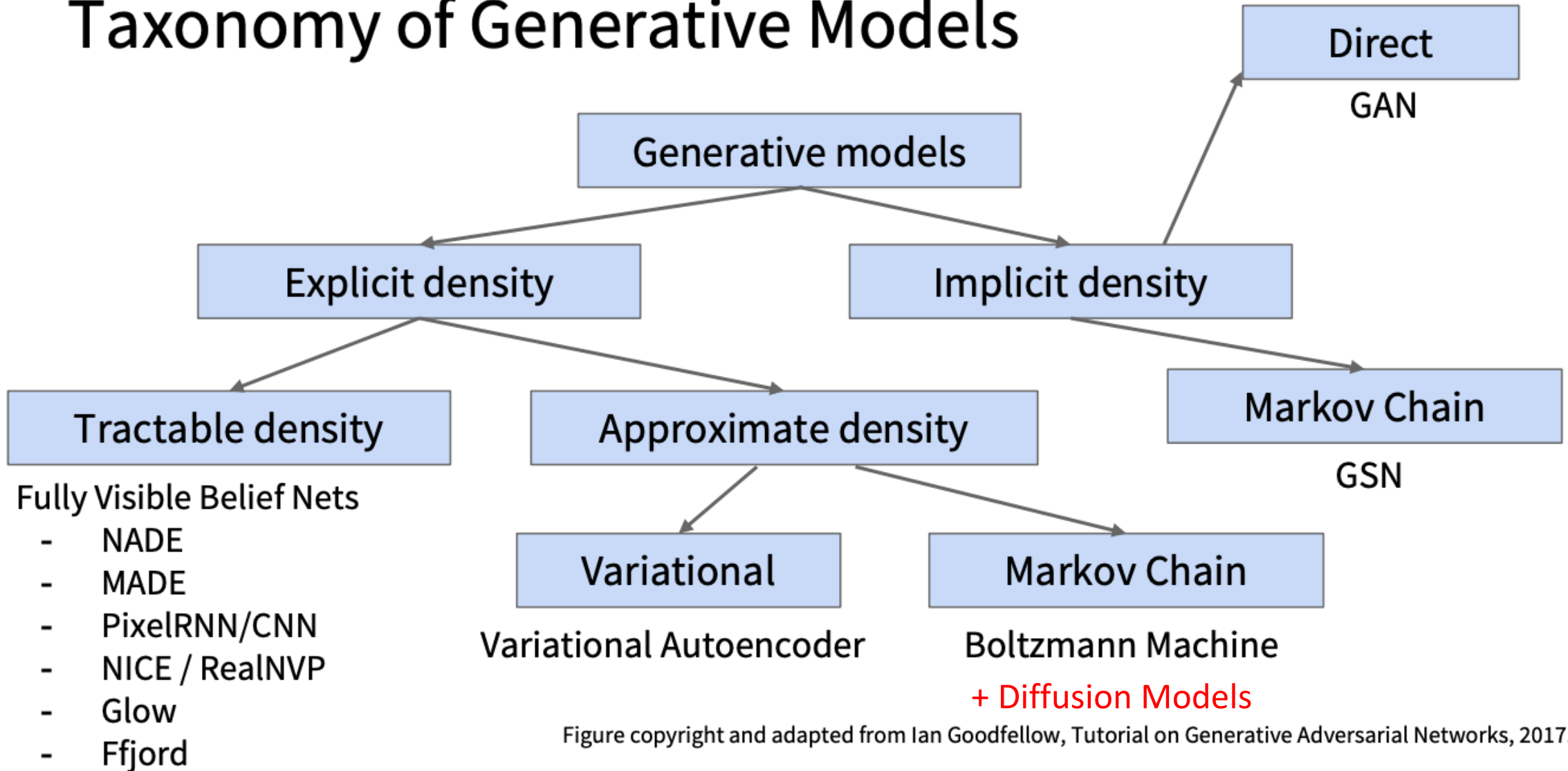
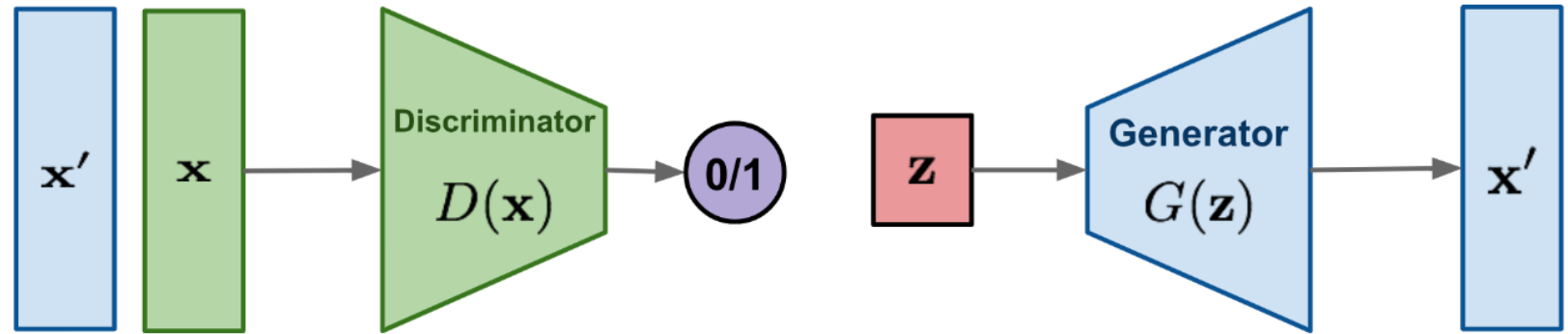
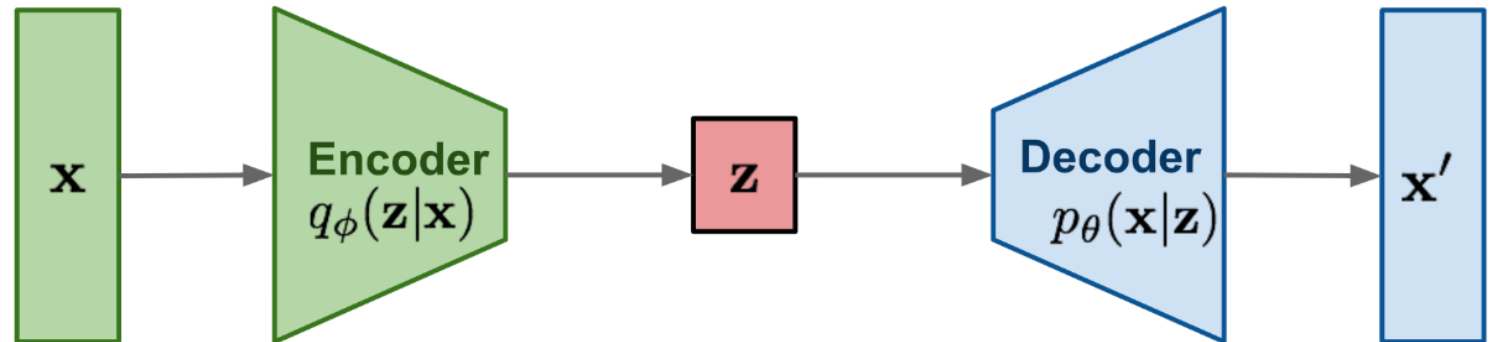


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

**GAN:** minimax the classification error loss.



**VAE:** maximize ELBO.



**Flow-based generative models:** minimize the negative log-likelihood

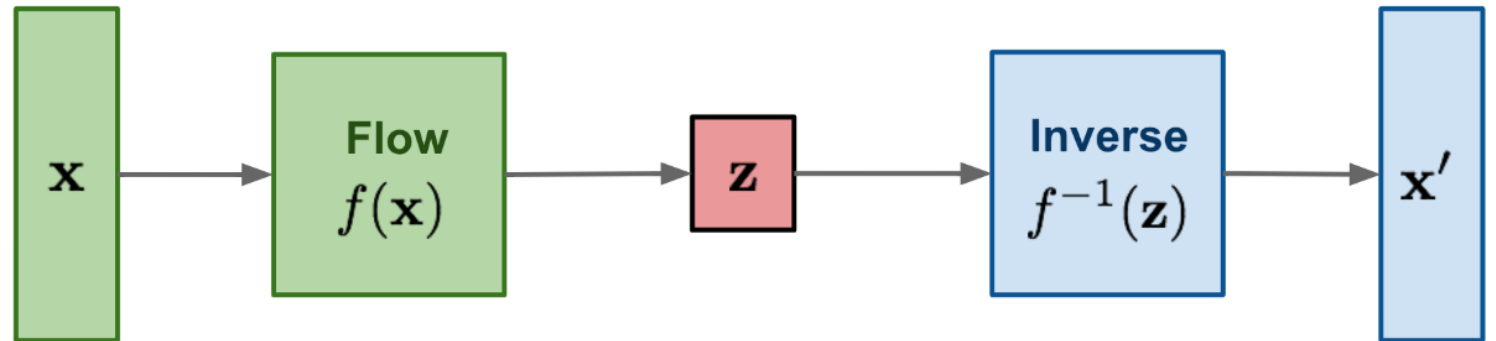


Figure: <https://lilianweng.github.io/posts/2018-10-13-flow-models/>

# Generative Adversarial Networks

# Generative Adversarial Networks (GANs)

- Originally proposed by Ian Goodfellow in 2014
- Won the “Test of Time” award at NeurIPS2024
  - <https://blog.neurips.cc/2024/11/27/announcing-the-neurips-2024-test-of-time-paper-awards/>
- It all started in a pub 😊
  - Full story here: <https://x.com/sherjilozair/status/1864013580624113817>

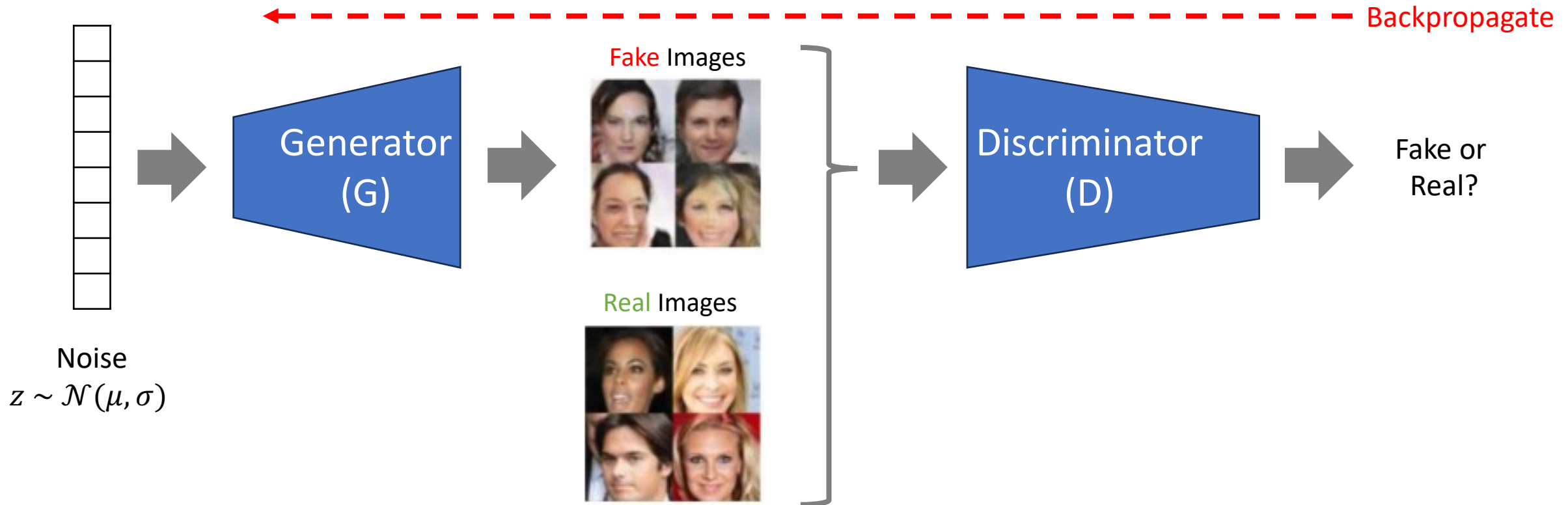
---

## Generative Adversarial Nets

---

Ian J. Goodfellow, Jean Pouget-Abadie\*, Mehdi Mirza, Bing Xu, David Warde-Farley,  
Sherjil Ozair,† Aaron Courville, Yoshua Bengio‡  
Département d’informatique et de recherche opérationnelle  
Université de Montréal  
Montréal, QC H3C 3J7

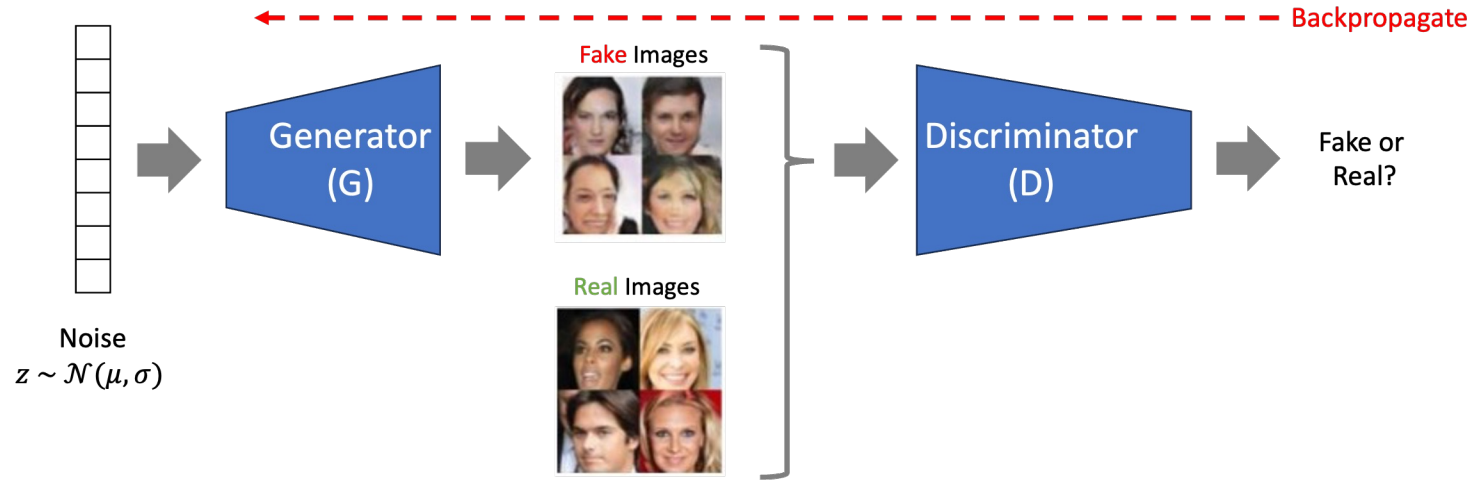
# Generative Adversarial Networks (GANs)



We have two networks:

- Generator (G): Generates a fake image given a noise (embedding) vector ( $z$ )
- Discriminator (D): Discriminates whether an image is fake or real.

# Generative Adversarial Networks (GANs)



- With two competing networks, we solve the following minimax game:

$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

- Discriminator's objective:

$$\max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

- Generator's objective:

$$\min_G V(D, G) = E_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

$D(x)$ : Probability that  $x$  is real (came from data).

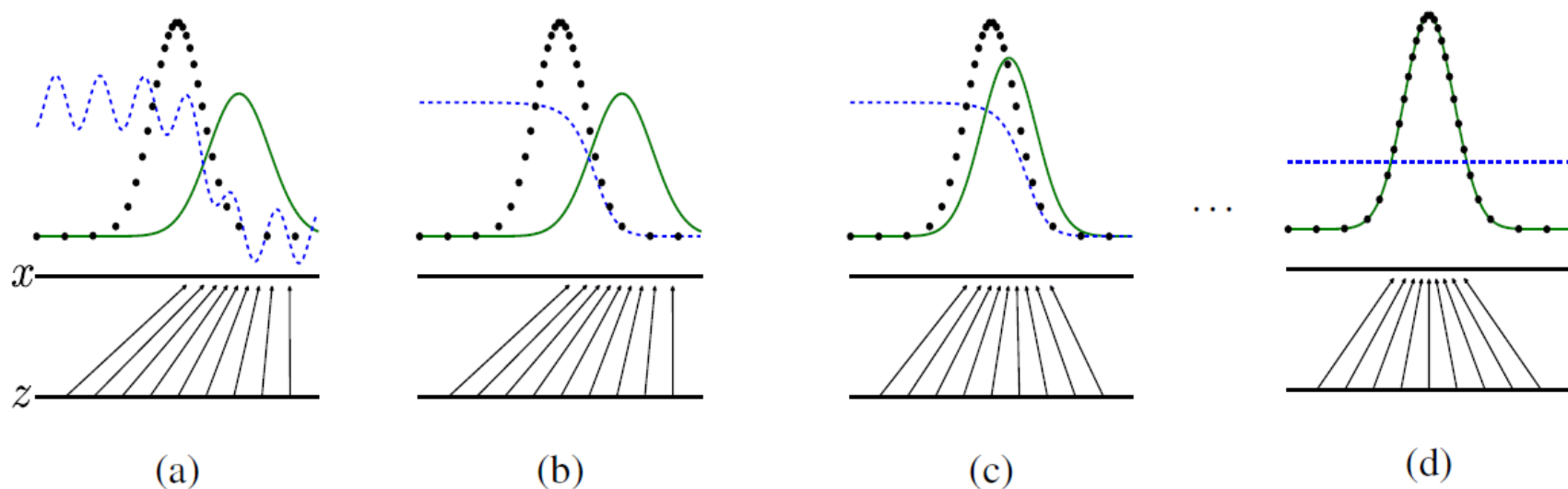


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution ( $D$ , blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line)  $p_x$  from those of the **g**enerative distribution  $p_g$  ( $G$ ) (green, solid line). The lower horizontal line is the domain from which  $z$  is sampled, in this case uniformly. The horizontal line above is part of the domain of  $x$ . The upward arrows show how the mapping  $x = G(z)$  imposes the non-uniform distribution  $p_g$  on transformed samples.  $G$  contracts in regions of high density and expands in regions of low density of  $p_g$ . (a) Consider an adversarial pair near convergence:  $p_g$  is similar to  $p_{\text{data}}$  and  $D$  is a partially accurate classifier. (b) In the inner loop of the algorithm  $D$  is trained to discriminate samples from data, converging to  $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$ . (c) After an update to  $G$ , gradient of  $D$  has guided  $G(z)$  to flow to regions that are more likely to be classified as data. (d) After several steps of training, if  $G$  and  $D$  have enough capacity, they will reach a point at which both cannot improve because  $p_g = p_{\text{data}}$ . The discriminator is unable to differentiate between the two distributions, i.e.  $D(x) = \frac{1}{2}$ .

