CENG501 – Deep Learning

Week 12

Spring 2026

Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on CENG501

What is a Vision-Language Model?

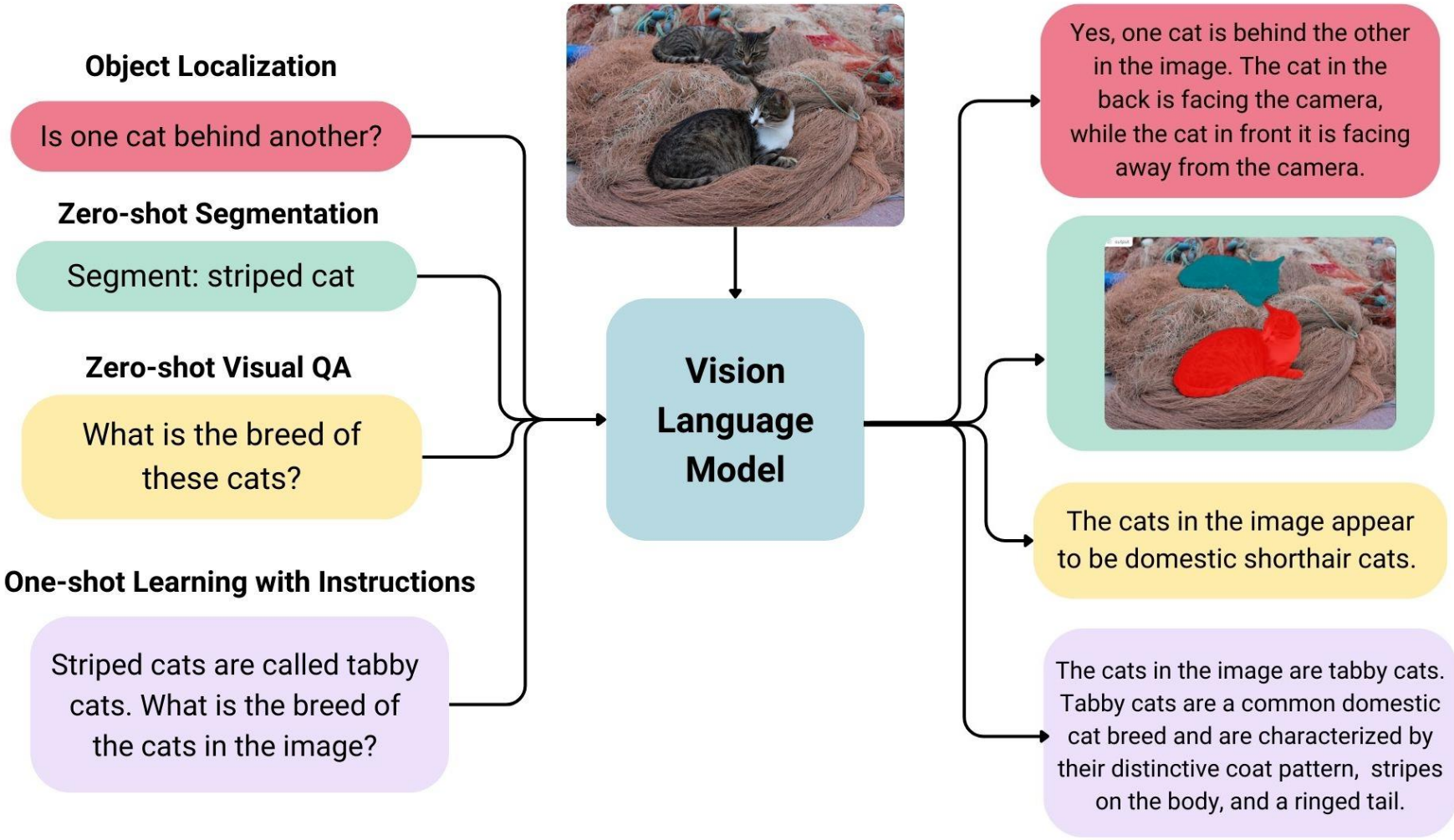


Fig: https://huggingface.co/blog/vlms

Previously on CENG501

Earlier Attempts:

VisualBERT: A Simple and Performant Baseline for Vision and Language

Liunian Harold Li[†], **Mark Yatskar**[*], **Da Yin**[°], **Cho-Jui Hsieh**[†] **& Kai-Wei Chang**[†]
[†]University of California, Los Angeles
[*]Allen Institute for Artificial Intelligence
[°]Peking University

2019

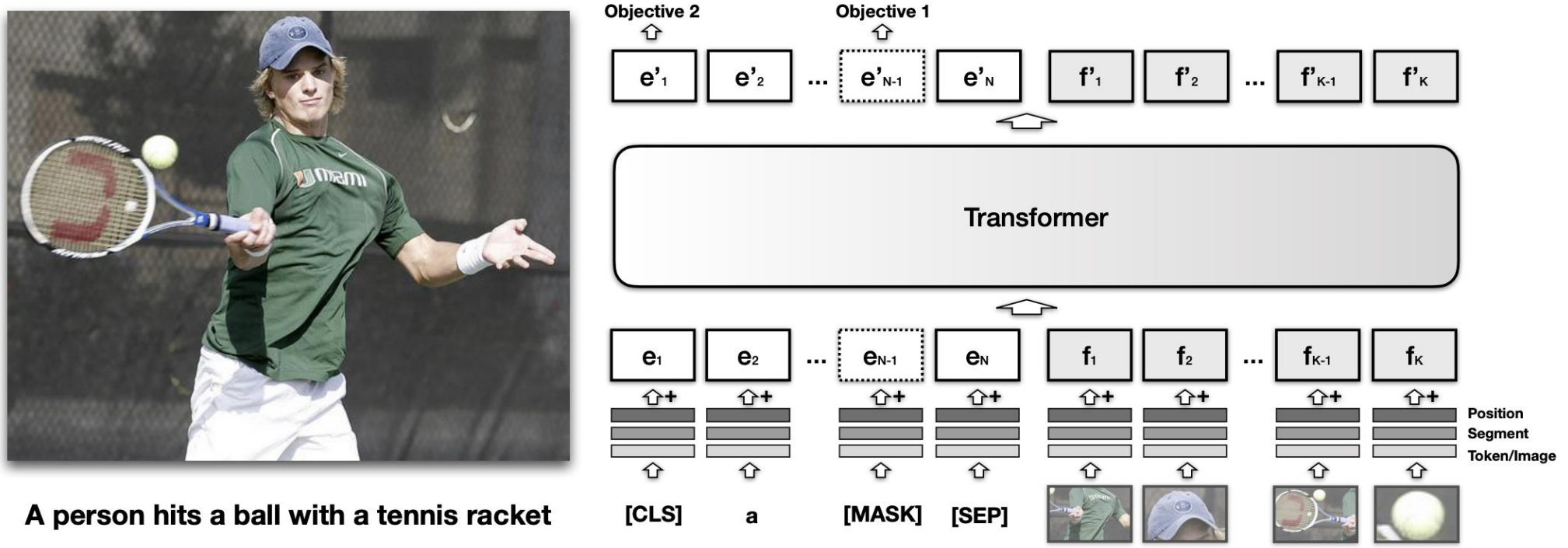
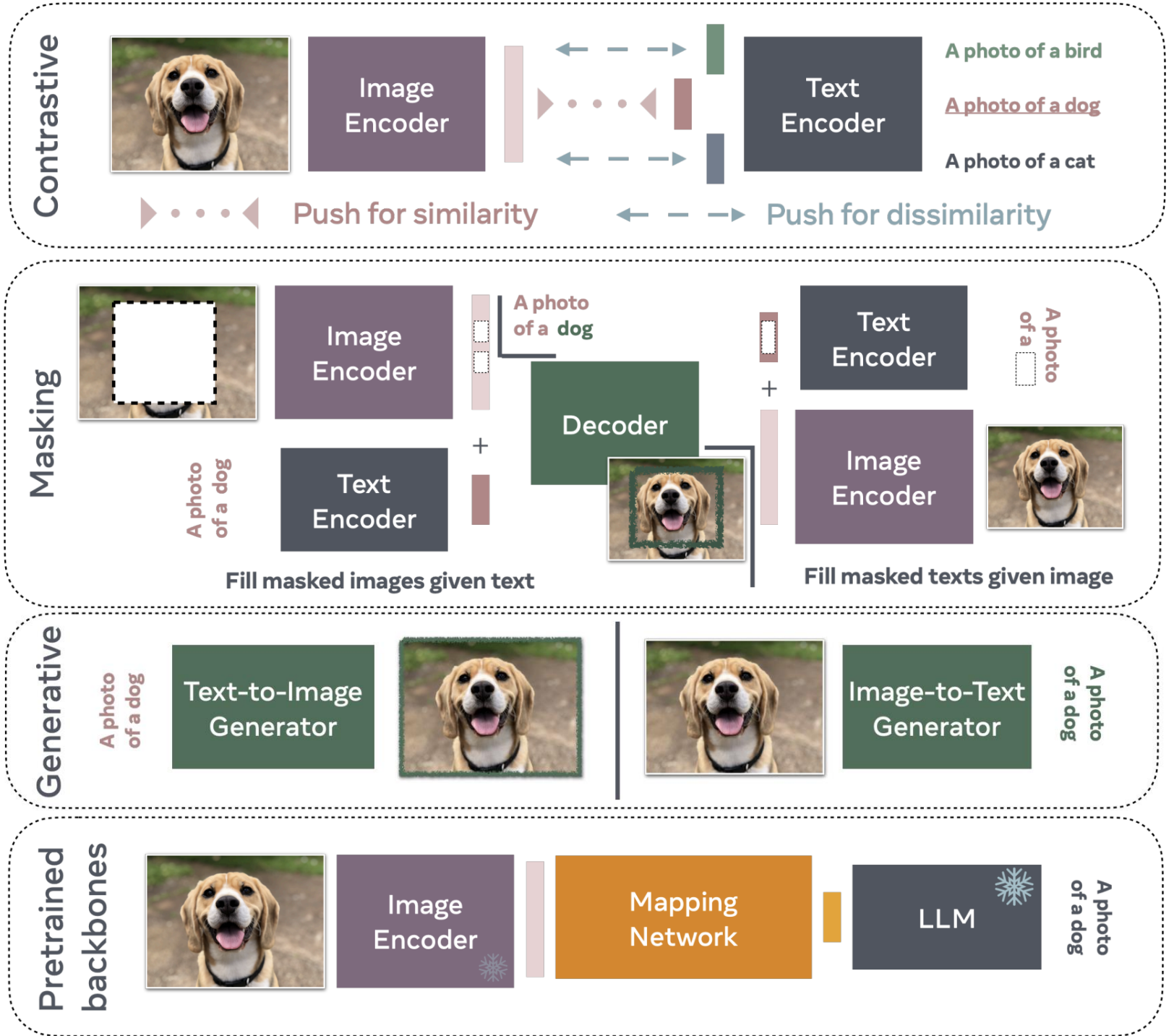


Figure 2: The architecture of VisualBERT. Image regions and language are combined with a Transformer to allow the self-attention to discover implicit alignments between language and vision. It is pre-trained with a masked language modeling (Objective 1), and sentence-image prediction task (Objective 2), on caption data and then fine-tuned for different tasks. See §3.3 for more details.

Previously on ENG501

"Modern" VLMs:

An Overview of the Approaches



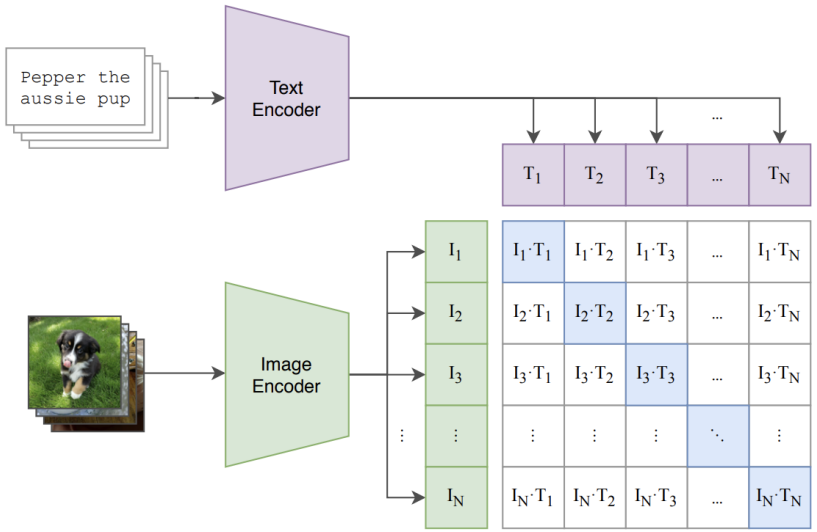
Bordes et al., "An Introduction to Vision-Language Modeling", 2024. https://arxiv.org/pdf/2405.17247

Learning Transferable Visual Models From Natural Language Supervision

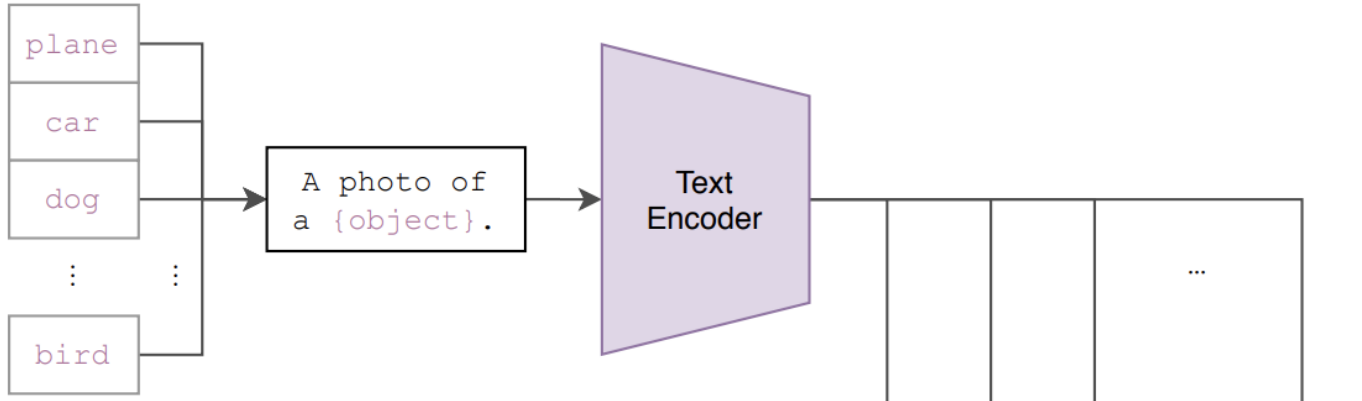
Alec Radford [*1] Jong Wook Kim [*1] Chris Hallacy [1] Aditya Ramesh [1] Gabriel Goh [1] Sandhini Agarwal [1]
Girish Sastry [1] Amanda Askell [1] Pamela Mishkin [1] Jack Clark [1] Gretchen Krueger [1] Ilya Sutskever [1]

2021

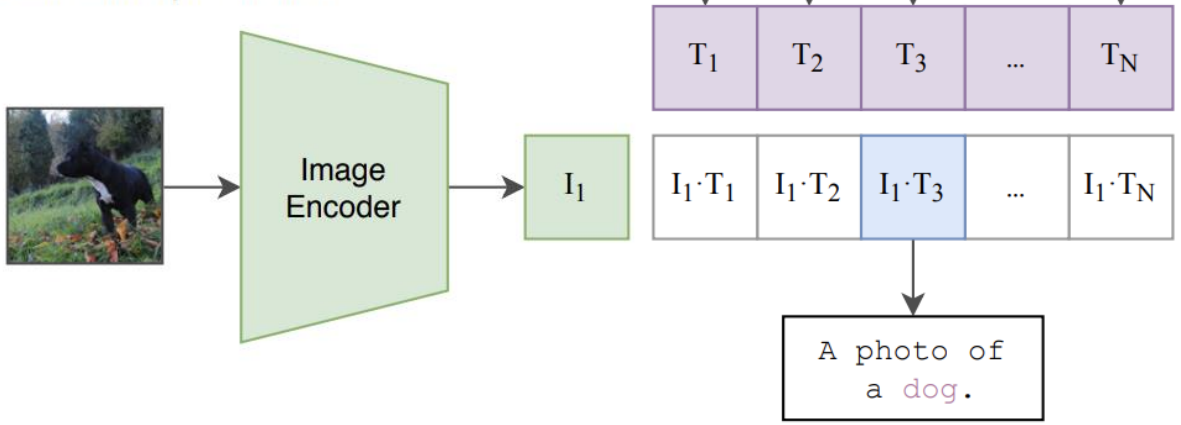
(1) Contrastive pre-training



(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



Previously on CENG501

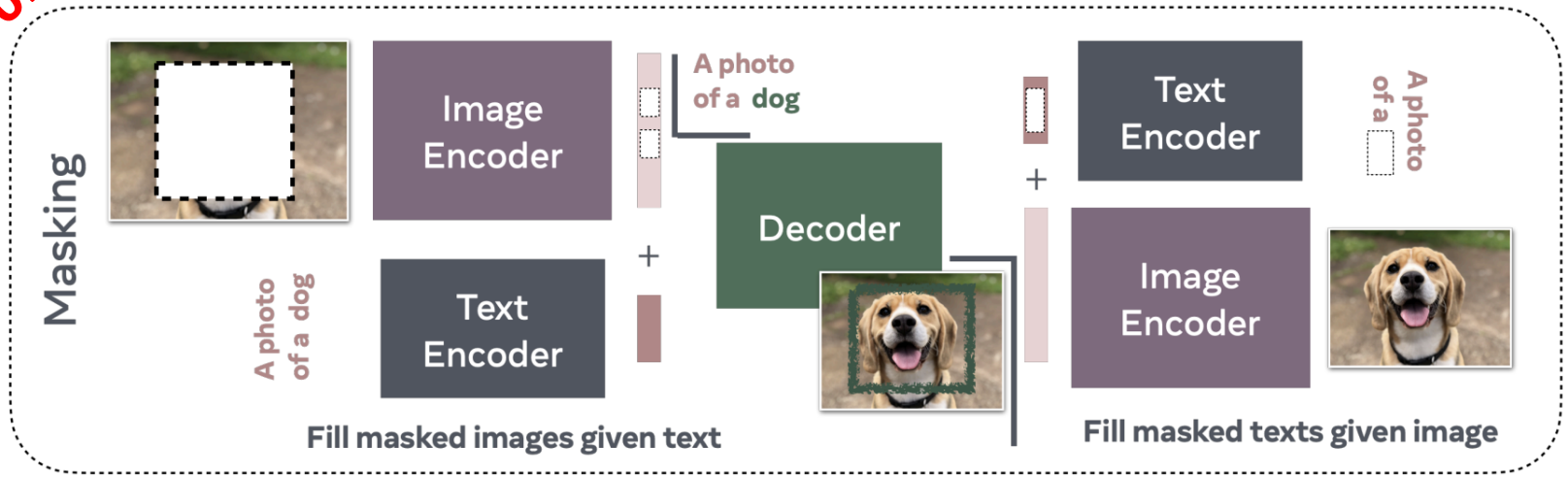


Fig: Bordes et al., “An Introduction to Vision-Language Modeling”, 2024.

