CENG501 – Deep Learning

Week 12

Fall 2024

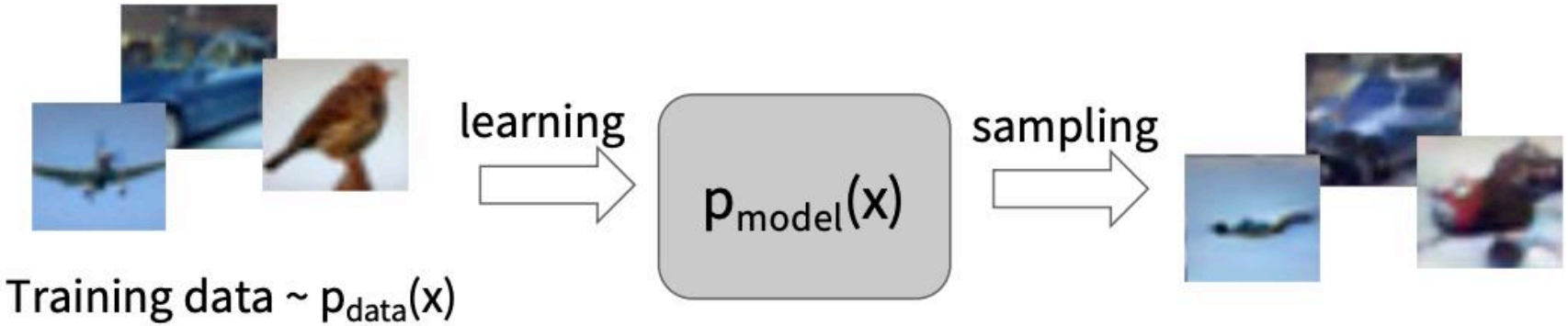
Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on CSENG501

Generative Modeling

- Learning the probability distribution of data



Objectives:

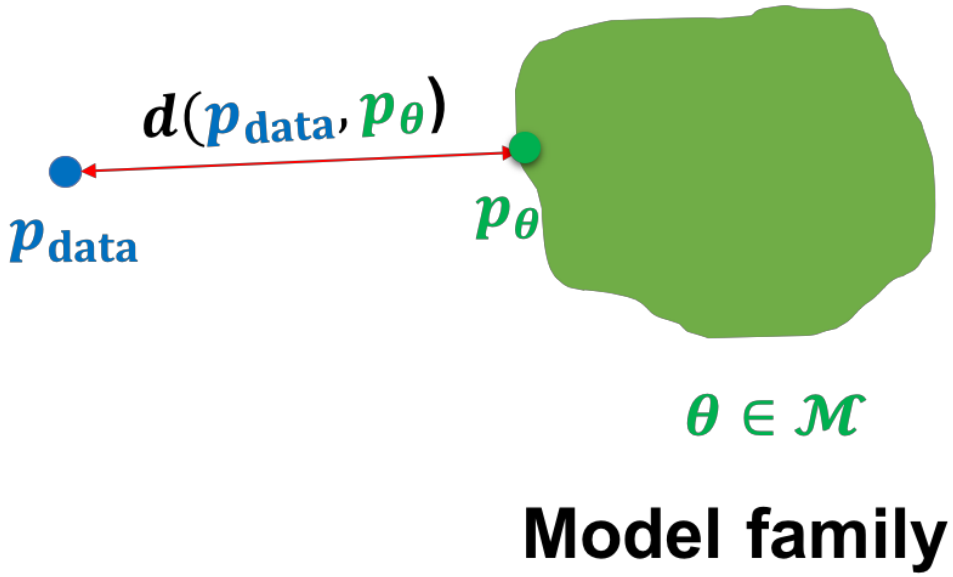
1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. Sampling new x from $p_{model}(x)$

Figure: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CSCENG501

Generative Modeling

- Learning the probability distribution of data



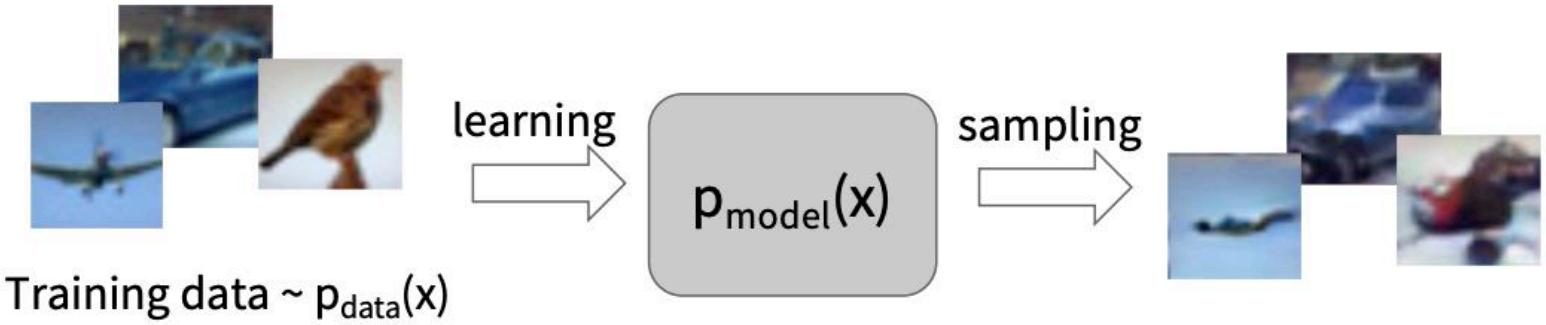
$p_\theta \equiv p_{\text{model}}$

Figure: https://deepgenerativemodels.github.io/notes/introduction/

Previously on ECENG501

Generative Modeling

- Learning the probability distribution of data



Formulate as density estimation problems:

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

Figure: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Taxonomy of Generative Models

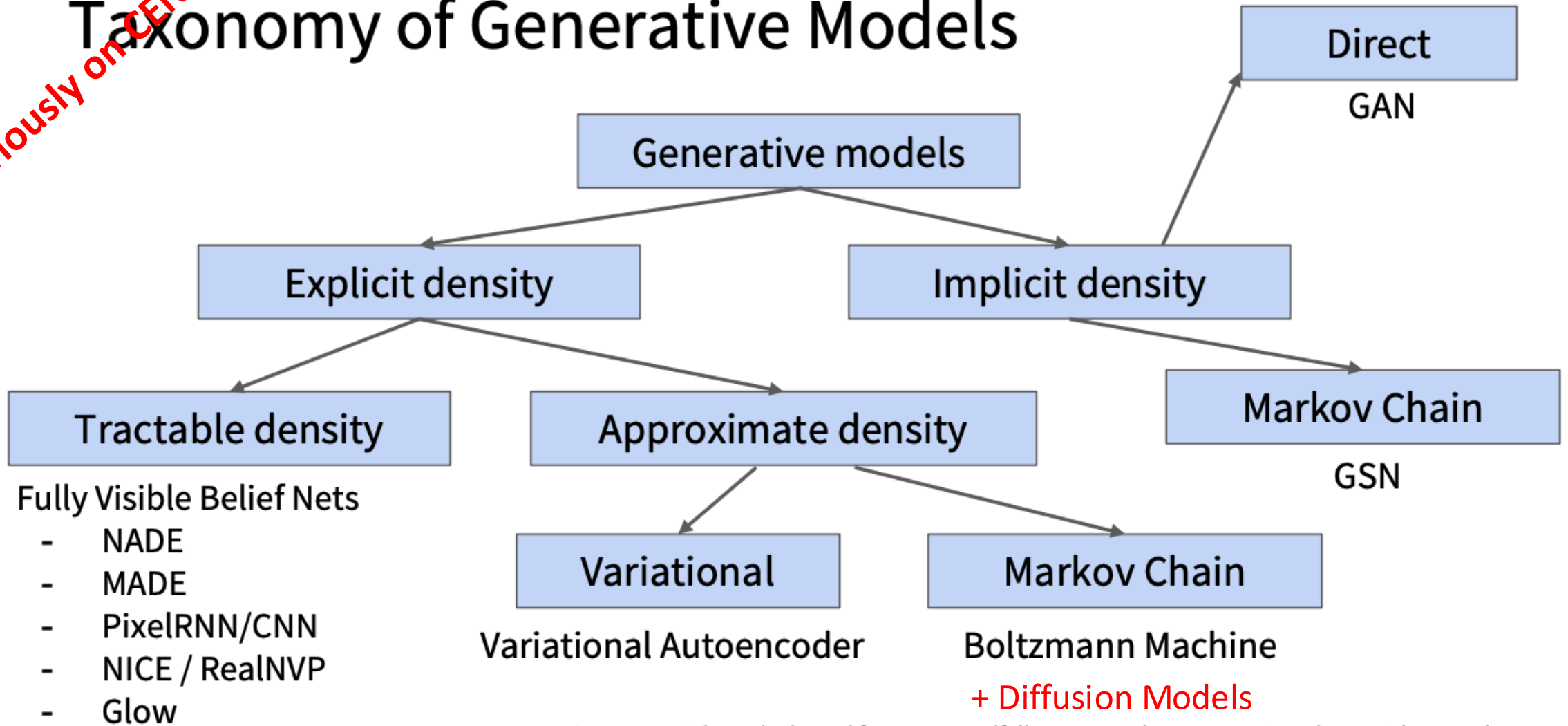


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Fully visible belief network (FVBN)

Explicit density model

$$p(x) = p(x_1, x_2, \ldots, x_n)$$

Likelihood of image x

Joint likelihood of each pixel in the image

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Fully visible belief network (FVBN)

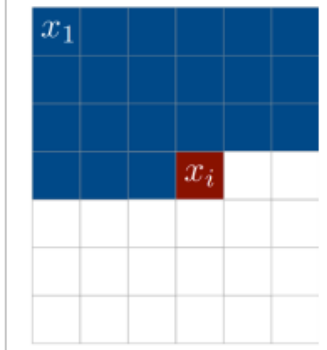
Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels



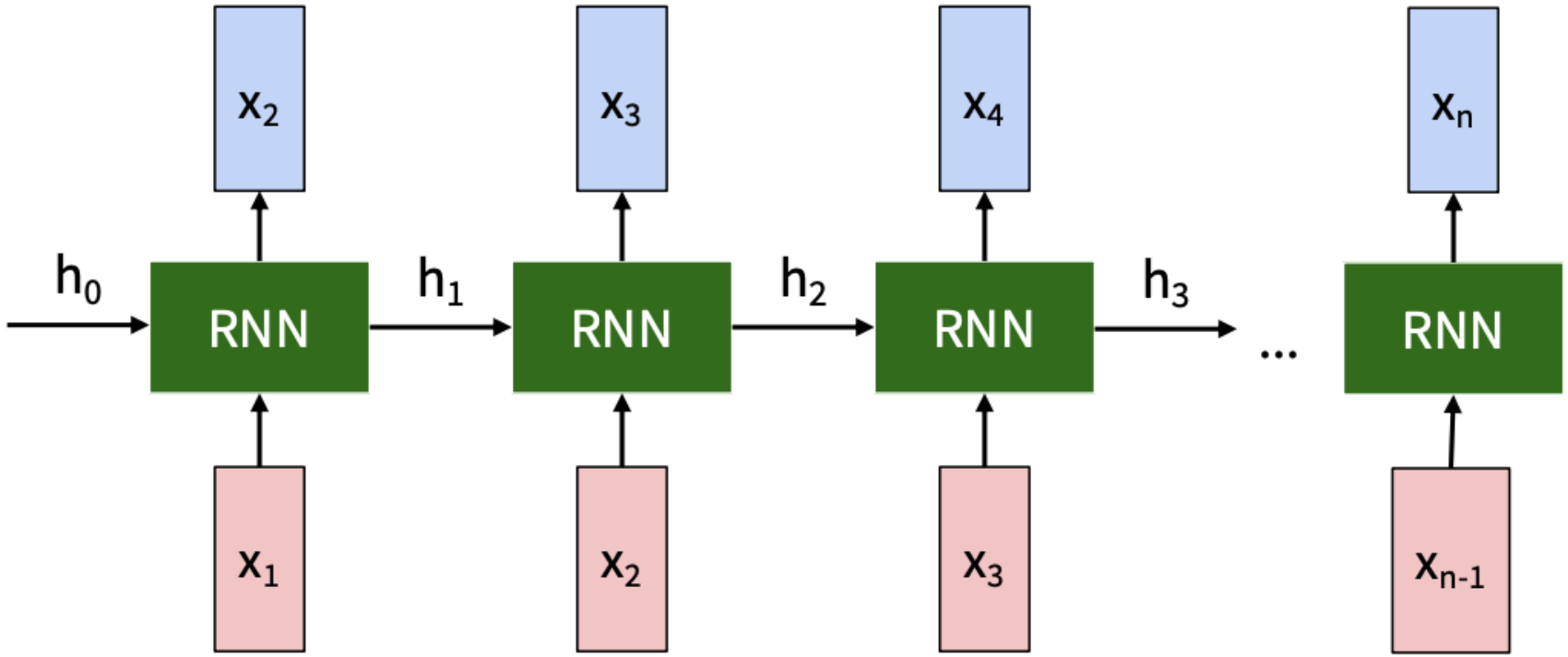
Then maximize likelihood of training data

Complex distribution over pixel values => Express using a neural network!

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Recurrent Neural Network



$$p(x_i|x_1, \ldots, x_{i-1})$$

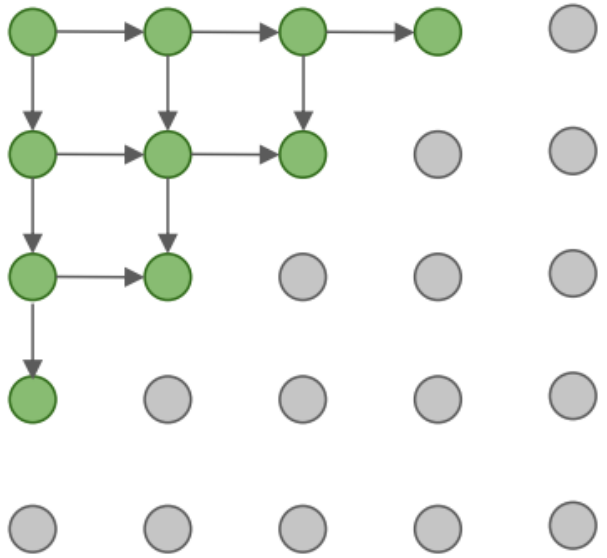
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

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!

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

PixelCNN [van der Oord et al. 2016]

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

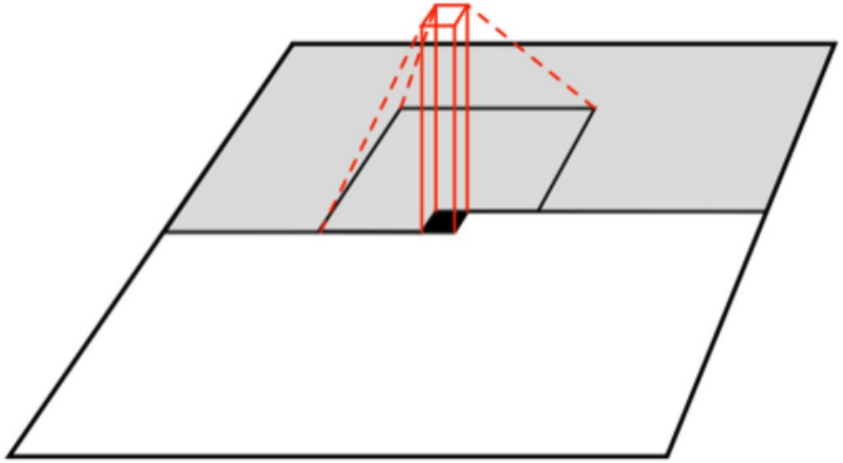


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

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++)

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Autoregressive Models vs Variational Autoencoders

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

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

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?