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator,  $k$ , is a hyperparameter. We used  $k = 1$ , the least expensive option, in our experiments.

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log (1 - D(G(z^{(i)}))) \right].$$

**end for**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Update the generator by descending its stochastic gradient:

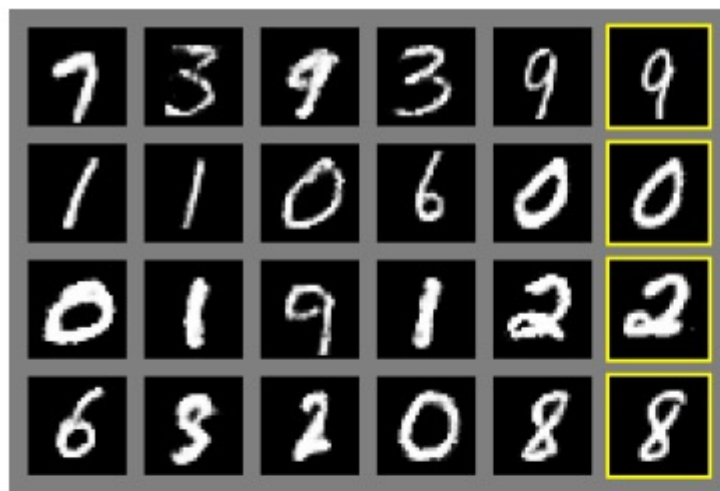
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)}))).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Discriminator

Generator



a)



b)



c)



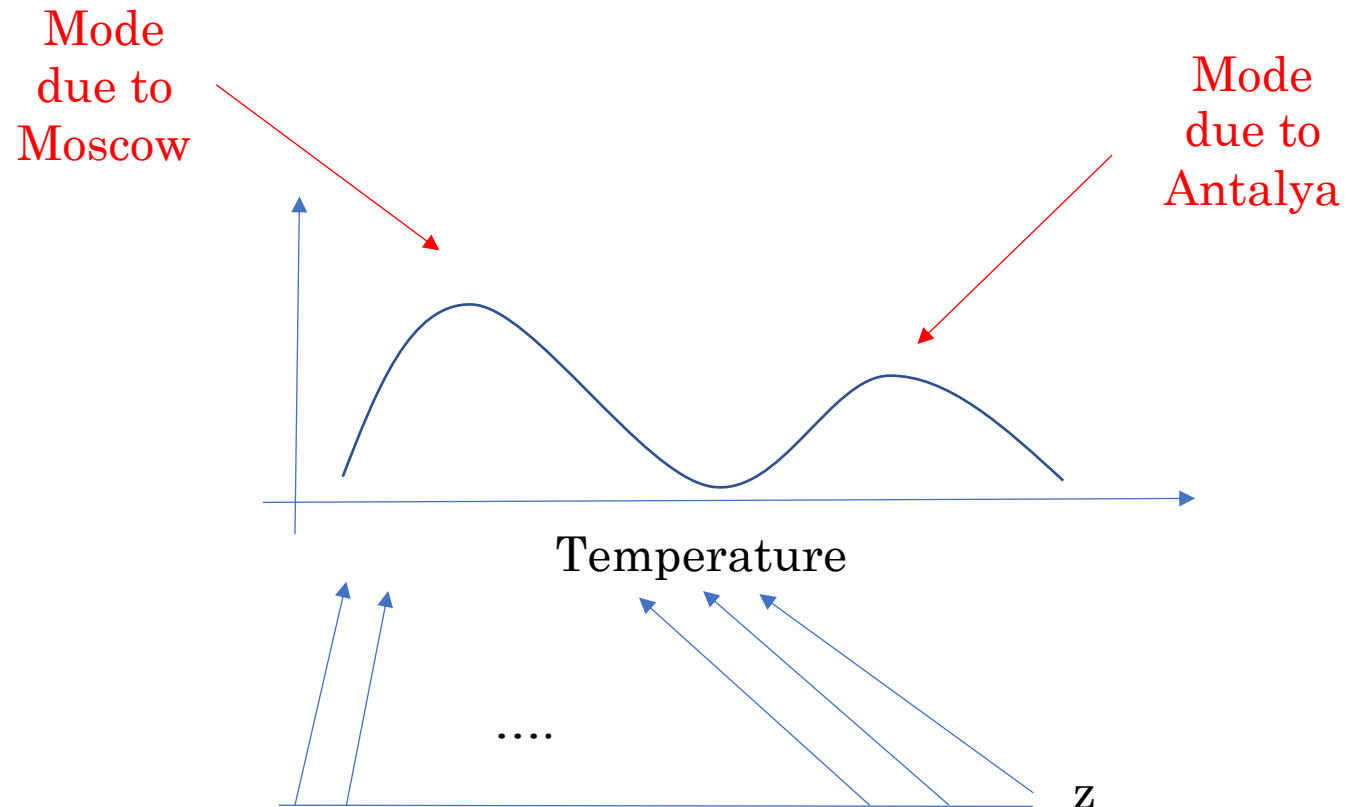
d)

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and “deconvolutional” generator)

Fig: Goodfellow et al., 2014.

# Mode collapse in GANs

- Problem:
  - The generator network maps the different  $z$  (embedding/noise) values into similar images.



# Mode collapse in GANs

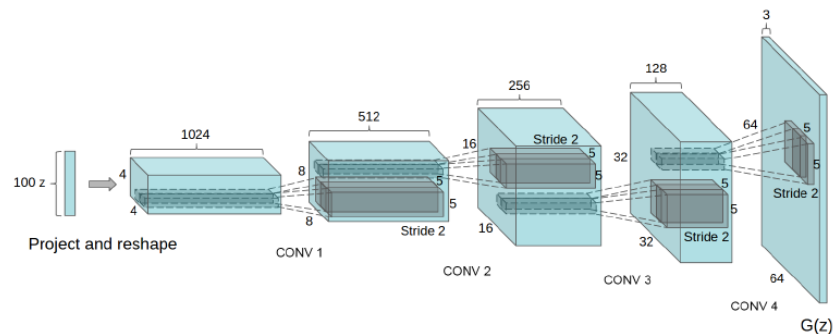
- Solutions:
  - Changing the training procedure (use batch discrimination instead of individual discrimination)
  - Experience replay (show old fake images again and again)
  - Use a different loss (+ enforce diversity)
  - ...
- Other tips and tricks:
  - <https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b>

# Deep Convolutional GAN

- GAN with convolutional layers
- More stable

## Architecture guidelines for stable Deep Convolutional GANs

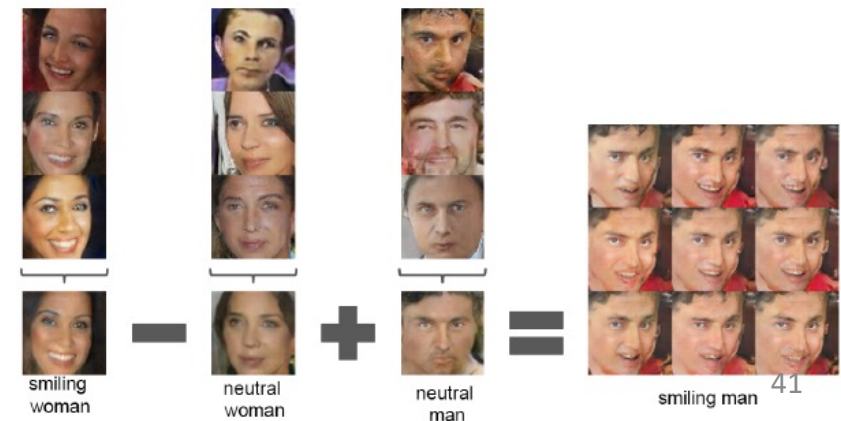
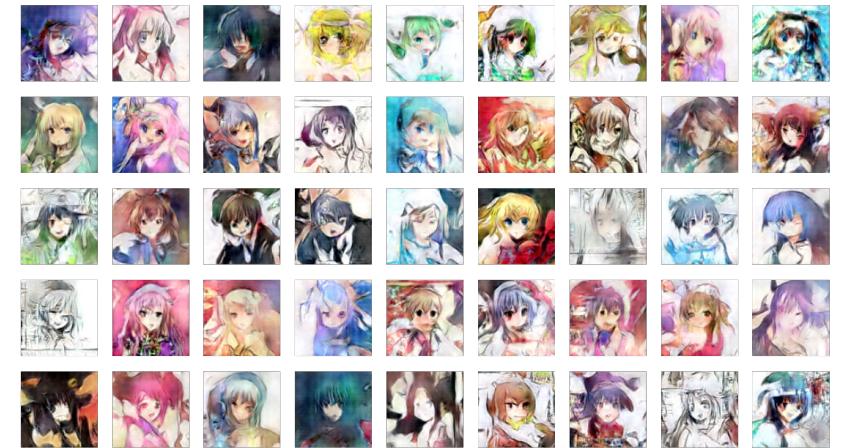
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.



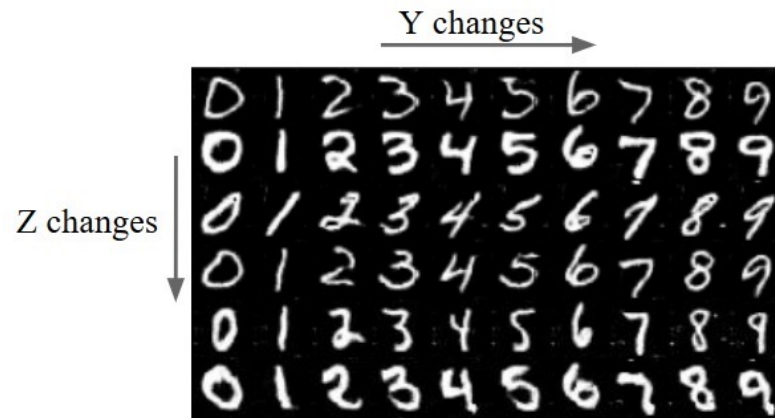
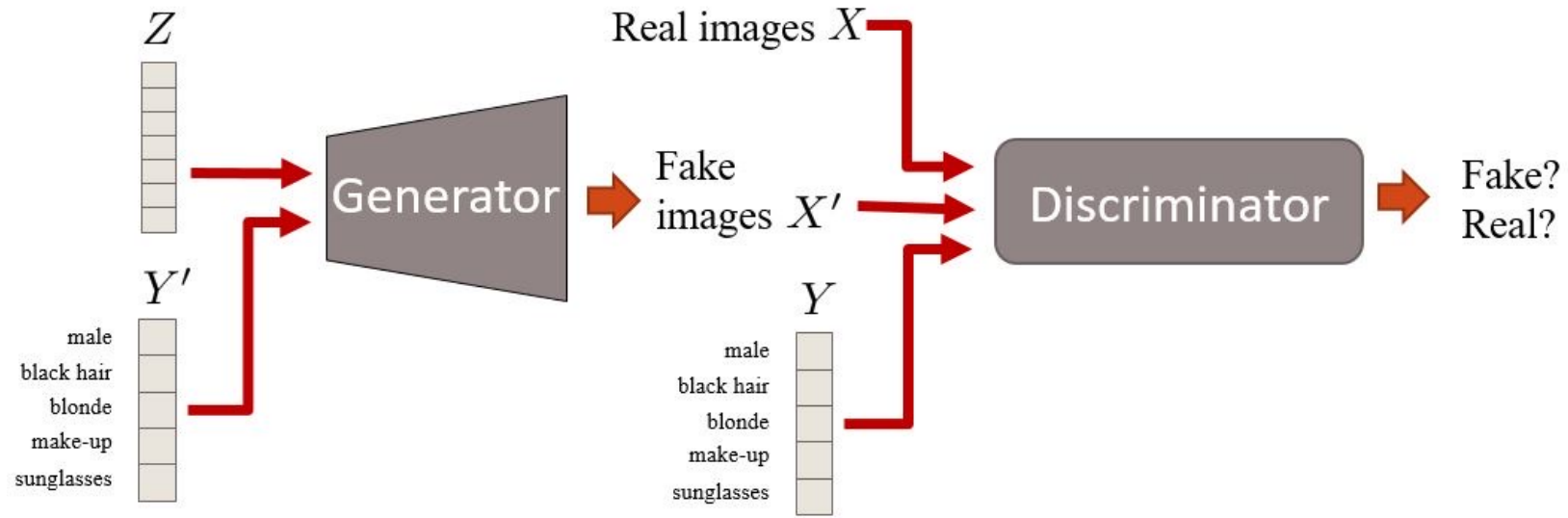
## UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz  
indico Research  
Boston, MA  
{alec,luke}@indico.io

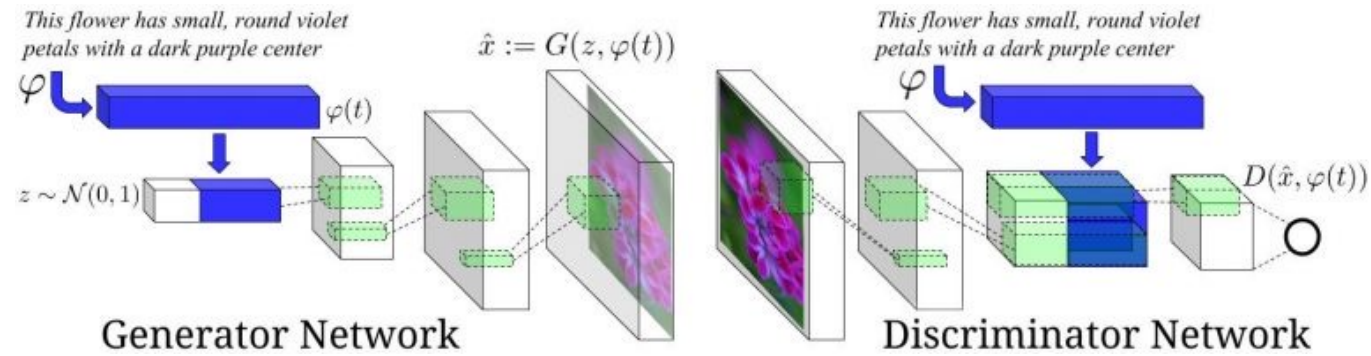
Soumith Chintala  
Facebook AI Research  
New York, NY  
soumith@fb.com



# Conditional GANs



# Text to image with GANs



*(this small bird has a pink breast and crown, and black primaries and secondaries)*



# Cycle GAN

## Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Jun-Yan Zhu\*   Taesung Park\*   Phillip Isola   Alexei A. Efros  
Berkeley AI Research (BAIR) laboratory, UC Berkeley

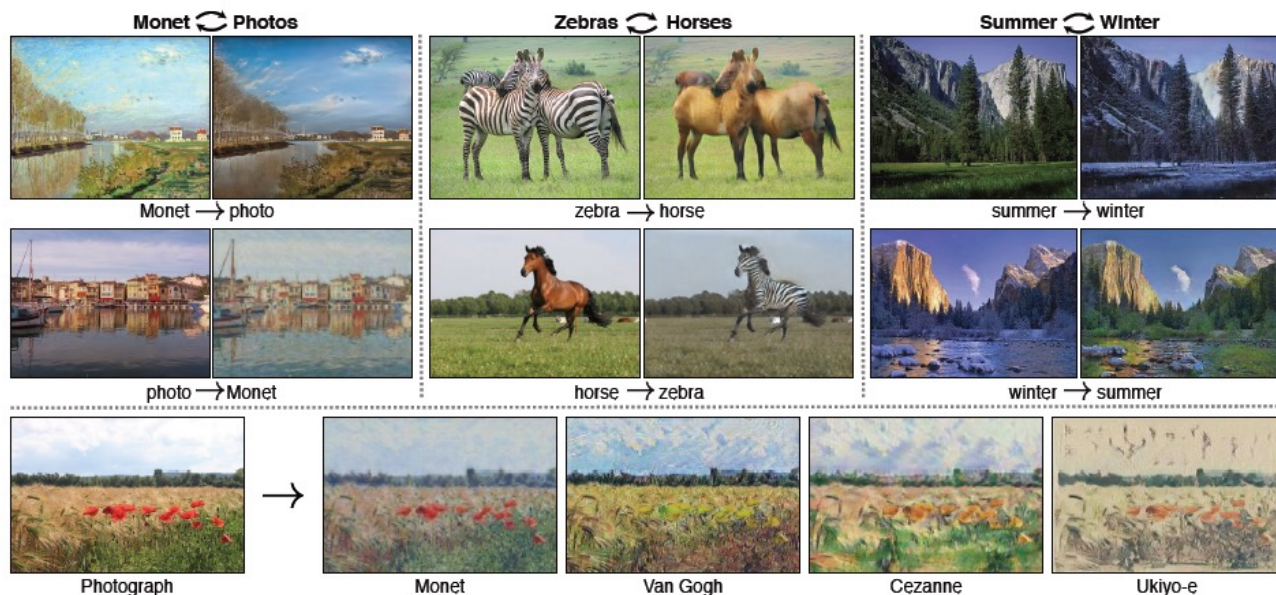
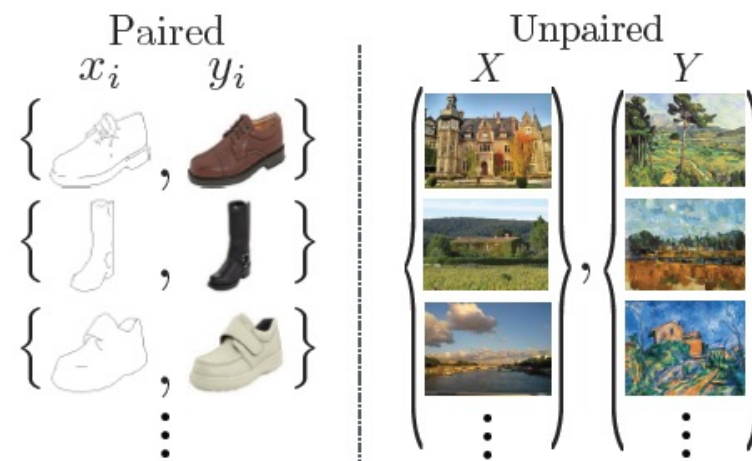


Figure 1: Given any two unordered image collections  $X$  and  $Y$ , our algorithm learns to automatically “translate” an image from one into the other and vice versa: (left) 1074 Monet paintings and 6753 landscape photos from Flickr; (center) 1177 zebras and 939 horses from ImageNet; (right) 1273 summer and 854 winter Yosemite photos from Flickr. Example application (bottom): using a collection of paintings of a famous artist, learn to render a user’s photograph into their style.