Masking Approaches

Previously on ENG501

FLAVA

GC: Global Contrastive Loss (same as CLIP)

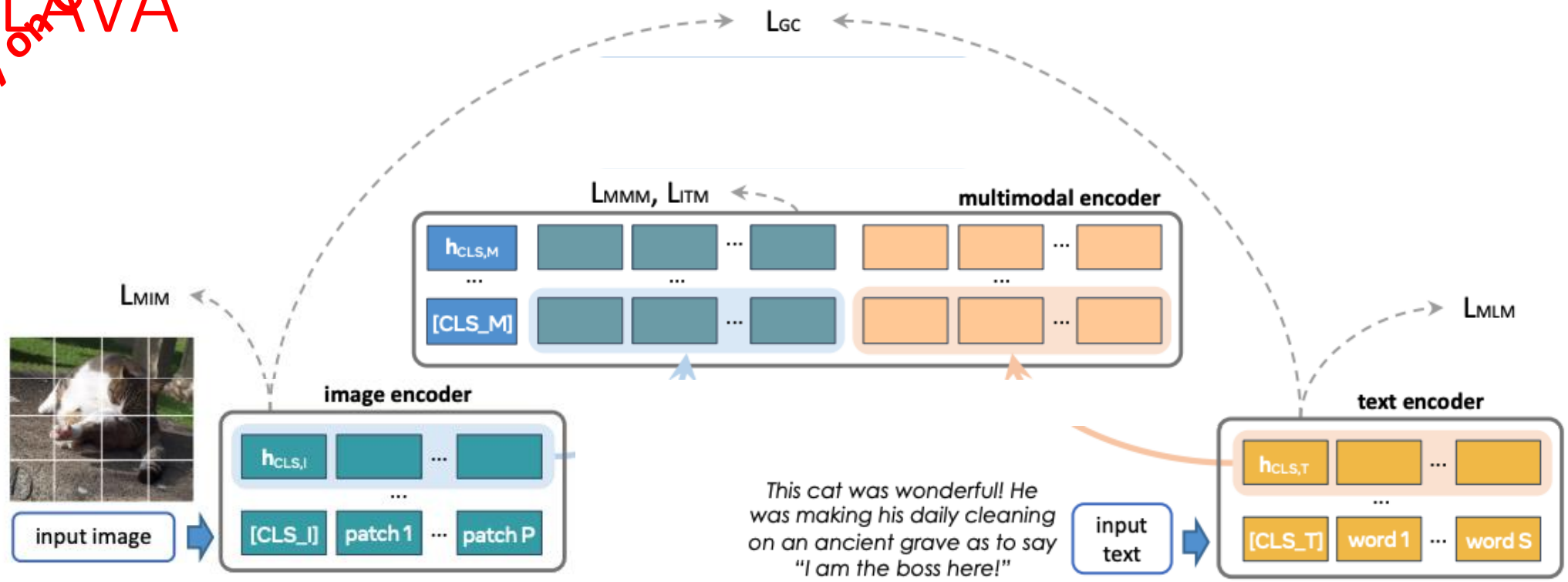
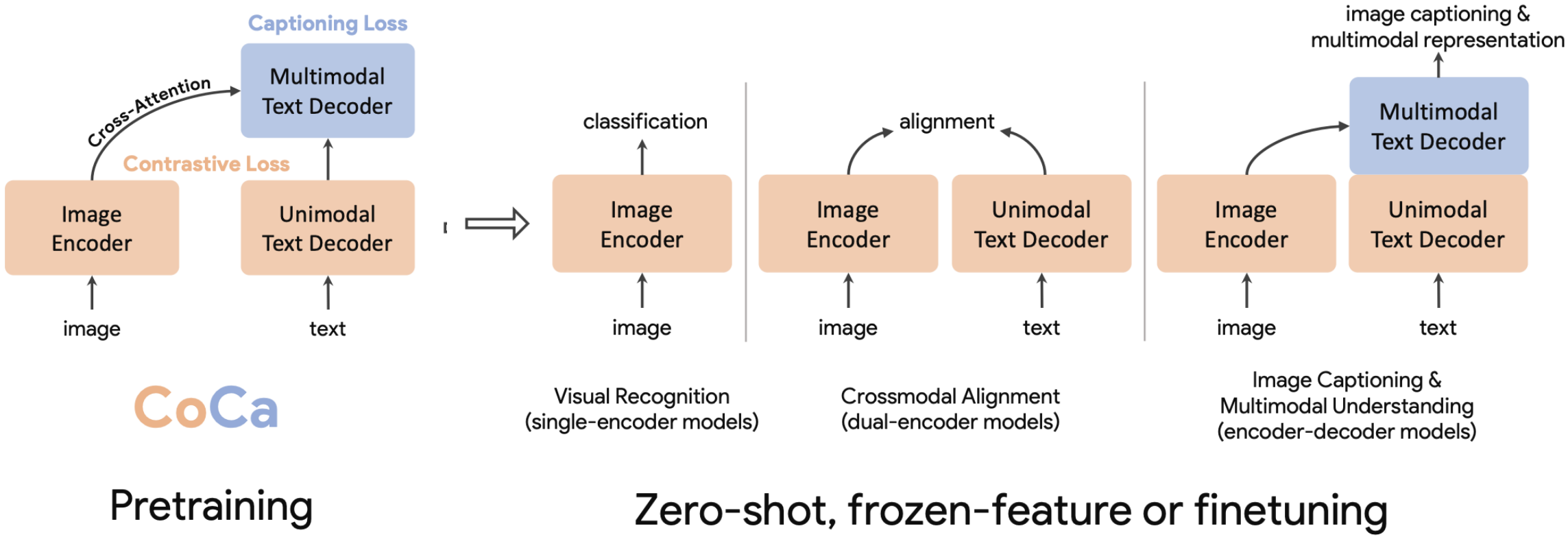


Figure 2. **An overview of our FLAVA model**, with an image encoder transformer to capture unimodal image representations, a text encoder transformer to process unimodal text information, and a multimodal encoder transformer that takes as input the encoded unimodal image and text and integrates their representations for multimodal reasoning. **During pretraining**, masked image modeling (MIM) and mask language modeling (MLM) losses are applied onto the image and text encoders over a single image or a text piece, respectively, while contrastive, masked multimodal modeling (MMM), and image-text matching (ITM) loss are used over paired image-text data. **For downstream tasks**, classification heads are applied on the outputs from the image, text, and multimodal encoders respectively for visual recognition, language understanding, and multimodal reasoning tasks.

CoCa

Contrastive Captioner (CoCa),
Yu et al., 2022.



Previously on CANG501

Chameleon

Chameleon: Mixed-Modal Early-Fusion Foundation Models, Meta, 2024.

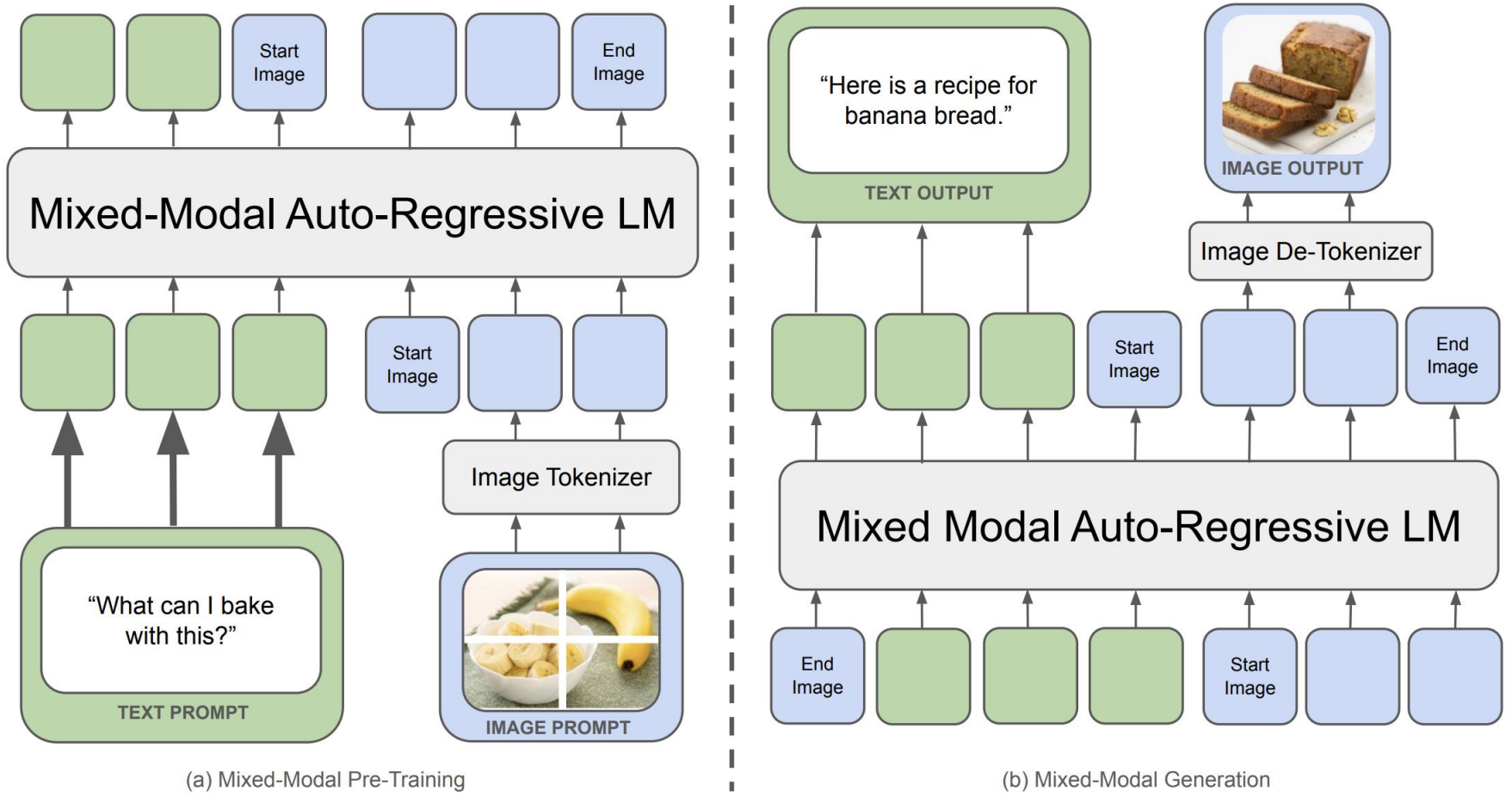


Figure 1 **Chameleon** represents all modalities — images, text, and code, as discrete tokens and uses a uniform transformer-based architecture that is trained from scratch in an end-to-end fashion on ∼10T tokens of interleaved mixed-modal data. As a result, **Chameleon** can both reason over, as well as generate, arbitrary mixed-modal documents. Text tokens are represented in green and image tokens are represented in blue.

Previously on CENG501

Fig: Bordes et al., “An Introduction to Vision-Language Modeling”, 2024.

Approaches Using Pre-trained Backbones

Previously on CENG501

Frozen

Multimodal Few-Shot Learning with Frozen Language Models, 2021

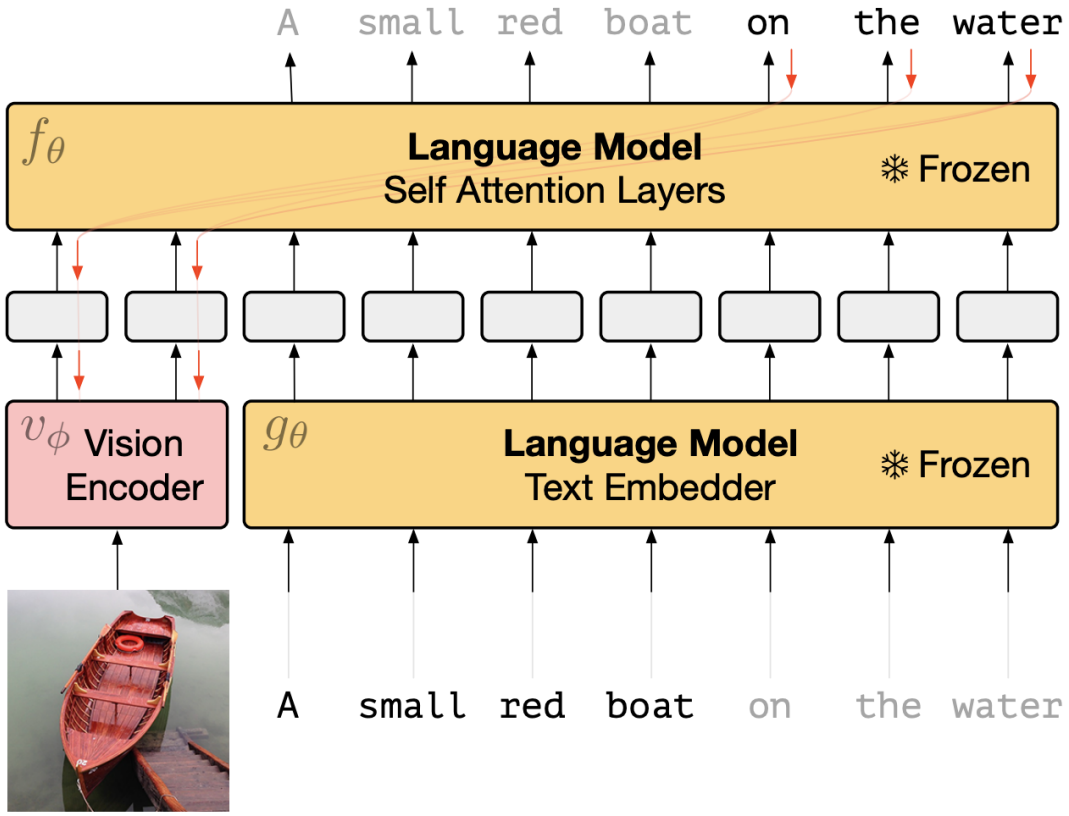


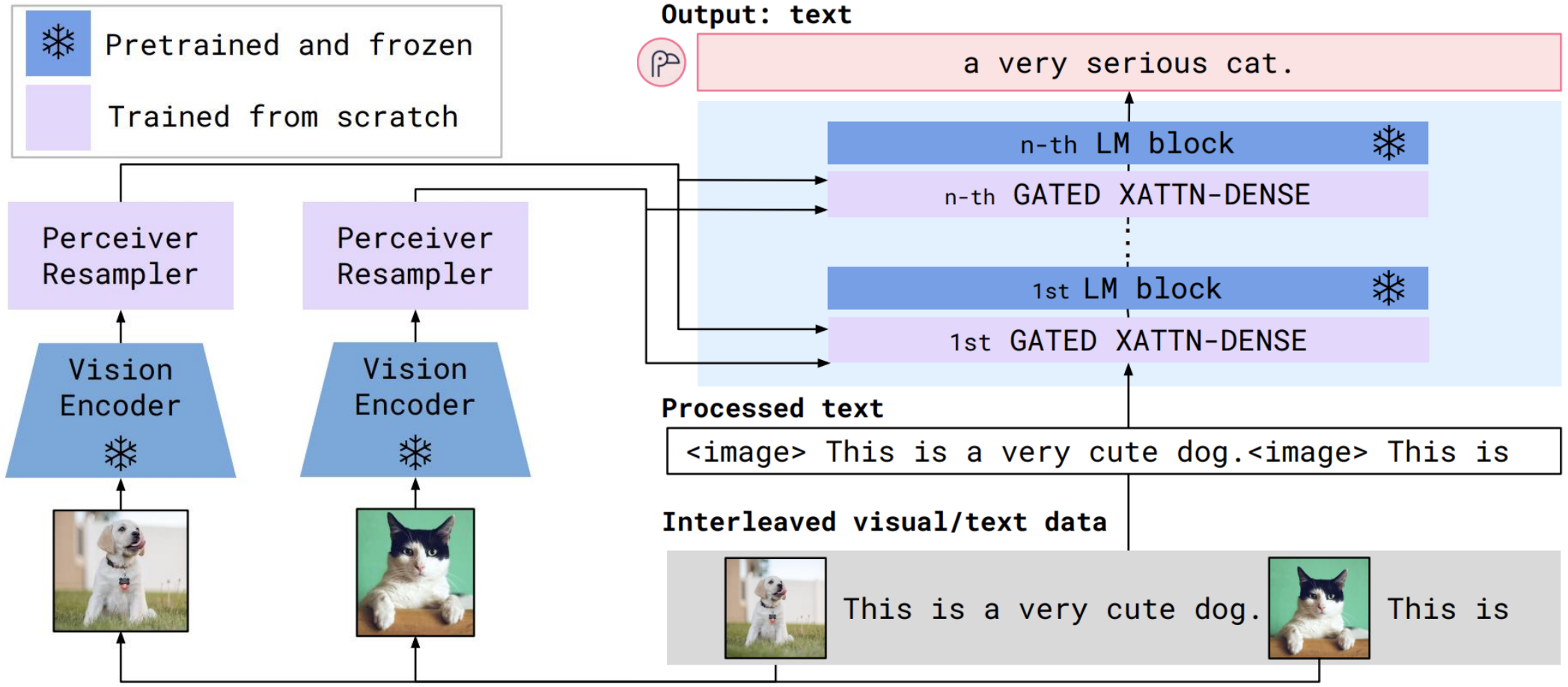
Figure 2: Gradients through a frozen language model's self attention layers are used to train the vision encoder.

Previously on ECENG501

Flamingo

Flamingo: a Visual Language Model for Few-Shot Learning, Deepmind, 2022

Vision Encoder: Normalizer-Free ResNet (NFNet)
Perceiver Sampler: Fixed # of queries attend to variable length of visual tokens. (input images can have different resolutions)
LLM: Chinchilla



Previously on CENG501

BLIP

BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation, Salesforce, 2022.