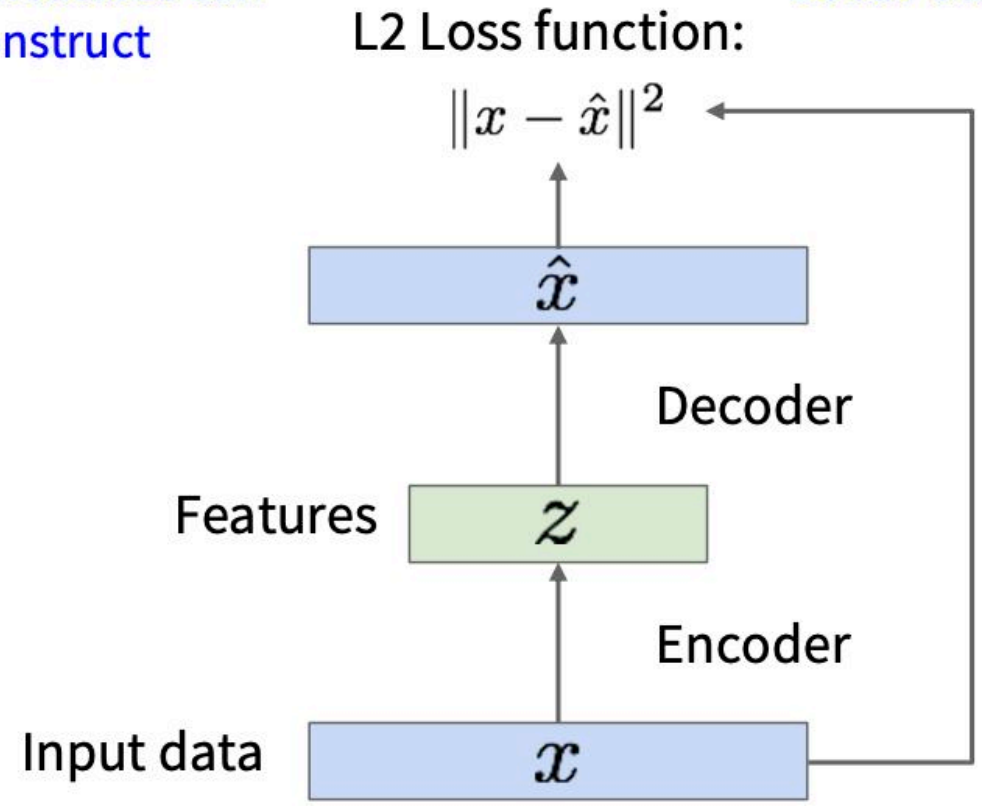
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

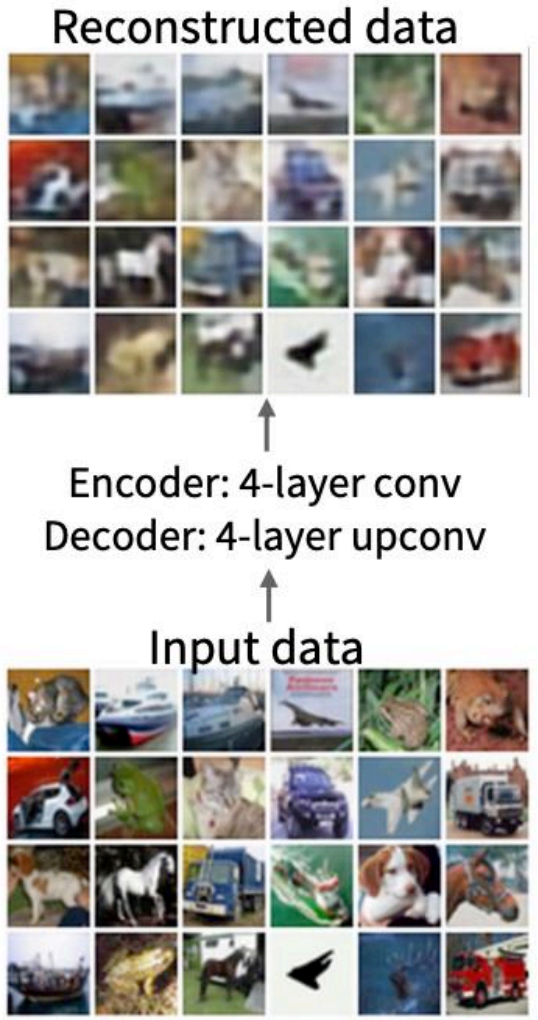
Previously on CSENG501

Recap: Autoencoders

Train such that features can be used to reconstruct original data

Doesn't use labels!





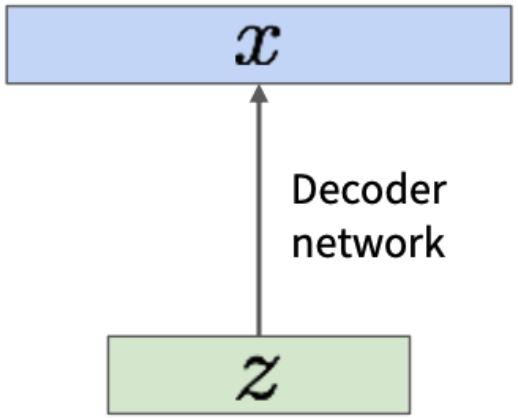
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Variational Autoencoders (VAEs)

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

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



We want to estimate the true parameters θ^* 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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Variational Autoencoders (VAEs): Intractability

😢 ✓ ✓

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

Posterior density: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

Intractable data likelihood

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Variational Autoencoders (VAEs)

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

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

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

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

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

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution close to prior

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

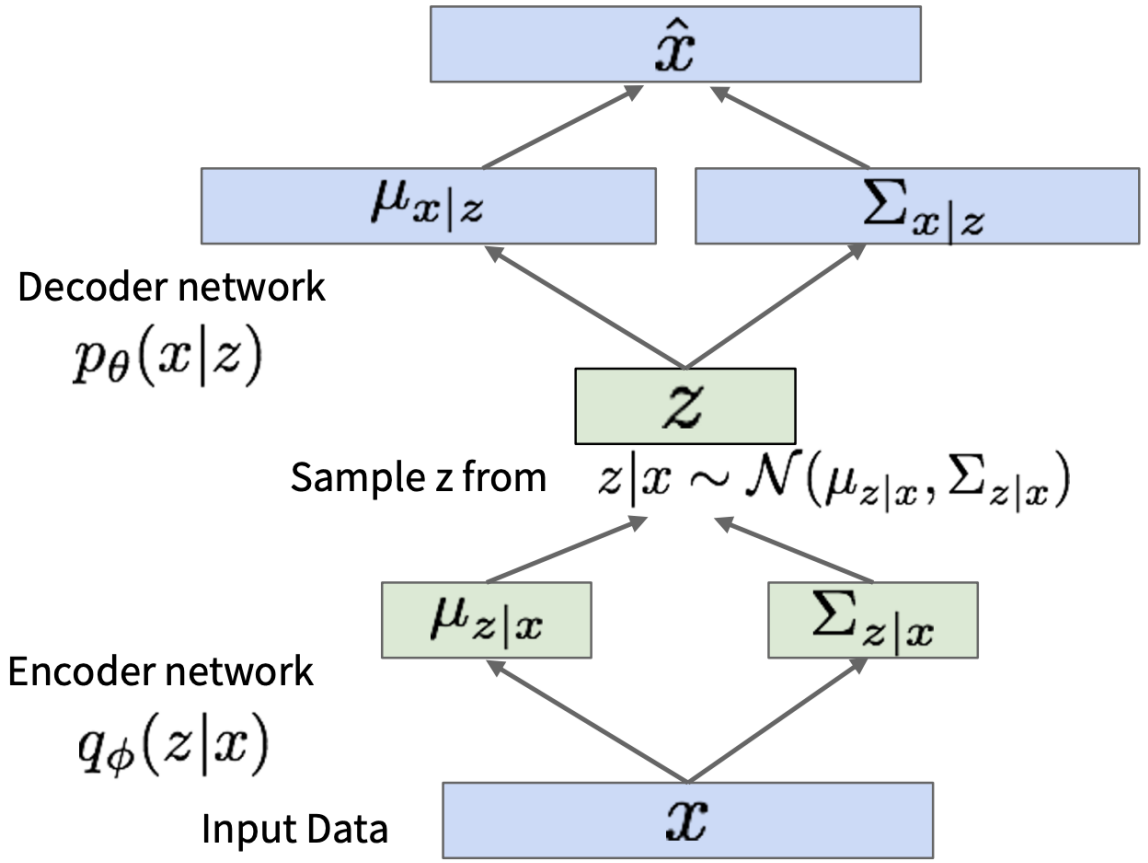
Previously on CENG501

Variational Autoencoders (VAEs)

Putting it all together: maximizing the likelihood lower bound

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

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



Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

Variational Autoencoders (VAEs)

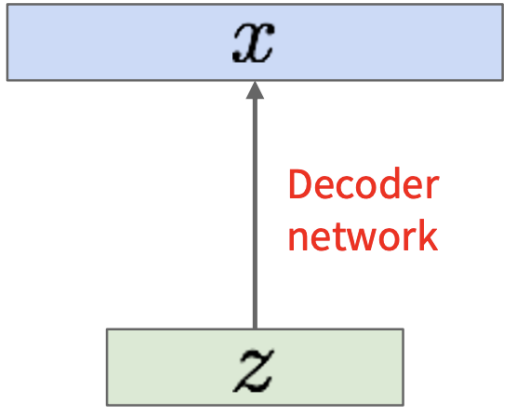
Our assumption about data generation process

Sample from true conditional

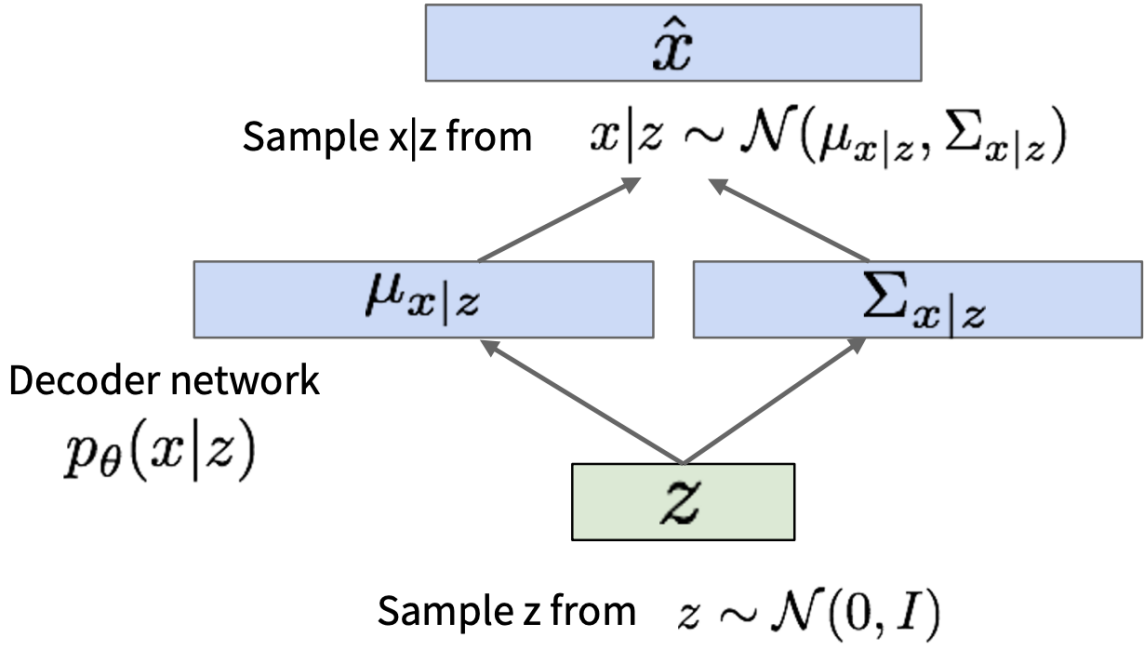
$p_{\theta^*}(x \mid z^{(i)})$

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

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 $q_\phi(z|x)$!

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

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

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.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on ECENG501

Normalizing Flow

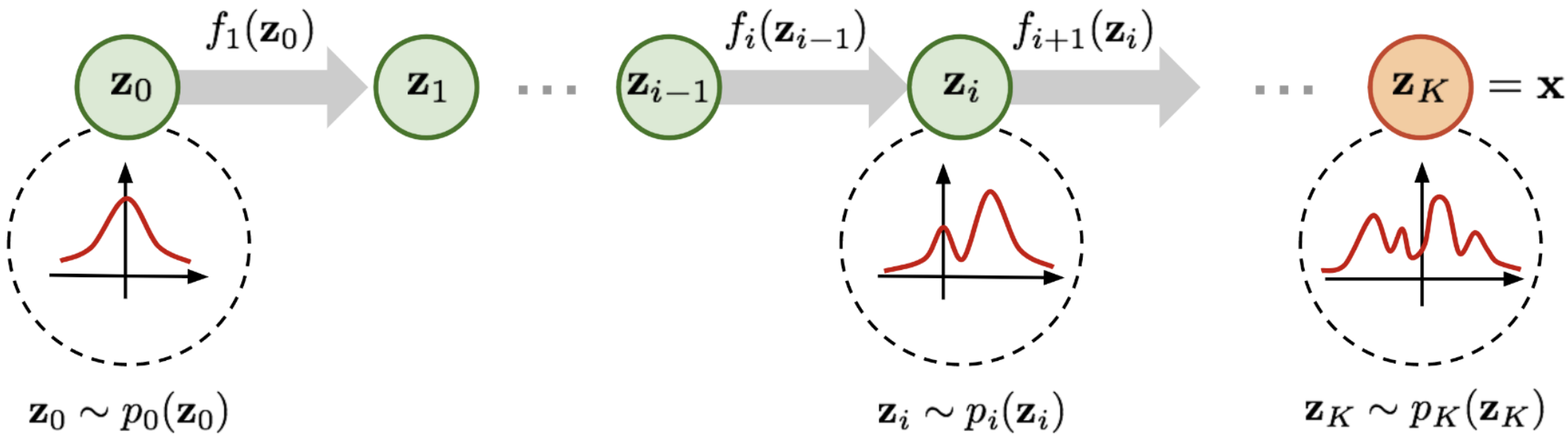


Fig. 2. Illustration of a normalizing flow model, transforming a simple distribution $p\_0(\mathbf{z}\_0)$ to a complex one $p\_K(\mathbf{z}\_K)$ step by step.

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

Previously on CENG501

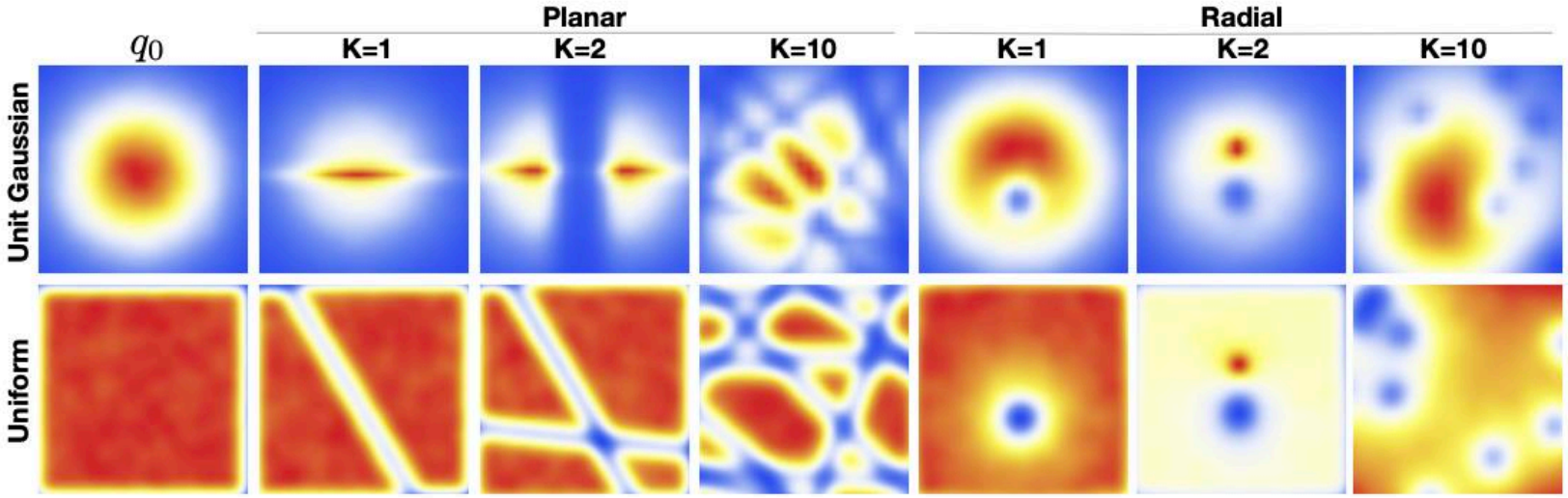


Figure 1. Effect of normalizing flow on two distributions.