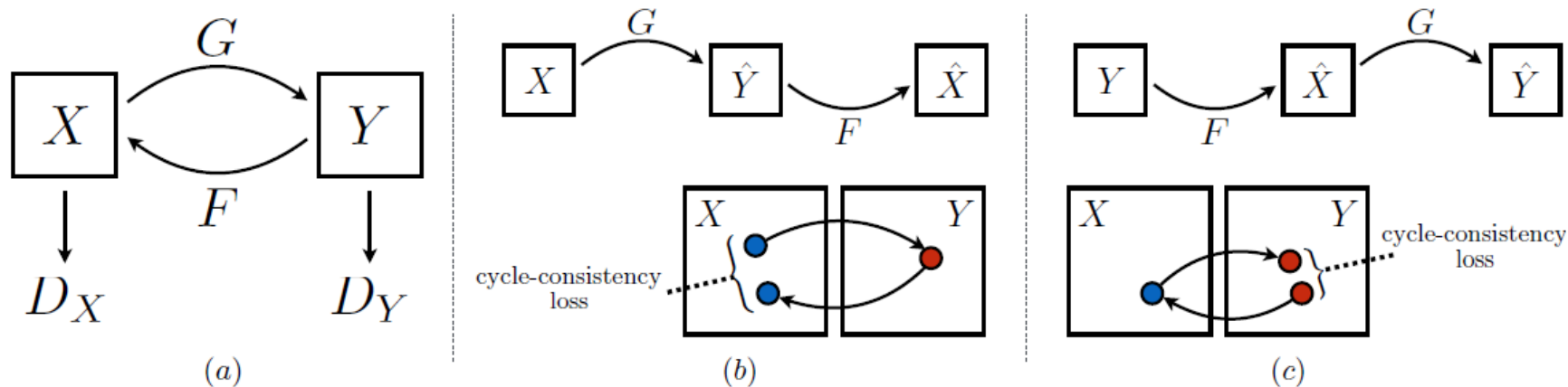


Figure 3: (a) Our model contains two mapping functions  $G : X \rightarrow Y$  and  $F : Y \rightarrow X$ , and associated adversarial discriminators  $D_Y$  and  $D_X$ .  $D_Y$  encourages  $G$  to translate  $X$  into outputs indistinguishable from domain  $Y$ , and vice versa for  $D_X$ ,  $F$ , and  $X$ . To further regularize the mappings, we introduce two “cycle consistency losses” that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss:  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ , and (c) backward cycle-consistency loss:  $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

$$\begin{aligned} \mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ & + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F), \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = & \mathbb{E}_{y \sim p_{\text{data}}(y)} [\log D_Y(y)] \\ & + \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(1 - D_Y(G(x)))] \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1]. \end{aligned}$$

# Cycle GAN

Winter



Summer



<https://junyanz.github.io/CycleGAN/>

# Example

<https://www.redsharknews.com/technology-computing/item/5191-nvidia-turns-summer-into-winter-with-ai-video-manipulation>

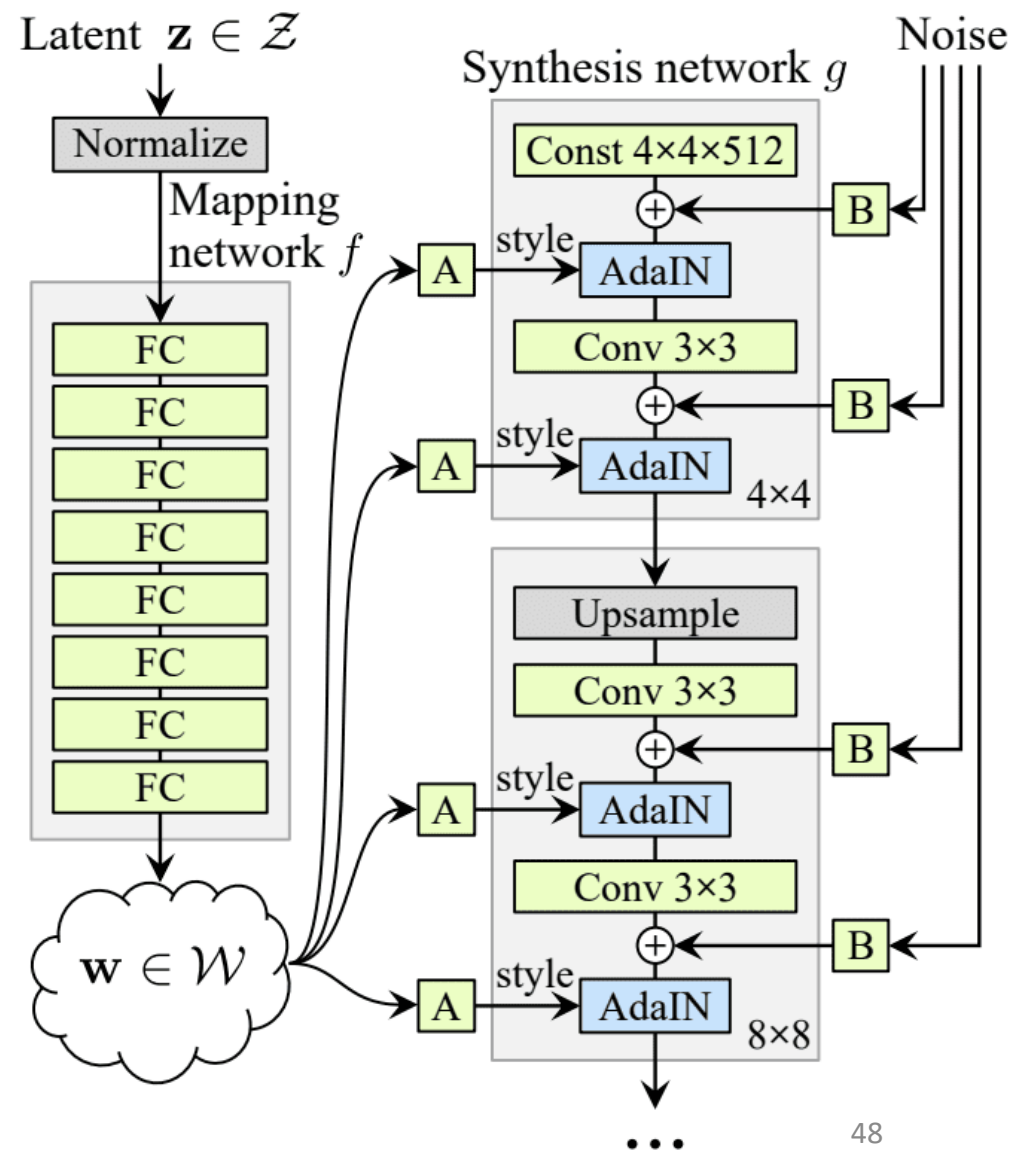
# StyleGAN

<https://github.com/NVlabs/stylegan2>

$\mathbf{z}$  (through  $\mathbf{w}$ ) controls the macro-structure and semantic attributes



A: FC layer.  
B: Noise broadcast.



# AdaIN

$$\text{BN}(x) = \gamma \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\text{IN}(x) = \gamma \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\text{AdaIN}(x, y) = \sigma(y) \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

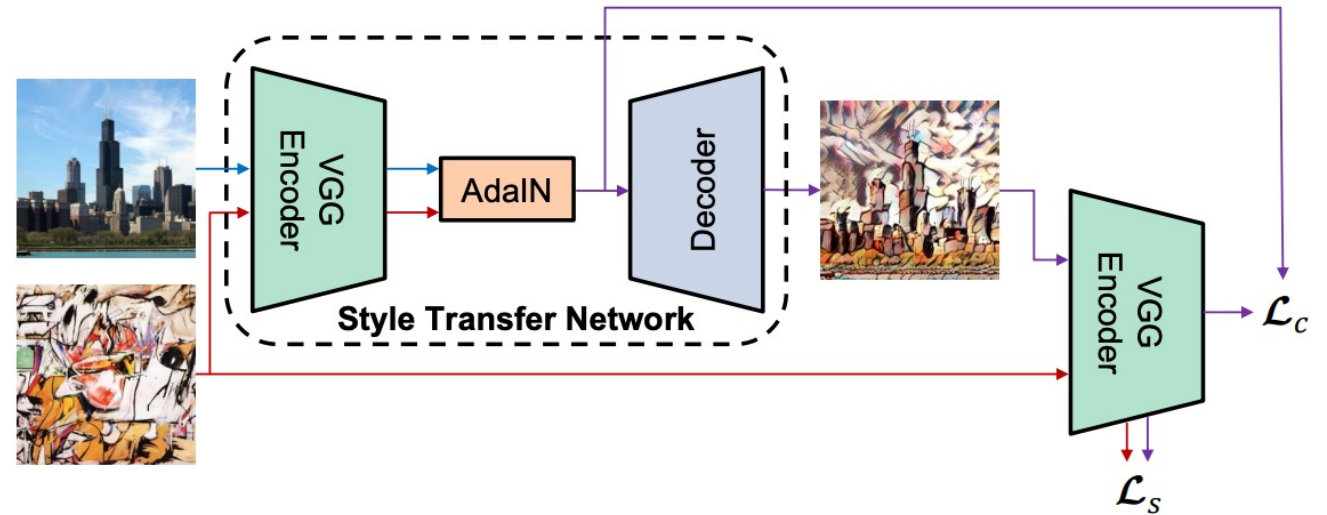
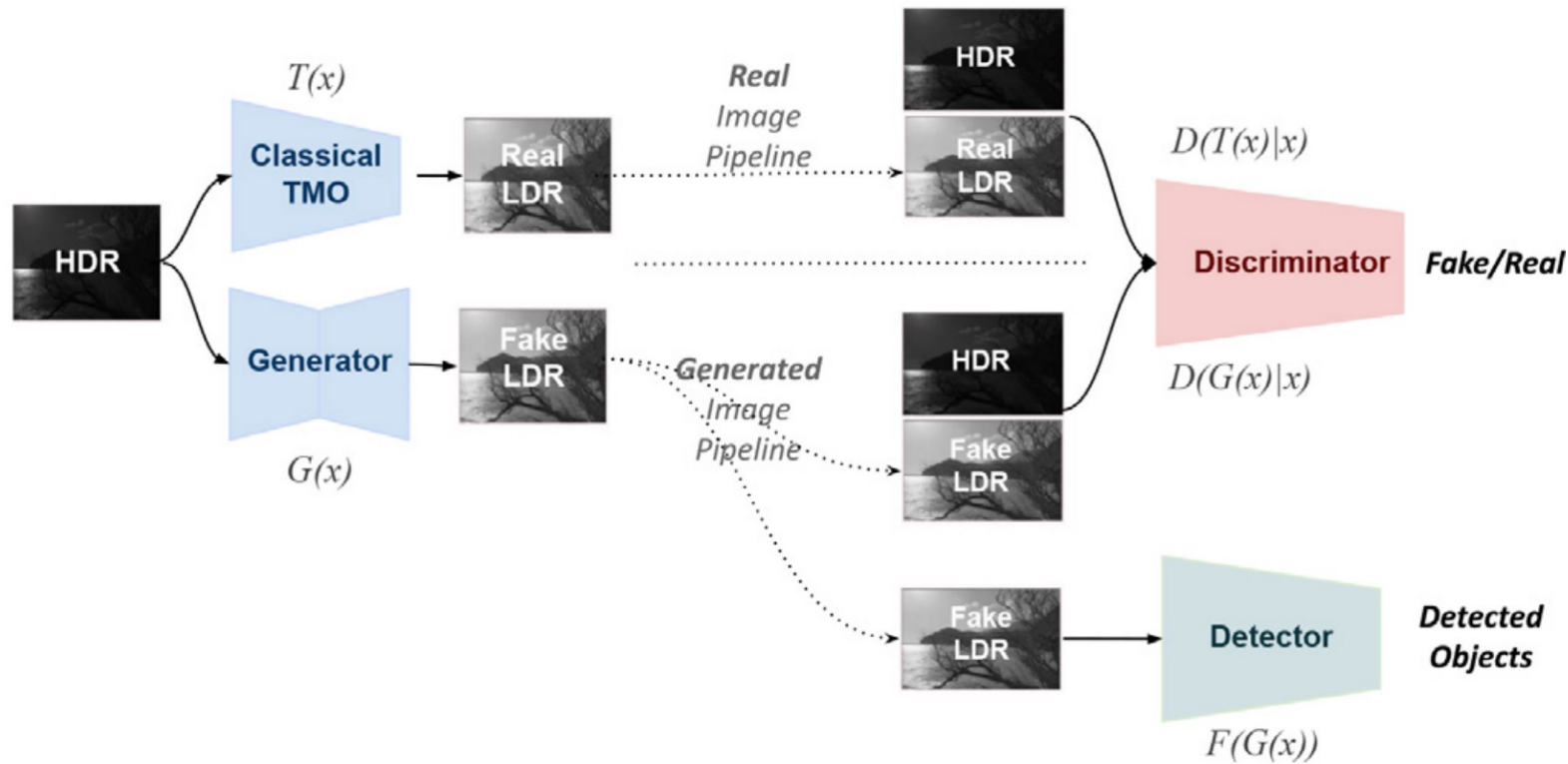


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss  $\mathcal{L}_c$  (Equ. 12) and a style loss  $\mathcal{L}_s$  (Equ. 13).

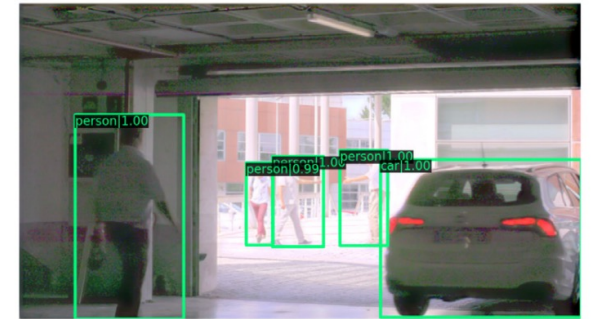
# Sample from Our Work



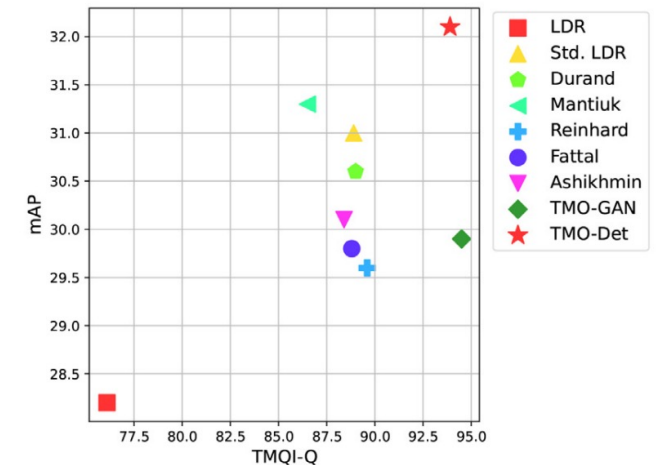
**Fig. 2.** Overall architecture diagram for the proposed method that combines object detection and tone-mapping objectives.



(a) RetinaNet results on an LDR image [1].



(b) TMO-Det detection & tone-mapped LDR image output.



(c) Detection vs. HDR quality.

# Energy-based Generative Models

---

Summary

Laureates

John J. Hopfield

Geoffrey E. Hinton

Prize announcement

Press release

Popular information

Advanced information

---

Share this



[The Royal Swedish Academy of Sciences](#) has decided to award the Nobel Prize in Physics 2024 to

**John J. Hopfield**

Princeton University, NJ, USA

**Geoffrey E. Hinton**

University of Toronto, Canada

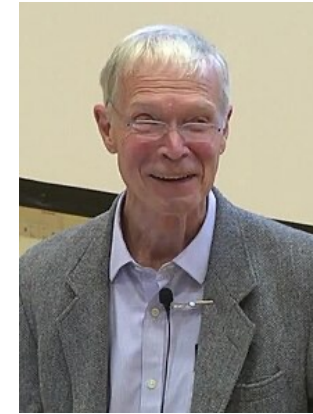
*“for foundational discoveries and inventions that enable machine learning with artificial neural networks”*

## They trained artificial neural networks using physics

---

This year’s two Nobel Laureates in Physics have used tools from physics to develop methods that are the foundation of today’s powerful machine learning. John Hopfield created an associative memory that can store and reconstruct images and other types of patterns in data. Geoffrey Hinton invented a method that can autonomously find properties in data, and so perform tasks such as identifying specific elements in pictures.

“**John Hopfield** invented a network that uses a method for saving and recreating patterns. We can imagine the nodes as pixels. The *Hopfield network* utilises physics that describes a material’s characteristics due to its atomic spin – a property that makes each atom a tiny magnet. The network as a whole is described in a manner equivalent to the energy in the spin system found in physics, and is trained by finding values for the connections between the nodes so that the saved images have low energy. When the Hopfield network is fed a distorted or incomplete image, it methodically works through the nodes and updates their values so the network’s energy falls. The network thus works stepwise to find the saved image that is most like the imperfect one it was fed with.”

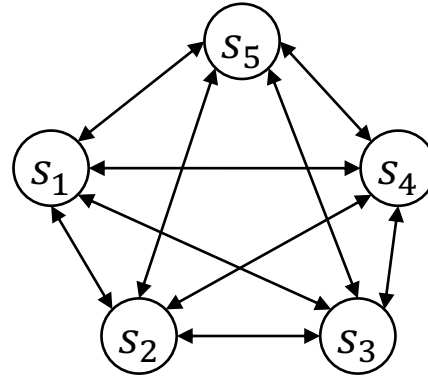


J. Hopfield  
(born in 1933)

“**Geoffrey Hinton** used the Hopfield network as the foundation for a new network that uses a different method: the *Boltzmann machine*. This can learn to recognise characteristic elements in a given type of data. Hinton used tools from statistical physics, the science of systems built from many similar components. The machine is trained by feeding it examples that are very likely to arise when the machine is run. The Boltzmann machine can be used to classify images or create new examples of the type of pattern on which it was trained. Hinton has built upon this work, helping initiate the current explosive development of machine learning.