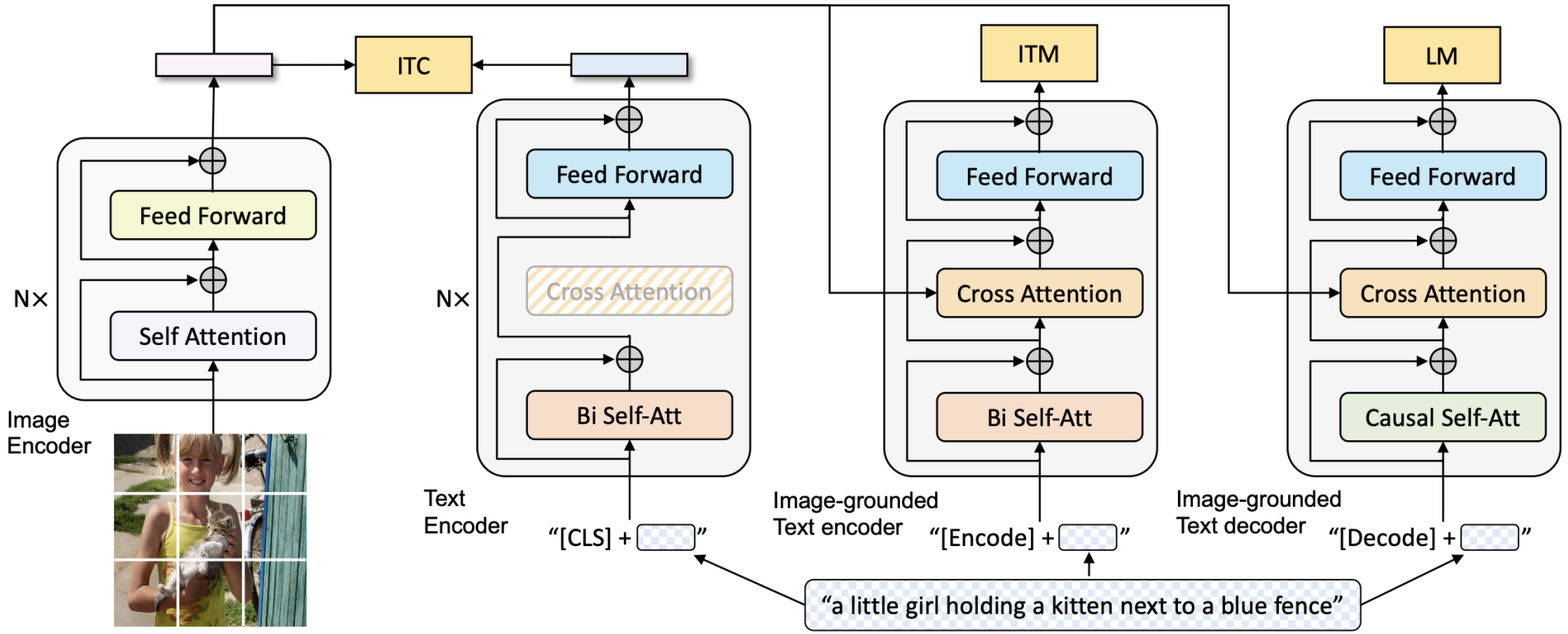


Figure 2. Pre-training model architecture and objectives of BLIP (same parameters have the same color). We propose multimodal mixture of encoder-decoder, a unified vision-language model which can operate in one of the three functionalities: (1) Unimodal encoder is trained with an image-text contrastive (ITC) loss to align the vision and language representations. (2) Image-grounded text encoder uses additional cross-attention layers to model vision-language interactions, and is trained with a image-text matching (ITM) loss to distinguish between positive and negative image-text pairs. (3) Image-grounded text decoder replaces the bi-directional self-attention layers with causal self-attention layers, and shares the same cross-attention layers and feed forward networks as the encoder. The decoder is trained with a language modeling (LM) loss to generate captions given images.

BLIP-2

BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models, Salesforce, 2023.

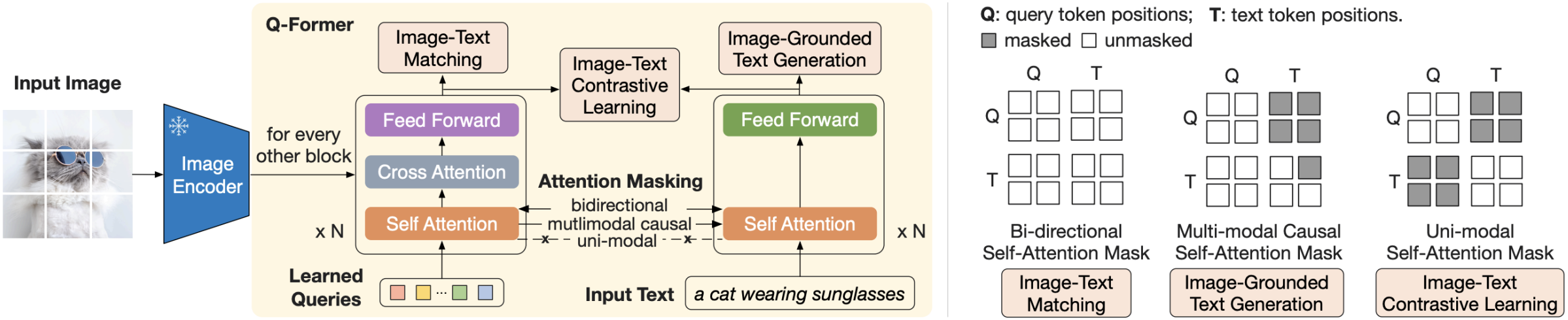


Figure 2. (**Left**) Model architecture of Q-Former and BLIP-2's first-stage vision-language representation learning objectives. We jointly optimize three objectives which enforce the queries (a set of learnable embeddings) to extract visual representation most relevant to the text. (**Right**) The self-attention masking strategy for each objective to control query-text interaction.

Previously on ENG501

BLIP-2

BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models, Salesforce, 2023.

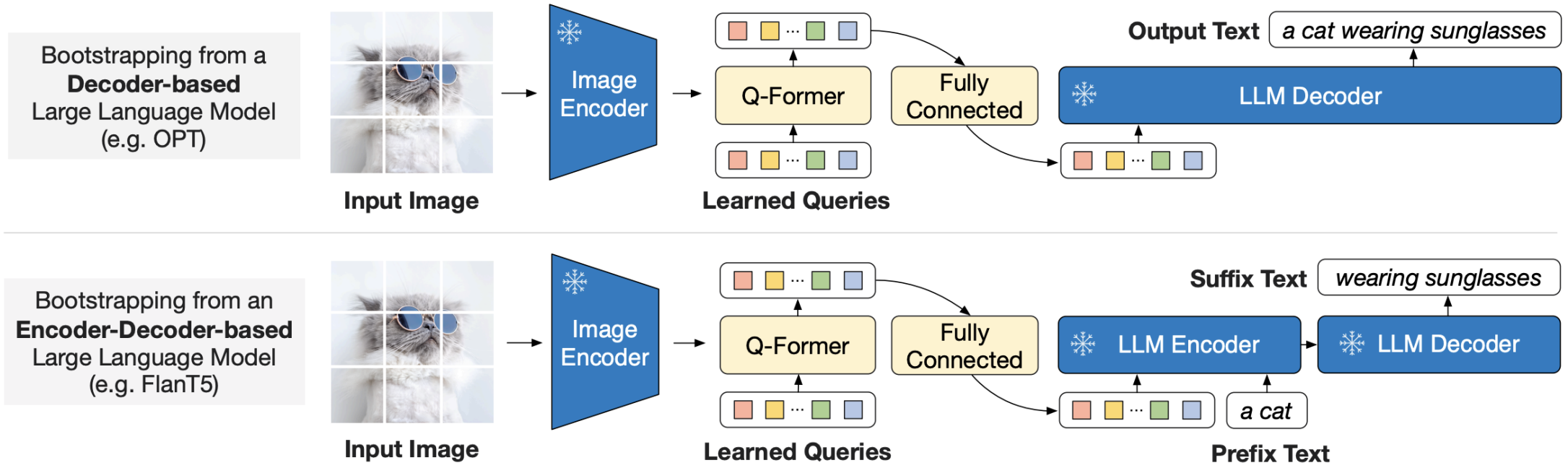


Figure 3. BLIP-2's second-stage vision-to-language generative pre-training, which bootstraps from frozen large language models (LLMs). (**Top**) Bootstrapping a decoder-based LLM (e.g. OPT). (**Bottom**) Bootstrapping an encoder-decoder-based LLM (e.g. FlanT5). The fully-connected layer adapts from the output dimension of the Q-Former to the input dimension of the chosen LLM.

Today

- Generative 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

Fırından Sıcak Sıcak

Taxonomy of Generative Models

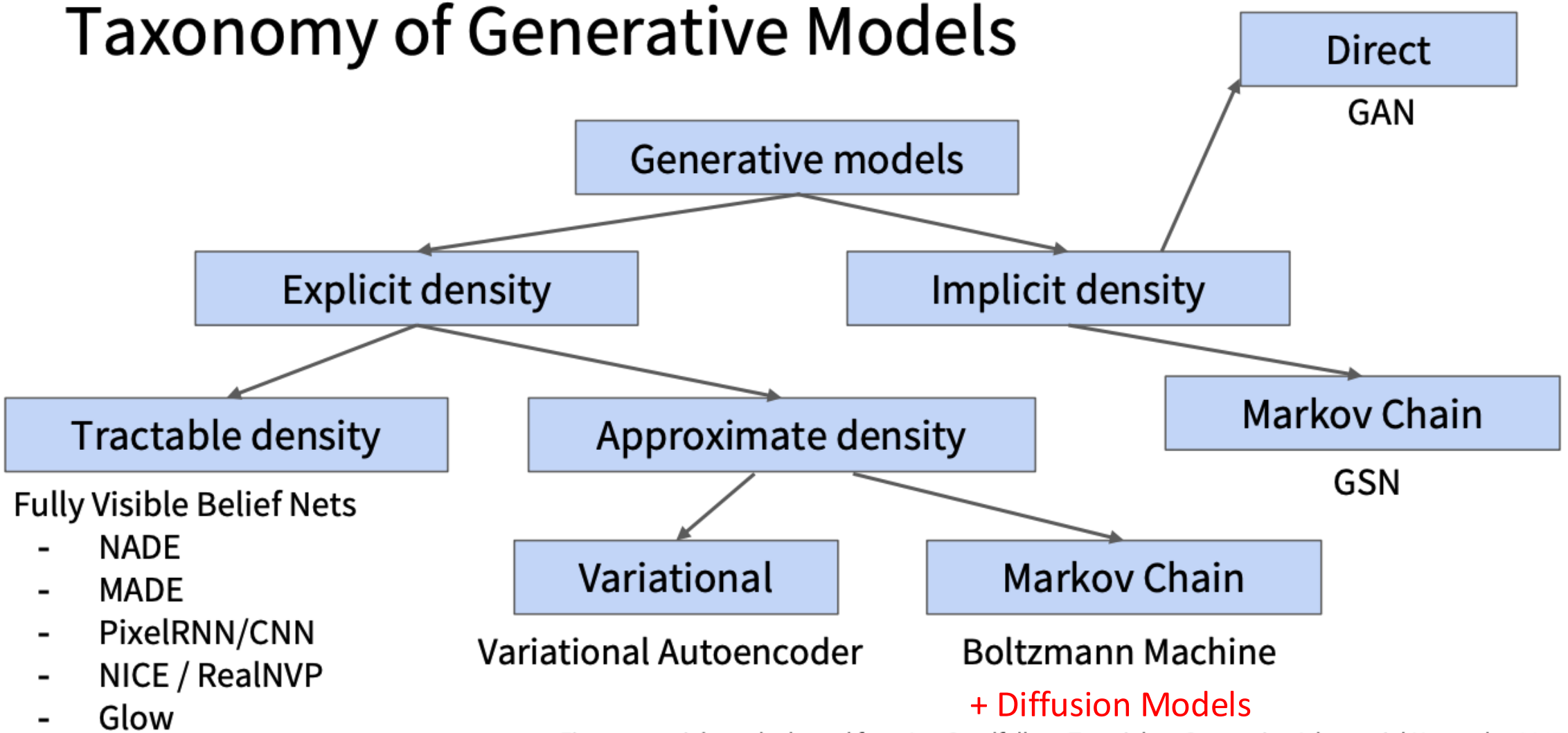
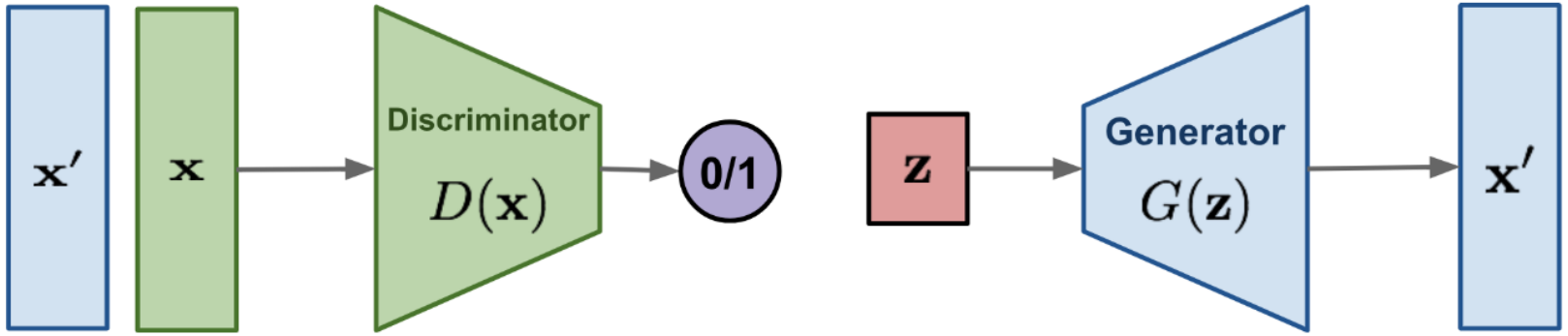


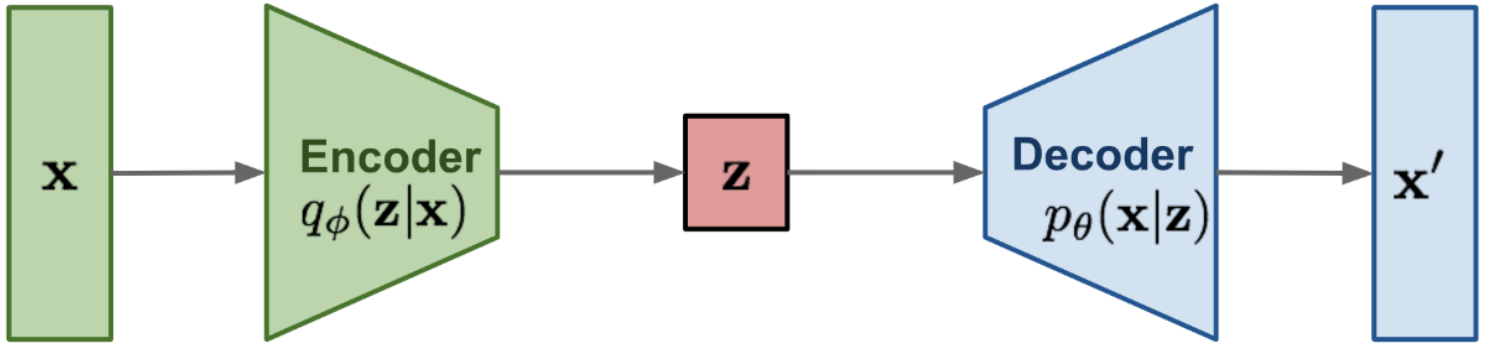
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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

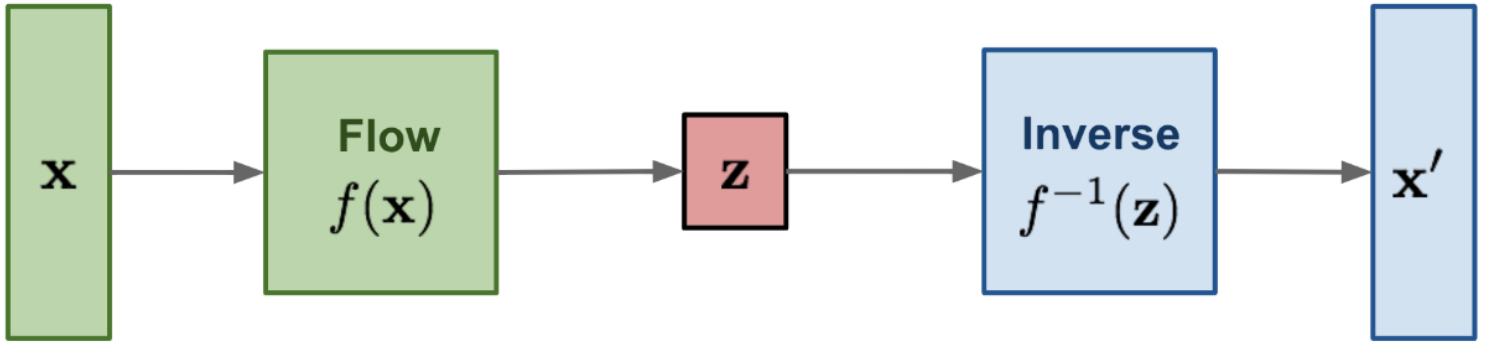
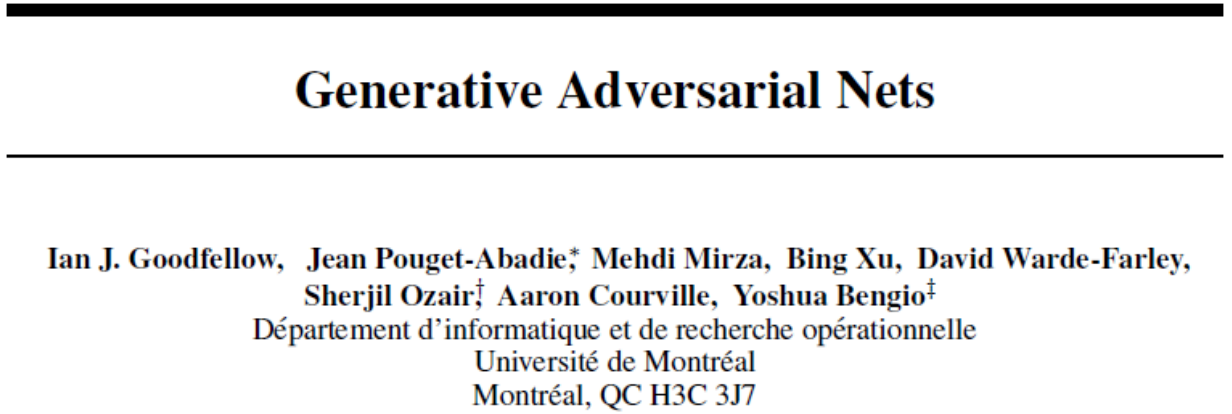


Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Generative Adversarial Networks

Generative Adversarial Networks (GANs)

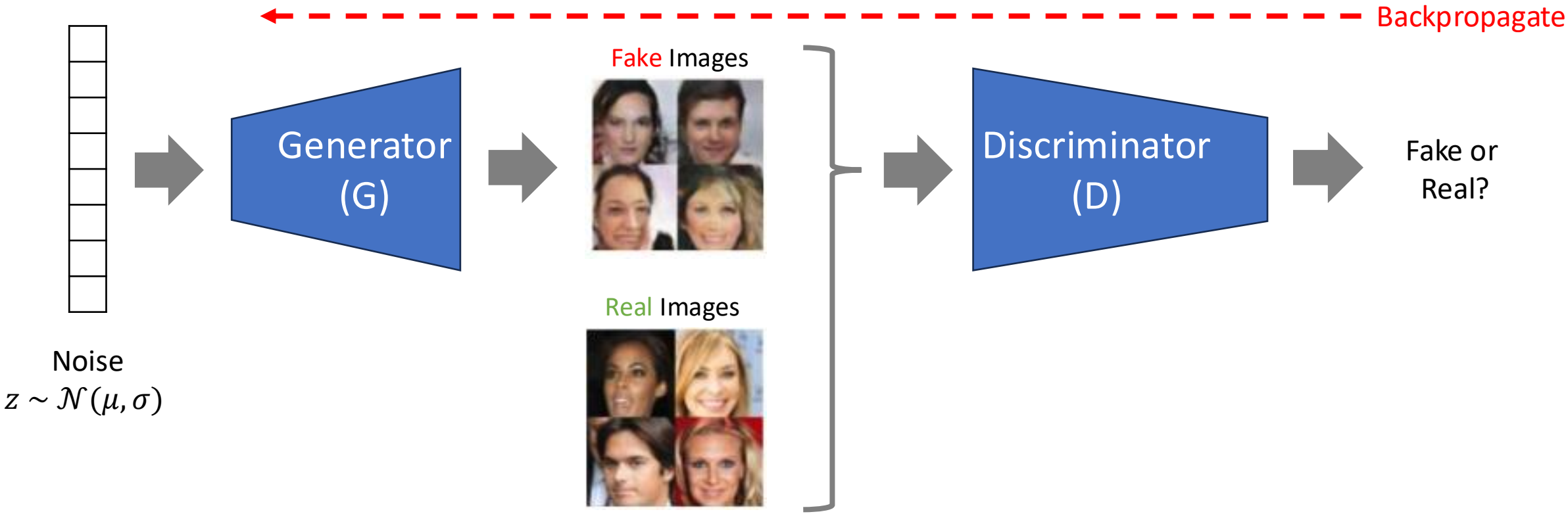
- Originally proposed by Ian Goodfellow in 2014
- Won the “Test of Time” award at NeurIPS2024
  - https://blog.neurips.cc/2024/11/27/announcing-the-neurips-2024-test-of-time-paper-awards/
- It all started in a pub ☺
  - Full story here: https://x.com/sherjilozair/status/1864013580624113817



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)

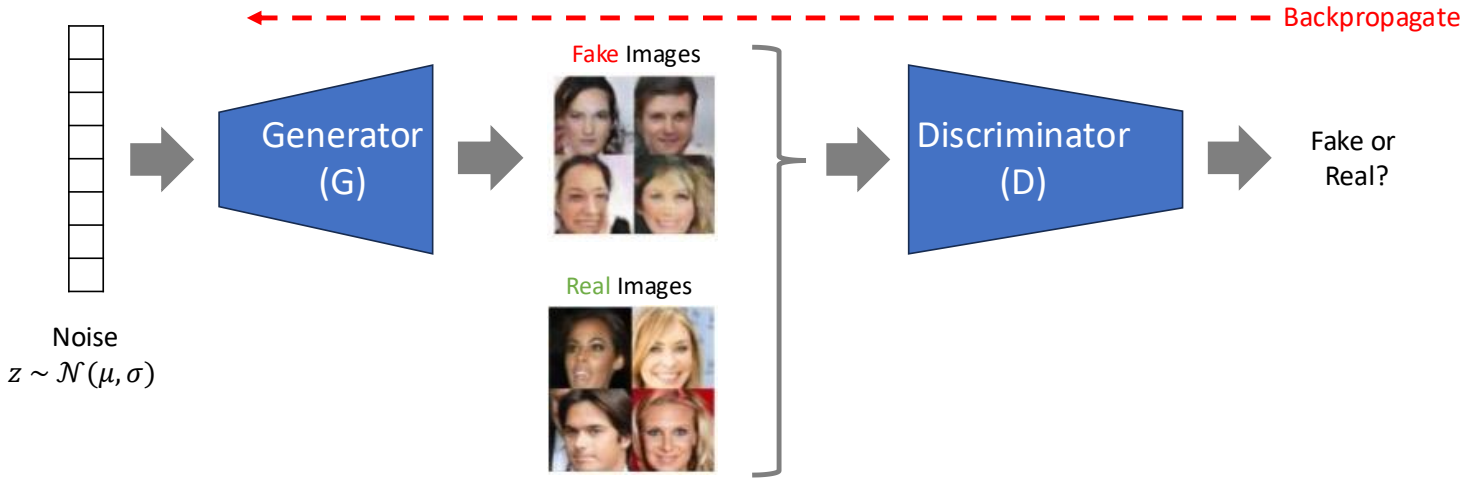


We have two networks:

- 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

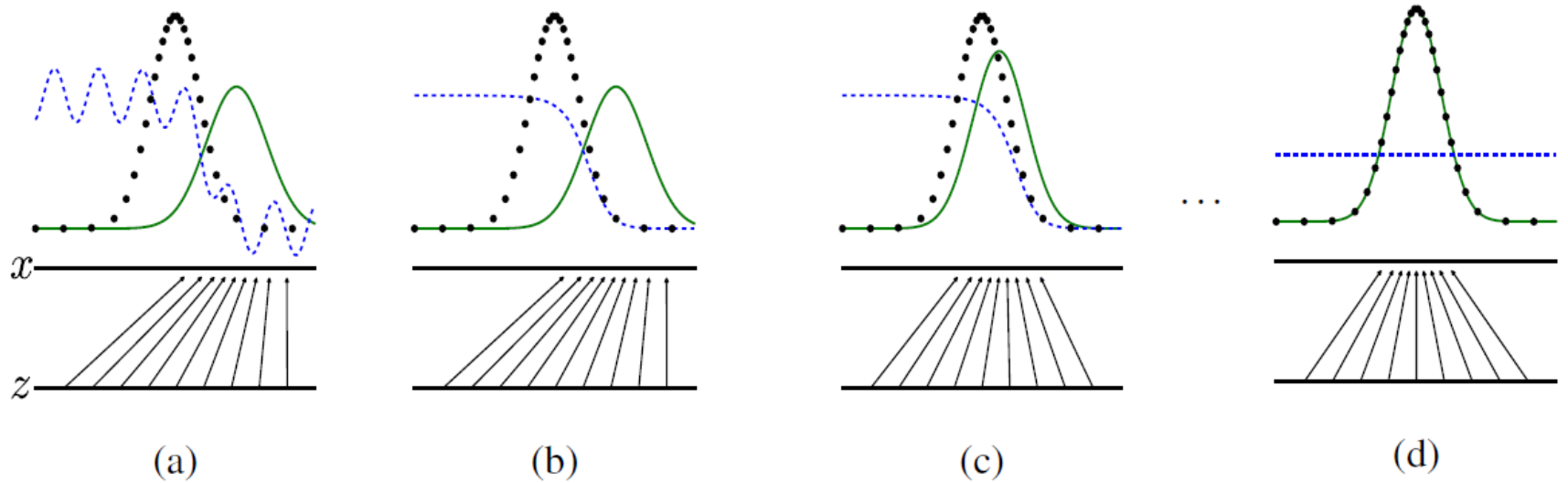


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution 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

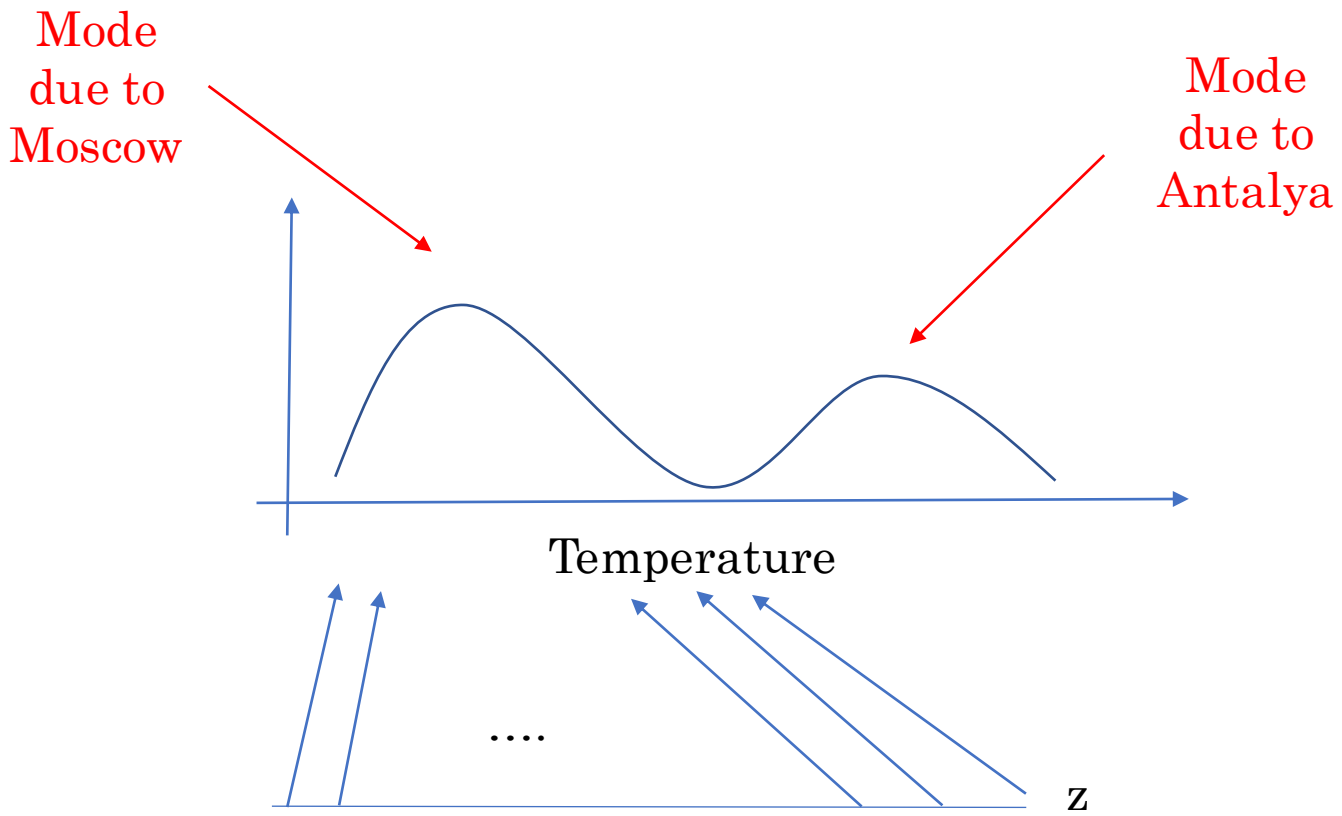
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

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

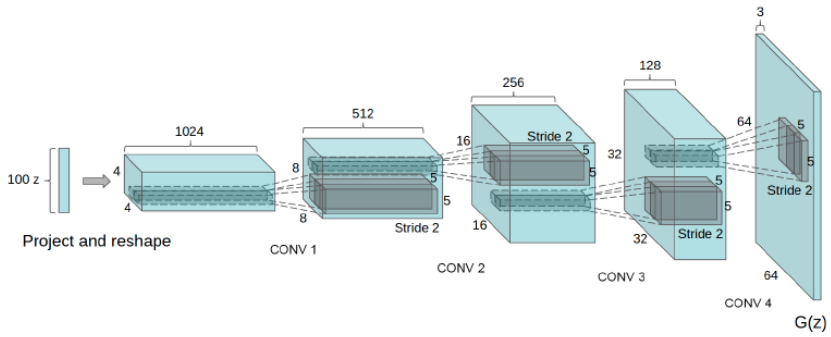
- Solutions:
  - Changing the training procedure (use batch discrimination instead of individual discrimination)
  - Experience replay (show old fake images again and again)
  - Use a different loss (+ enforce diversity)
  - …
- Other tips and tricks:
  - https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

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.

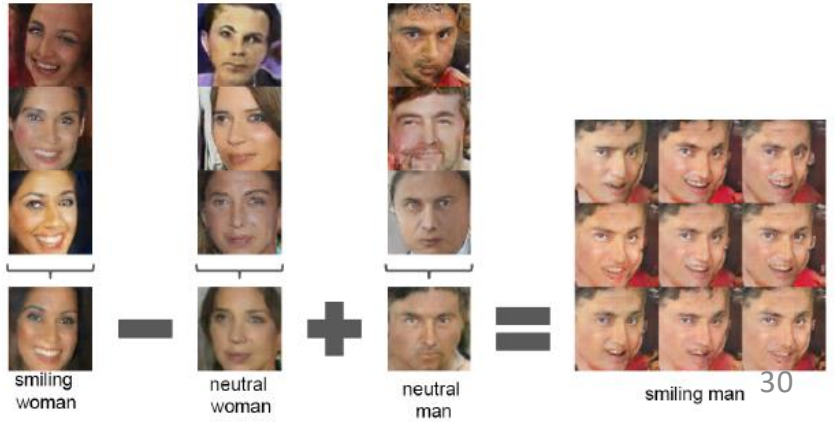


Under review as a conference paper at ICLR 2016

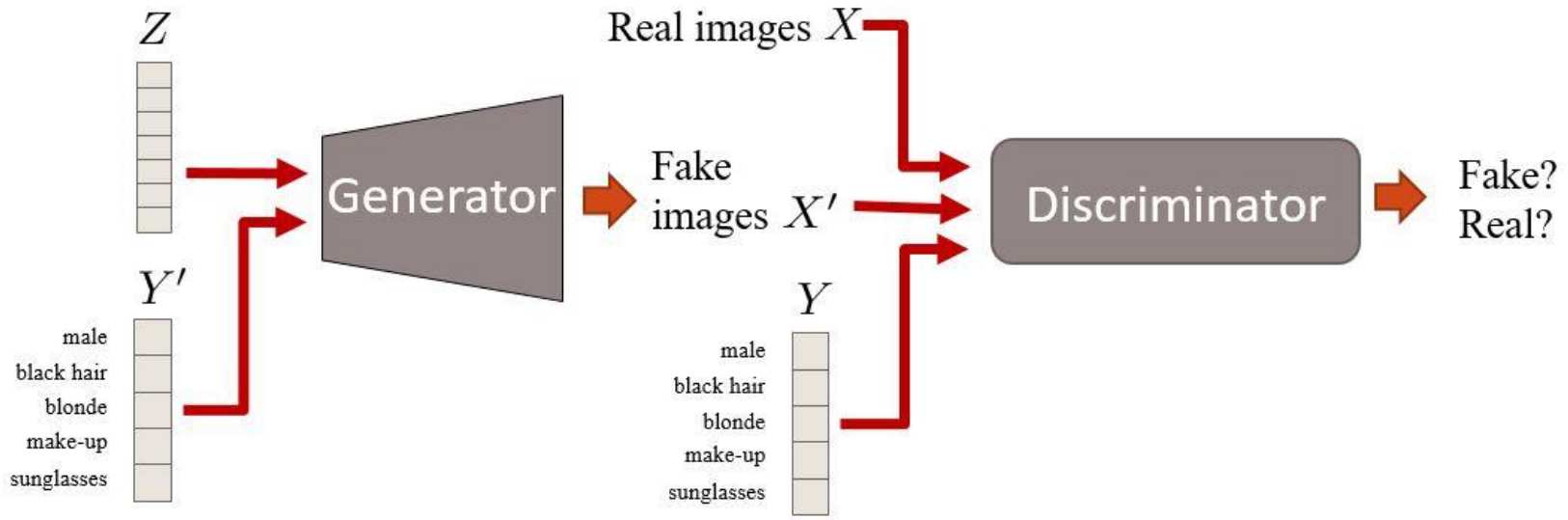
UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

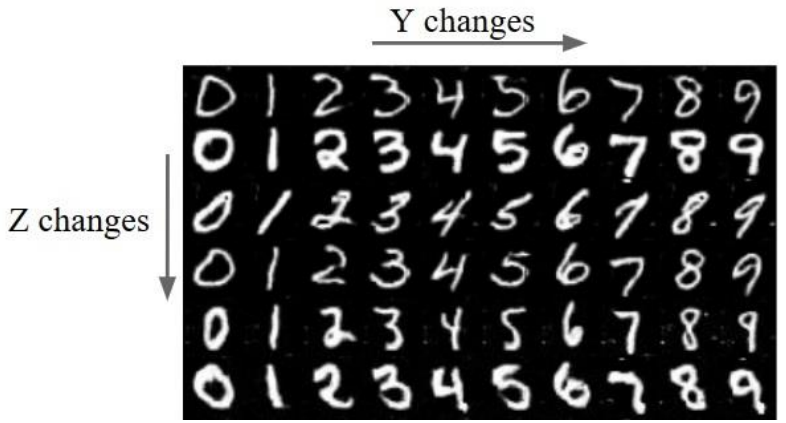
Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