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

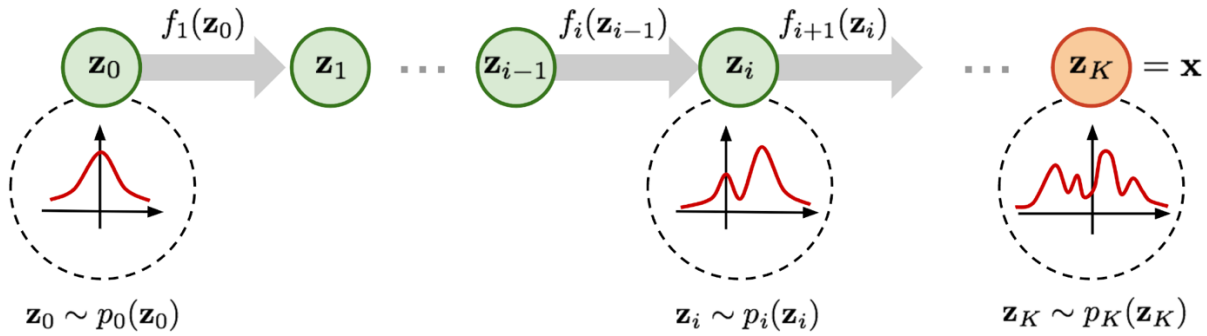
Previously on CENG501

Training

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

Pseudo-code

1. $\mathbf{x} \leftarrow$ Sample a batch
2. $\mathbf{z}_0 \sim p_\theta(\mathbf{z}_0 \,|\mathbf{x})$
3. $\mathbf{z}_K \leftarrow f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0)$
4. $\Delta\theta \propto -\nabla_\theta \, d(\mathbf{x}, \mathbf{z}_K)$



Previously on ENG501

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

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

Previously on CENG501

Normalizing Flows

- 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

CENG796 DEEP GENERATIVE MODELS

Course Code:	5710796
METU Credit (Theoretical-Laboratory hours/week):	3(3-0)
ECTS Credit:	8.0
Department:	Computer Engineering
Language of Instruction:	English
Level of Study:	Graduate
Course Coordinator:	Assoc.Prof.Dr. RAMAZAN GÖKBERK CİNBİŞ
Offered Semester:	Fall Semesters.

Course Objectives

At the end of the course, the students will be expected to:

- Comprehend a variety of deep generative models.
- Apply deep generative models to several problems.
- Know the open issues in learning deep generative models, and have a grasp of the current research directions.

Course Content

Deep generative modeling with Autoregressive models; Energy-based models; Adversarial models; Variational models.

Administrative Notes

- No quiz this week

- Time plan for the projects
  1. Milestone (November 24, midnight):
     - Github repo will be ready
     - Read & understand the paper
     - Download the datasets
     - Prepare the Readme file excluding the results & conclusion
  2. Milestone (December 8, midnight)
     - The results of the first experiment
  3. Milestone (January 5, midnight)
     - Final report (Readme file)
     - Repo with all code & trained models

Taxonomy of Generative Models

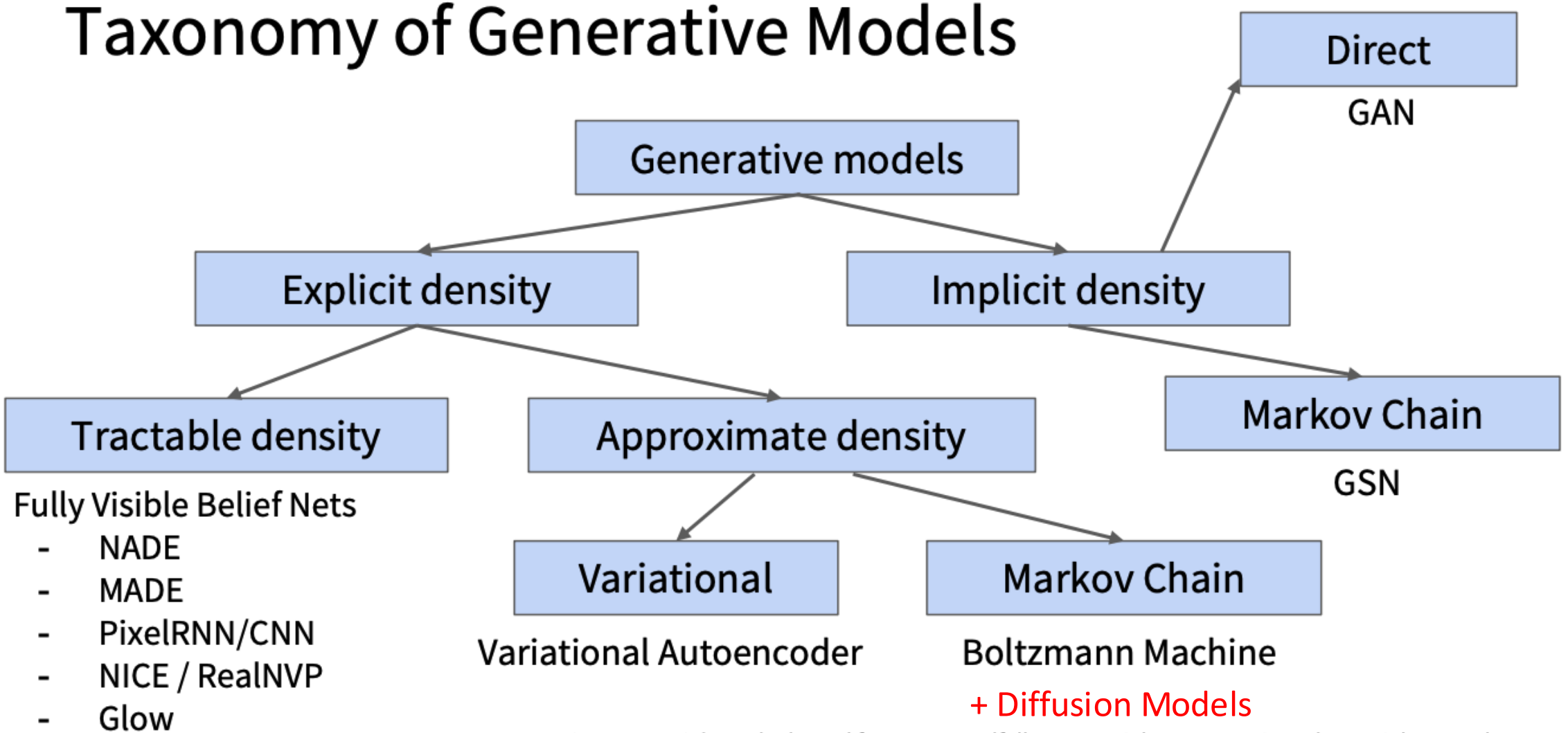
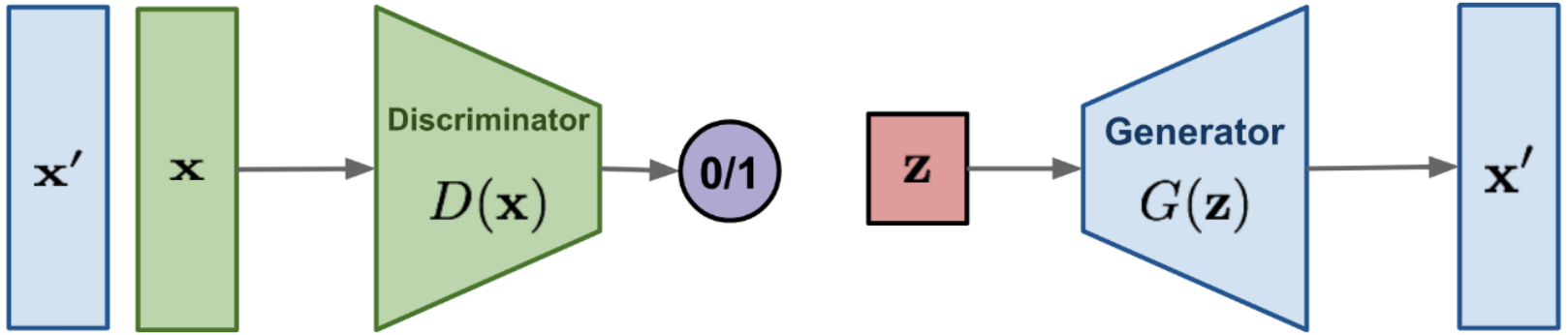


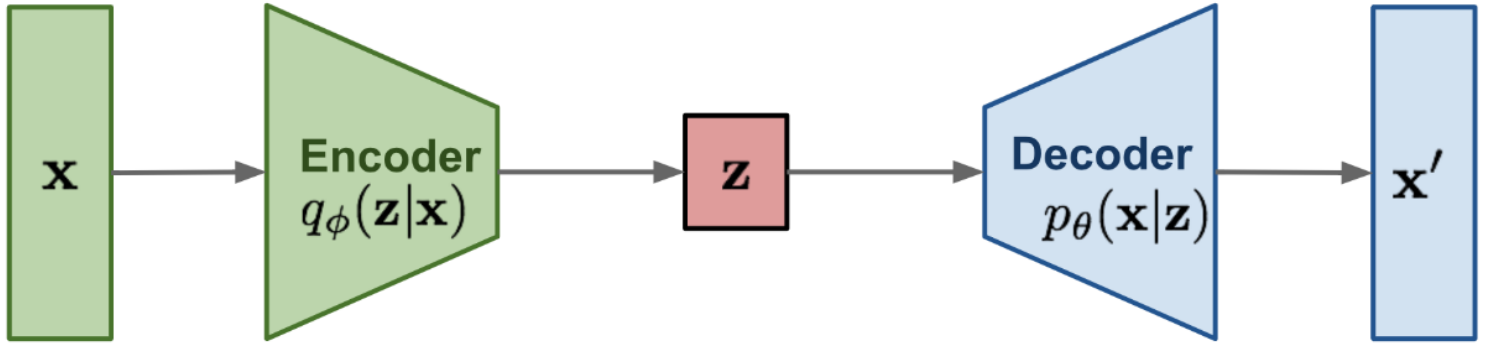
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

GAN: minimax the classification error loss.



VAE: maximize ELBO.



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

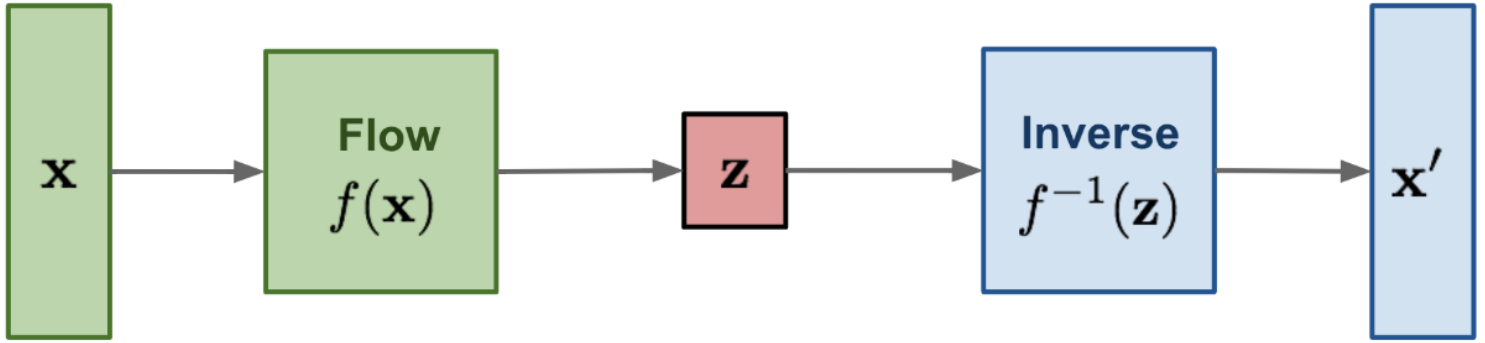
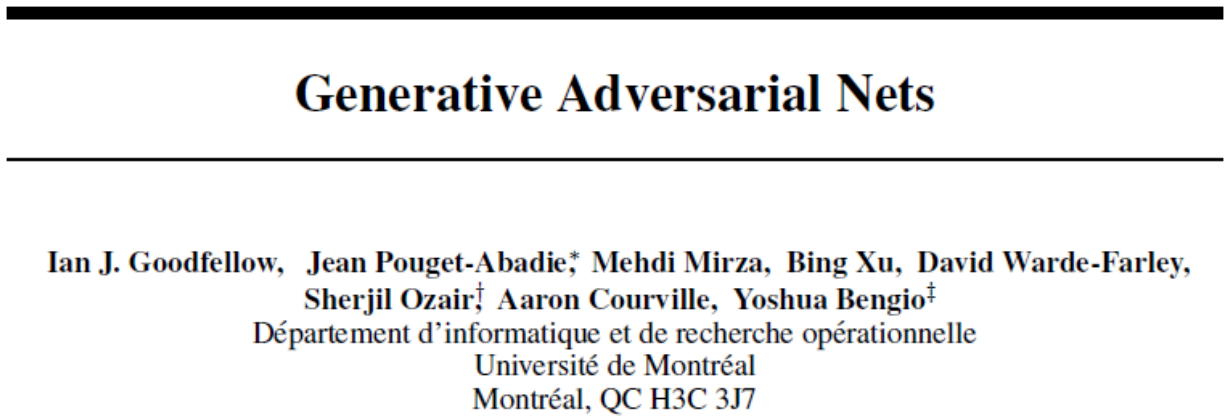


Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Generative Adversarial Networks

Generative Adversarial Networks (GANs)

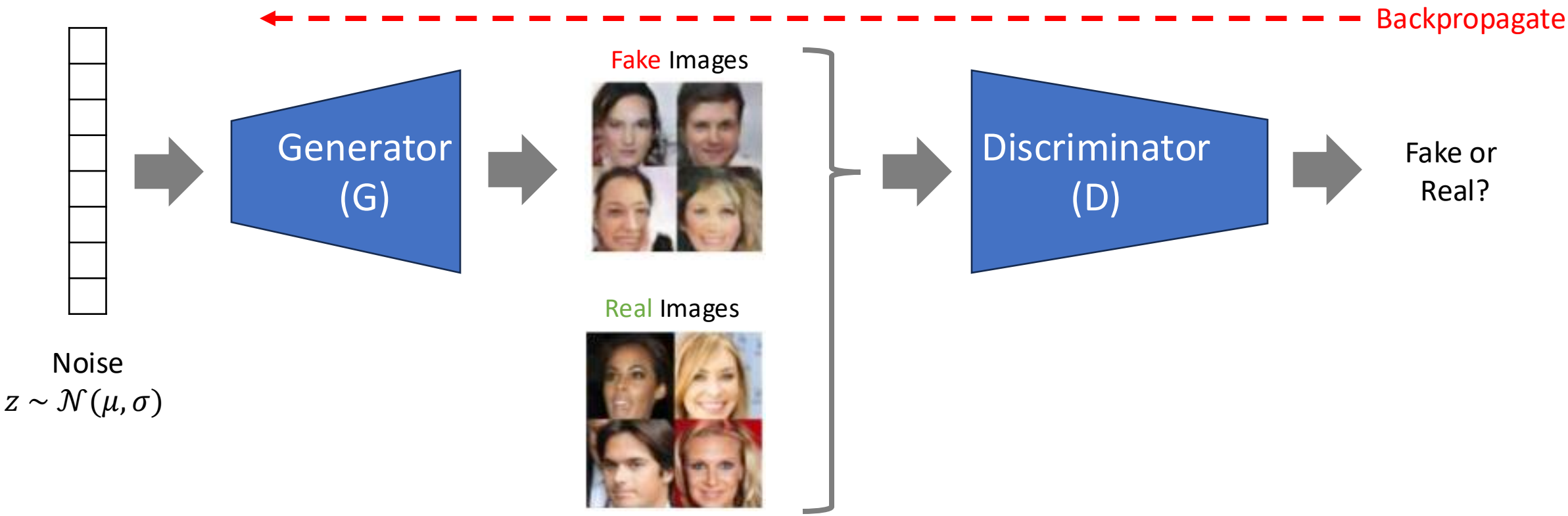
- Originally proposed by Ian Goodfellow in 2014
- Won the “Test of Time” award this year at NeurIPS2024
  - https://blog.neurips.cc/2024/11/27/announcing-the-neurips-2024-test-of-time-paper-awards/
- It all started in a pub ☺
  - Full story here: https://x.com/sherjilozair/status/1864013580624113817



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)