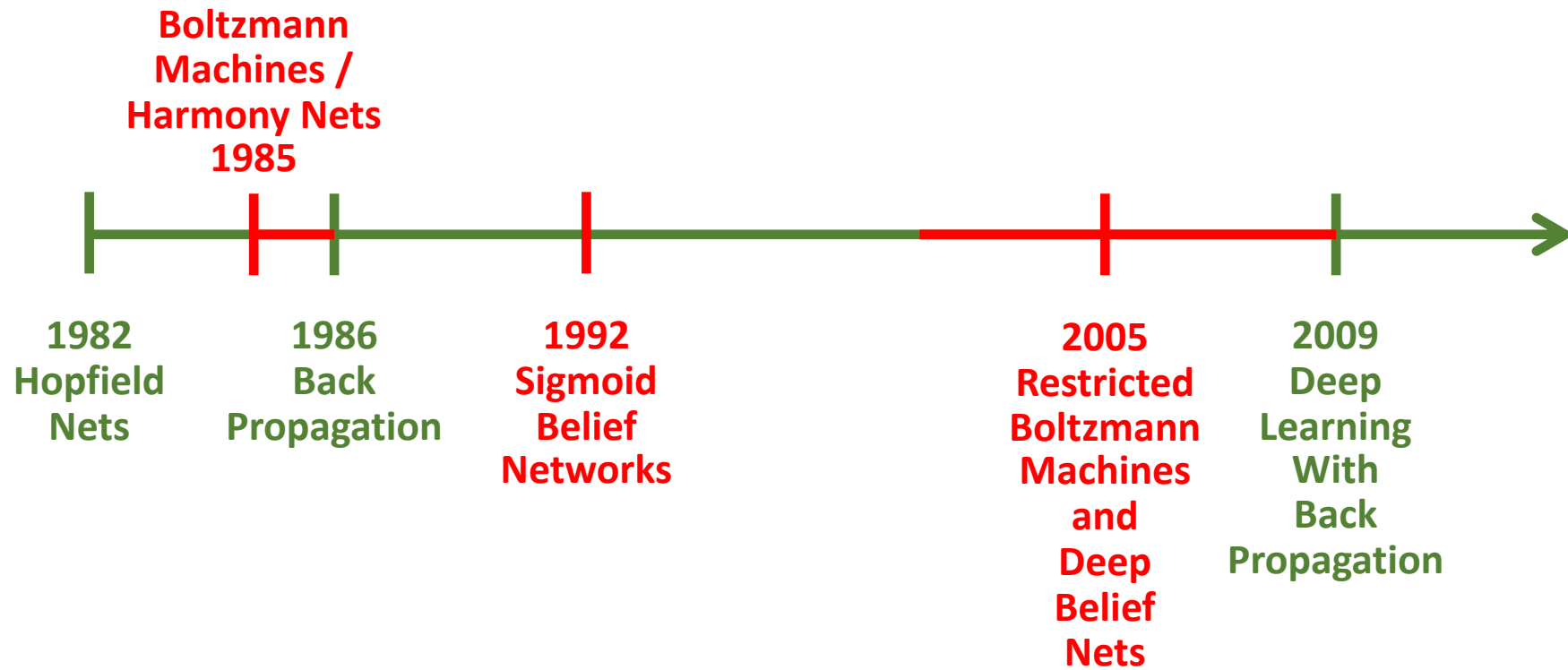
G. Hinton  
(born in 1947)



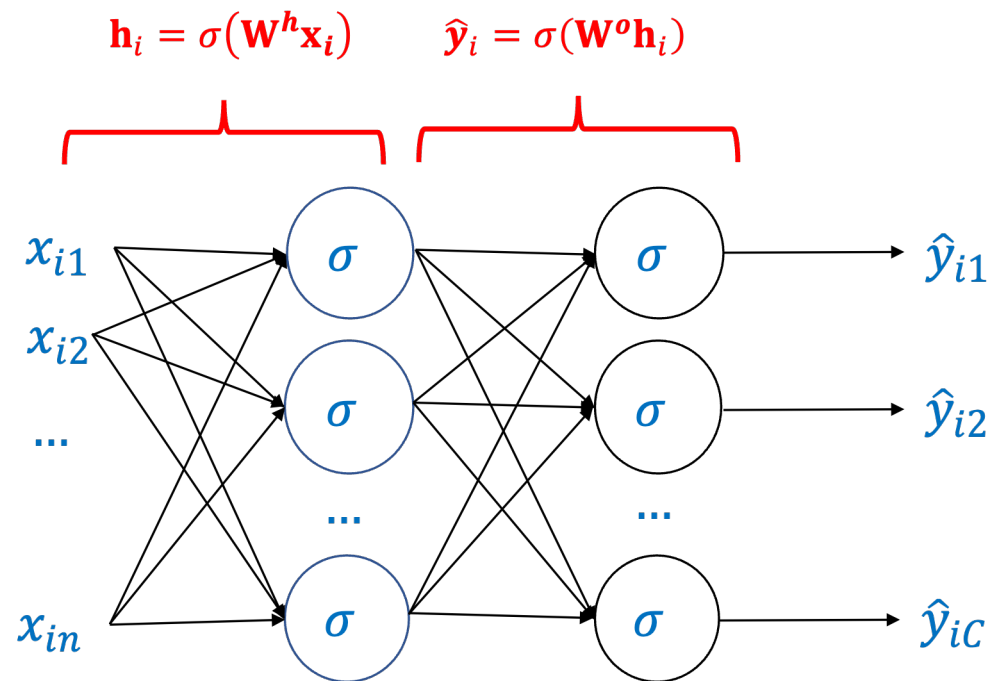
# Hopfield Networks

(Associative Memory, Ising Model, Spin-glass System)

Neural networks and physical systems with emergent collective computational properties, Hopfield and Tank, Proceedings of the National Academy of Sciences, 1982.



# Artificial Neural Networks



Hidden activations:

$$h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(\text{net}_{ij}^h)$$

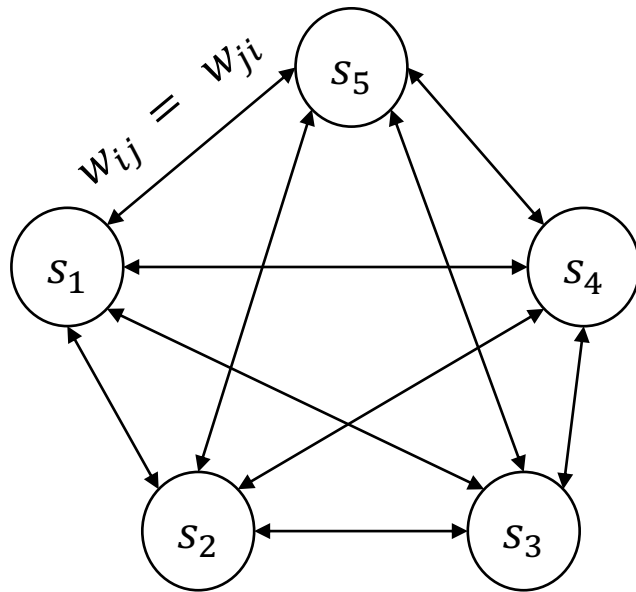
Output layer:

$$\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(\text{net}_{ic}^o)$$

The loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^N \sum_{c \in \mathcal{C}} (\hat{y}_{ic} - y_{ic})^2$$

# Hopfield Networks



- $s_i = -1$  or  $+1$

- Then,

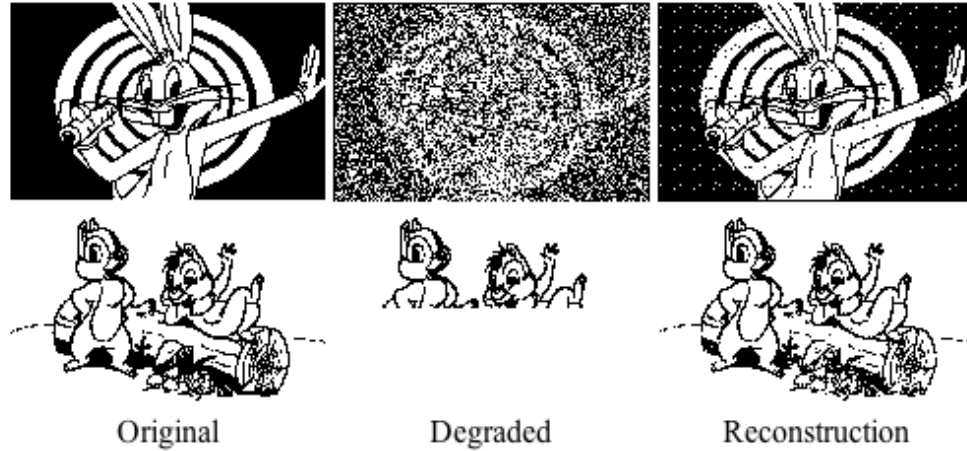
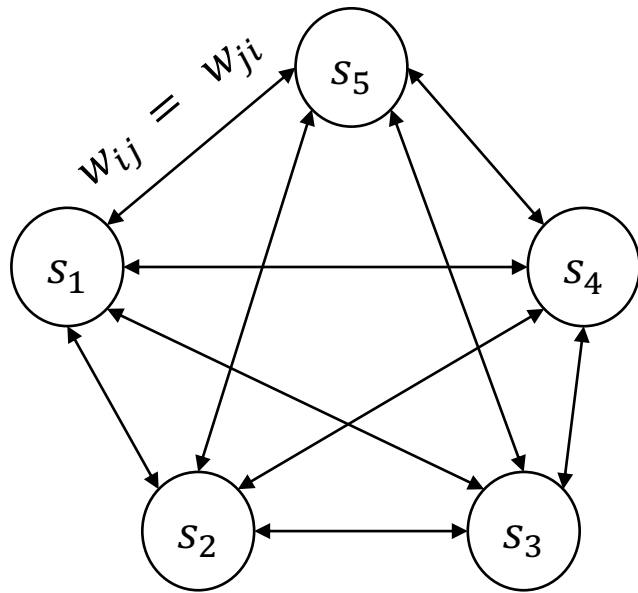
$$s_i \leftarrow \begin{cases} +1, & \sum_j w_{ij} s_j \geq \theta_i \\ -1, & \text{otherwise} \end{cases}$$

- $\theta_i$ : threshold of neuron  $i$ . Mostly we set this to zero.

- In short:

$$s_i = \text{sgn} \left( \left[ \sum_j w_{ij} s_j \right] - \theta_i \right)$$

# Hopfield Networks

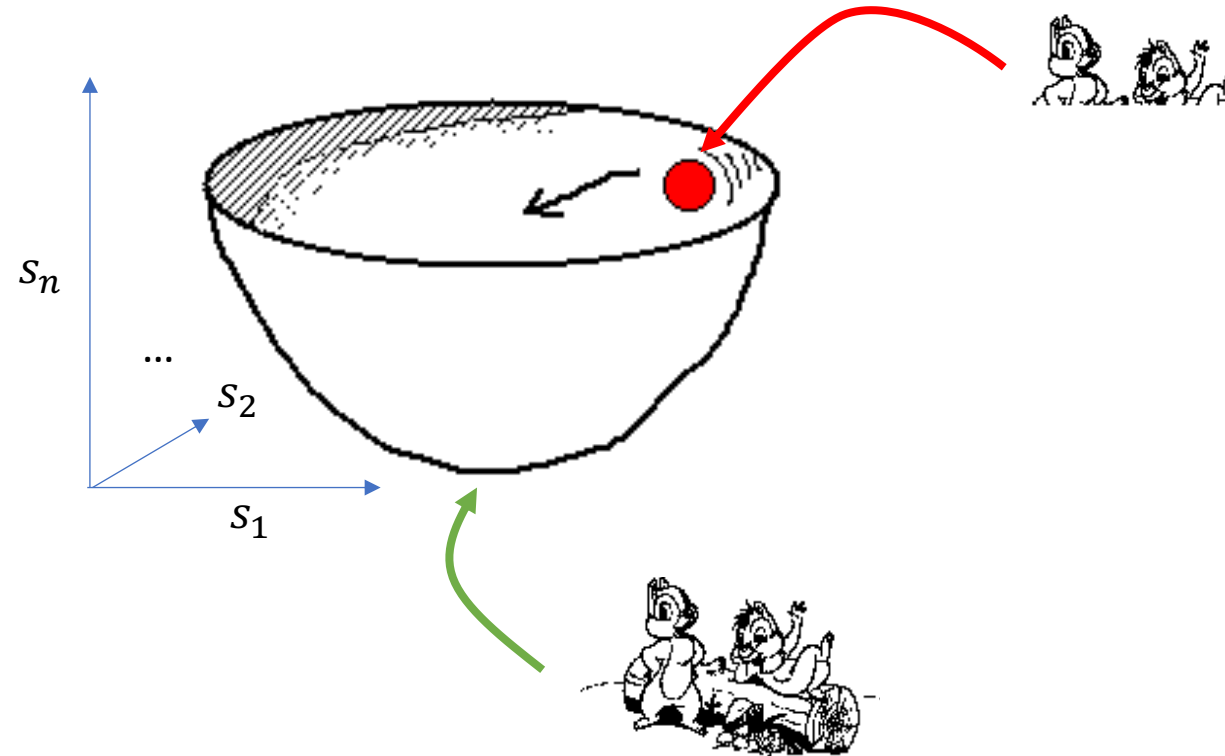
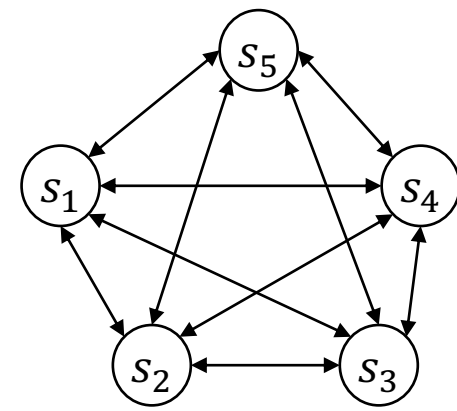


# Hopfield Networks

- “Training” on a set of patterns causes them to become attractors
- Degraded input is mapped to nearest attractor



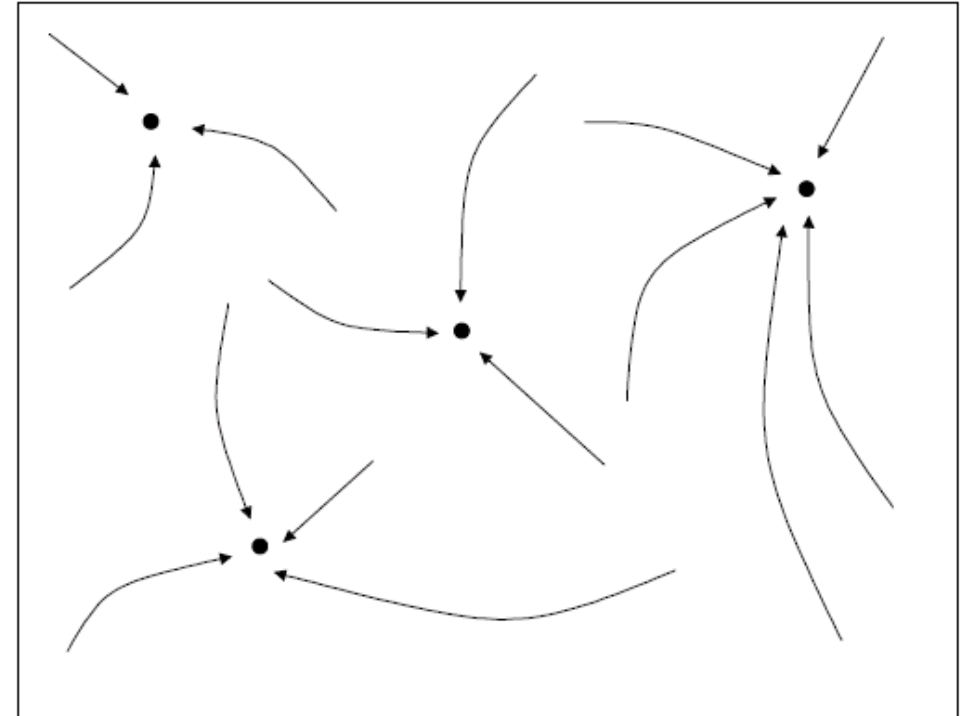
# Hopfield Networks as Content-addressable Memory



Adapted from: E. Sahin

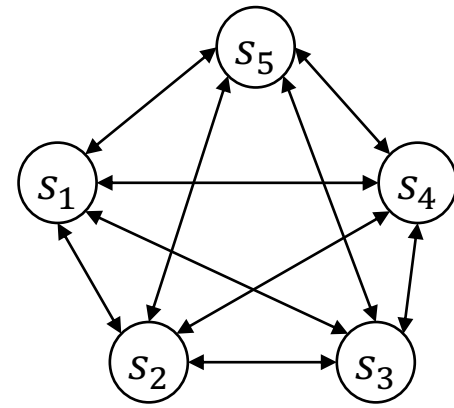
# Hopfield Networks as Content-addressable Memory

- CAM can be defined as a system whose stable points can be set as a set of pre-defined states.
- The stored patterns divide the state space into locally stable points, called “basins of attraction” in dynamical systems theory.



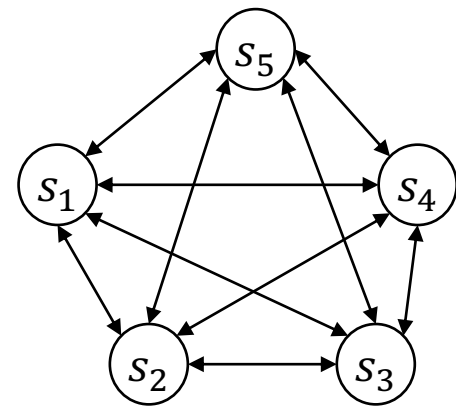
Adapted from: E. Sahin

# Hopfield Networks: Updating Neurons



- Three possible schemes:
  - Synchronously: all units updated at each step.
  - Asynchronously I: at each time step select a random unit for update.
  - Asynchronously II: each unit independently chooses to update itself with some probability per unit time.
  
- Use asynchronously I and keep updating until no neuron changes its state.