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



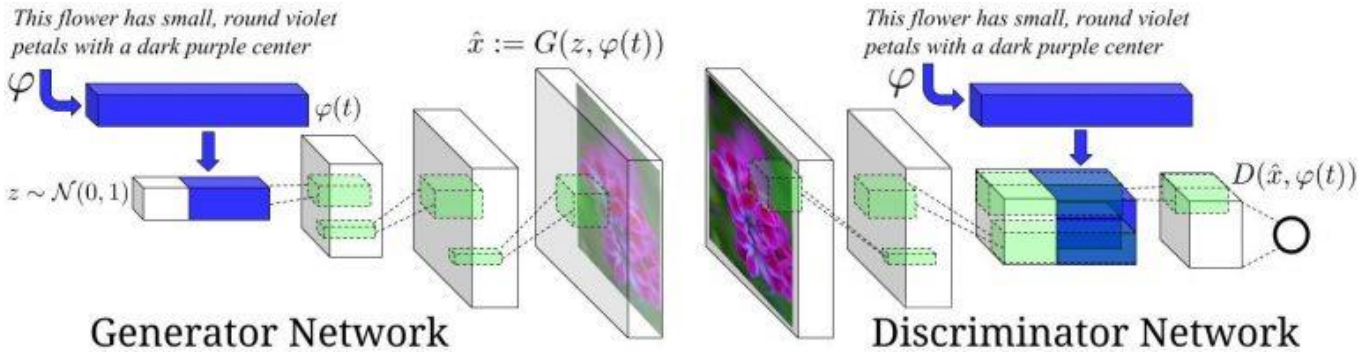
Conditional GANs





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

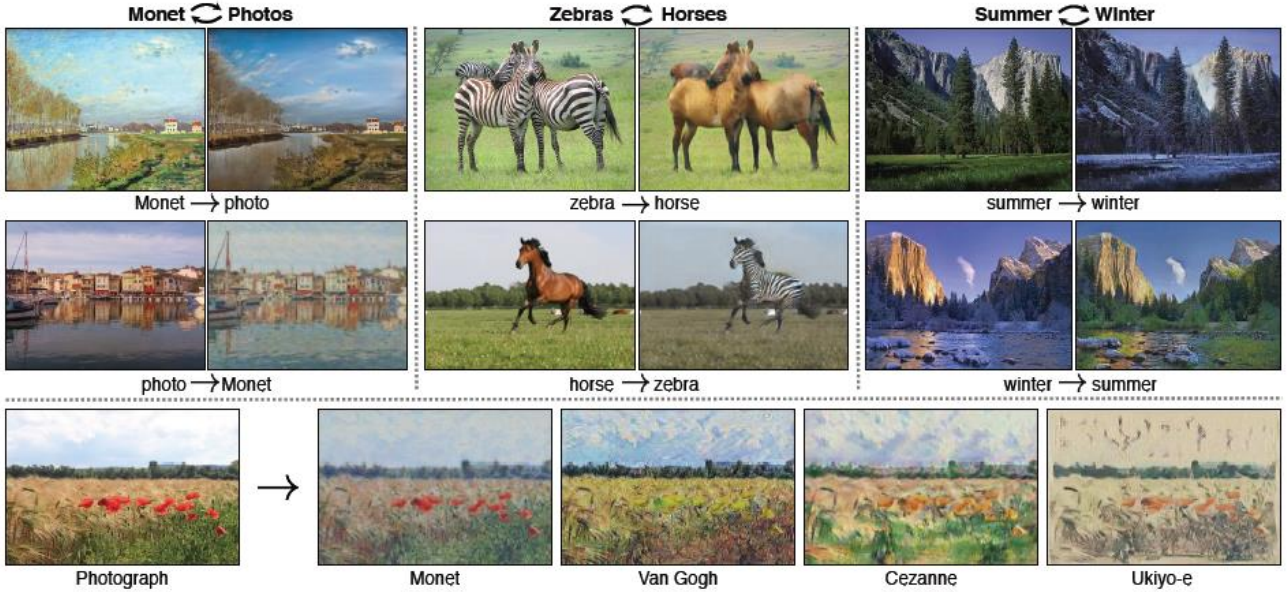
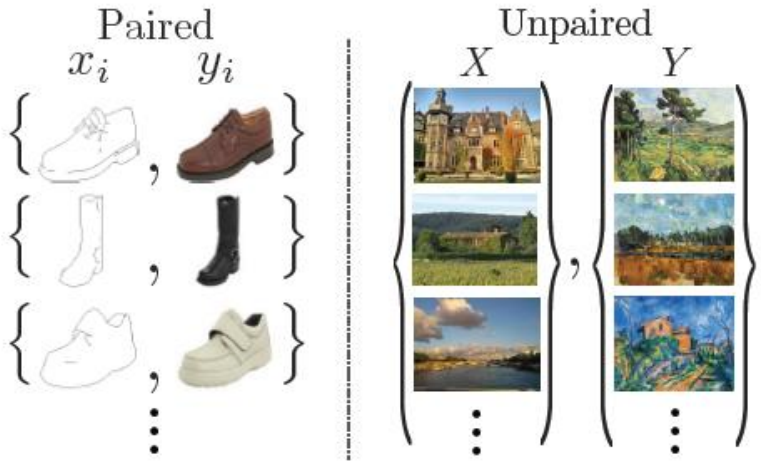


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* 1074 Monet paintings and 6753 landscape photos from Flickr; *(center)* 1177 zebras and 939 horses from ImageNet; *(right)* 1273 summer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

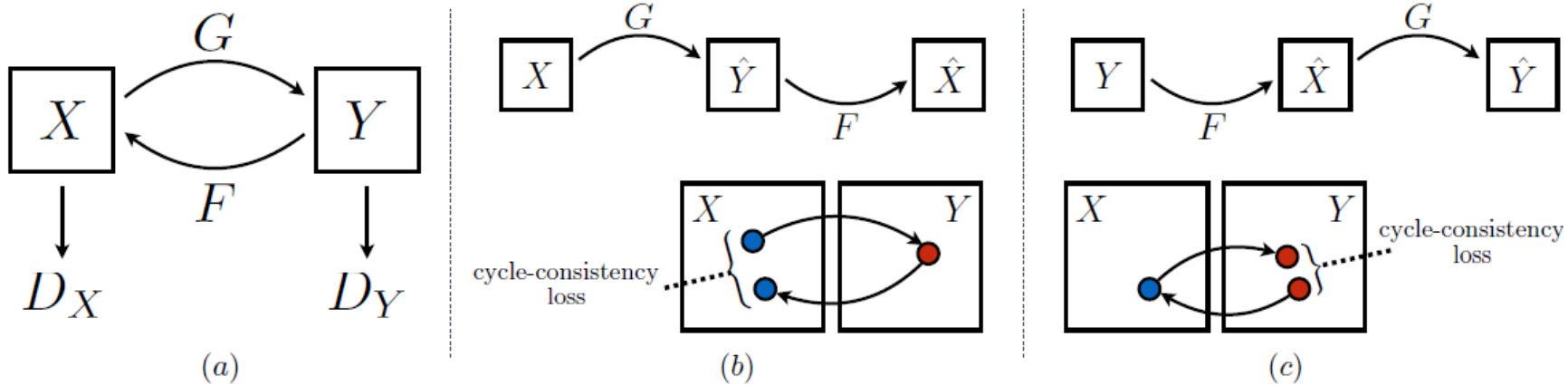


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings,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

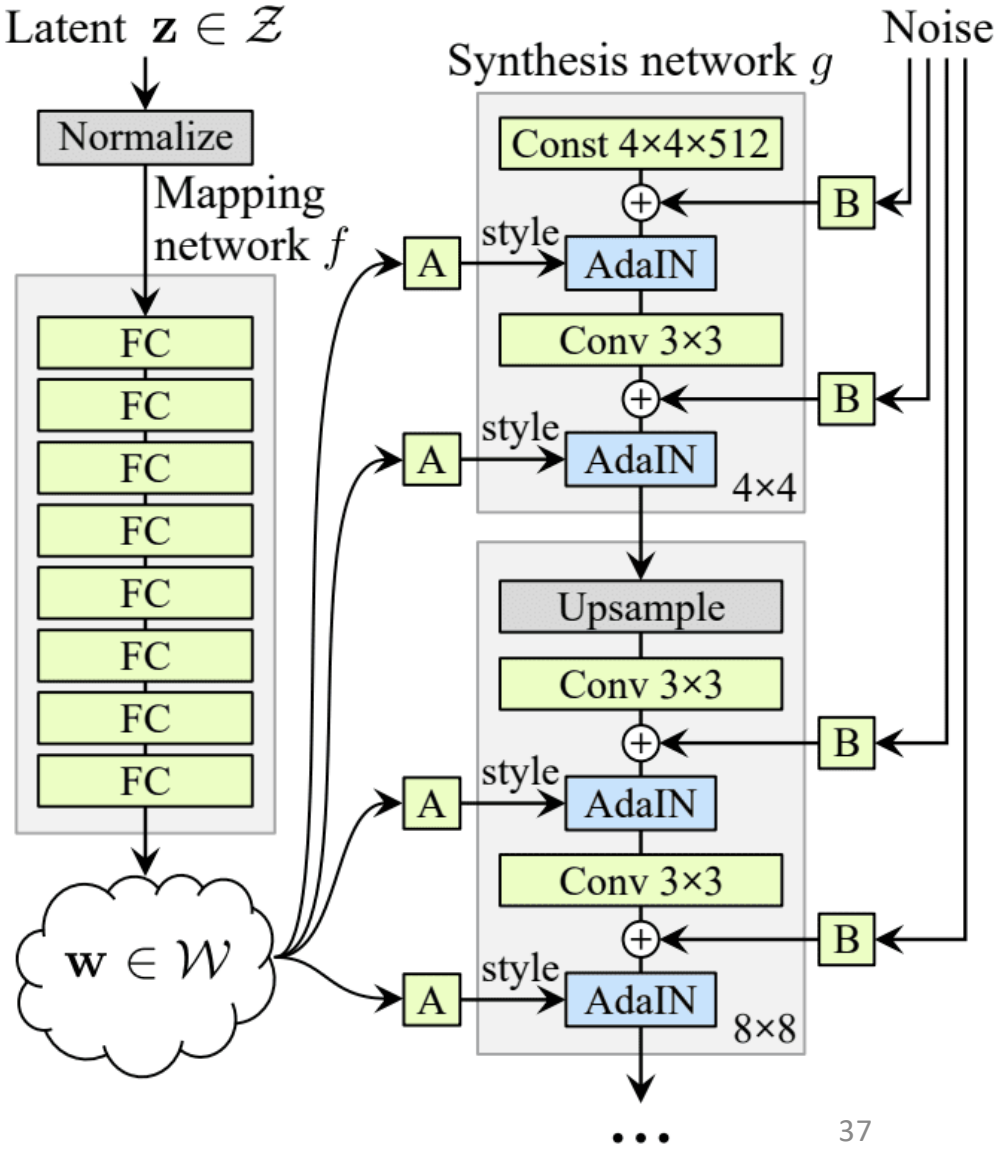
Example

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

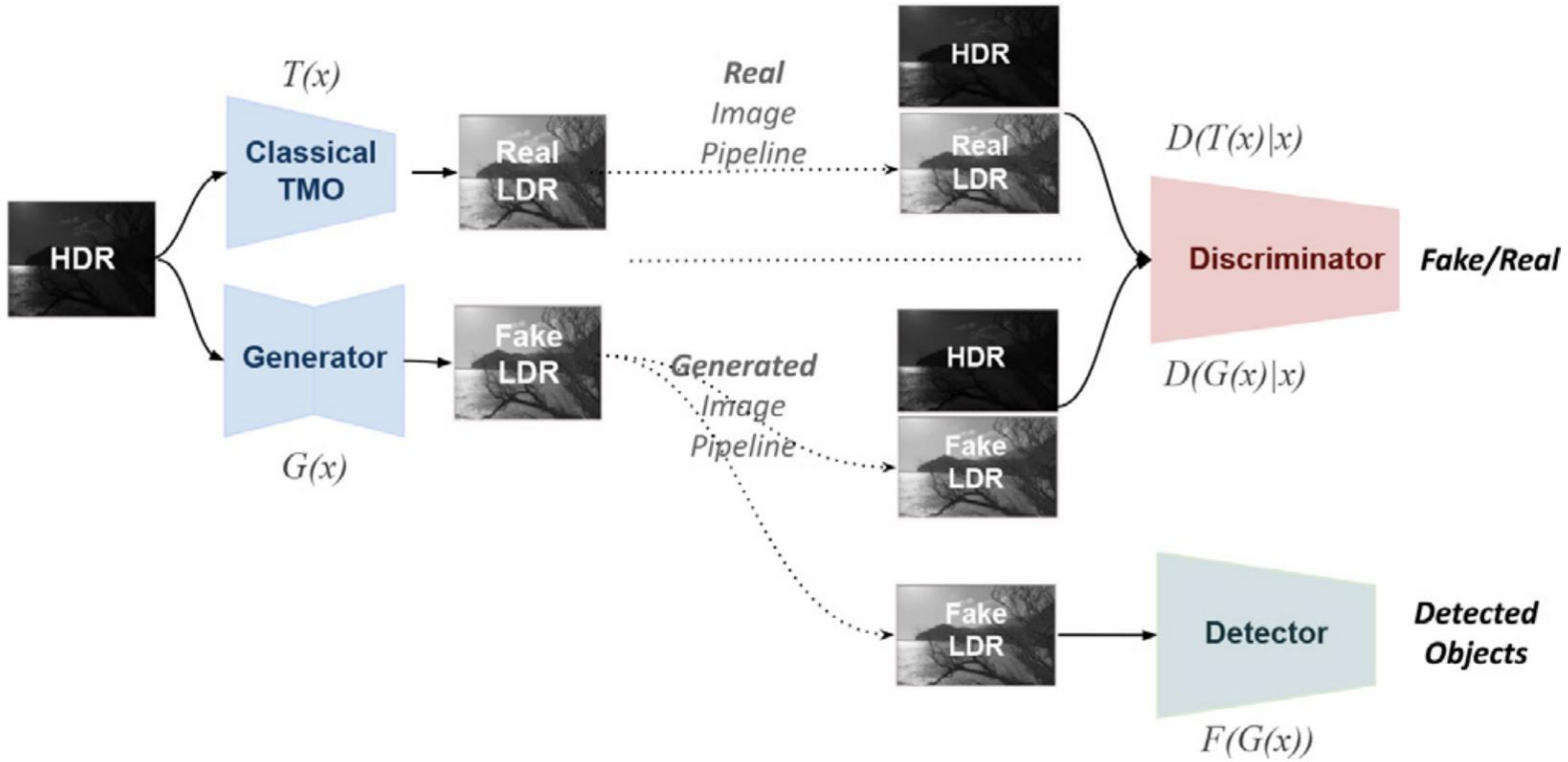
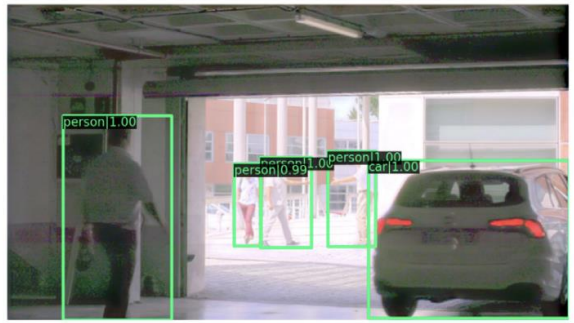
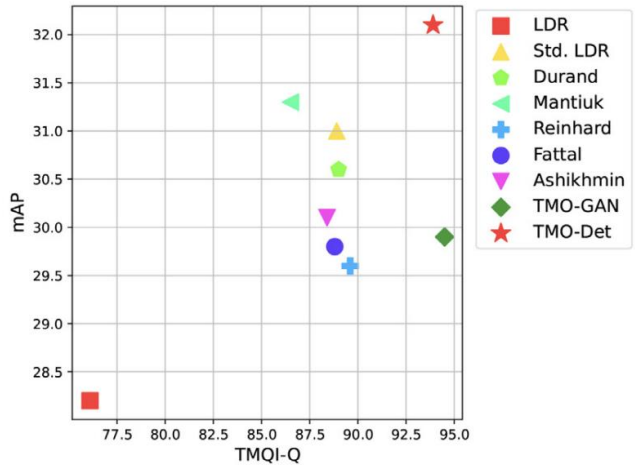


Fig. 2. Overall architecture diagram for the proposed method that combines object detection and tone-mapping objectives.

(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

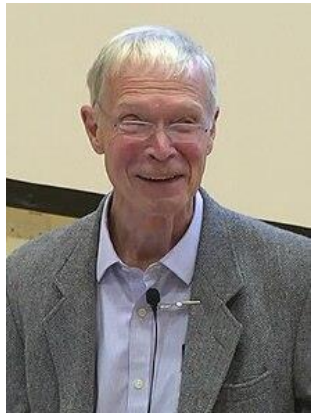
"for foundational discoveries and inventions that enable machine learning with artificial neural networks"

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.”