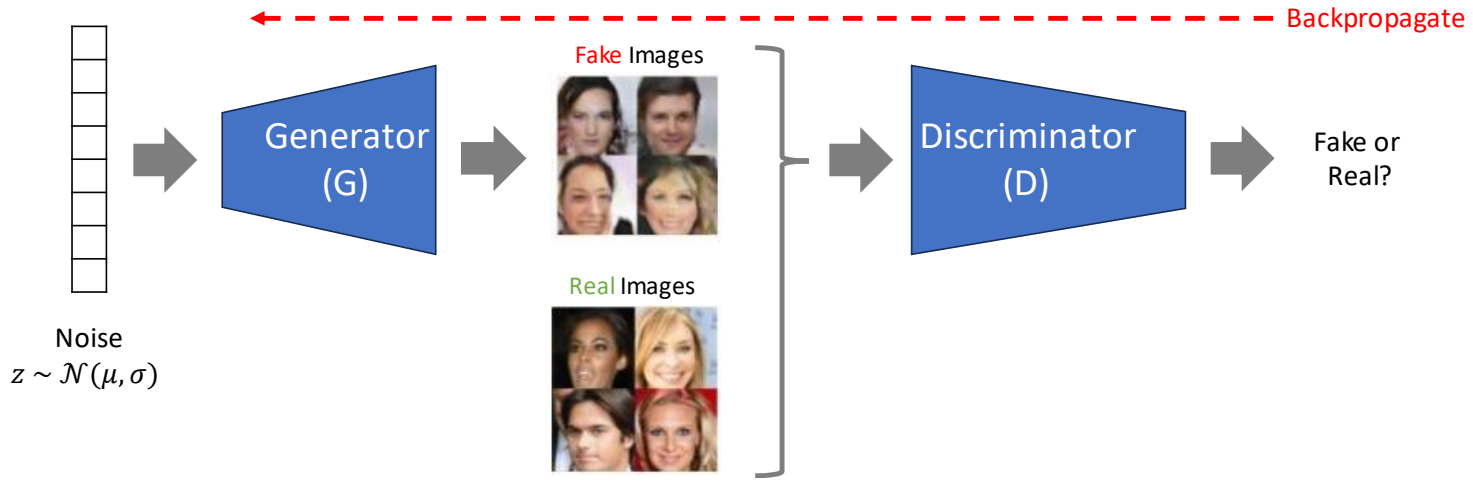


We have two networks:

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

http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them

Generative Adversarial Networks (GANs)



- With two competing networks, we solve the following minimax game:
  $\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$
- Discriminator's objective:
  $\max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$
- Generator's objective:
  $\min_G V(D,G) = E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$

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

Adapted from: http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them

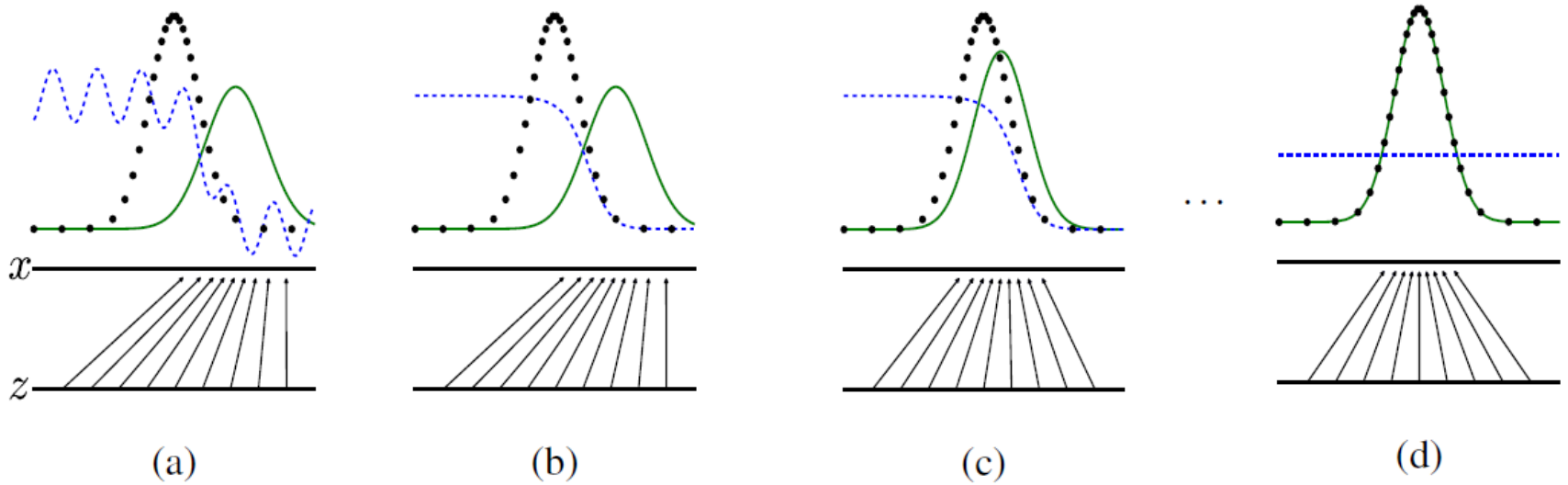


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ 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_{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 $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ 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. $D(\boldsymbol{x}) = \frac{1}{2}$.

Fig: Goodfellow et al., 2014.

Alg: Goodfellow et al., 2014.

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

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

end for

Discriminator

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

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

Generator

end for

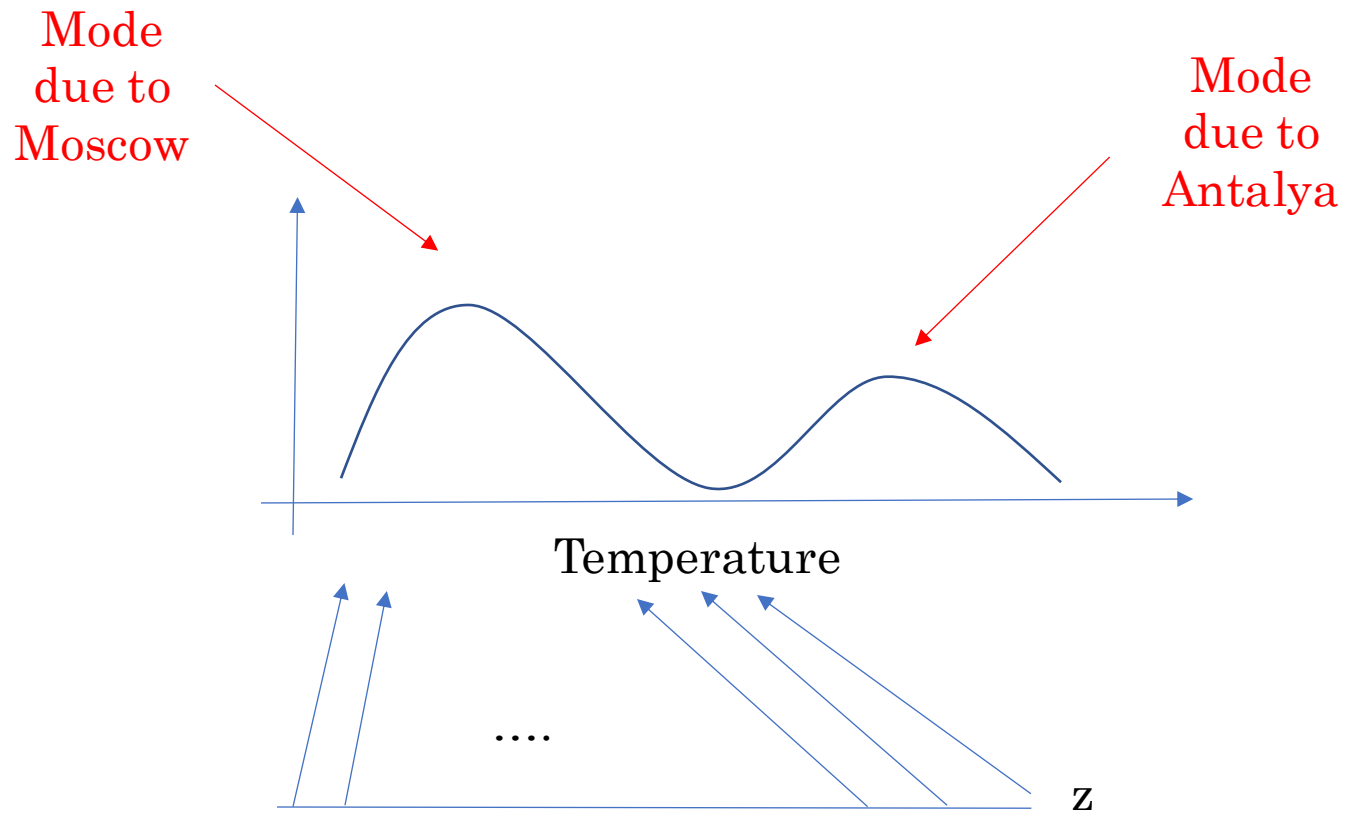
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

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

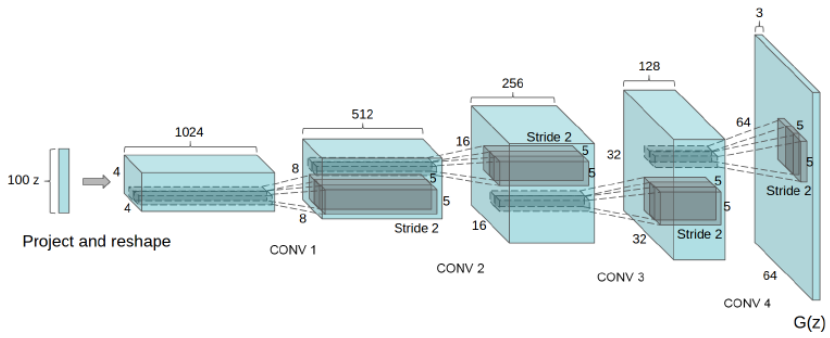
- Solutions:
  - Changing the training procedure (use batch discrimination instead of individual discrimination)
  - Experience replay (show old fake images again and again)
  - Use a different loss (+ enforce diversity)
  - …
- Other tips and tricks:
  - https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

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.

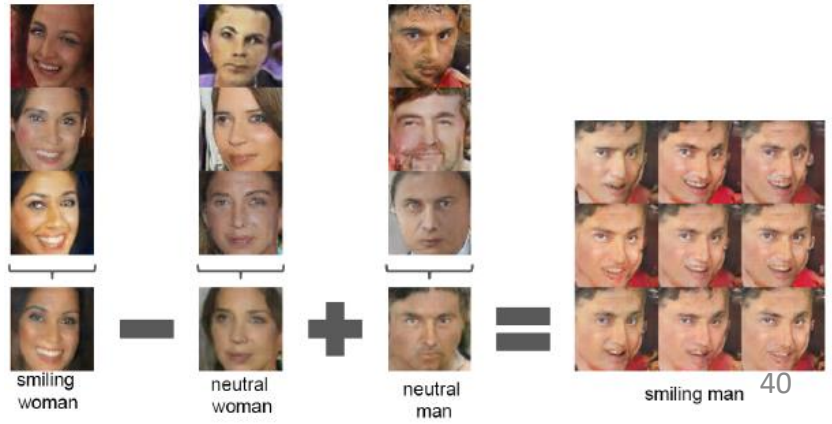


Under review as a conference paper at ICLR 2016

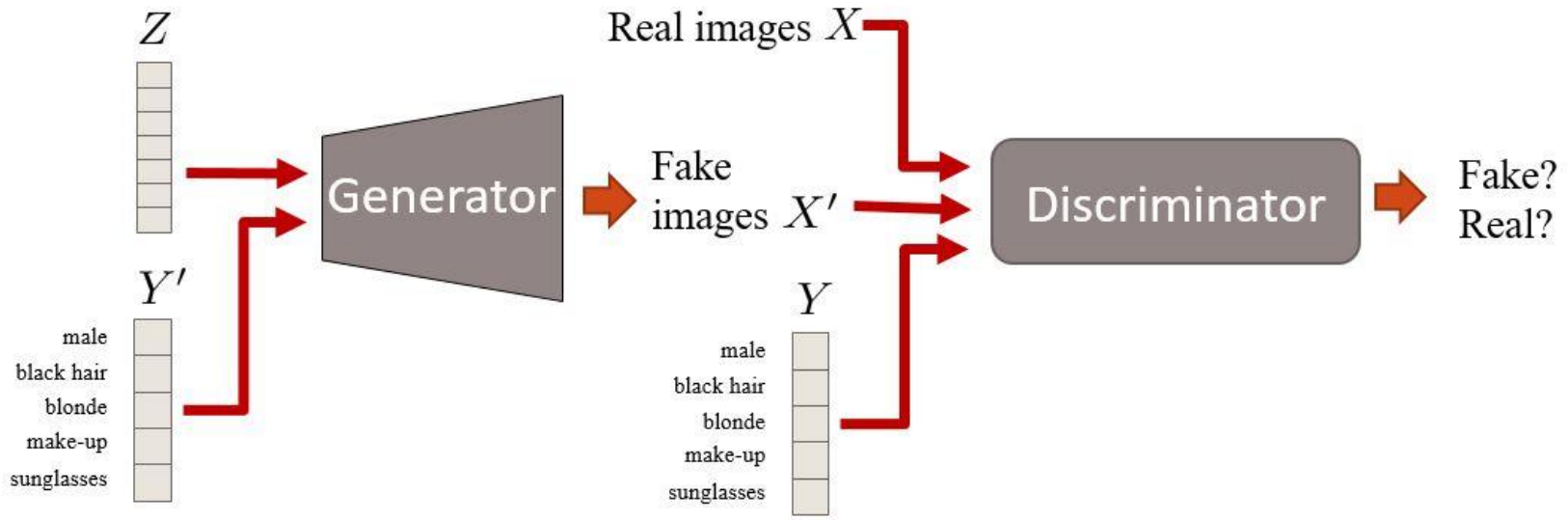
UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

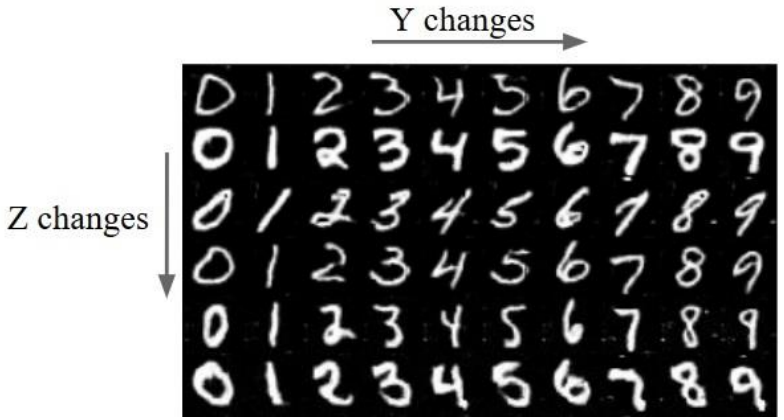
Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

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



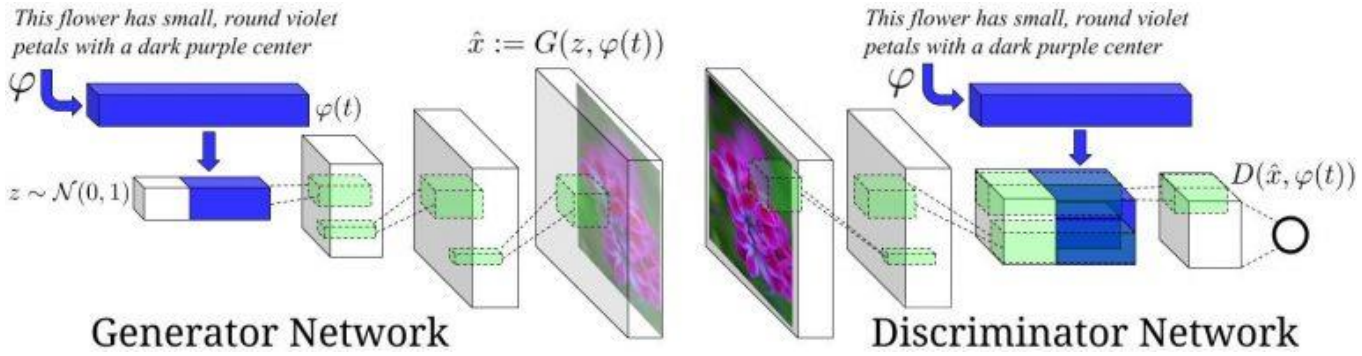
Conditional GANs





http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them

Text to image with GANs



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

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee, 2016.