# Hopfield Networks: Learning to Store a Single Pattern



- Assume that we want to store pattern  $\mathcal{P}$
- i.e., find  $w_{ij}$  so that we have:

$$s_i = \text{sgn} \left( \sum_j w_{ij} \mathcal{P}_j \right) = \mathcal{P}_i$$

- A solution:

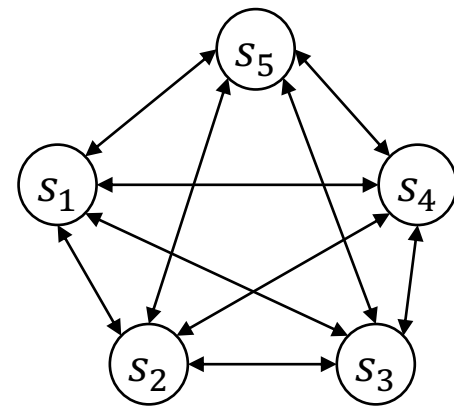
$$w_{ij} = \frac{1}{N} \mathcal{P}_i \mathcal{P}_j$$

since

$$s_i = \text{sgn} \left( \sum_j w_{ij} \mathcal{P}_j \right) = \text{sgn} \left( \sum_j \frac{1}{N} \mathcal{P}_i \overbrace{\mathcal{P}_j \mathcal{P}_j}^1 \right) = \mathcal{P}_i$$

- If more than half of the bits are the same as  $\mathcal{P}$ , the network will recall  $\mathcal{P}$  (it is an attractor of the system)

# Hopfield Networks: Learning to Store Many Patterns



- For storing K patterns:

$$w_{ij} = \frac{1}{N} \sum_{k=1 \dots K} \mathcal{P}_i^k \mathcal{P}_j^k$$

- Hebbian Learning Rule
  - “Neurons that fire together wire together” –Donald Hebb

# Hopfield Networks: Example

- Patterns:

$$\mathcal{P}^1 = (-1, -1, -1, +1) \text{ and } \mathcal{P}^2 = (+1, +1, +1, +1)$$

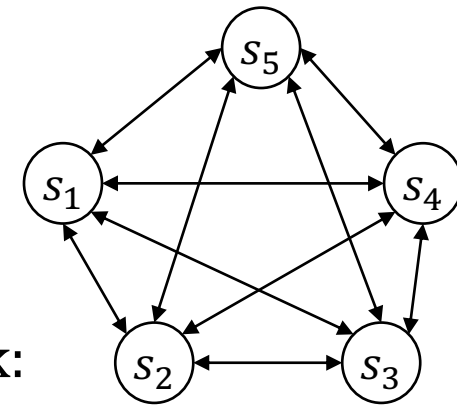
- Weights (using  $w_{ij} = \frac{1}{N} \sum_{k=1,2} \mathcal{P}_i^k \mathcal{P}_j^k$ ):

$$\frac{1}{4} \begin{bmatrix} 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

$$\text{(e.g., } w_{12} = \frac{1}{4} (-1 \times -1 + 1 \times 1) = 2/4)$$

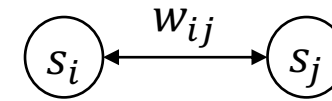
- Inputs and reconstructions (using  $s_i = \text{sgn}([\sum_j w_{ij} s_j] - \theta_i)$ ):
  - $\mathcal{P}^3 = (-1, -1, -1, +1) \Rightarrow \text{Recall: } (-1, -1, -1, +1)$
  - $\mathcal{P}^4 = (-1, +1, +1, +1) \Rightarrow \text{Recall: } (+1, +1, +1, +1)$

# Hopfield Networks: An Energy Perspective



- We can define a scalar representing the energy of the state of the network:

$$E = - \sum_i \sum_{j < i} w_{ij} s_i s_j + \sum_i \theta_i s_i$$



- This is called energy since when you update neurons randomly, it **either decreases or stays the same**.
- Repeatedly updating the network will eventually make the network converge to a **local minimum**, i.e., a **stable state**.

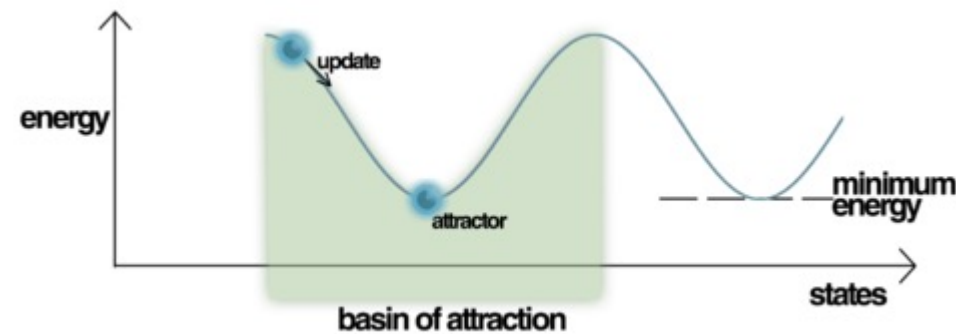


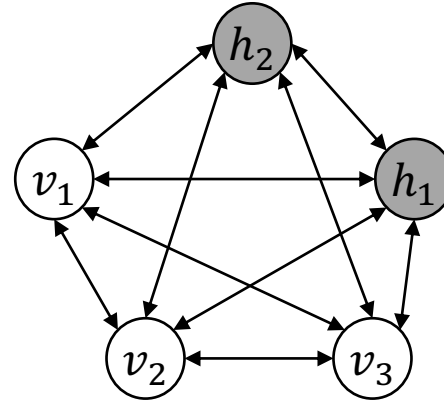
Fig: Wikipedia

# Hopfield Networks

- An associative memory
- Inspired many models in Machine Learning

Skipping:

- Stability conditions
- Storage capacity
- Increasing robustness
- Extension for continuous-valued patterns
- ...



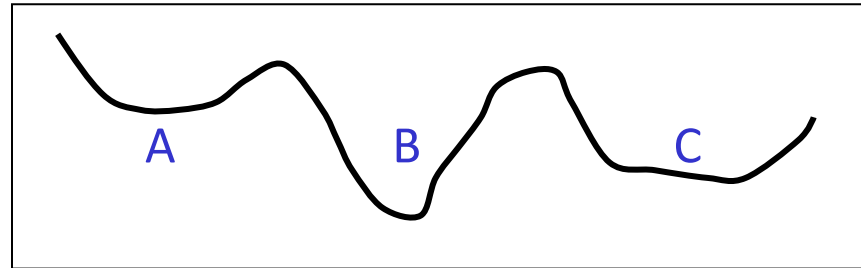
# Boltzmann Machines

(Sherrington–Kirkpatrick model with external field, Stochastic Ising Model, Markov Random Field)

Hinton, G. E. and Sejnowski, T. J. (1983). Optimal Perceptual Inference. Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, Washington DC, pp. 448-453.

# Boltzmann Machines: Motivation

- A Hopfield net always makes decisions that reduce the energy.
  - This makes it impossible to escape from local minima.



- Add some randomness to escape from poor minima.
  - Start with a lot of noise so it's easy to cross “energy barriers”.
- This may mean we **occasionally increase the energy**
  - Slowly reduce the noise so that the system ends up in a deep minimum.
  - This is “**simulated annealing**”.

# Boltzmann Machines: Boltzmann (Gibbs) Distribution

- Probability of particles in a state ( $\mathbf{s}$ ) in a system:

$$\propto e^{-E(\mathbf{s})/kT},$$

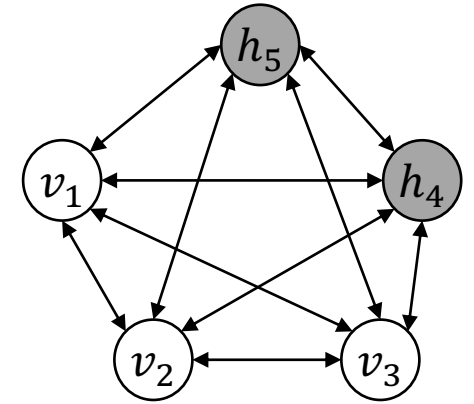
where  $E(\mathbf{s})$ : the energy of the state  $\mathbf{s}$ ,  $k$ : Boltzmann's constant,  $T$ : temperature.

- The probability that a system will be in a certain state:

$$p_i = p(\mathbf{s}_i) = \frac{e^{-E(\mathbf{s}_i)/kT}}{\sum_{j=1}^M e^{-E(\mathbf{s}_j)/kT}}$$

where  $E(\mathbf{s}_i)$  is the energy of state  $\mathbf{s}_i$ .

# Boltzmann Machines vs. Hopfield Networks



- They have the same energy definition ( $\mathbf{s} = \{v_m\} \cup \{h_n\}$ ):

$$E(\mathbf{s}) = - \sum_i \sum_{j < i} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

Differences:

- Updates are stochastic
- We have hidden neurons now
  - Hidden variables  $\rightarrow$  Bigger class of distributions that can be modeled  $\rightarrow$  In principle, we can model distributions of arbitrary complexity

# Boltzmann Machines: Probability of a Neuron's State

- Turning on a neuron  $i$  (i.e.,  $s_i$  is changed to 1 from 0) causes change  $\Delta E_i$  in energy:

$$\begin{aligned}\Delta E_i &= E_{i=0} - E_{i=1} \\ &= -kT \ln(Z p_{i=0}) - (-kT \ln(Z p_{i=1})) \\ &= -kT \ln\left(\frac{Z p_{i=0}}{Z p_{i=1}}\right) = -kT \ln\left(\frac{p_{i=0}}{p_{i=1}}\right) \\ &= -kT \ln\left(\frac{1-p_{i=1}}{p_{i=1}}\right)\end{aligned}$$

Using:

$$p_i = \frac{e^{-E_i/kT}}{Z}$$

- This yields the famous logistic / sigmoid function:

$$p_{i=1} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)}$$

- $\Delta E_i > 0 \Rightarrow$  Energy is reduced  $\Rightarrow$  High  $p_{i=1}$
- $\Delta E_i < 0 \Rightarrow$  Energy is increased  $\Rightarrow$  Low  $p_{i=1}$

# Boltzmann Machines: Interpretation of a State's Probability

$$p_{i=1} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)}$$

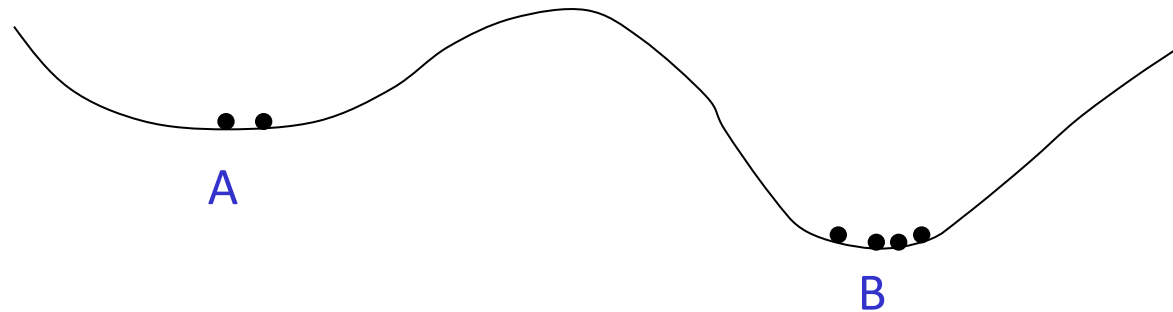
- a. If  $T = 0$ ,
    - $p_{i=1} \approx 1$  if  $\Delta E_i$  is positive (energy reduced).
    - If  $\Delta E_i$  is negative,  $p_{i=1} \approx 0$ .
  - b. If  $T$  is high, then  $p_{i=1} \approx 1/2$ .
    - Half the chance is given to updating the neuron.
  - c. For a fixed  $T$ , if  $\Delta E_i$  is zero, same as case (b).
  - d. For a fixed  $T$ , if  $\Delta E_i$  is very high, same as case (a).
- When the temperature is high, the network covers the whole state space.
  - In the cooling phase, when the temperature is small, the network converges to a minima, hopefully the global one.

# Boltzmann Machines: How temperature affects transition probabilities

High temperature  
transition  
probabilities

$$p(A \rightarrow B) = 0.2$$

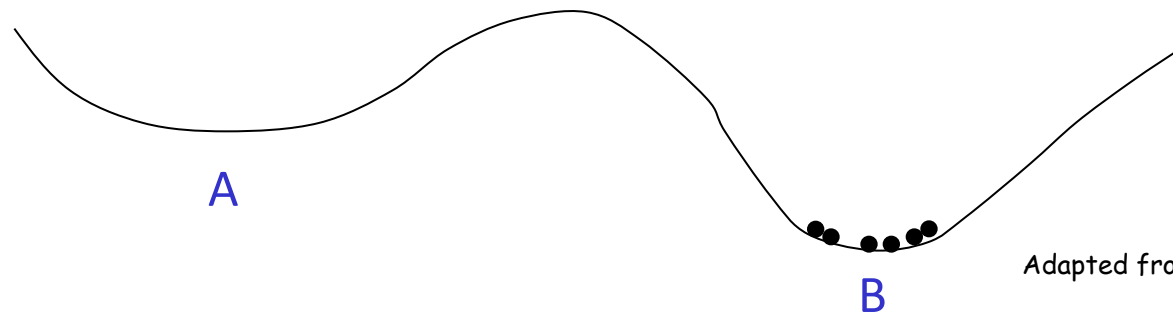
$$p(A \leftarrow B) = 0.1$$



Low temperature  
transition  
probabilities

$$p(A \rightarrow B) = 0.001$$

$$p(A \leftarrow B) = 0.000001$$



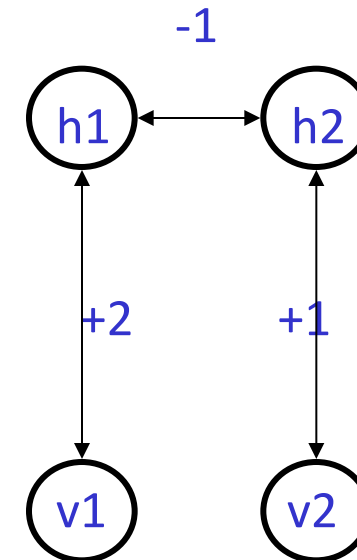
Adapted from G. Hinton

# Boltzmann Machines: An Example

$$E(\mathbf{s}) = - \sum_i \sum_{j < i} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

<b>v</b>	<b>h</b>	$-E$	$e^{-E}$	$p(\mathbf{v}, \mathbf{h})$	$p(\mathbf{v})$
11	11	2	7.39	.186	0.466
11	10	2	7.39	.186	
11	01	1	2.72	.069	
11	00	0	1	.025	
10	11	1	2.72	.069	0.305
10	10	2	7.39	.186	
10	01	0	1	.025	
10	00	0	1	.025	
01	11	0	1	.025	0.144
01	10	0	1	.025	
01	01	1	2.72	.069	
01	00	0	1	.025	
00	11	-1	0.37	.009	0.084
00	10	0	1	.025	
00	01	0	1	.025	
00	00	0	1	.025	

total = 39.70



Adapted from G. Hinton

# Boltzmann Machines: Thermal Equilibrium

- We select a neuron and update its state according to the following probability:

$$p_{i=on} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)}$$

- If this is repeated long enough for a certain temperature, the state of the network will depend on the state's energy, and not on the initial state.
- In this condition, the log probabilities of global states become linear in their energies.
- This is called thermal equilibrium.
- Start from a high temperature, gradually decrease it until thermal equilibrium, we may converge to a distribution where energy level is close to the global minimum. →  
Simulated Annealing.

# Boltzmann Machines: Thermal Equilibrium

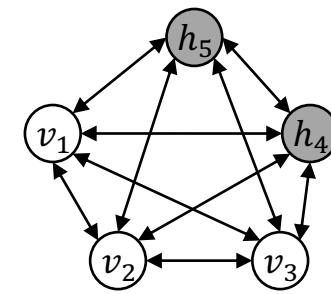
- How do we understand we have reached it?
  - The average activation of neurons don't change over time.
  - i.e., the probability of being in a state does not change.
- The initial state is not important!
- At low temperature:
  - There is a strong bias for states with low energy
  - But this makes it too slow to converge to thermal eq.
- At high temperature:
  - Not a strong bias for low energy
  - Equilibrium is reached faster

# Boltzmann Machines: Simulated Annealing



[https://en.m.wikipedia.org/wiki/File:Hill\\_Climbing\\_with\\_Simulated\\_Annealing.gif](https://en.m.wikipedia.org/wiki/File:Hill_Climbing_with_Simulated_Annealing.gif)

# Boltzmann Machines: Training



- Two sets of neurons: Visible units ( $V$ ) and Hidden units ( $H$ )
- Two distributions
  - Over the training set:  $P^+(V)$
  - Without the training set:  $P^-(V)$

- Minimize the difference between  $P^+(V)$  and  $P^-(V)$ :

$$G = D_{KL}(P^+(V) \parallel P^-(V)) = \sum_{\mathbf{v}} P^+(\mathbf{v}) \ln \left( \frac{P^+(\mathbf{v})}{P^-(\mathbf{v})} \right),$$

a summation over all possible states of  $V$ .

- $G$  is a function of weights.
  - We can use gradient descent on  $G$  to update the weights to minimize it.