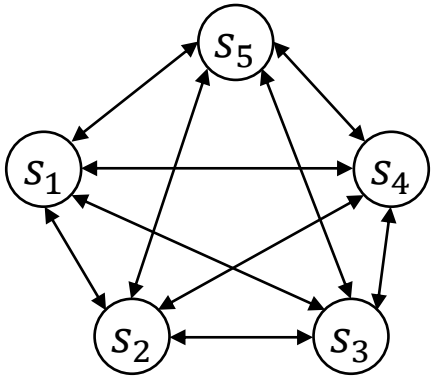
J. Hopfield
(born in 1933)

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/

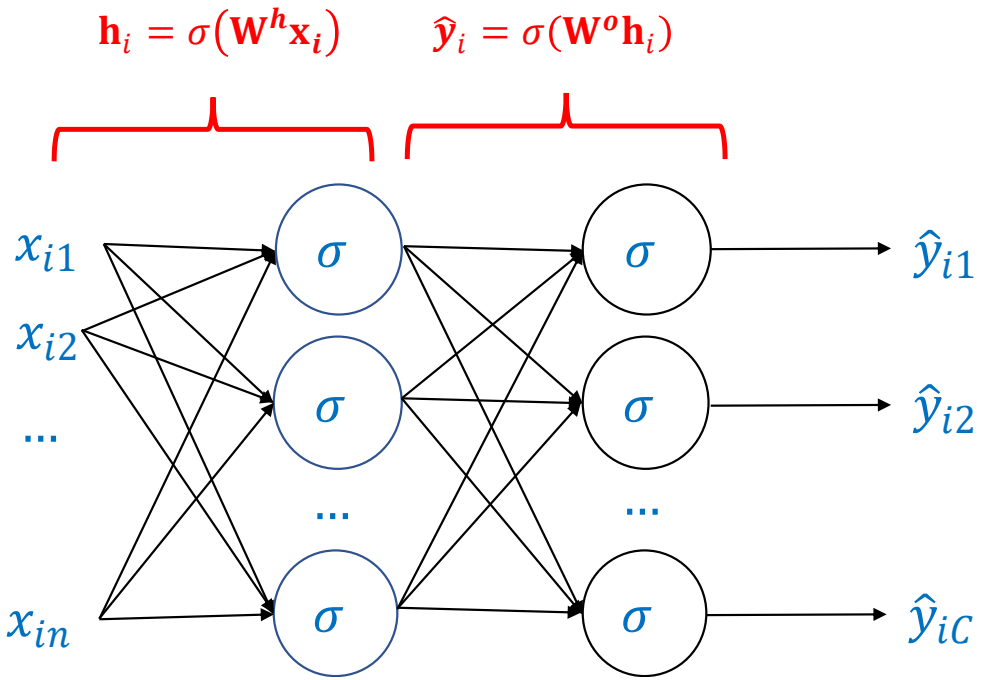


Hopfield Networks

(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:

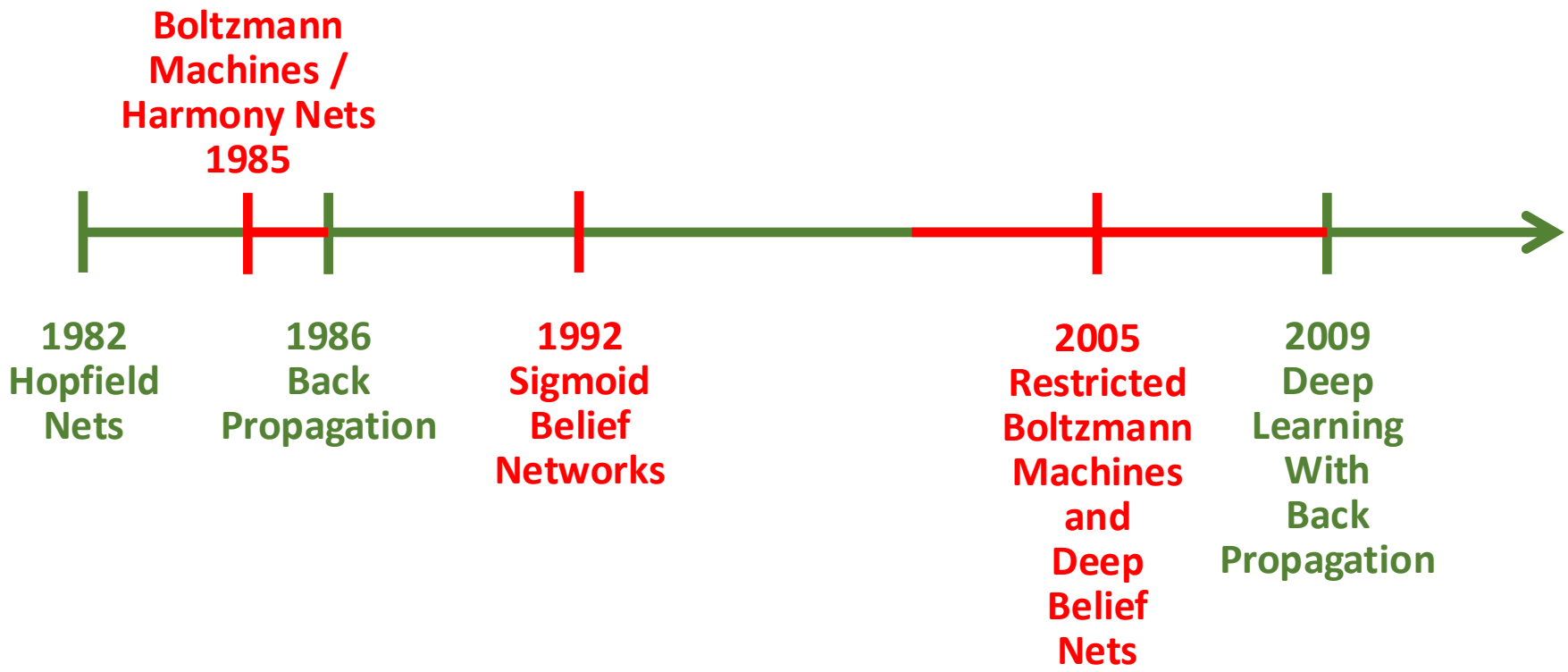
$$h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(net_{ij}^h)$$

Output layer:

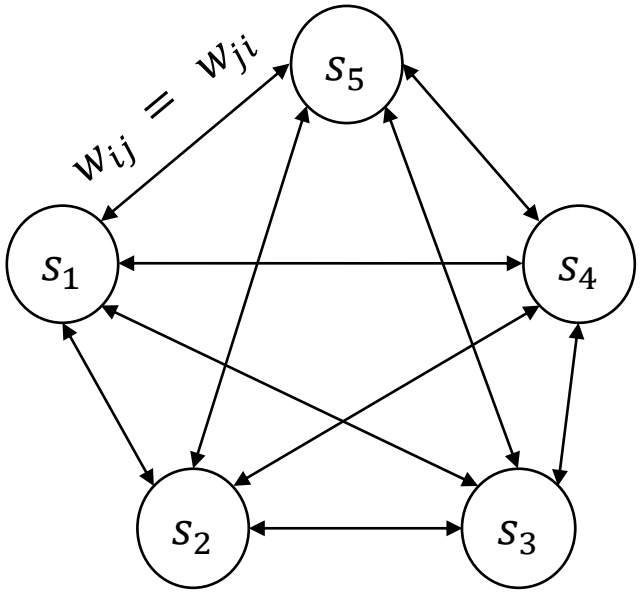
$$\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$$

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



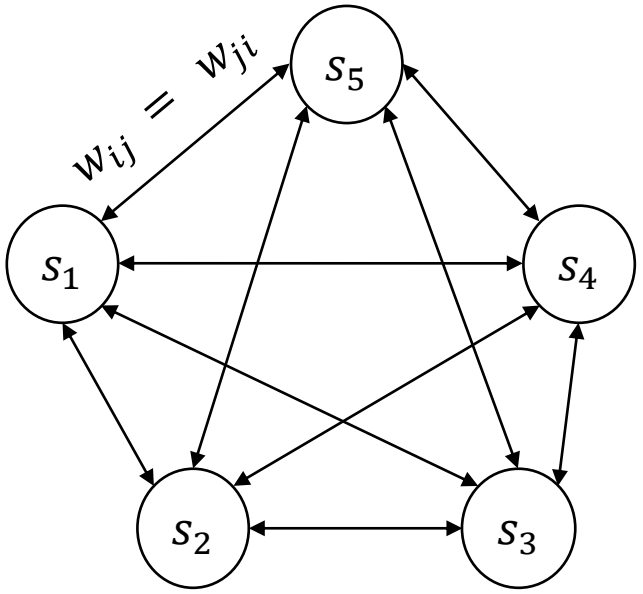
- $s_i = -1 \text{ or } +1$
- Then,

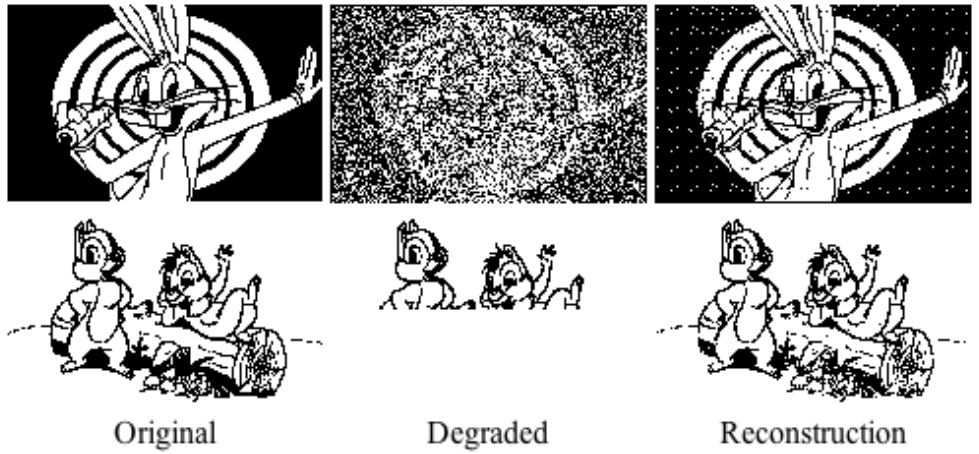
$$s_i \leftarrow \begin{cases} +1, & \sum_j w_{ij} s_j \geq \theta_i \\ -1, & \text{otherwise} \end{cases}$$

- 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



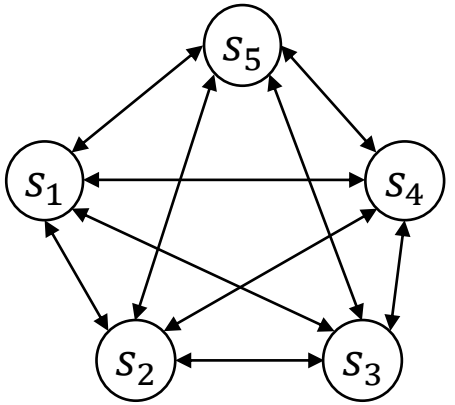


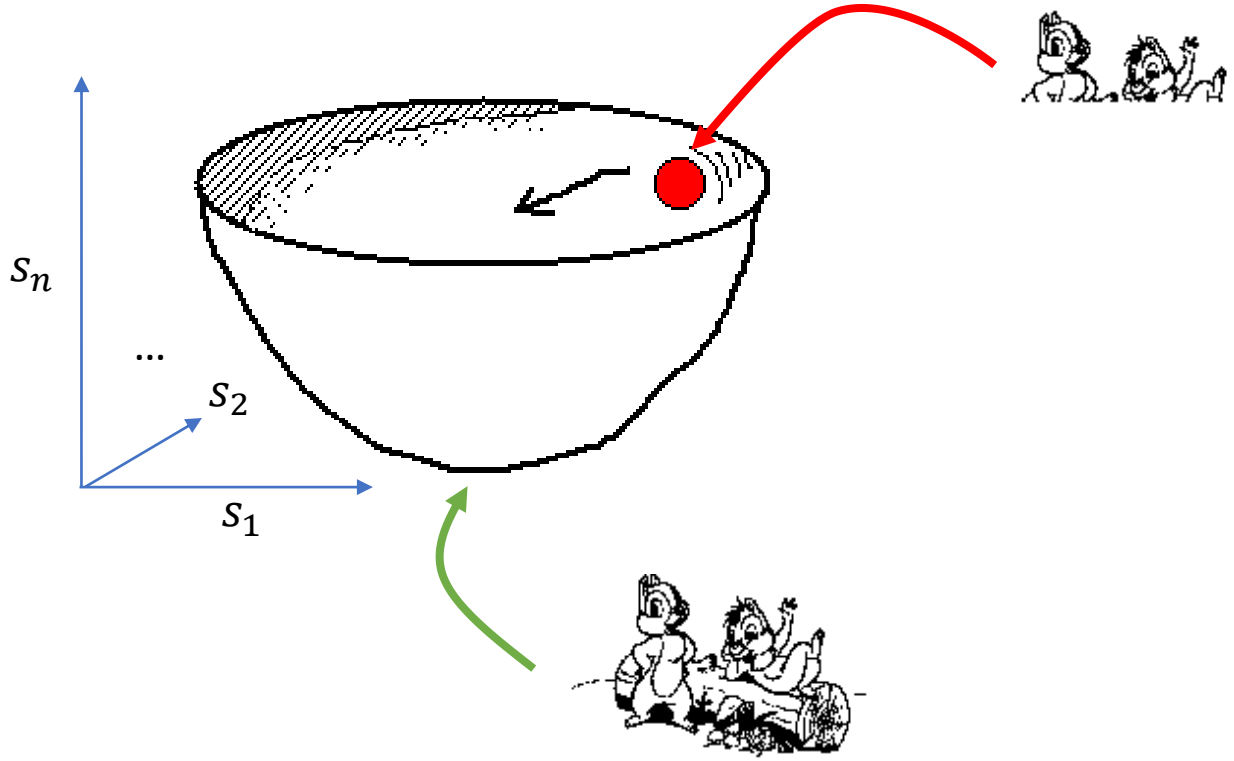
Hopfield Networks

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

- Degraded input is mapped to nearest attractor

Hopfield Networks as Content-addressable Memory



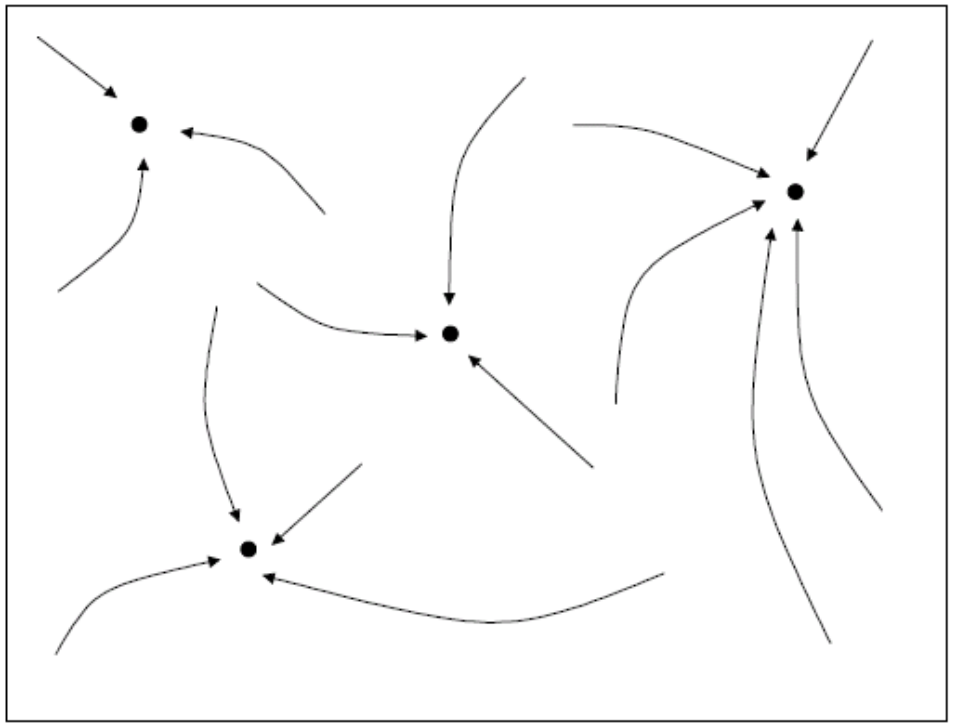


Adapted from: E. Sahin

Hopfield Networks as Content-addressable Memory

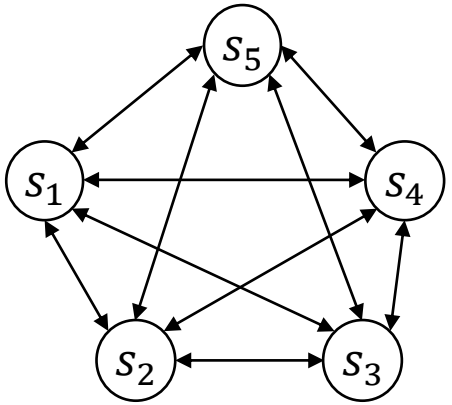
- CAM can be defined as a system whose stable points can be set as a set of pre-defined states.

- 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

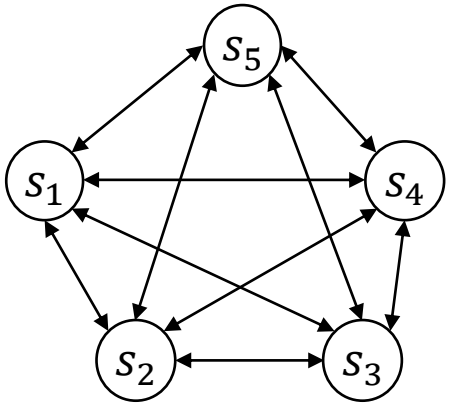


- Three possible schemes:
  - Synchronously: all units updated at each step.
  - Asynchronously I: at each time step select a random unit for update.
  - Asynchronously II: each unit independently chooses to update itself with some probability per unit time.

- 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:

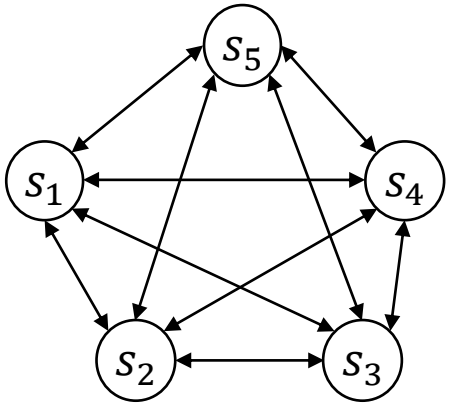
$$w_{ij} = \frac{1}{N}\mathcal{P}_i\mathcal{P}_j$$

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

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

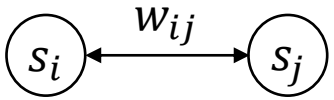
  (e.g., $w_{12} = \frac{1}{4}(-1 \times -1 + 1 \times 1) = 2/4$)
- Inputs and reconstructions (using $s_i = \mathrm{sgn}\left(\left[\sum_j w_{ij} s_j\right] - \theta_i\right)$):
  - $\mathcal{P}^3 = (-1, -1, -1, +1) \Rightarrow$ Recall: $(-1, -1, -1, +1)$
  - $\mathcal{P}^4 = (-1, +1, +1, +1) \Rightarrow$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

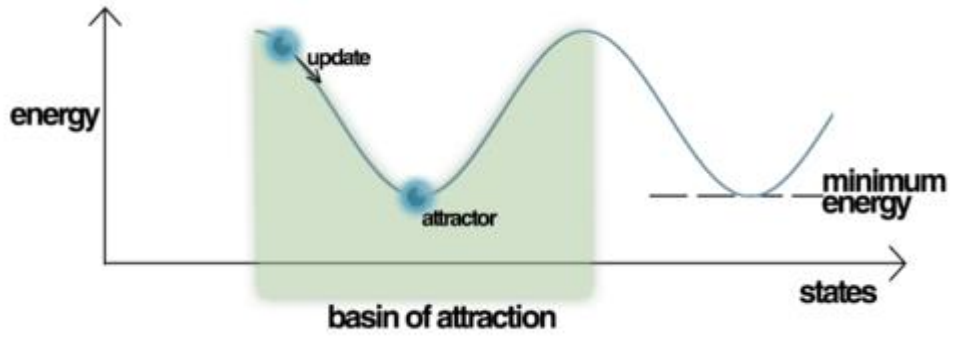


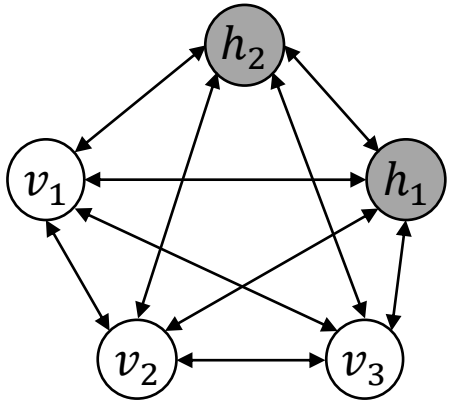
Fig: Wikipedia

Hopfield Networks

- An associative memory
- Inspired many models in Machine Learning

Skipping:

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



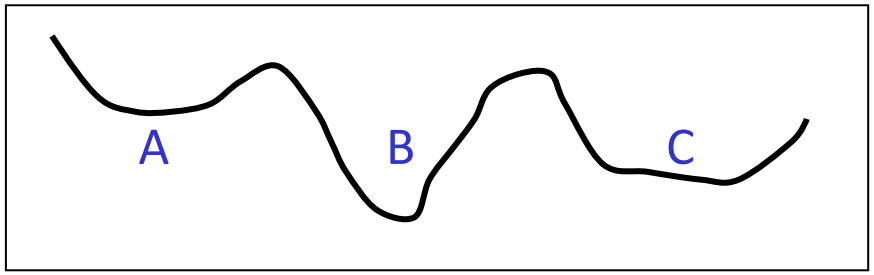
Boltzmann Machines

(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 $\mathbf{s}$,

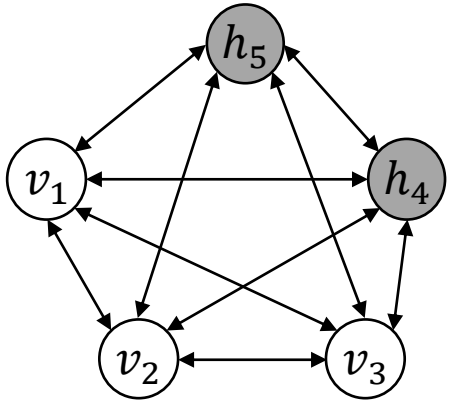
k: Boltzmann's constant, T: temperature.