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

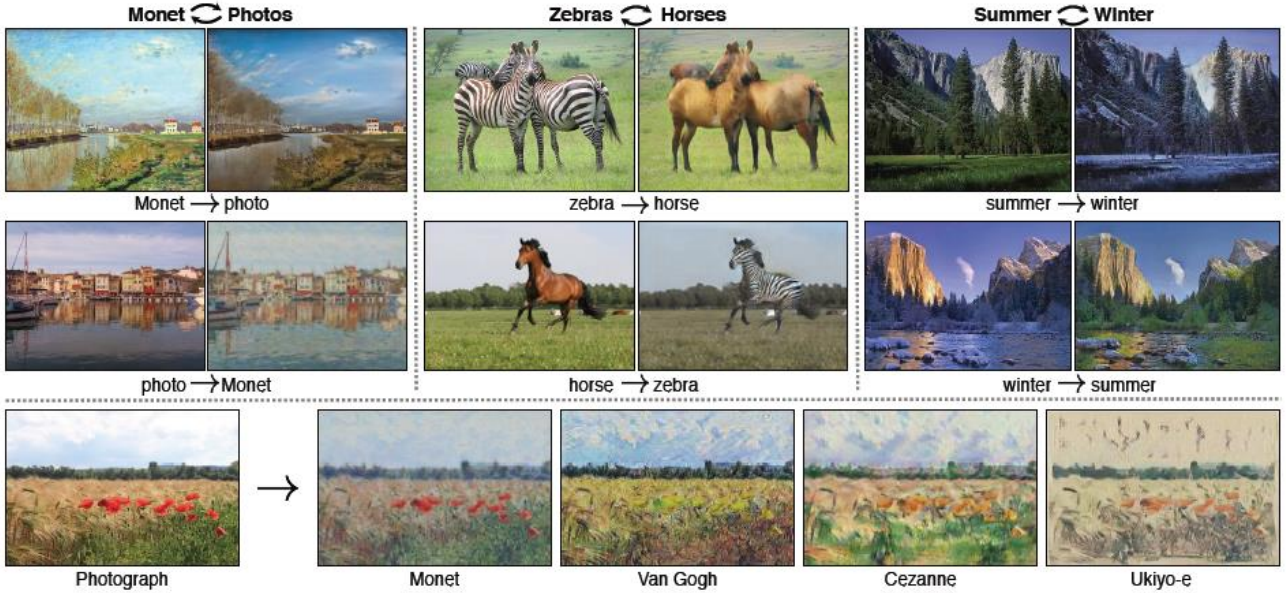
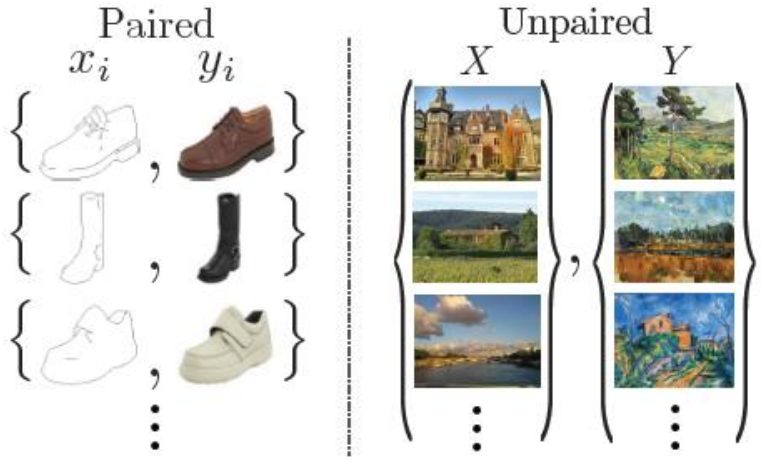


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* 1074 Monet paintings and 6753 landscape photos from Flickr; *(center)* 1177 zebras and 939 horses from ImageNet; *(right)* 1273 summer 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.



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

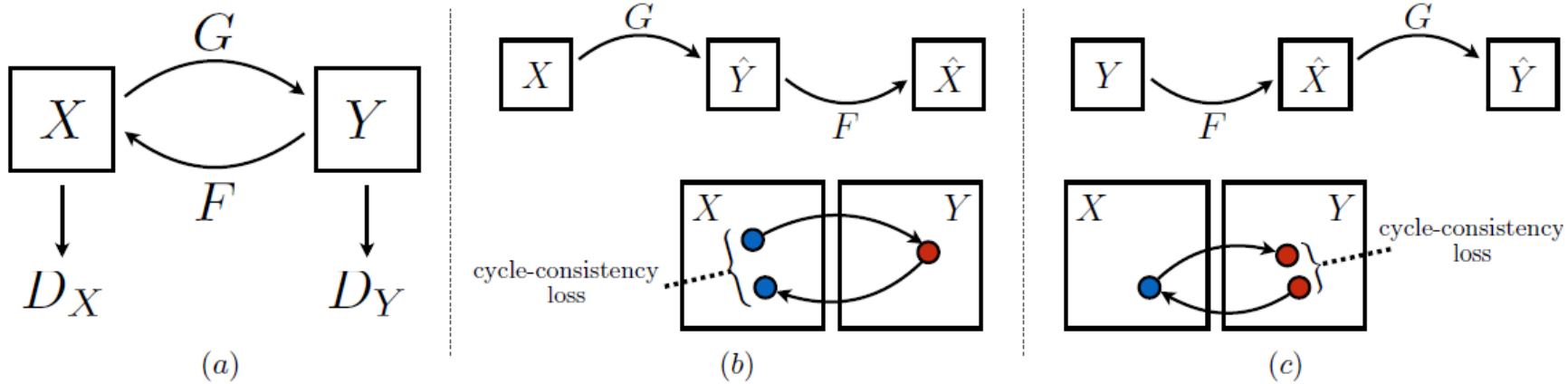


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, 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/

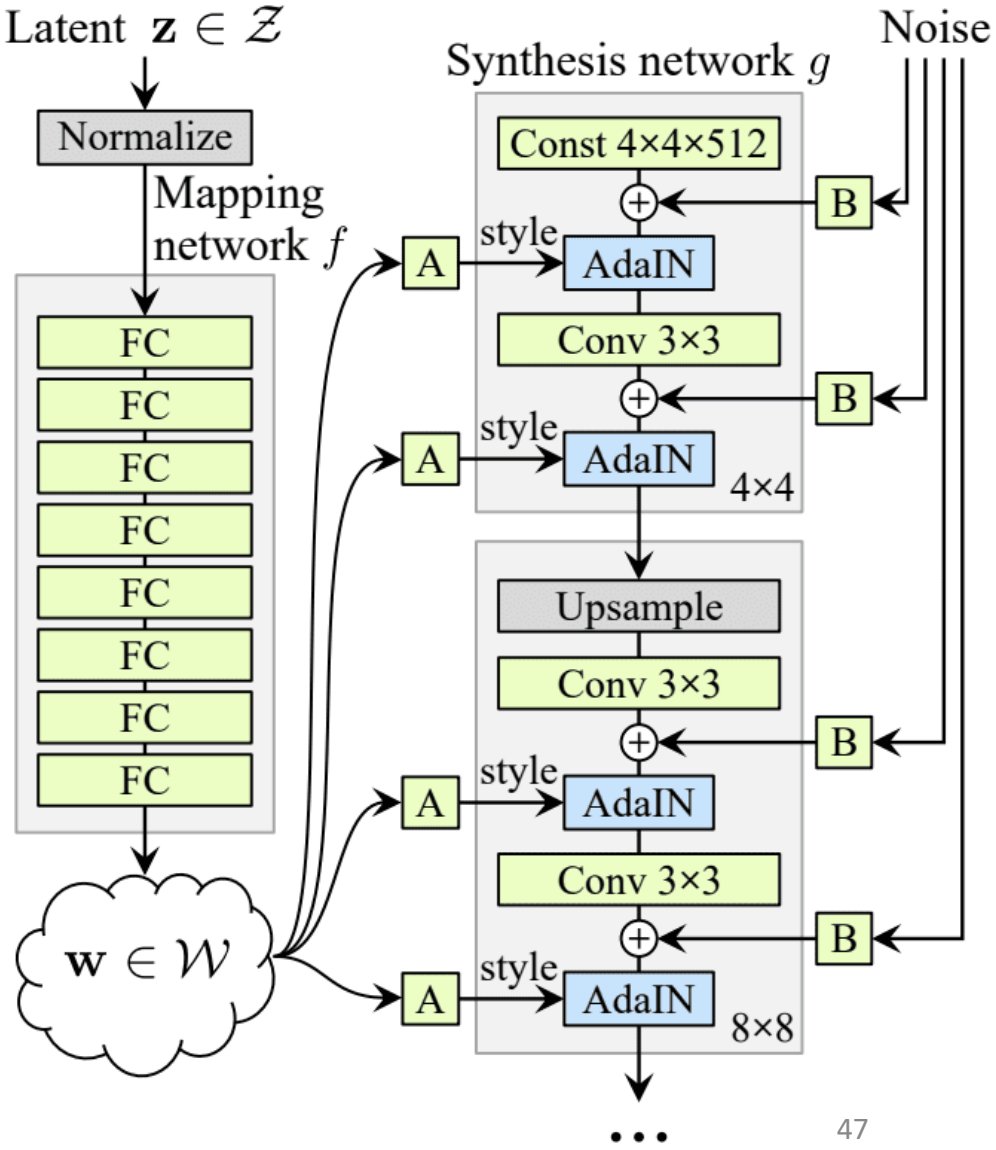
Example

https://www.digitaltrends.com/cool-tech/nvidia-ai-winter-summer-car/

GAN -- state of the art

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

https://github.com/NVlabs/stylegan2



The zoo of GANs

- https://deephunt.in/the-gan-zoo-79597dc8c347

Sample from Our Work

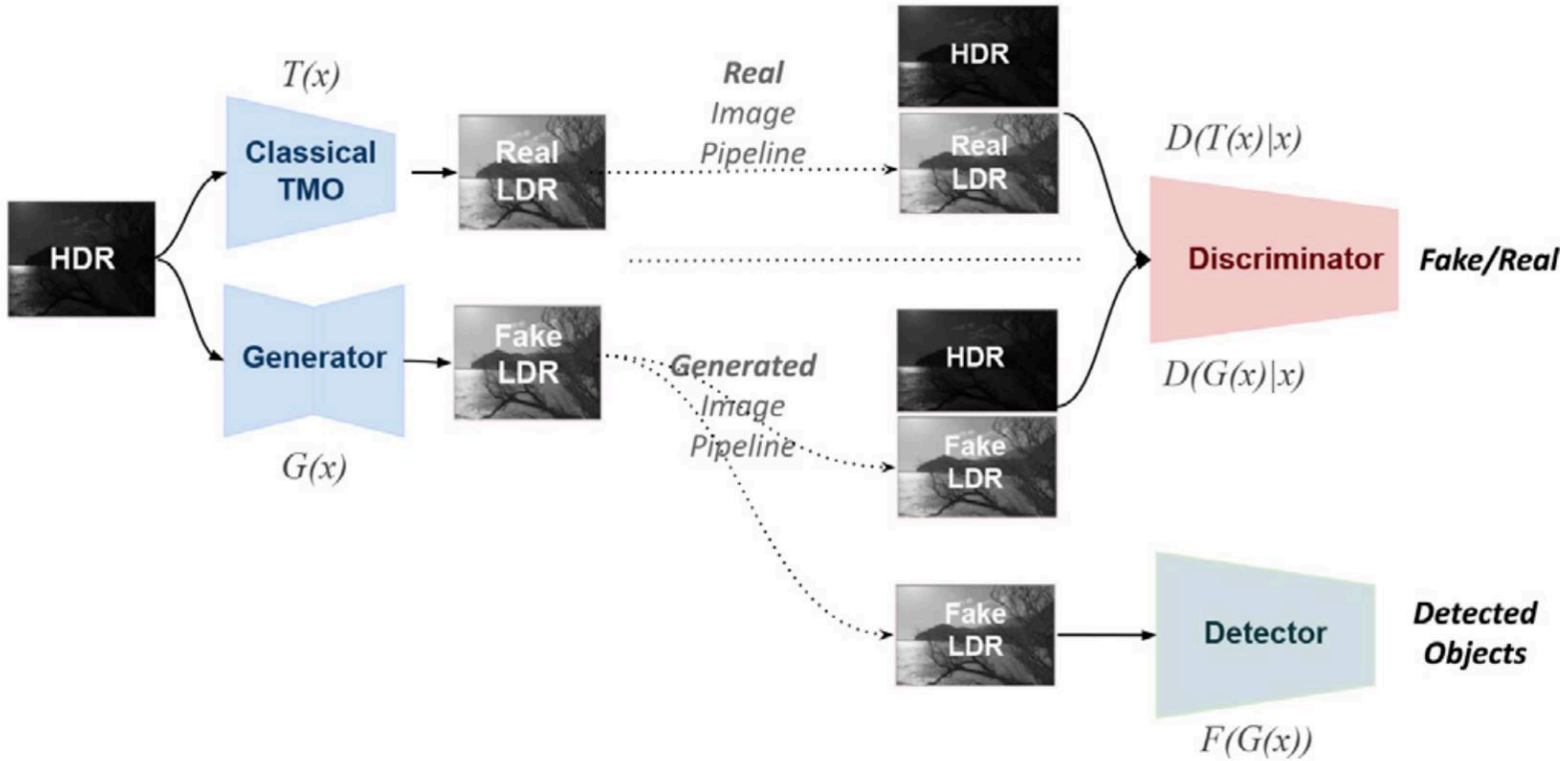
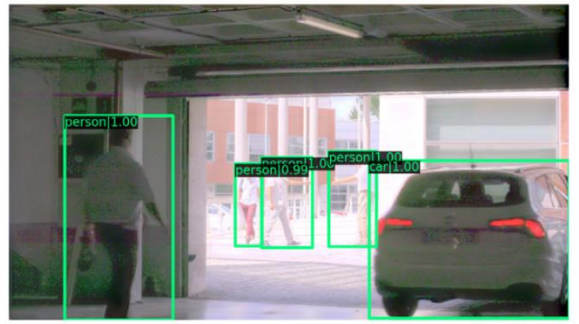
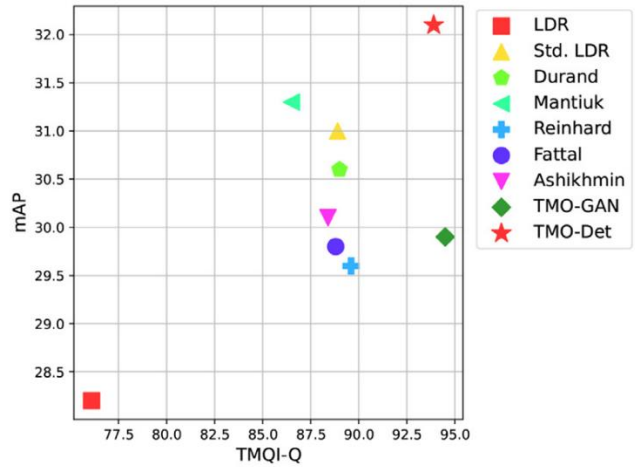


Fig. 2. Overall architecture diagram for the proposed method that combines object detection and tone-mapping objectives.

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



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



(c) Detection vs. HDR quality.

I. H. Kocdemir, A. Koz, A. O. Akyuz, A. Chalmers, A. Alatan, **S. Kalkan**, *"TMO-Det: Deep Tone-mapping Optimized with and for Object Detection"*, Pattern Recognition Letters, 172:230-236, 2023.

Energy-based Generative Models

Share this

8 October 2024

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

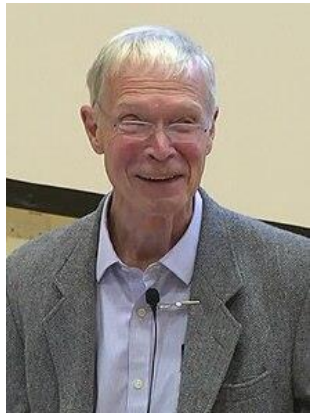
"for foundational discoveries and inventions that enable machine learning with artificial neural networks"

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.

https://www.nobelprize.org/prizes/physics/2024/press-release/

“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.”