# Boltzmann Machines: Training

- Two phases:
  - Positive phase: visible units are initialized to a sample from the training set.
  - Negative phase: the network runs freely. The units are not initialized to external data.
- Then:

$$\frac{\partial G}{\partial w_{ij}} = \frac{1}{R} [p_{ij}^+ - p_{ij}^-]$$

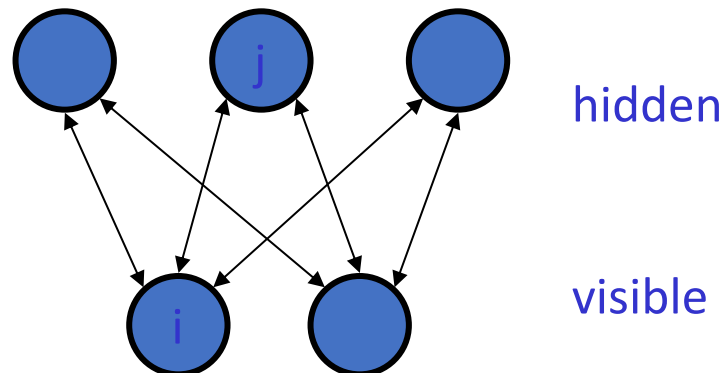
- R: learning rate
  - $p_{ij}^+$ : probability that both units are on at thermal equilibrium on the positive phase.
  - $p_{ij}^-$ : probability that both units are on at thermal equilibrium on the negative phase.
- $w_{ij} = w_{ij} - \frac{\partial G}{\partial w_{ij}}$
  - Needs only local information (compare it to backprop)

# Why Boltzmann Machines Failed

- Too slow
    - loop over training epochs
      - loop over training examples
        - loop over 2 phases (+ and -)
          - loop over annealing schedule for T
            - loop until thermal equilibrium reached
- Sensitivity to annealing schedule
- Difficulty determining when equilibrium is reached
- As learning progresses, weights get larger, energy barriers get hard to break -> becomes even slower
- Backprop was invented shortly after
  - The need to perform pattern completion wasn't necessary for most problems (feedforward nets sufficed)

# Restricted Boltzmann Machines

- Invented by Smolensky (1986), improved by Hinton et al. (2006)
- RBM: Boltzmann Machine with restricted connectivity
- Connections between hidden-visible units only!
- Smolensky called it Harmonium or Harmony networks



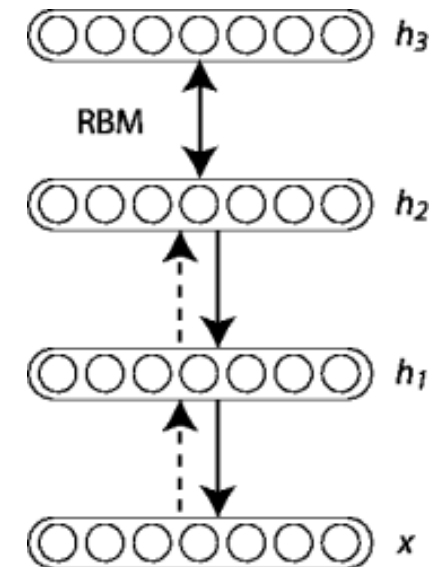
Adapted from G. Hinton

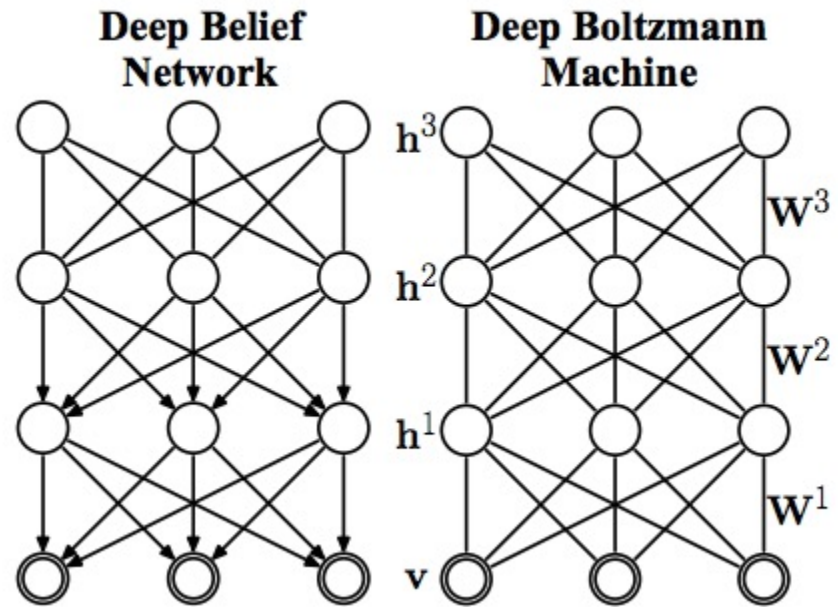
# Deep Belief Networks

- A stacked RBM
- First used by Hinton & Salakhutdinov (2006)
- Models the distribution:

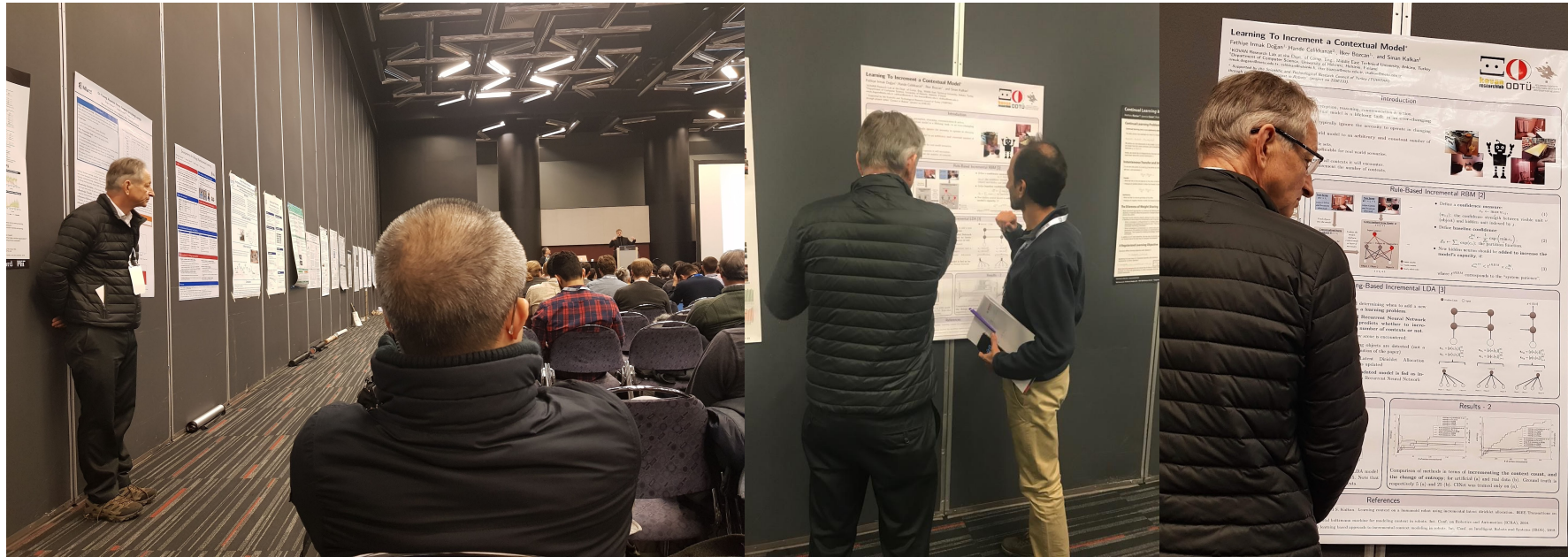
$$P(x, h^1, \dots, h^\ell) = \left( \prod_{k=0}^{\ell-2} P(h^k | h^{k+1}) \right) P(h^{\ell-1}, h^\ell)$$

- Training is similar to autoencoders



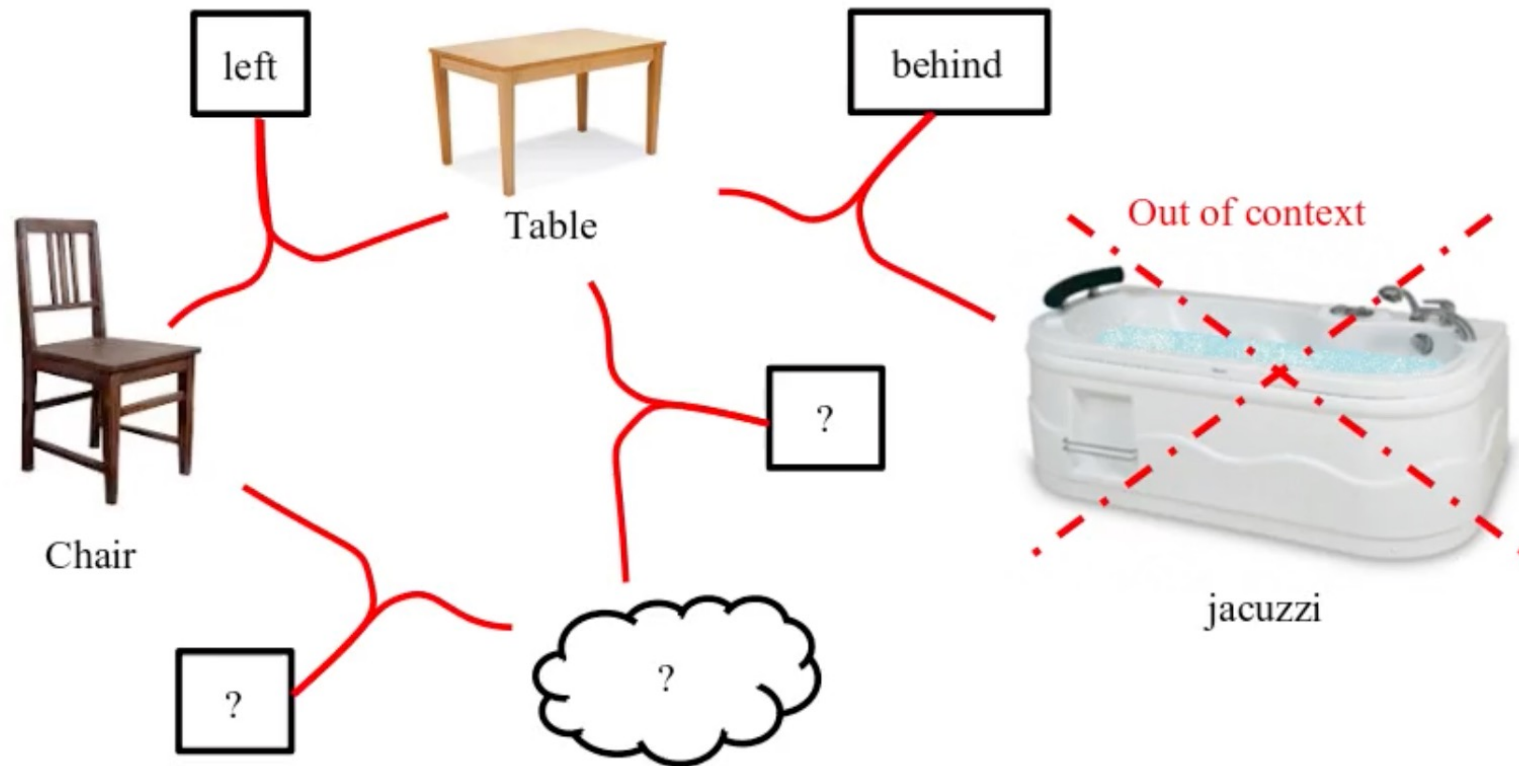


Wang, H., & Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.



# Our Work Using Boltzmann Machines

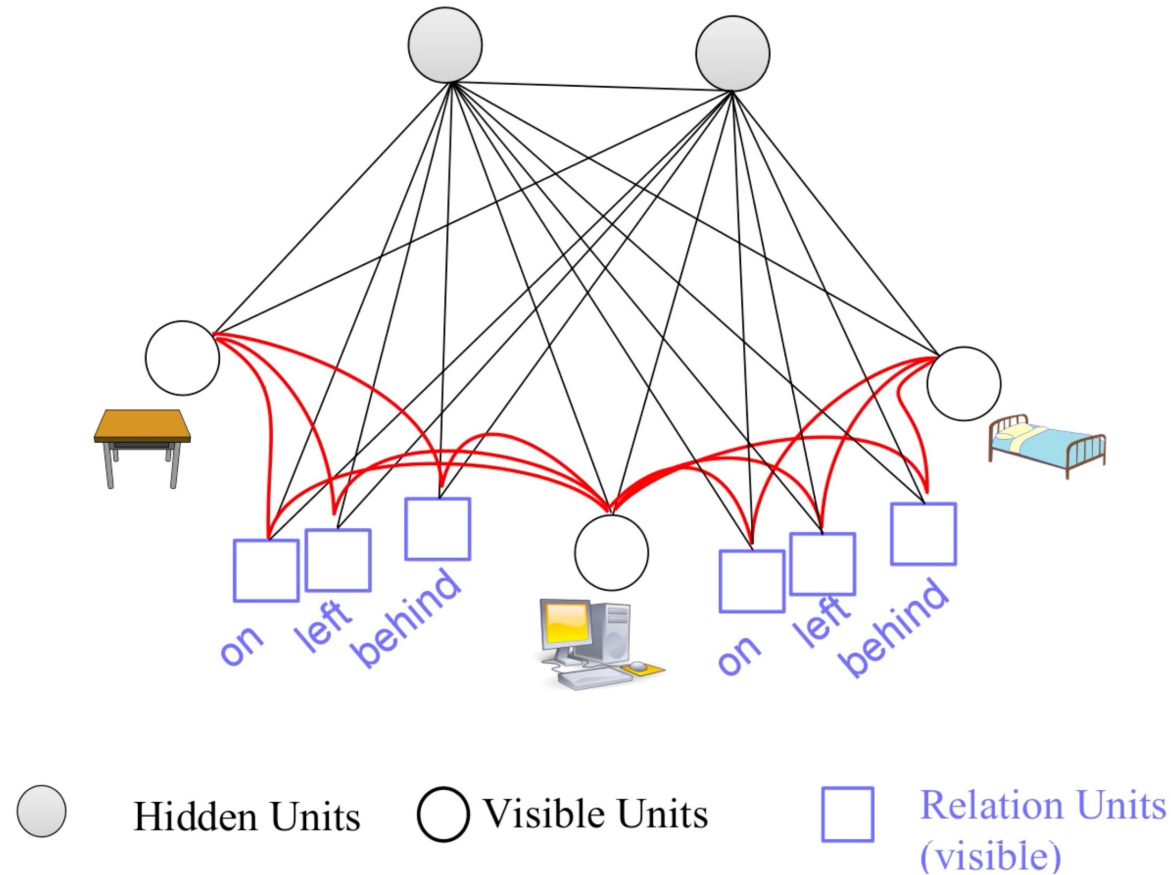
# Boltzmann Machines: Our Work



I. Bozcan, S. Kalkan, "COSMO: Contextualized Scene Modeling with Boltzmann Machines", Robotics and Autonomous Systems journal, 113:132-148, 2019.

I. Bozcan, Y. Oymak, I. Z. Alemdar, S. Kalkan, "What is (missing or wrong) in the scene? A Hybrid Deep Boltzmann Machine For Contextualized Scene Modeling", International Conference on Robotics and Automation (ICRA), pp. 1-6, IEEE, 2018.

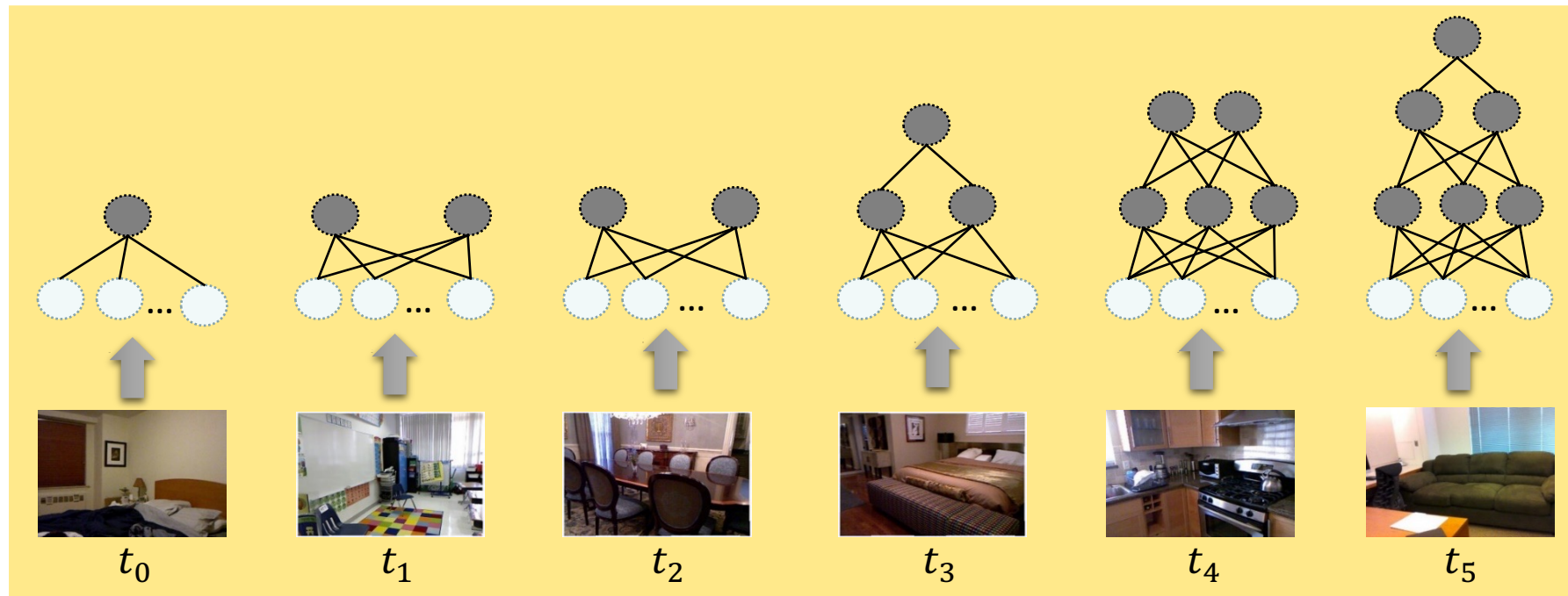
# Boltzmann Machines: Our Work



I. Bozcan, S. Kalkan, "COSMO: Contextualized Scene Modeling with Boltzmann Machines", Robotics and Autonomous Systems journal, 113:132-148, 2019.

I. Bozcan, Y. Oymak, I. Z. Alemdar, S. Kalkan, "What is (missing or wrong) in the scene? A Hybrid Deep Boltzmann Machine For Contextualized Scene Modeling", International Conference on Robotics and Automation (ICRA), pp. 1-6, IEEE, 2018.