- 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:

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

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).

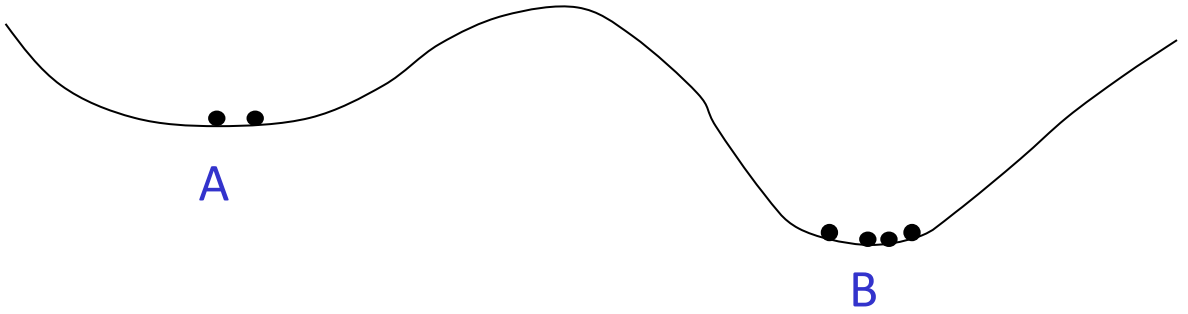
d. For a fixed T, if ΔE_i is very high, same as case (a).

- 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

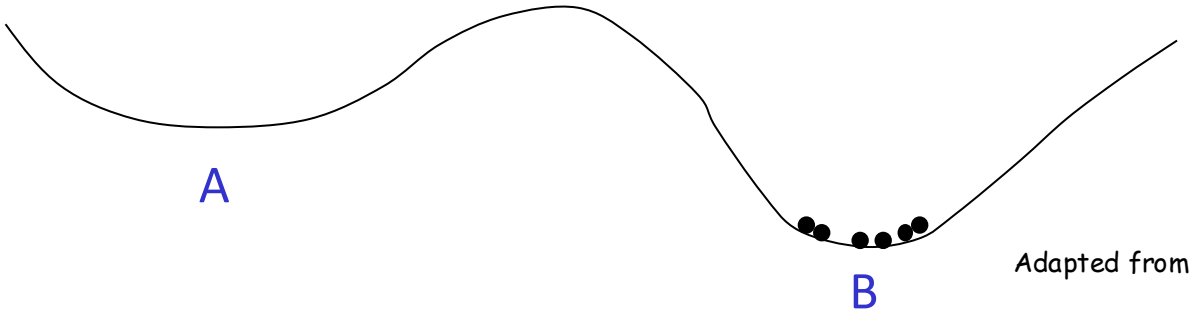
High temperature transition probabilities

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



Low temperature transition probabilities

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

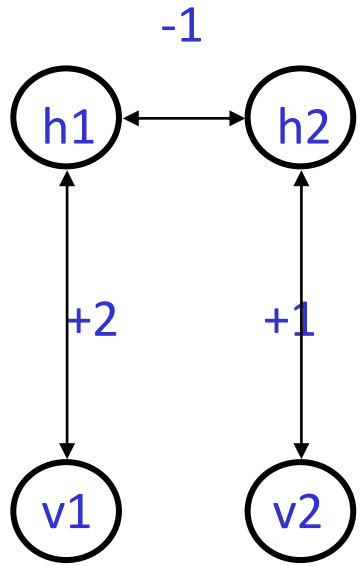


Adapted from G. Hinton

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:

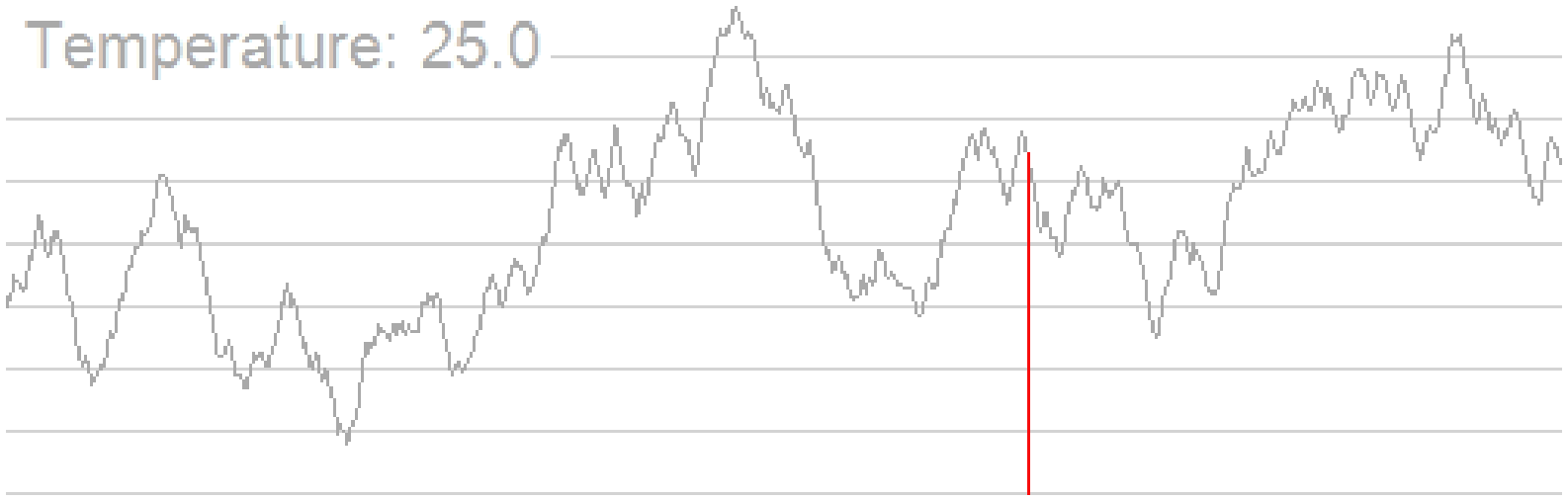
$$p_{i=on} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)}$$

- 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

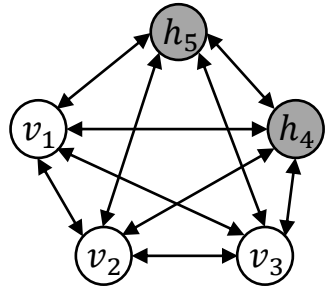
- How do we understand we have reached it?
  - The average activation of neurons don't change over time.
  - i.e., the probability of being in a state does not change.
- The initial state is not important!
- At low temperature:
  - There is a strong bias for states with low energy
  - But this makes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

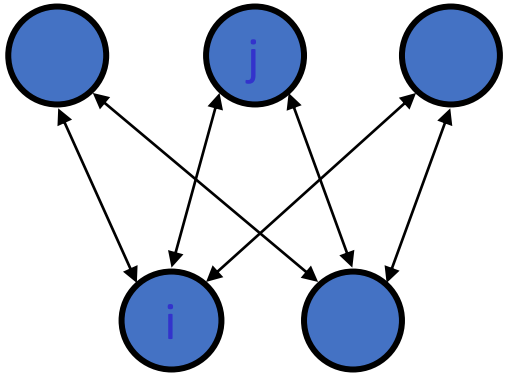
- Too slow
  - loop over training epochs
    loop over training examples
    loop over 2 phases (+ and -)
    loop over annealing schedule for T
    loop until thermal equilibrium reached

- 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



hidden

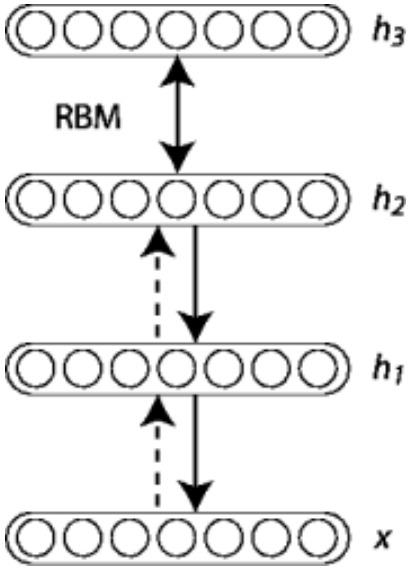
visible

Adapted from G. Hinton

Deep Belief Networks

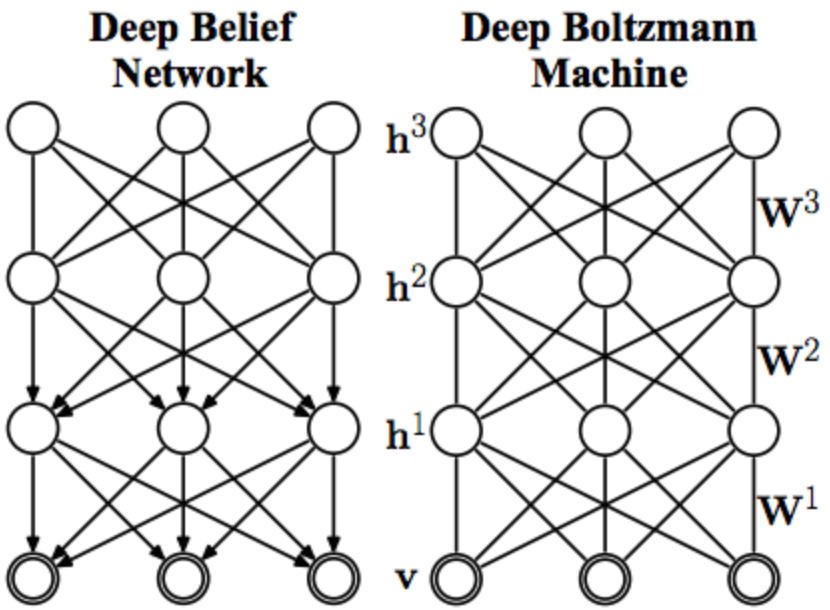
- A stacked RBM
- First used by Hinton & Salakhutdinov (2006)
- Models the distribution:

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

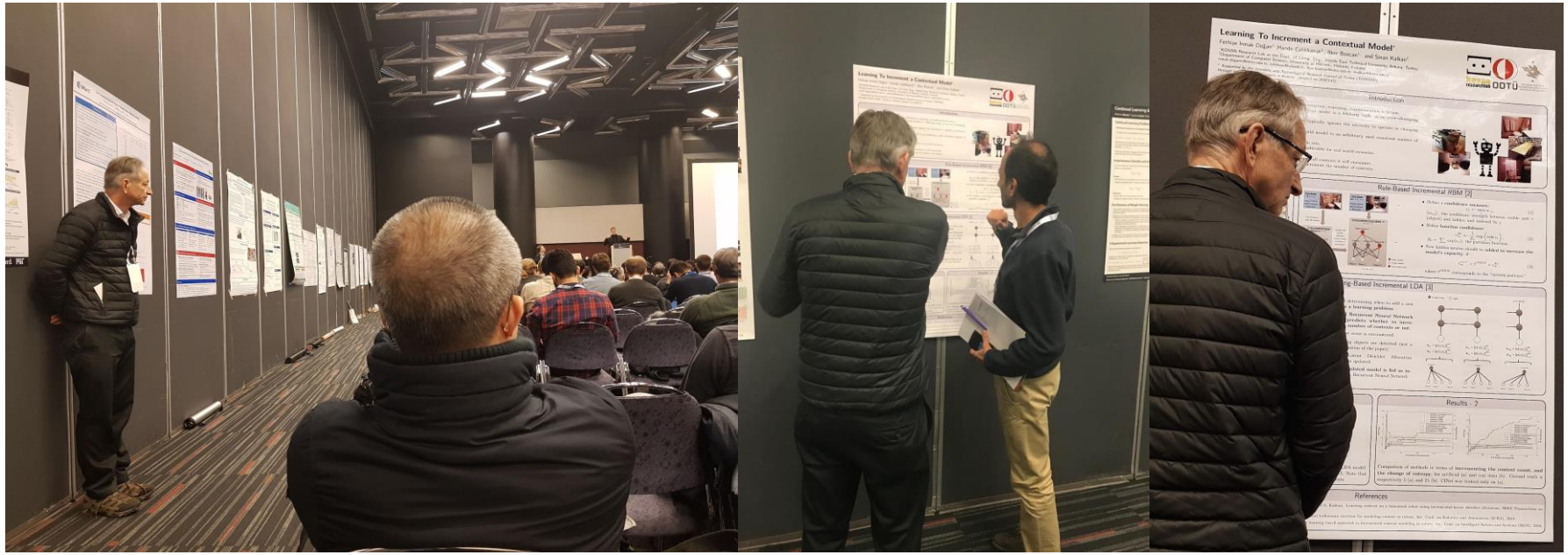


- Training is similar to autoencoders

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

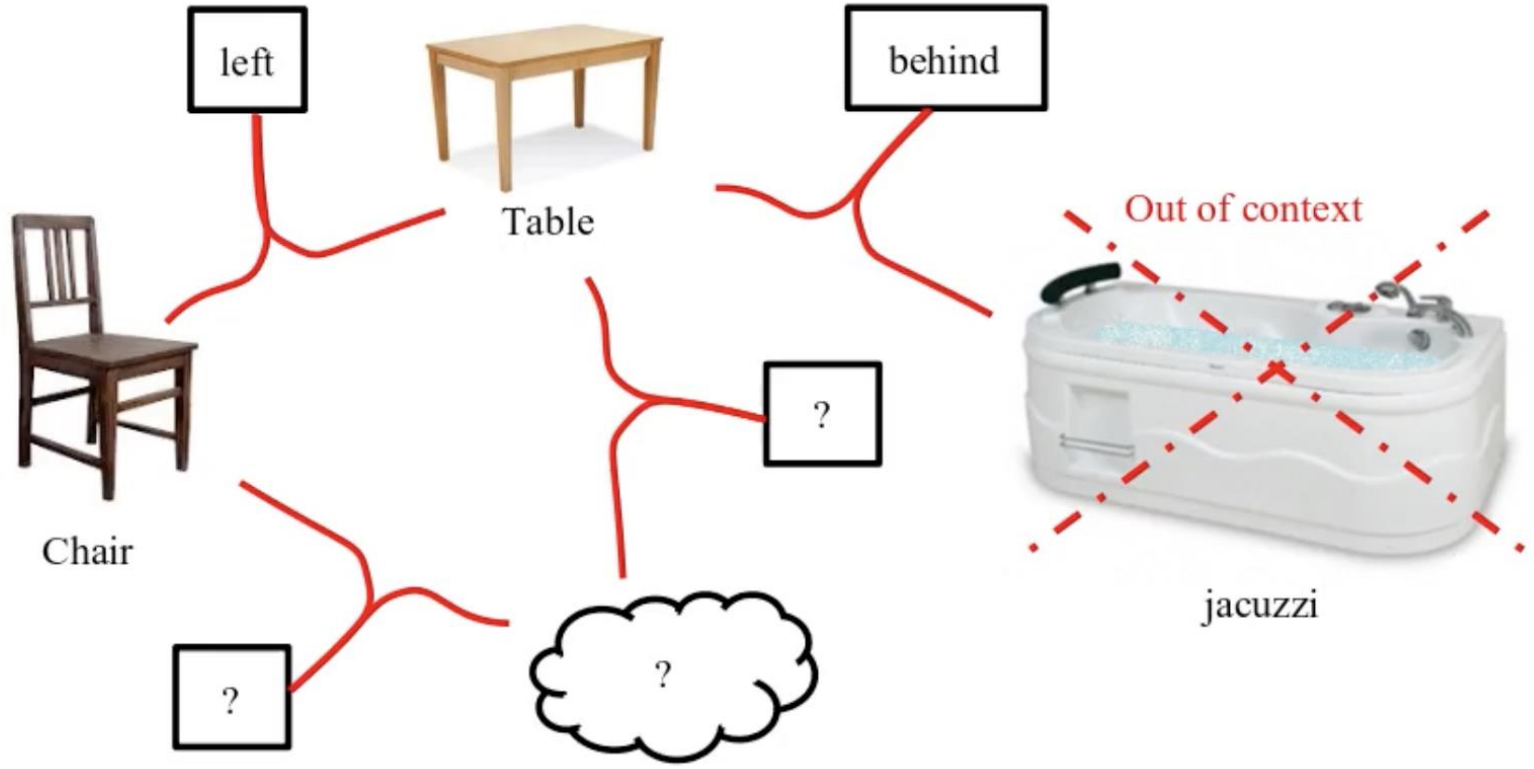


Wang, H., & Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.