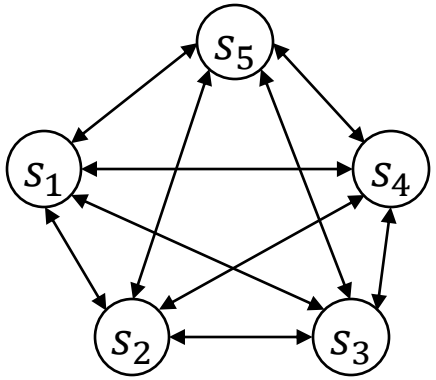
J. Hopfield
(born in 1933)

https://www.nobelprize.org/prizes/physics/2024/press-release/

“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)

https://www.nobelprize.org/prizes/physics/2024/press-release/

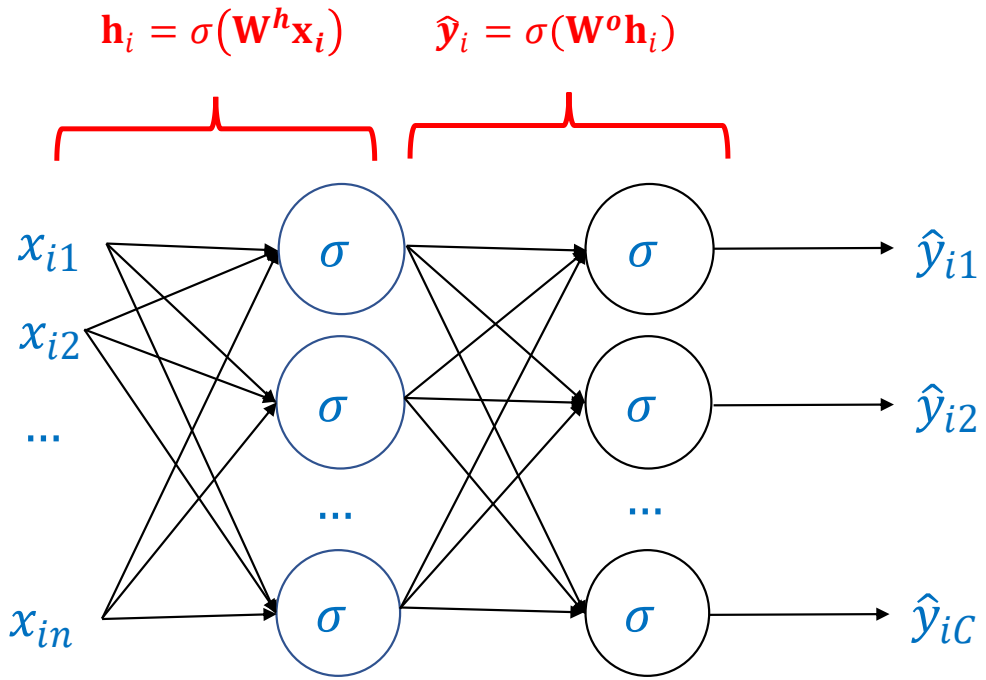


Hopfield Networks

(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:

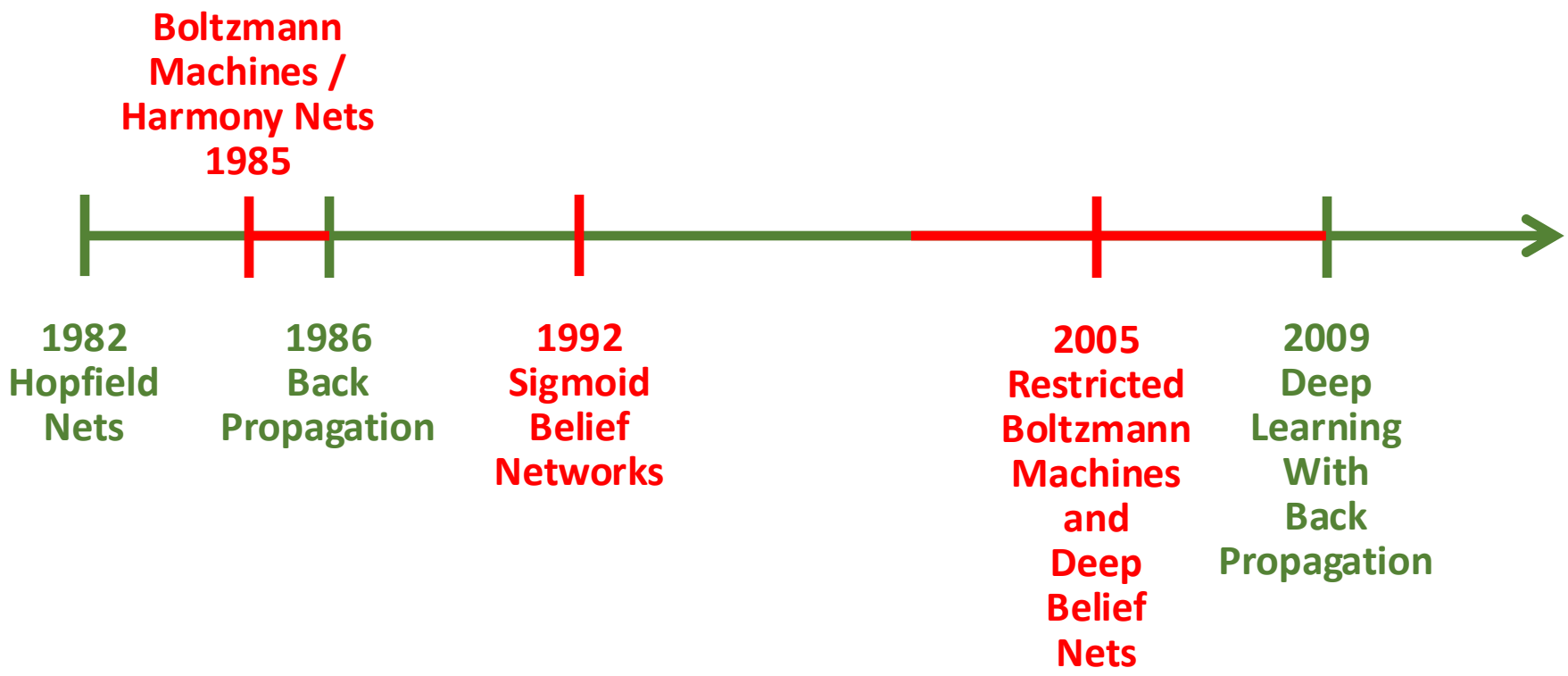
$$h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(net_{ij}^h)$$

Output layer:

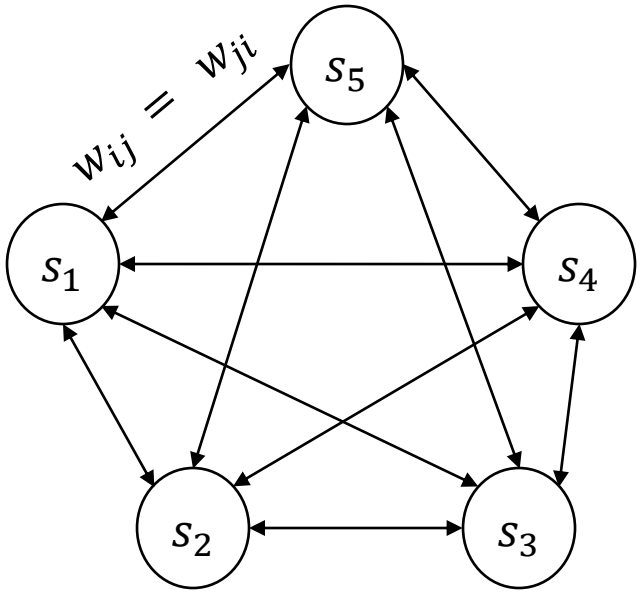
$$\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$$

The loss function:

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


Boltzmann Machines / Harmony Nets 1985
1982 Hopfield Nets
1986 Back Propagation
1992 Sigmoid Belief Networks
2005 Restricted Boltzmann Machines and Deep Belief Nets
2009 Deep Learning With Back Propagation

Hopfield Networks



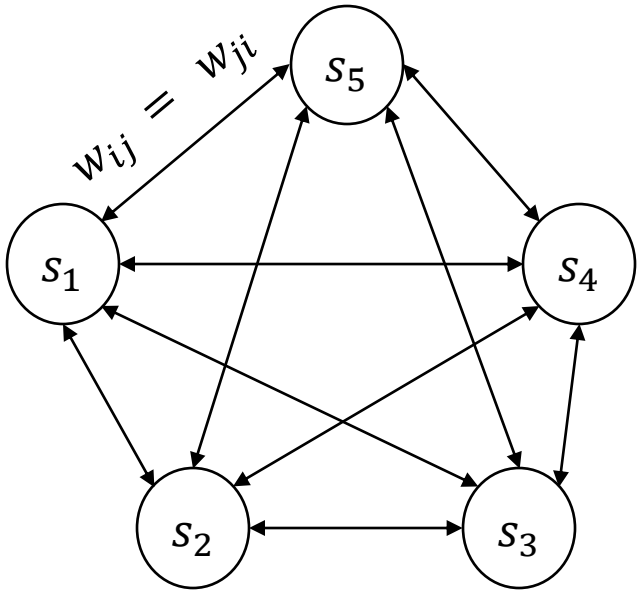
- $s_i = -1 \text{ or } +1$
- Then,

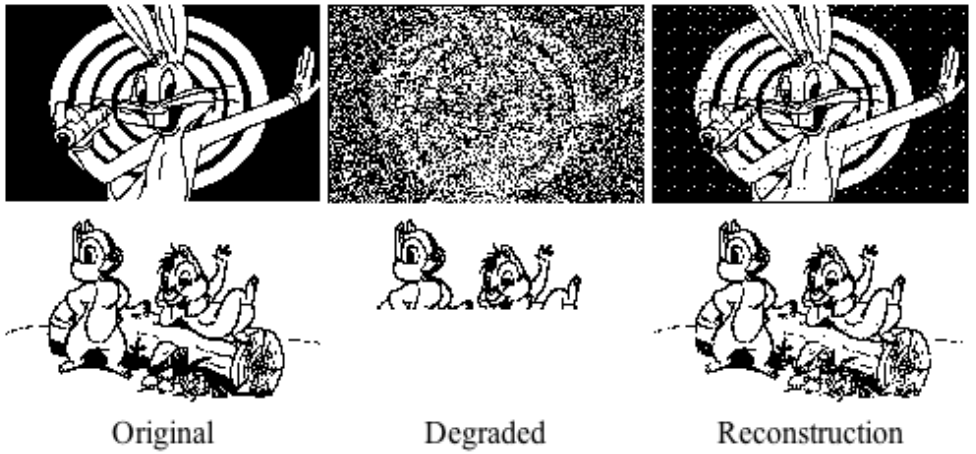
$$s_i \leftarrow \begin{cases} +1, & \sum_j w_{ij} s_j \geq \theta_i \\ -1, & \text{otherwise} \end{cases}$$

- θ_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



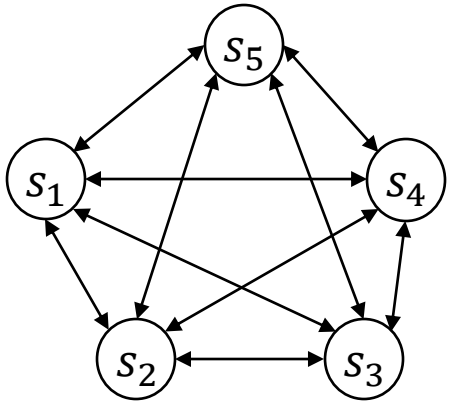


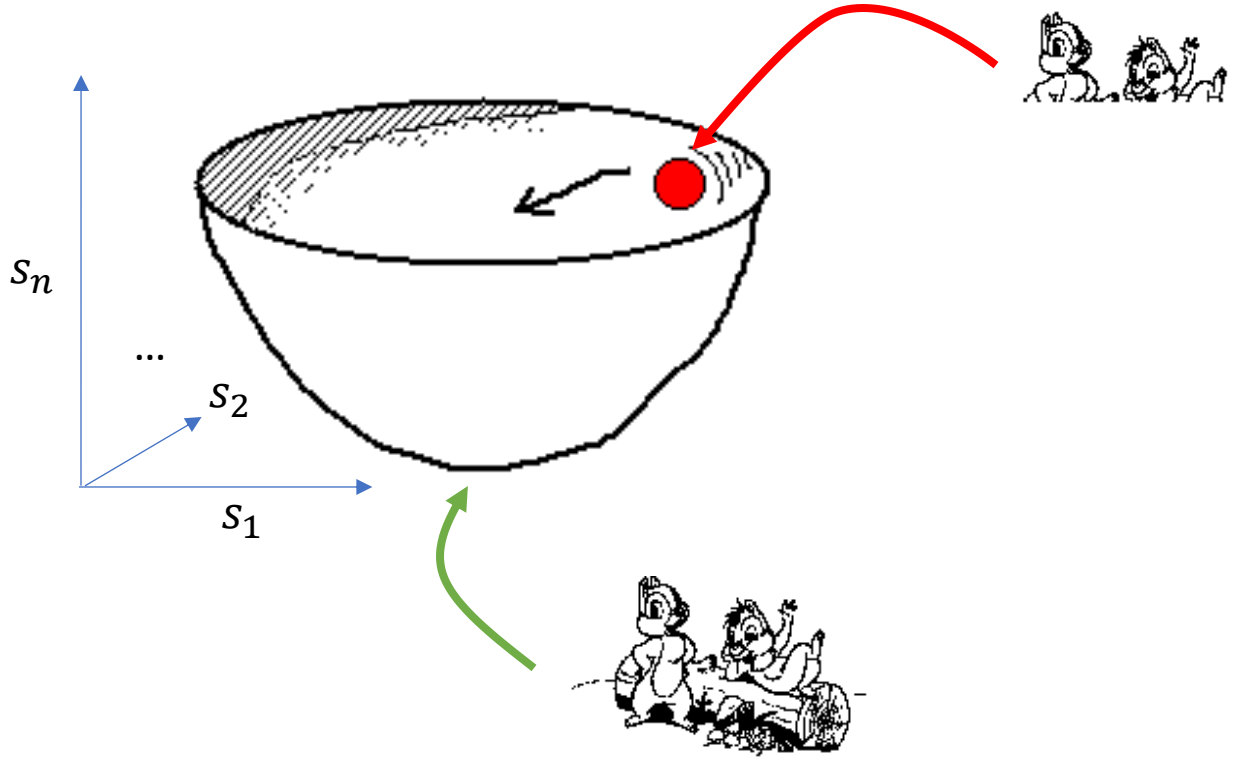
Hopfield Networks

- “Training” on a set of patterns causes them to become attractors

- Degraded input is mapped to nearest attractor

Hopfield Networks as Content-addressable Memory

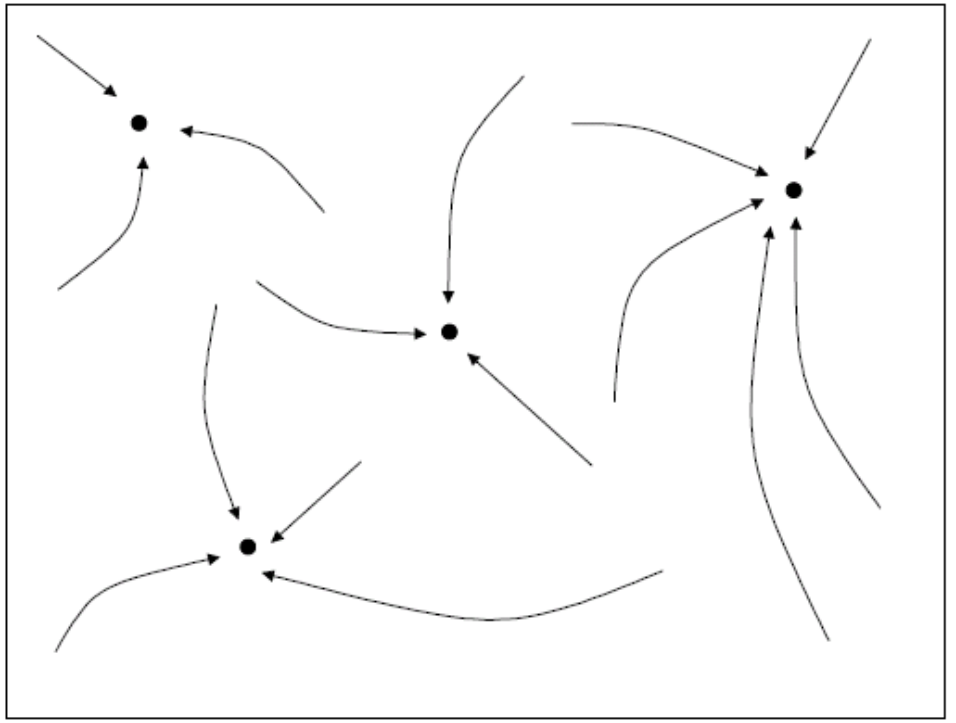




Adapted from: E. Sahin

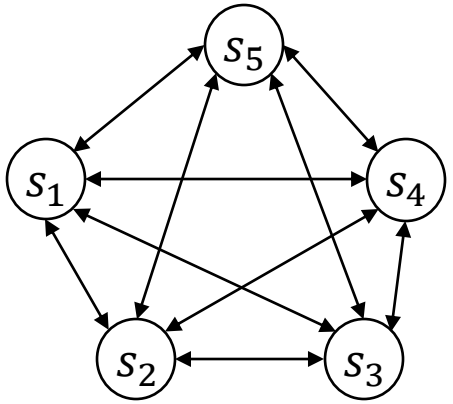
Hopfield Networks as Content-addressable Memory

- 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

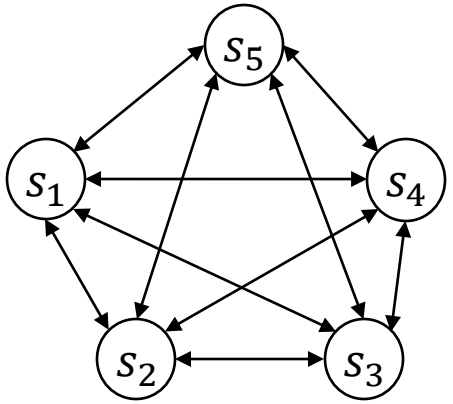


- Three possible schemes:
  - Synchronously: all units updated at each step.
  - Asynchronously I: at each time step select a random unit for update.
  - Asynchronously II: each unit independently chooses to update itself with some probability per unit time.