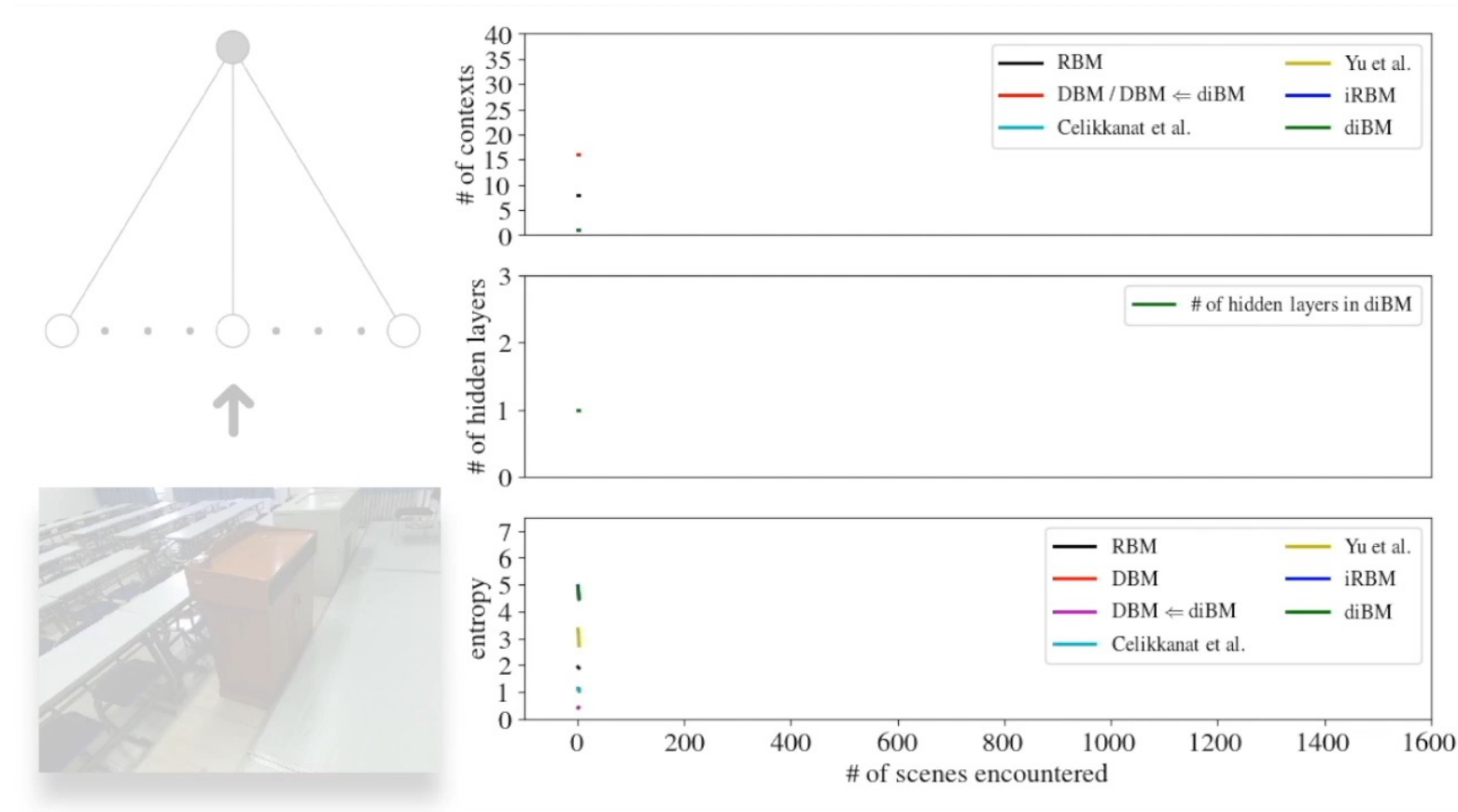
# Boltzmann Machines: Our Work



F. I. Dogan, H. Celikkanat, I. Bozcan, S. Kalkan, "Learning to Increment a Contextual Model", Continual Learning Workshop at Neural Information Processing Systems (NIPS) Conference, Canada, 2018.

F. I. Dogan, H. Celikkanat, S. Kalkan, "A Deep Incremental Boltzmann Machine for Modeling Context in Robots", International Conference on Robotics and Automation (ICRA), pp. 2411-2416, IEEE, 2018.

# Boltzmann Machines: Our Work



F. I. Dogan, H. Celikkanat, I. Bozcan, S. Kalkan, "Learning to Increment a Contextual Model", Continual Learning Workshop at Neural Information Processing Systems (NIPS) Conference, Canada, 2018.

F. I. Dogan, H. Celikkanat, S. Kalkan, "A Deep Incremental Boltzmann Machine for Modeling Context in Robots", International Conference on Robotics and Automation (ICRA), pp. 2411-2416, IEEE, 2018.

# Diffusion-based Generative Models

## ELBO Recap

### Why use ELBO?

Directly maximizing  $p(x)$  is very difficult:

- it involves either marginalizing over the entire latent space  $Z$  (intractable for complex models) OR
- It involves having access to the ground truth latent encoder  $p(z|x)$

### ELBO:

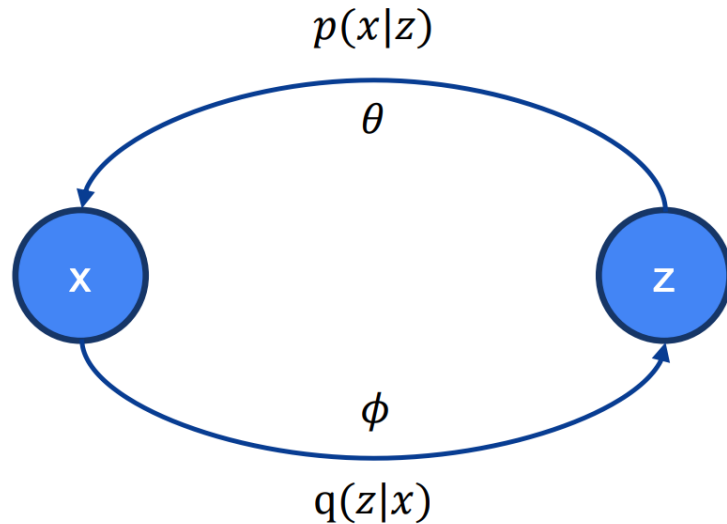
$$\log(p(x)) \geq \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p(x, z)}{q_\phi(z|x)} \right]$$

**Question:** Why does the  $\geq$  show up here?  $\rightarrow$  With the derivation in the appendix, we see a  $D_{KL}(q_\phi(z|x) || p(z|x))$  term show up which is always  $\geq 0$ .

### Applying chain-rule of probabilities:

$$ELBO = \underbrace{\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)]}_{\text{Reconstruction}} - \underbrace{D_{KL}(q_\phi(z|x) || p(z))}_{\text{Prior matching}}$$

## Variational Autoencoder Recap



Latent variable sampling:  $z \sim \mathcal{N}(z; \mu_\phi(x), \sigma_\phi^2(x))$

Reparameterization trick:  $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \epsilon \sim \mathcal{N}(0, I)$

Training:

- Jointly optimize  $\theta$  and  $\phi$
- Maximize **ELBO**

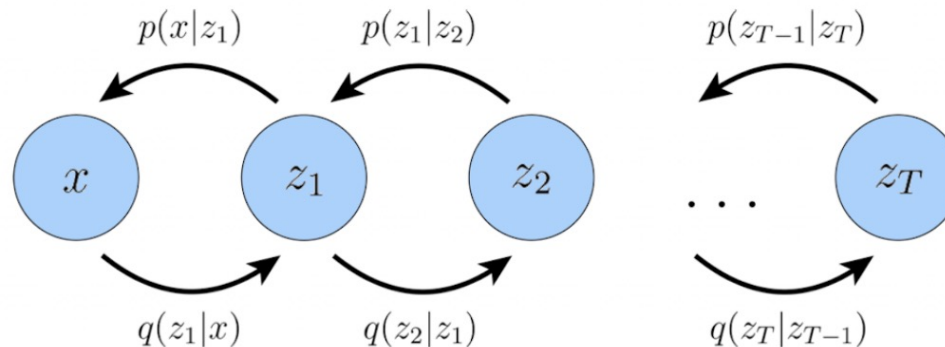
Empirically, we found that two things make VAEs work really well:

1. Increasing the depth of the networks
2. Introducing a hierarchy of latent variables (latent variables of latent variables)

$x \leftarrow z_1 \leftarrow z_2 \leftarrow \dots \leftarrow z_T$ , such that each latent is conditioned on all previous latents.

We are particularly interested in such HAVEs that where the process is a **Markovian chain - MHVAE**

# Markovian Hierarchical Variational Autoencoder



**Joint probability:**

$$p(x, z_{1:T}) = p(z_T) p_\theta(x | z_1) \prod_{t=2}^T p_\theta(z_{t-1} | z_t)$$

**Posterior probability:**

$$q_\phi(z_{1:T} | x) = q_\phi(z_1 | x) \prod_{t=2}^T q_\phi(z_t | z_{t-1})$$

**Updated ELBO:**

$$\log(p(x)) \geq \mathbb{E}_{q_\phi(z_{1:T} | x)} \left[ \log \frac{p(x, z_{1:T})}{q_\phi(z_{1:T} | x)} \right]$$

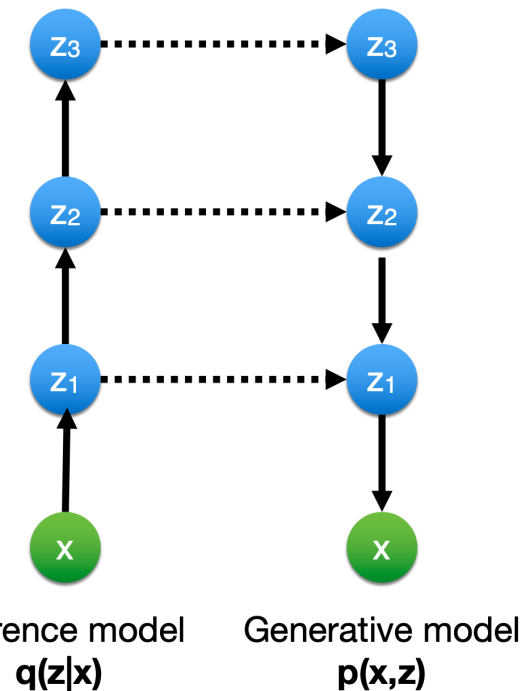


Fig: [https://cs231n.stanford.edu/slides/2023/lecture\\_15.pdf](https://cs231n.stanford.edu/slides/2023/lecture_15.pdf)

## Diffusion Models

Diffusion models are essentially **MHVAEs** with **3 restrictions**:

1. Latent dimension is the same as the data dimension
2. The encoder has no parameters to be learnt. It is defined to be a linear gaussian such that the  $t^{\text{th}}$  gaussian is centered around the previous latent  $z_{t-1}$
3. The parameters for the gaussians are scheduled such that the final latent is a standard gaussian.

$$z_T \sim \mathcal{N}(z_T; 0, \mathbf{I})$$

The first restriction allows for some mild abuse of notation:

$$q_\phi(x_{1:T} | x_0) = \prod_{t=1}^T q_\phi(x_t | x_{t-1})$$

(We are using x instead of z)

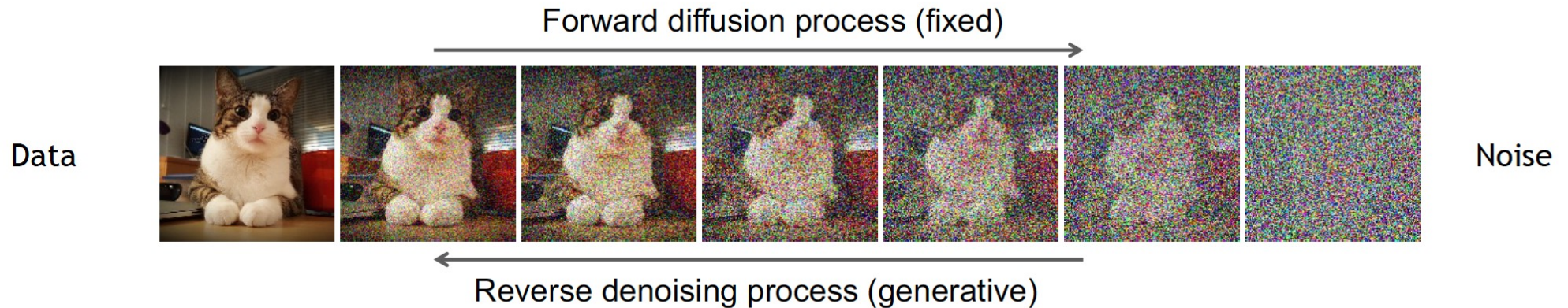
$$p(x_{0:T}) = p(x_T) \prod_{t=1}^T p_\theta(x_{t-1} | x_t)$$

# Denoising Diffusion Models

## Learning to generate by denoising

Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising



[Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015](#)

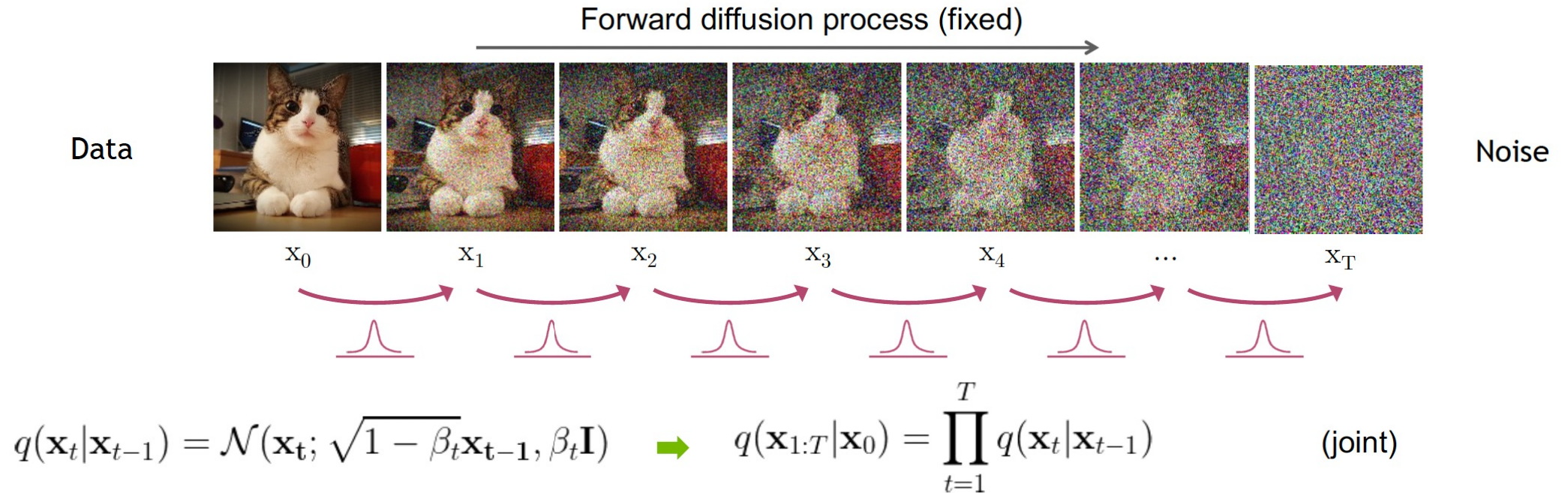
[Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020](#)

[Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021](#)

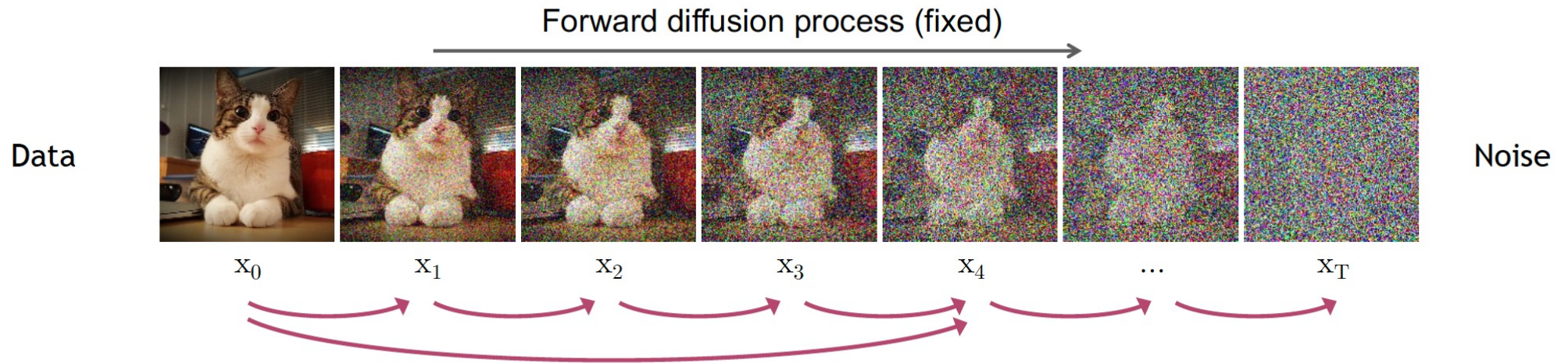
18

# Forward Diffusion Process

The formal definition of the forward process in T steps:



# Diffusion Kernel



Define  $\bar{\alpha}_t = \prod_{s=1}^t (1 - \beta_s)$   $\rightarrow$   $q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$  (Diffusion Kernel)

For sampling:  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon$  where  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

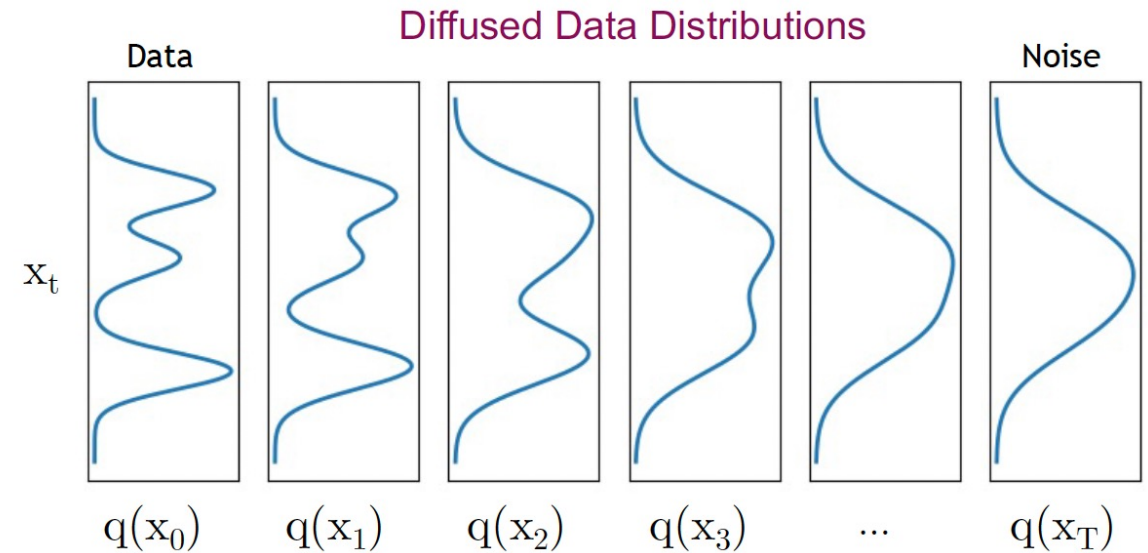
$\beta_t$  values schedule (i.e., the noise schedule) is designed such that  $\bar{\alpha}_T \rightarrow 0$  and  $q(\mathbf{x}_T | \mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

# What happens to a distribution in the forward diffusion?

So far, we discussed the diffusion kernel  $q(\mathbf{x}_t|\mathbf{x}_0)$  but what about  $q(\mathbf{x}_t)$ ?

$$q(\mathbf{x}_t) = \int \underbrace{q(\mathbf{x}_0, \mathbf{x}_t)}_{\text{Joint dist.}} d\mathbf{x}_0 = \int \underbrace{q(\mathbf{x}_0)}_{\text{Input data dist.}} \underbrace{q(\mathbf{x}_t|\mathbf{x}_0)}_{\text{Diffusion kernel}} d\mathbf{x}_0$$

The diffusion kernel is Gaussian convolution.



We can sample  $\mathbf{x}_t \sim q(\mathbf{x}_t)$  by first sampling  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$  and then sampling  $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$  (i.e., ancestral sampling).