- Use asynchronously I and keep updating until no neuron changes its state.

Adapted from E. Sahin

Hopfield Networks: Learning to Store a Single Pattern



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

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

- A solution:

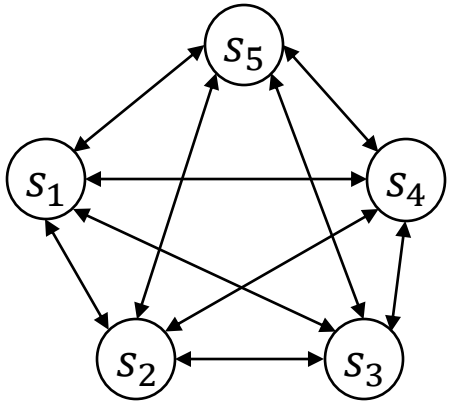
$$w_{ij} = \frac{1}{N}\mathcal{P}_i\mathcal{P}_j$$

since

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

- 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:

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

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

Hopfield Networks: Example

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

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

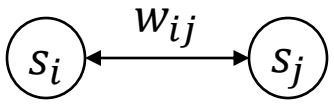
  (e.g., $w_{12} = \frac{1}{4}(-1 \times -1 + 1 \times 1) = 2/4$)
- Inputs and reconstructions (using $s_i = \mathrm{sgn}\left(\left[\sum_j w_{ij} s_j\right] - \theta_i\right)$):
  - $\mathcal{P}^3 = (-1, -1, -1, +1) \Rightarrow$ Recall: $(-1, -1, -1, +1)$
  - $\mathcal{P}^4 = (-1, +1, +1, +1) \Rightarrow$ Recall: $(+1, +1, +1, +1)$

Adapted from E. Sahin

Hopfield Networks: An Energy Perspective

- We can define a scalar for 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.

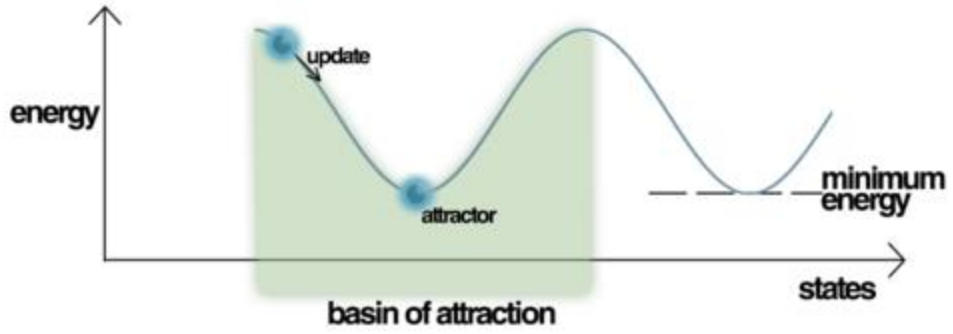


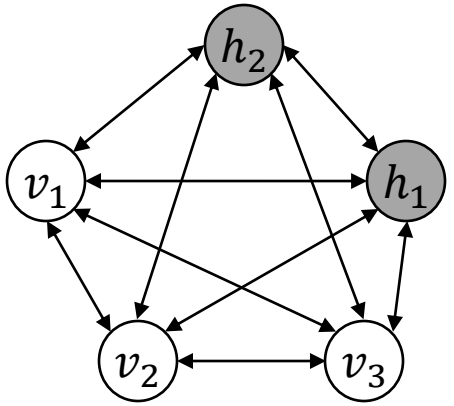
Fig: Wikipedia

Hopfield Networks

- An associative memory
- Inspired many models in Machine Learning

Skipping:

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



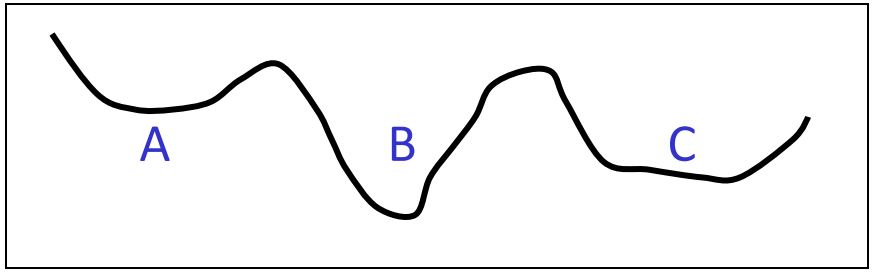
Boltzmann Machines

(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".

Adapted from G. Hinton

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 $\mathbf{s}$,

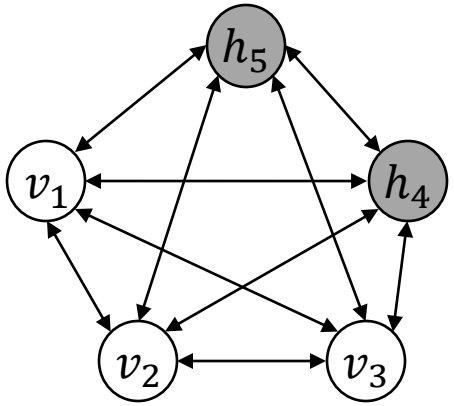
k: Boltzmann's constant, T: temperature.

- 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. Hopefield 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 ➔ Bigger class of distributions that can be modeled ➔ 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 ΔE_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:

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

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

Boltzmann Machines:
Interpretation of a State's Probability

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

a. If T = 0,
   - $p_{i=1} \approx 1$ if ΔE_i is positive (energy reduced).
   - If ΔE_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 ΔE_i is zero, same as case (b).

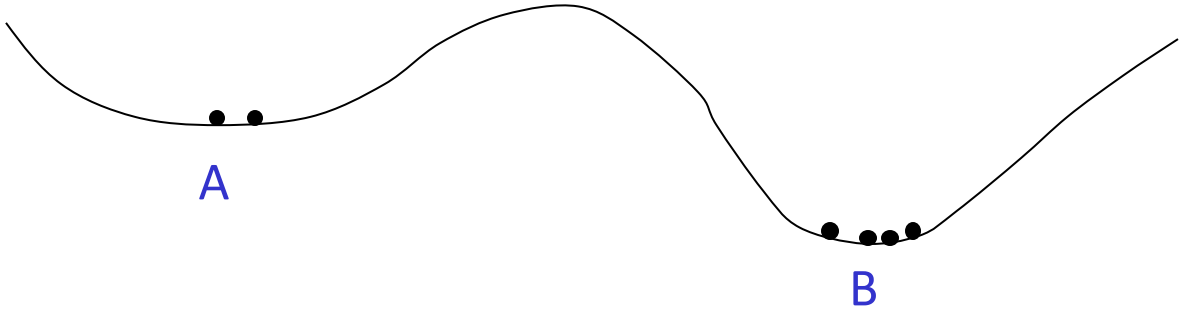
d. For a fixed T, if ΔE_i is very high, same as case (a).

- 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

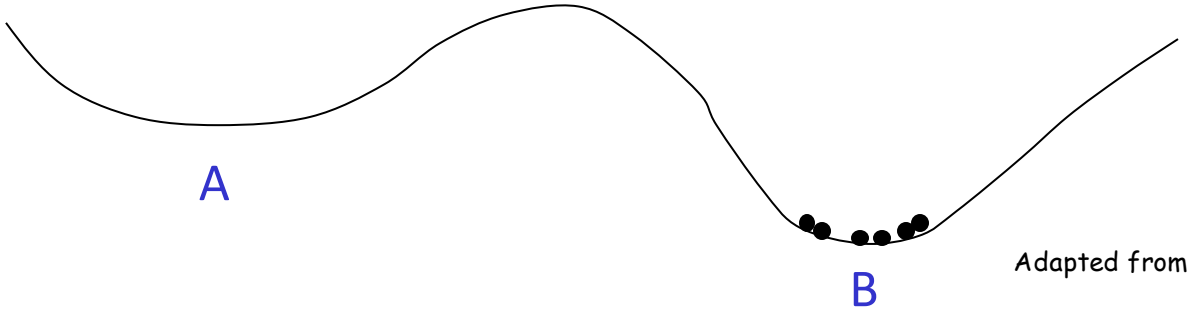
High temperature transition probabilities

$$p(A \rightarrow B) = 0.2$$
$$p(A \leftarrow B) = 0.1$$



Low temperature transition probabilities

$$p(A \rightarrow B) = 0.001$$
$$p(A \leftarrow B) = 0.000001$$

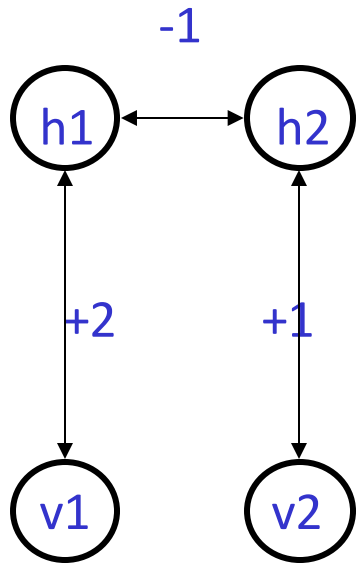


Adapted from G. Hinton

Boltzmann Machines: An Example

v	h	$-E$	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1 1	1 1	2	7.39	.186	0.466
1 1	1 0	2	7.39	.186	
1 1	0 1	1	2.72	.069	
1 1	0 0	0	1	.025	
1 0	1 1	1	2.72	.069	0.305
1 0	1 0	2	7.39	.186	
1 0	0 1	0	1	.025	
1 0	0 0	0	1	.025	
0 1	1 1	0	1	.025	0.144
0 1	1 0	0	1	.025	
0 1	0 1	1	2.72	.069	
0 1	0 0	0	1	.025	
0 0	1 1	-1	0.37	.009	0.084
0 0	1 0	0	1	.025	
0 0	0 1	0	1	.025	
0 0	0 0	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:

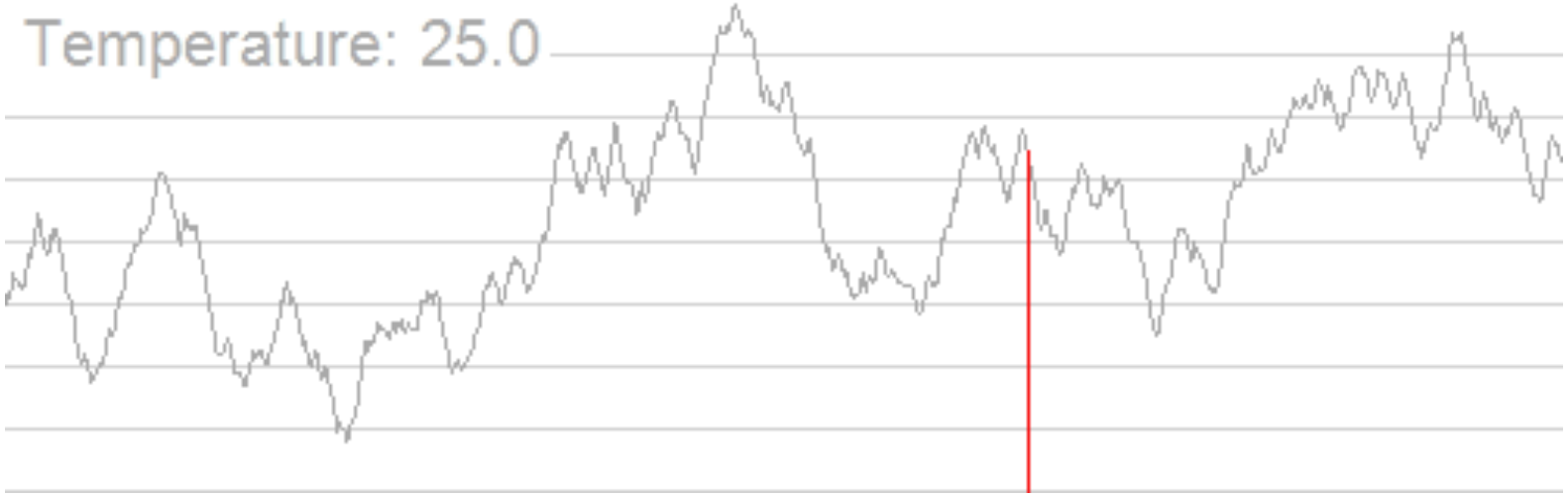
$$p_{i=on} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)}$$

- 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

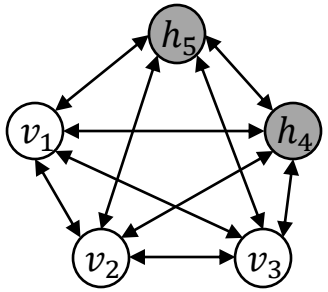
- How do we understand we have reached it?
  - The average activation of neurons don't change over time.
  - i.e., the probability of being in a state does not change.
- The initial state is not important!
- At low temperature:
  - There is a strong bias for states with low energy
  - But this makes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

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 $P^-(V)$:

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

  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:

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

  - 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)

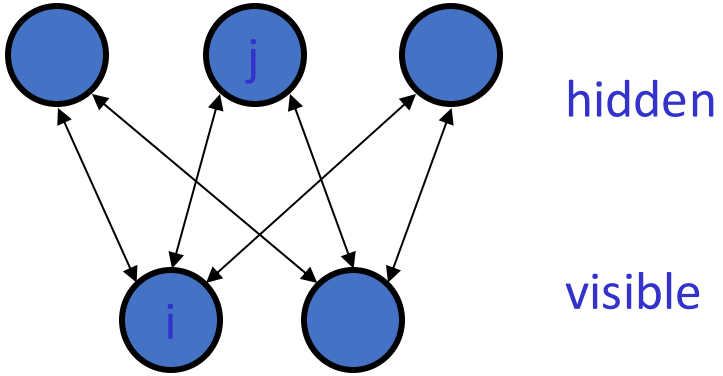
Why Boltzmann Machines Failed

- 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)

Slide: Michael Mozer

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



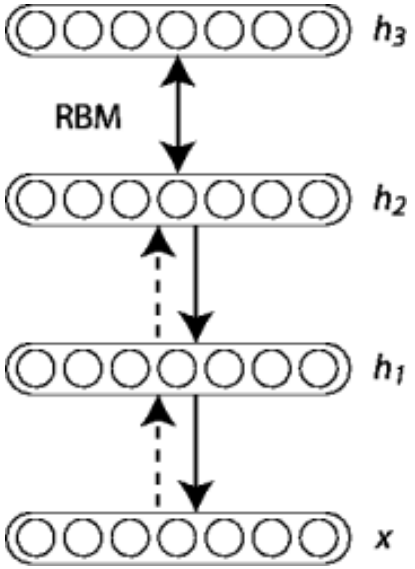
Adapted from G. Hinton

Deep Belief Networks

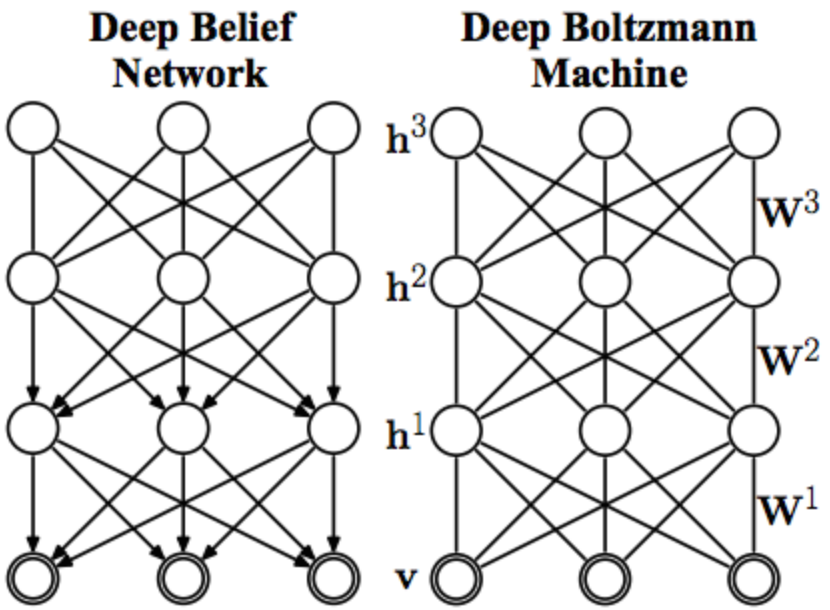
- A stacked RBM
- First used by Hinton & Salakhutdinov (2006)
- Models the distribution:

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

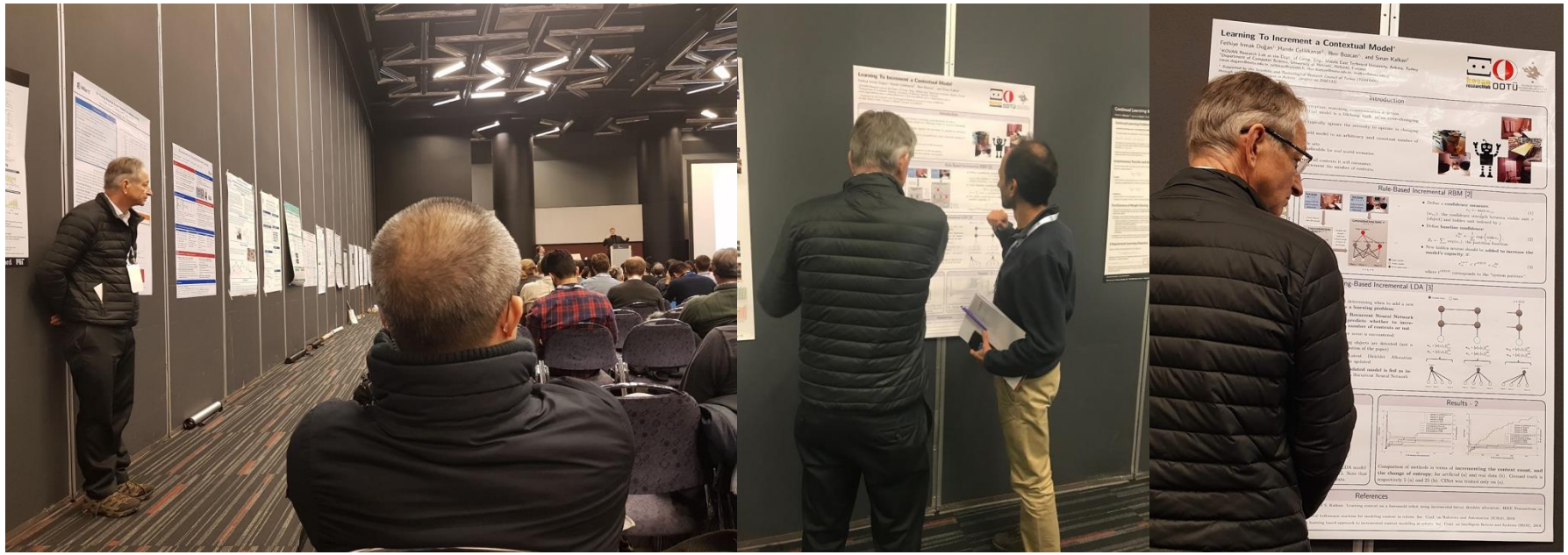
- Training is similar to autoencoders



From: http://deeplearning.net/tutorial/DBN.html

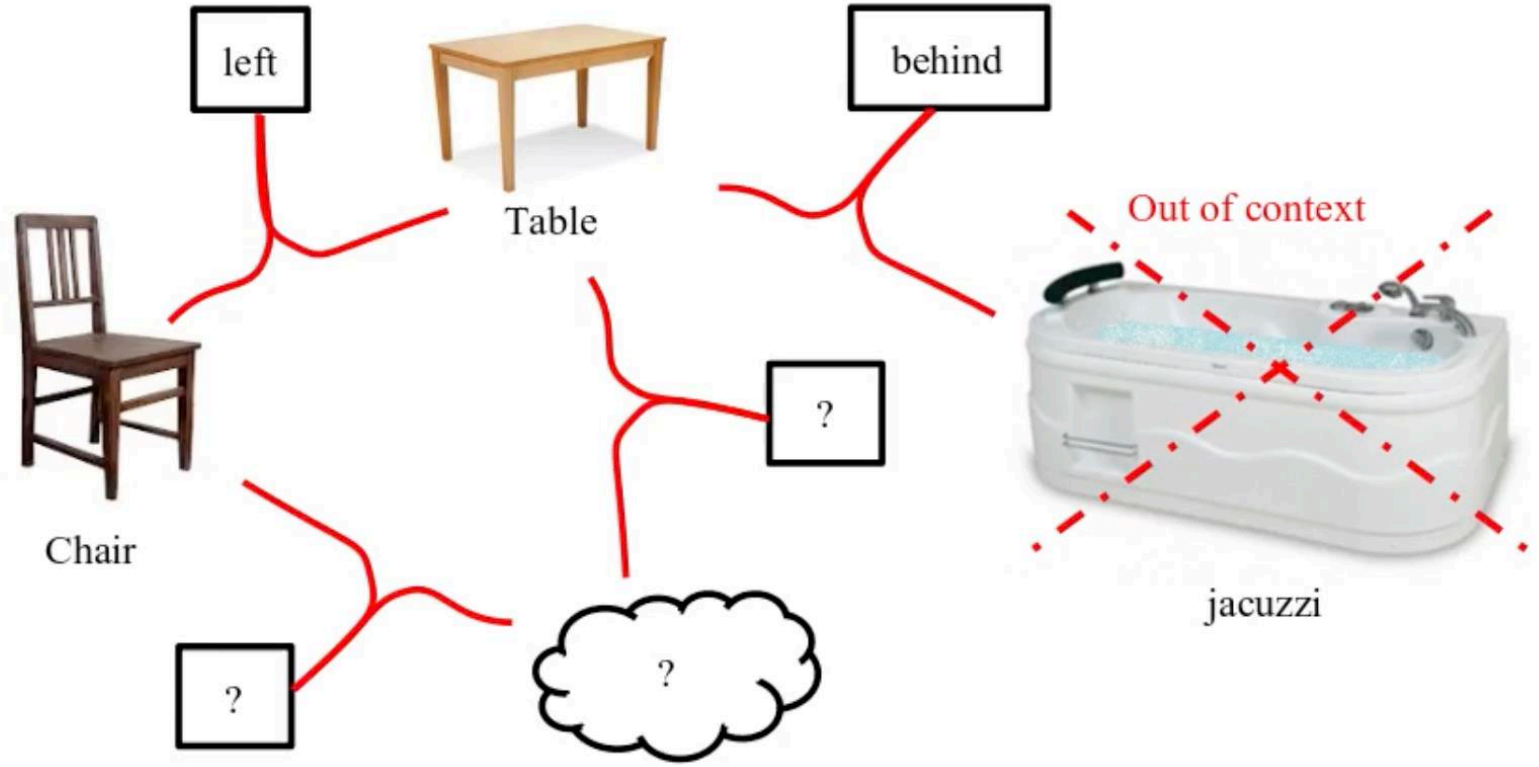


Wang, H., & Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.



Our Work Using Boltzmann Machines

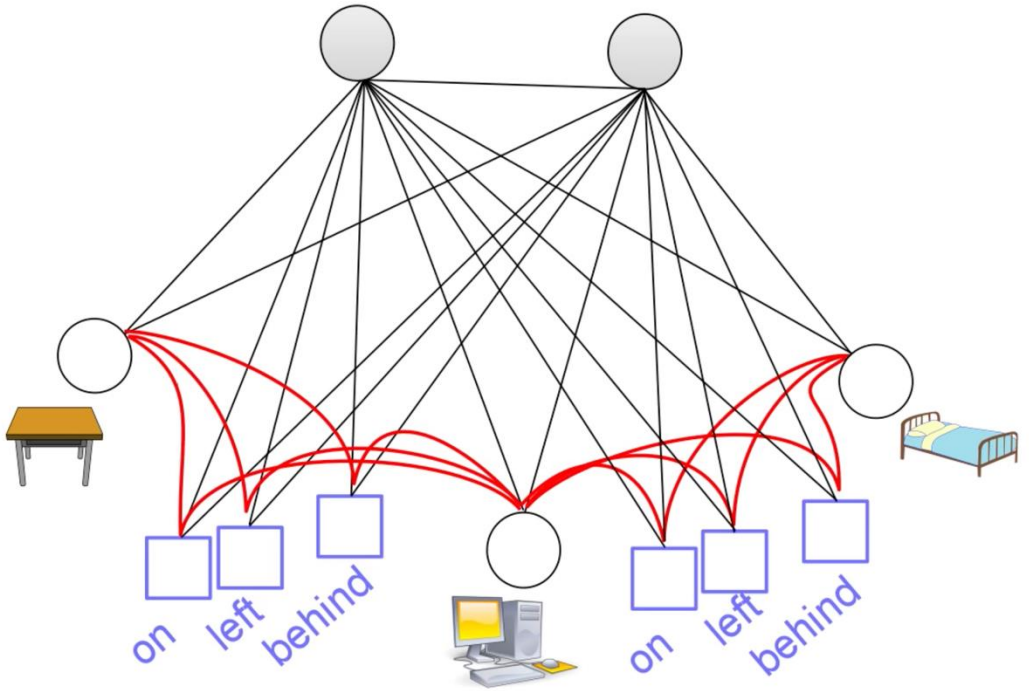
Boltzmann Machines: Our Work



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

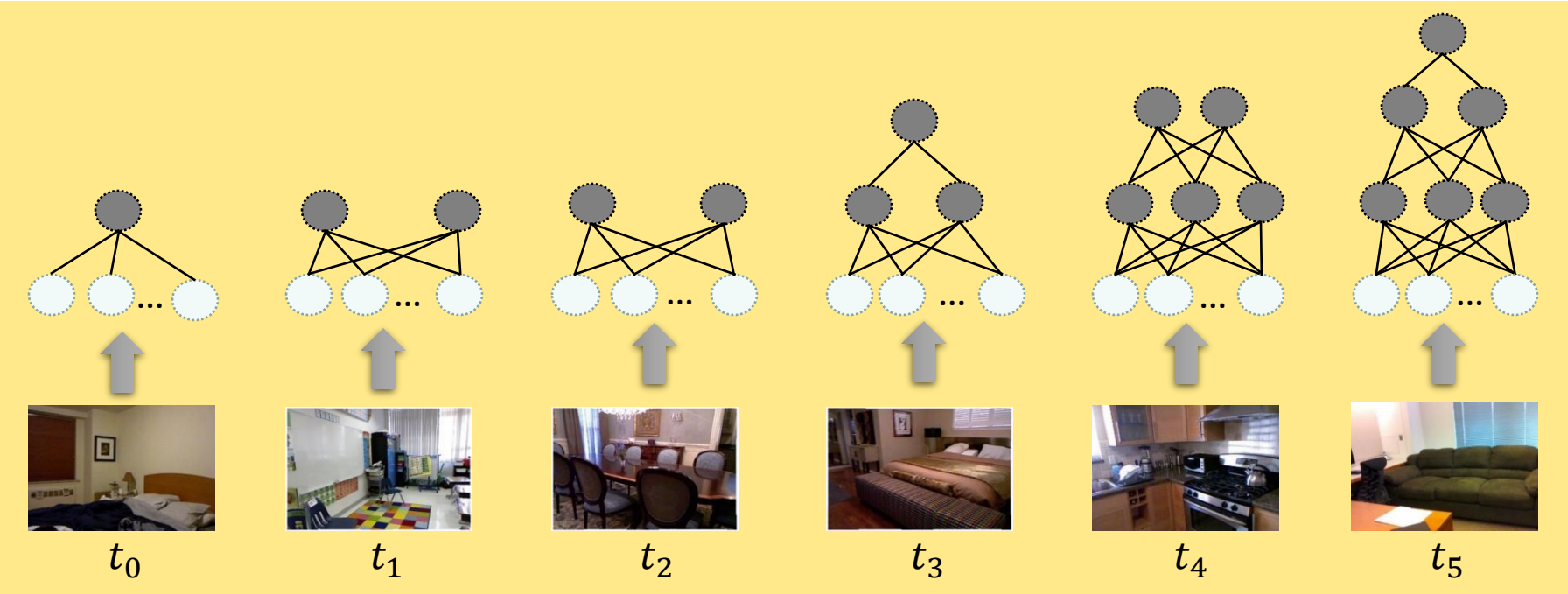


Hidden Units Visible Units Relation Units (visible)

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.

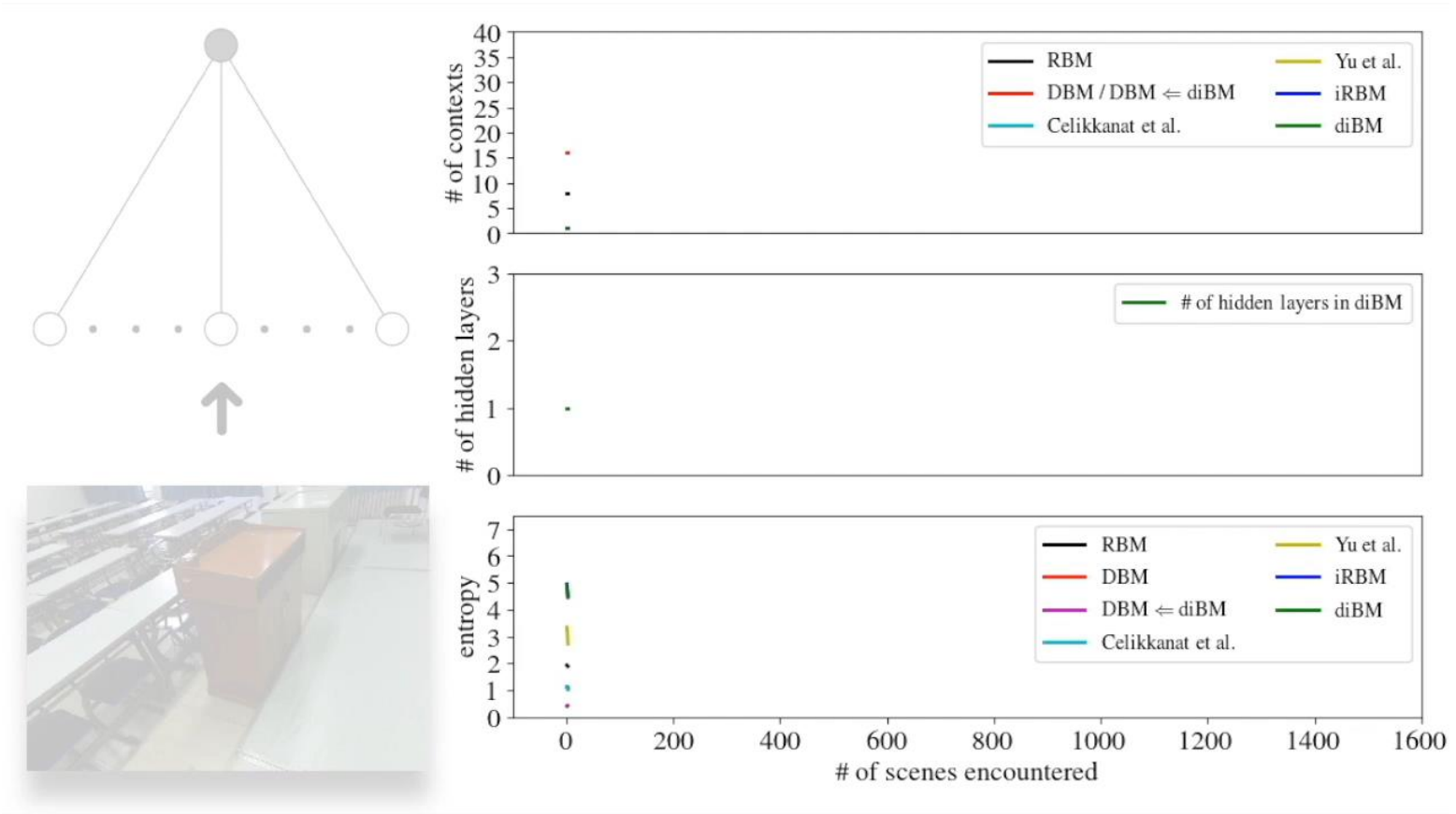
Boltzmann Machines: Our Work



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.