# Generative Learning by Denoising

Recall, that the diffusion parameters are designed such that  $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

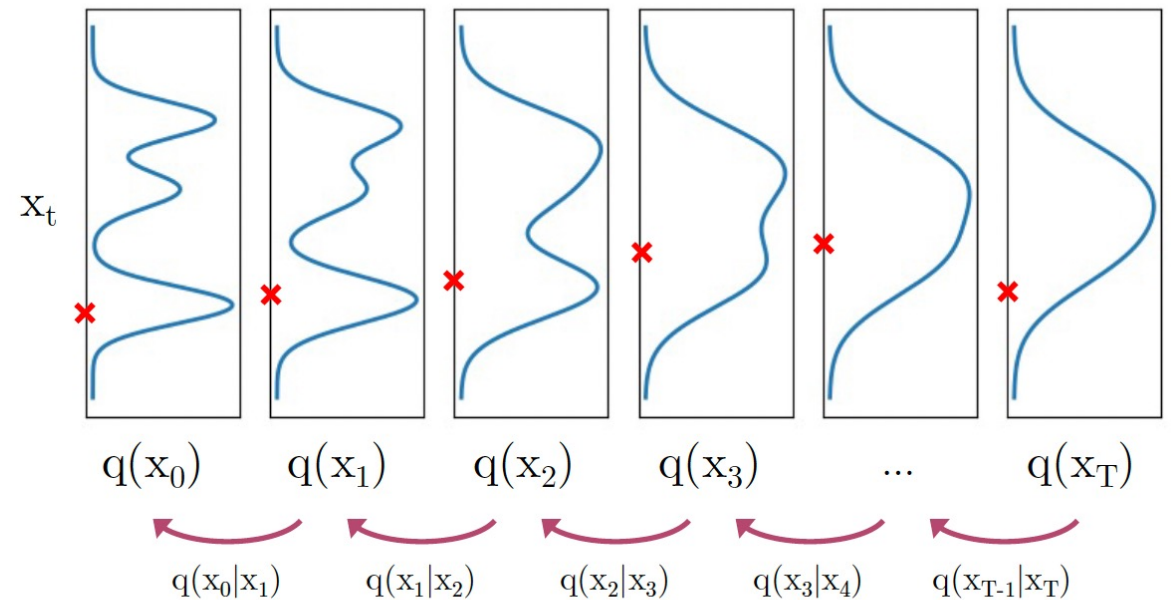
**Generation:**

Sample  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample  $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$

True Denoising Dist.

Diffused Data Distributions



# Generative Learning by Denoising

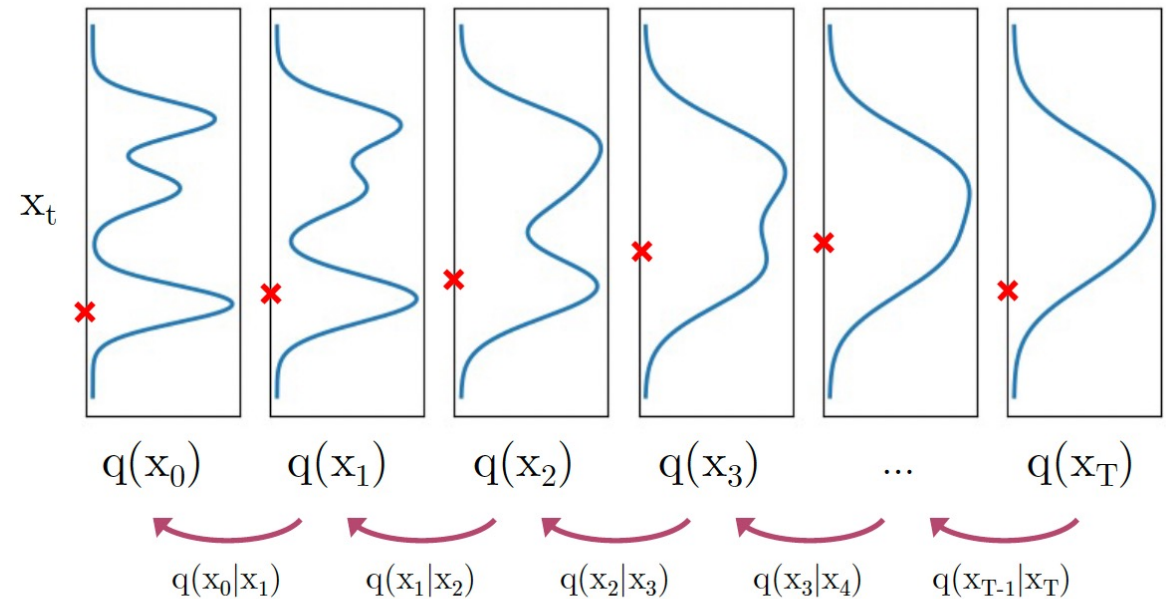
Recall, that the diffusion parameters are designed such that  $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

**Generation:**

Sample  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample  $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$

Diffused Data Distributions



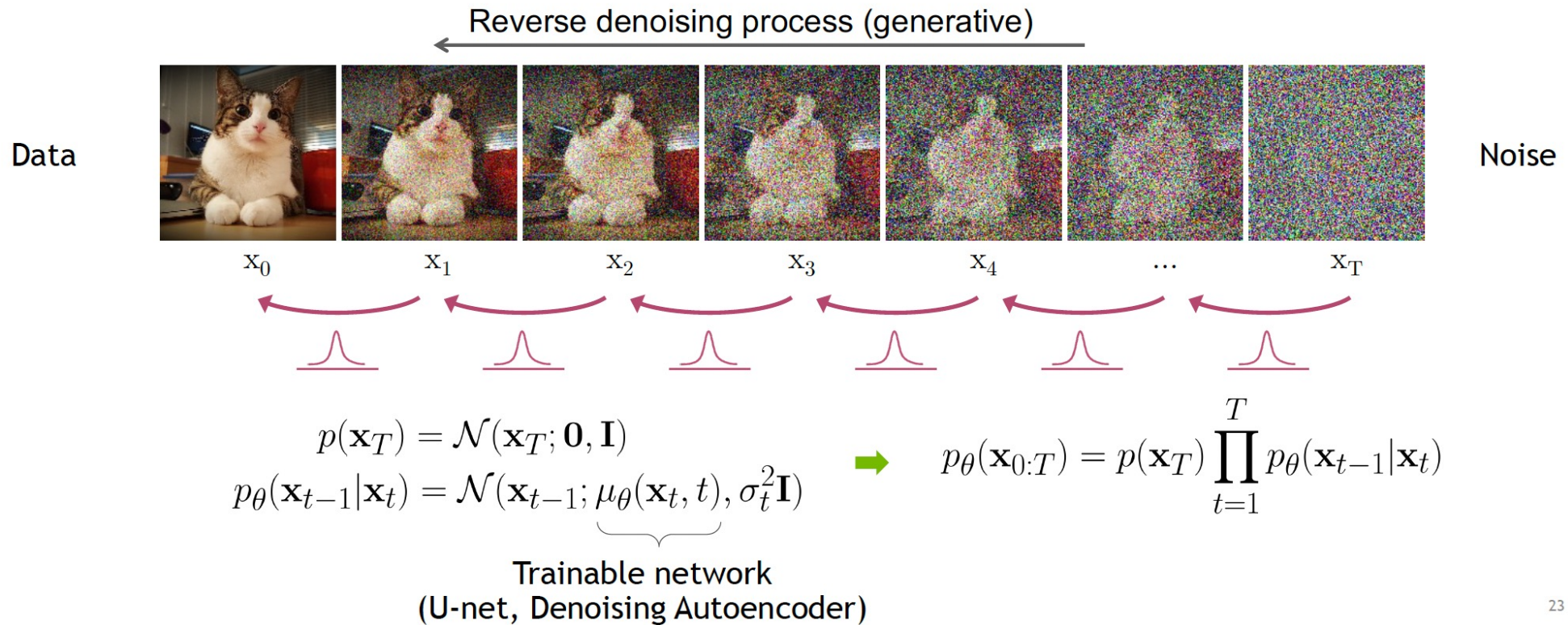
In general,  $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$  is intractable.

Can we approximate  $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ ? Yes, we can use a **Normal distribution** if  $\beta_t$  is small in each forward diffusion step.

22

# Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



# Learning Denoising Model

## Variational upper bound

For training, we can form variational upper bound that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)} [-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[ -\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] =: L$$

[Sohl-Dickstein et al. ICML 2015](#) and [Ho et al. NeurIPS 2020](#) show that:

$$L = \mathbb{E}_q \left[ \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]$$

where  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}),$$

$$\text{where } \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{1 - \beta_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t$$

# Parameterizing the Denoising Model

Since both  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  and  $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$  are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q \left[ \frac{1}{2\sigma_t^2} \|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2 \right] + C$$

Recall that  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon$ . [Ho et al. NeurIPS 2020](#) observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1 - \beta_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon \right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1 - \beta_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right)$$

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \frac{\beta_t^2}{2\sigma_t^2(1 - \beta_t)(1 - \bar{\alpha}_t)} \|\epsilon - \underbrace{\epsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t)}_{\mathbf{x}_t}\|^2 \right] + C$$

# Training Objective Weighting

## Trading likelihood for perceptual quality

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \underbrace{\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \epsilon, t)\|^2 \right]$$

The time dependent  $\lambda_t$  ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small  $t$ 's.

[Ho et al. NeurIPS 2020](#) observe that simply setting  $\lambda_t = 1$  improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1, T)} \left[ \|\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \epsilon}_{\mathbf{x}_t}, t)\|^2 \right]$$

For more advanced weighting see [Choi et al., Perception Prioritized Training of Diffusion Models, CVPR 2022](#).

# The Three Terms

$$L = \mathbb{E}_q \left[ \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} - \underbrace{\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]$$

## 3.1 Forward process and $L_T$

We ignore the fact that the forward process variances  $\beta_t$  are learnable by reparameterization and instead fix them to constants (see Section 4 for details). Thus, in our implementation, the approximate posterior  $q$  has no learnable parameters, so  $L_T$  is a constant during training and can be ignored.

the standard normal prior  $p(\mathbf{x}_T)$ . To obtain discrete log likelihoods, we set the last term of the reverse process to an independent discrete decoder derived from the Gaussian  $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \sigma_1^2 \mathbf{I})$ :

$$p_\theta(\mathbf{x}_0|\mathbf{x}_1) = \prod_{i=1}^D \int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)} \mathcal{N}(x; \mu_\theta^i(\mathbf{x}_1, 1), \sigma_1^2) dx \tag{13}$$

$$\delta_+(x) = \begin{cases} \infty & \text{if } x = 1 \\ x + \frac{1}{255} & \text{if } x < 1 \end{cases} \quad \delta_-(x) = \begin{cases} -\infty & \text{if } x = -1 \\ x - \frac{1}{255} & \text{if } x > -1 \end{cases}$$

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[ \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t) \right\|^2 \right] \tag{14}$$

# Summary

## Training and Sample Generation

---

### Algorithm 1 Training

---

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

---

### Algorithm 2 Sampling

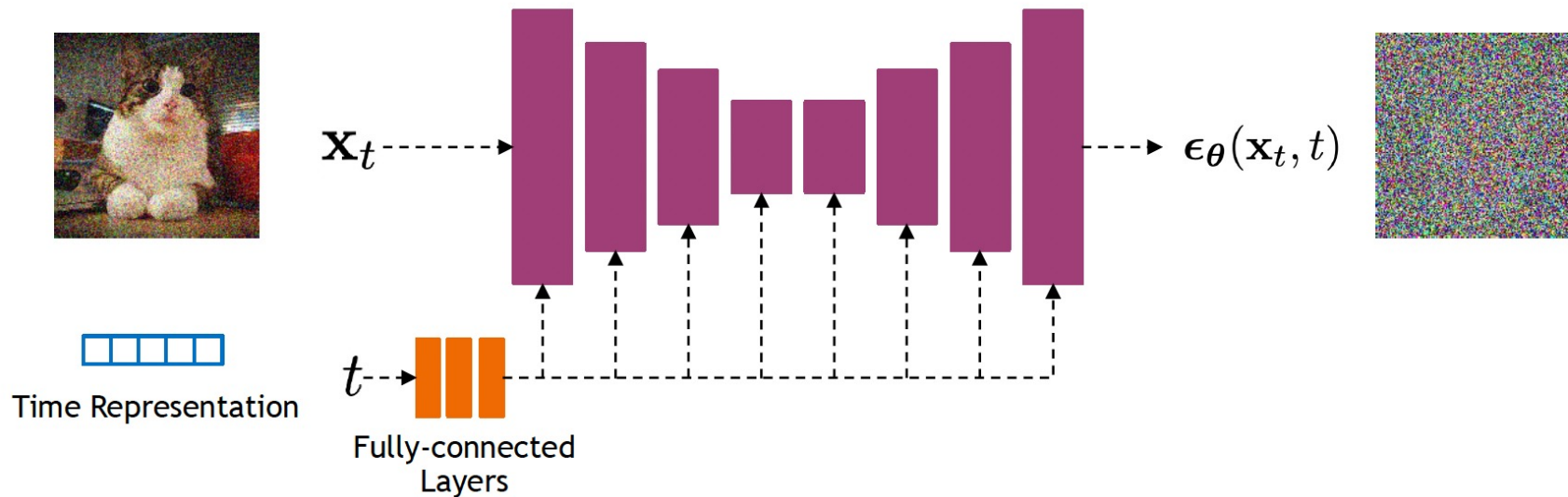
---

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

# Implementation Considerations

## Network Architectures

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent  $\epsilon_{\theta}(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see [Dharivwal and Nichol NeurIPS 2021](#))

28

# Diffusion Parameters

## Noise Schedule

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



$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$$

Above,  $\beta_t$  and  $\sigma_t^2$  control the variance of the forward diffusion and reverse denoising processes respectively.

Often a linear schedule is used for  $\beta_t$ , and  $\sigma_t^2$  is set equal to  $\beta_t$ .

[Kingma et al. NeurIPS 2022](#) introduce a new parameterization of diffusion models using signal-to-noise ratio (SNR), and show how to learn the noise schedule by minimizing the variance of the training objective.

We can also train  $\sigma_t^2$  while training the diffusion model by minimizing the variational bound ([Improved DPM by Nichol and Dhariwal ICML 2021](#)) or after training the diffusion model ([Analytic-DPM by Bao et al. ICLR 2022](#)).

# Latent Diffusion Models (Stable Diffusion)

Main differences:

- Use a pretrained encoder ( $\mathcal{E}$ ) and a decoder ( $\mathcal{D}$ )
- Conditioning with cross-attention

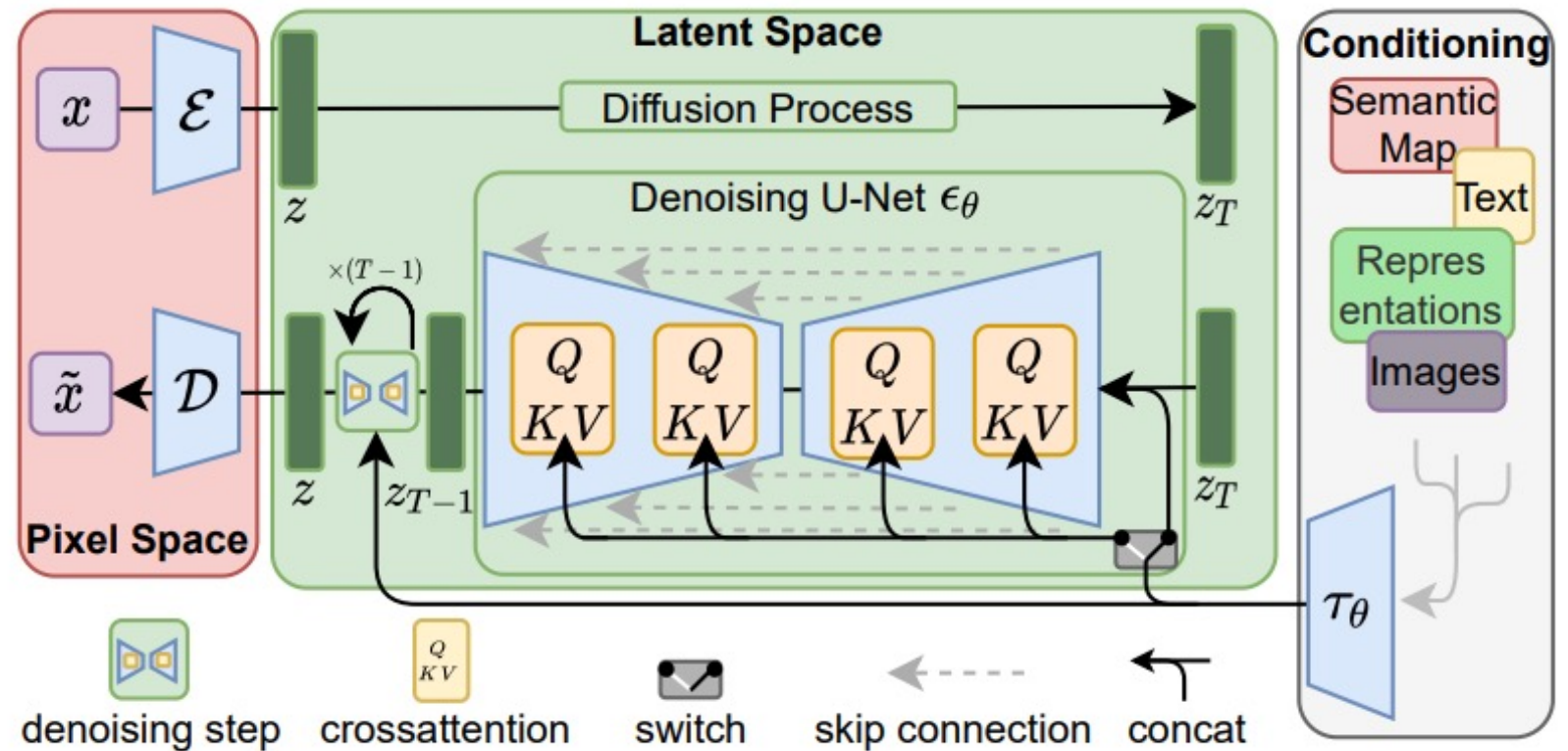


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3

## Diffusion models - Summary

- Diffusion models are **Markovian Hierarchical VAEs** with extra restrictions
- The loss is the vanilla VAE ELBO loss with an added denoising term
- The encoder has **0 parameters**
- The true denoising posterior can be **exactly calculated**
- The problem can be reformulated as a noise prediction problem
- There's a ton of math underlying a rather simple intuition