Our Work Using Boltzmann Machines

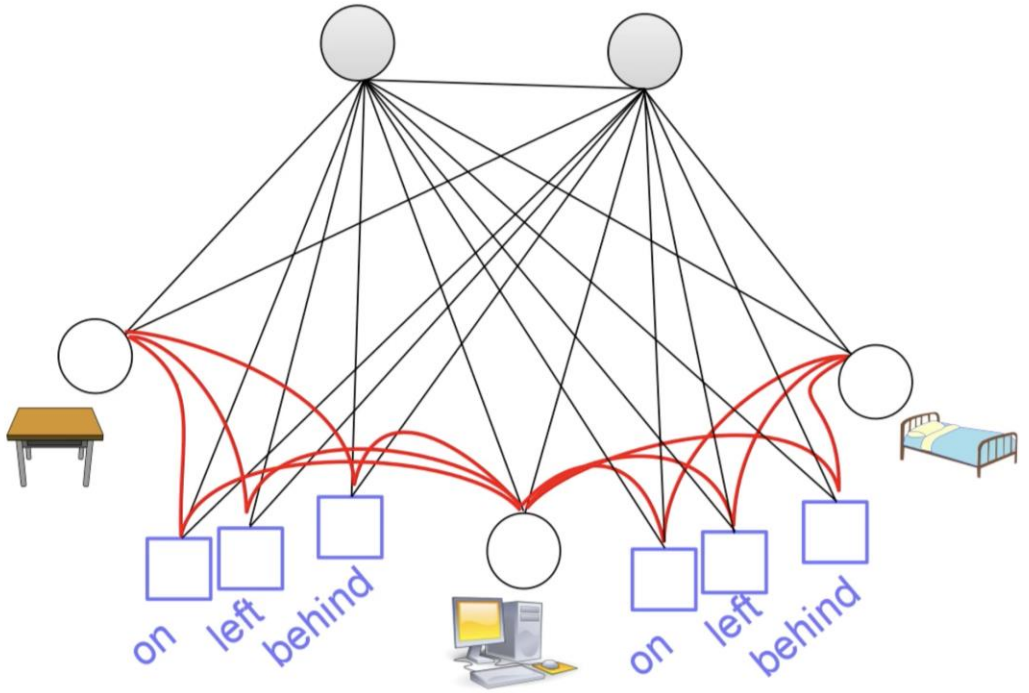
Boltzmann Machines: Our Work



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

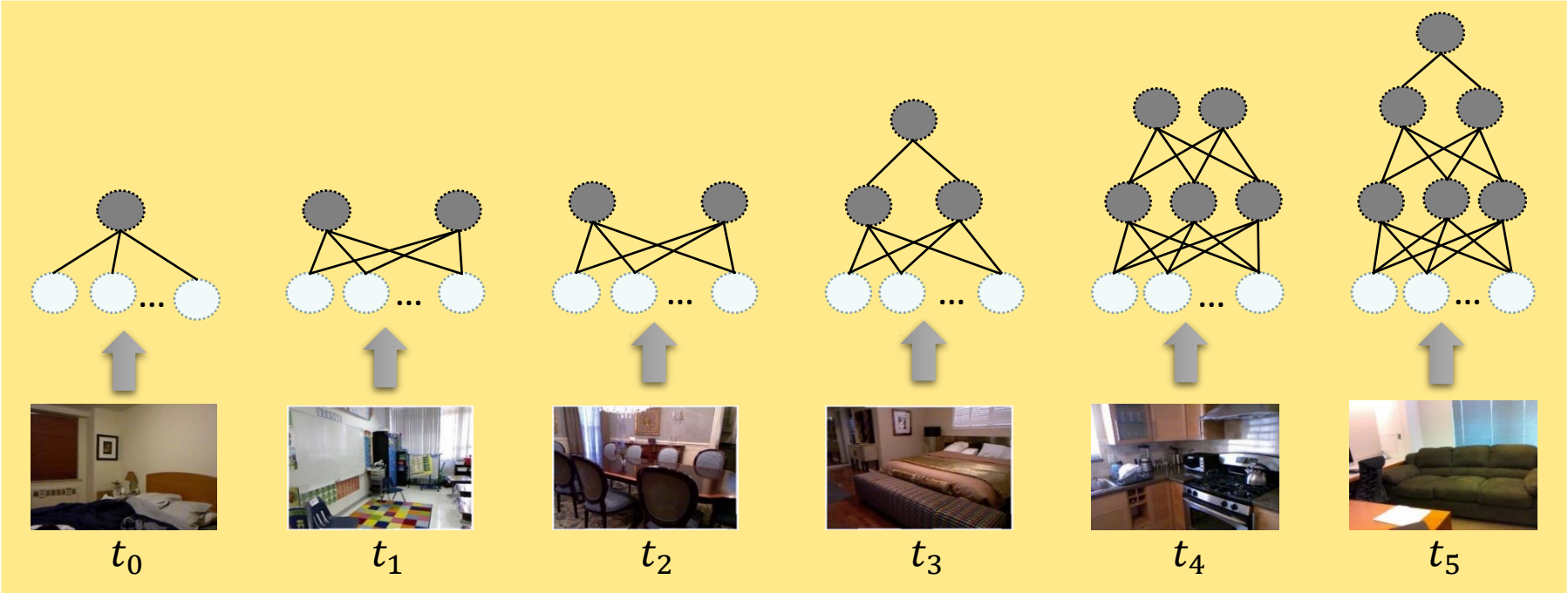


Hidden Units Visible Units Relation Units (visible)

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.

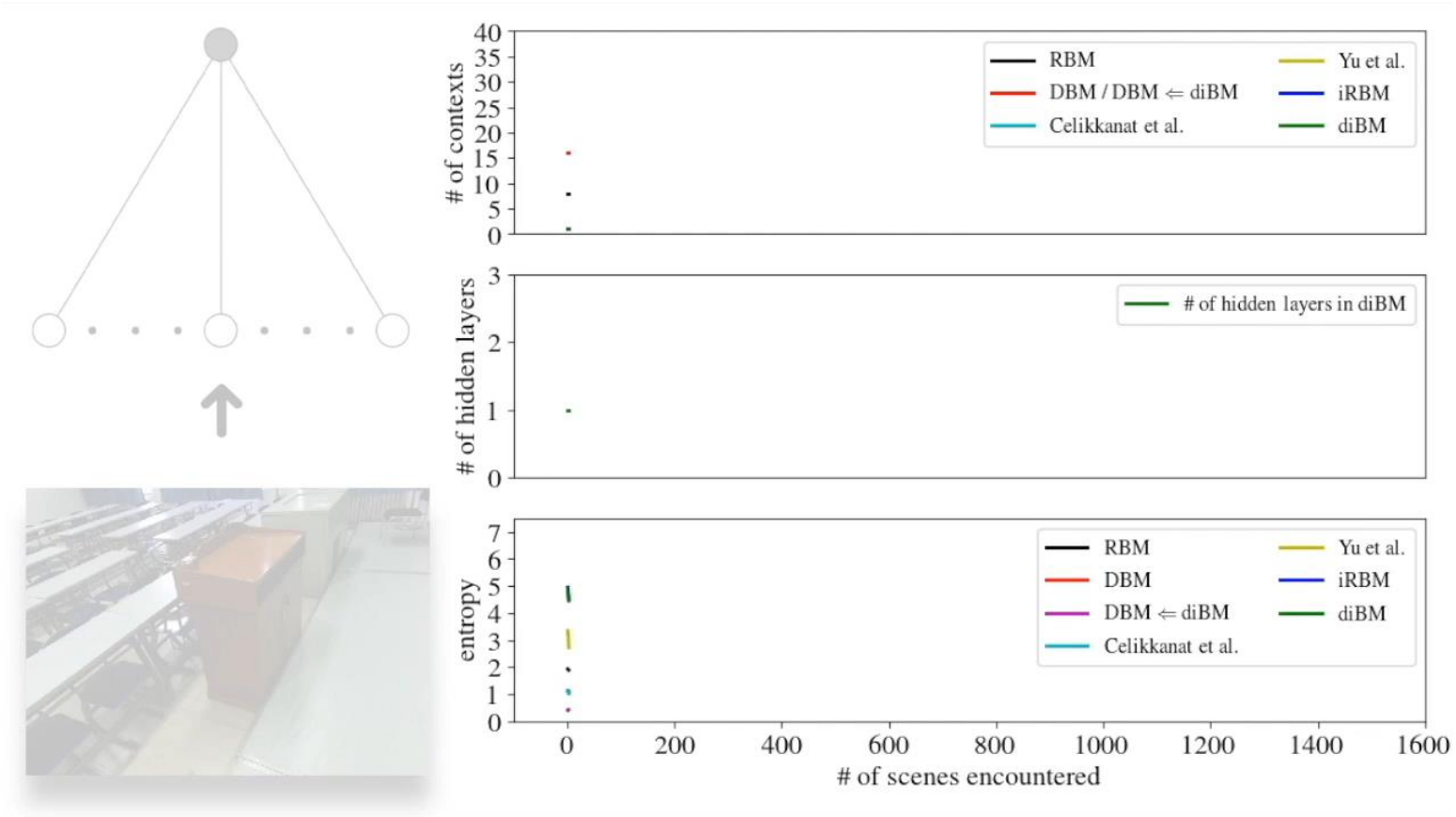
Boltzmann Machines: Our Work



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.

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 $\boldsymbol{Z}$ (intractable for complex models) OR
- It involves having access to the ground truth latent encoder $p(z|x)$

ELBO:

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

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

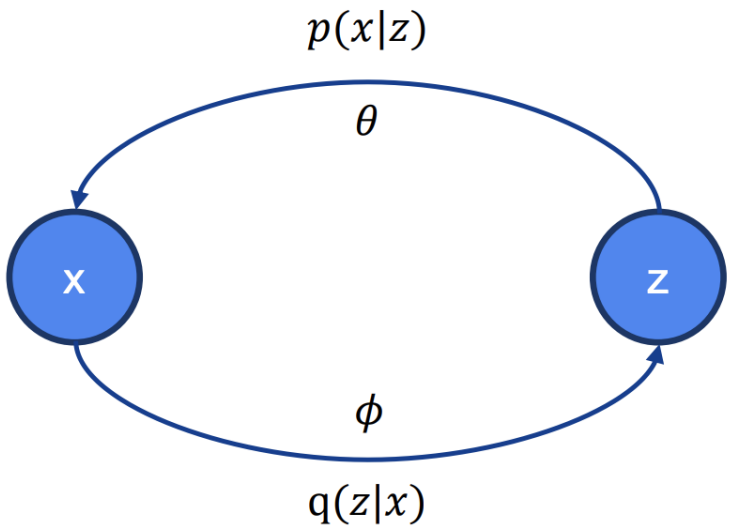
Applying chain-rule of probabilities:

$$ELBO = \mathbb{E}_{q_\phi(z \mid x)}[\log p_\theta(x|z)] - D_{KL}\big(q_\phi(z \mid x)| \,|p(z)\big)$$

Reconstruction Prior matching

Slide: https://deeplearning.cs.cmu.edu/F23/document/slides/lec23.diffusion.updated.pdf

Variational Autoencoder Recap



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

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

Training:

- Jointly optimize θ and ϕ
- Maximize **ELBO**

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

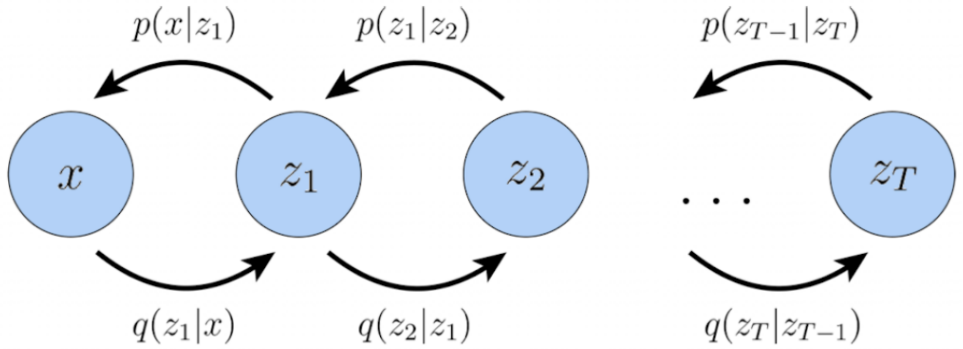
1. Increasing the depth of the networks
2. Introducing a hierarchy of latent variables (latent variables of latent variables)

$x \leftarrow z_1 \leftarrow z_2 \leftarrow \ \ldots \leftarrow z_T$, 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**

Slide: https://deeplearning.cs.cmu.edu/F23/document/slides/lec23.diffusion.updated.pdf

Markovian Hierarchical Variational Autoencoder



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

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

Updated ELBO:

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

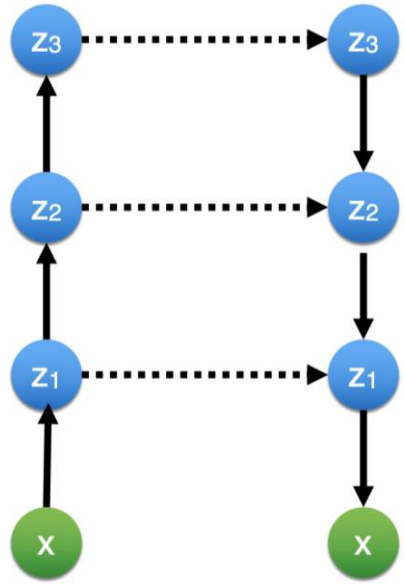


Fig: https://cs231n.stanford.edu/slides/2023/lecture_15.pdf

Slide: https://deeplearning.cs.cmu.edu/F23/document/slides/lec23.diffusion.updated.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^{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.

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

The first restriction allows for some mild abuse of notation:

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

(We are using x instead of z)

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

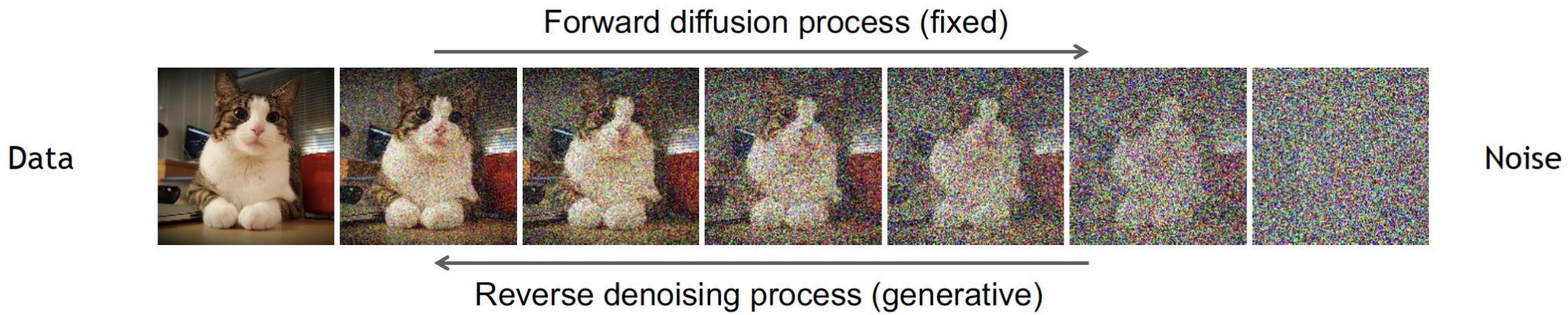
Slide: https://deeplearning.cs.cmu.edu/F23/document/slides/lec23.diffusion.updated.pdf

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

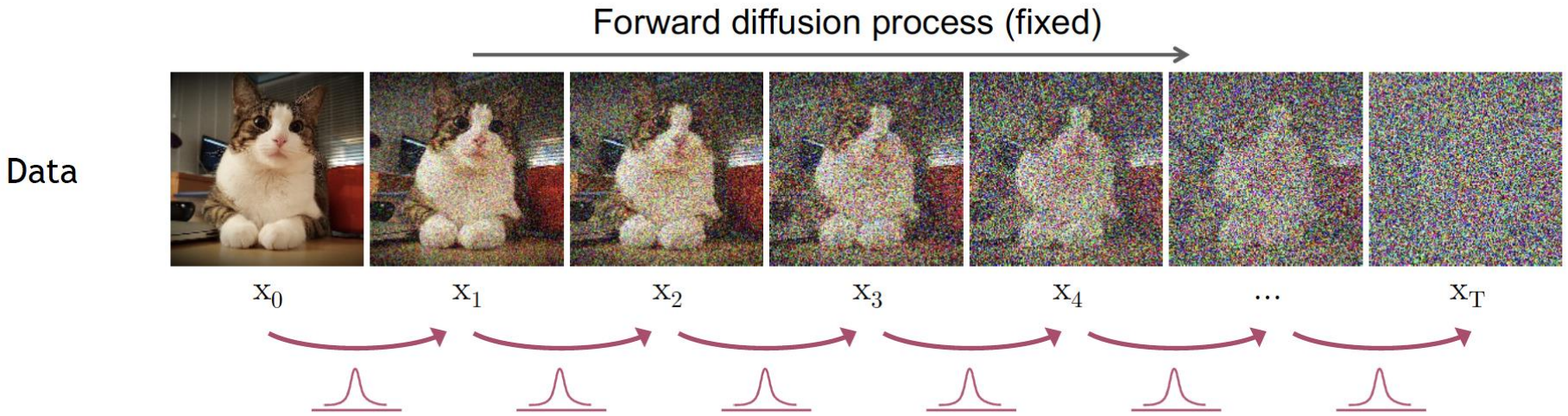


Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Forward Diffusion Process

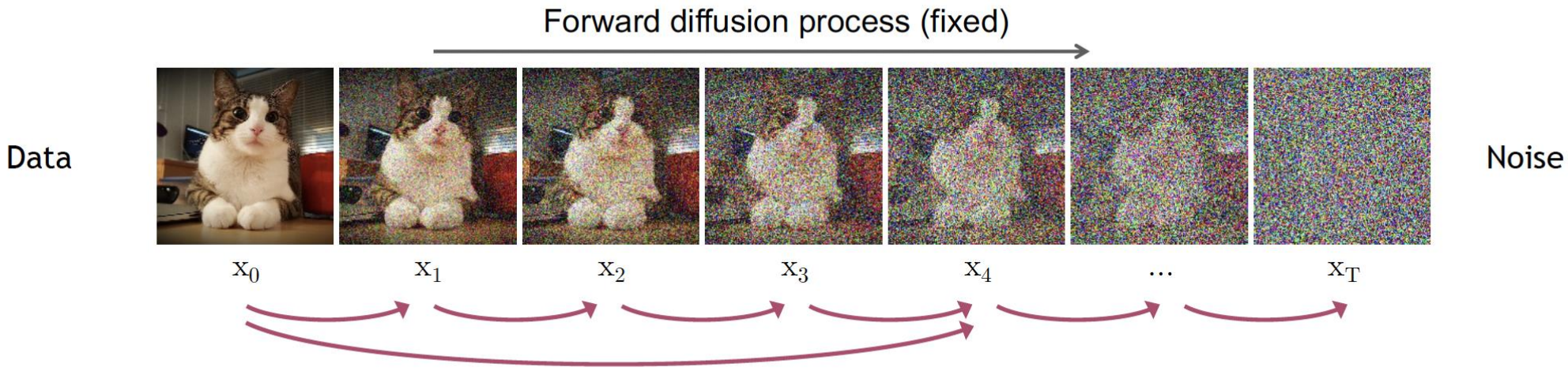
The formal definition of the forward process in T steps:



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Diffusion Kernel



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

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

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

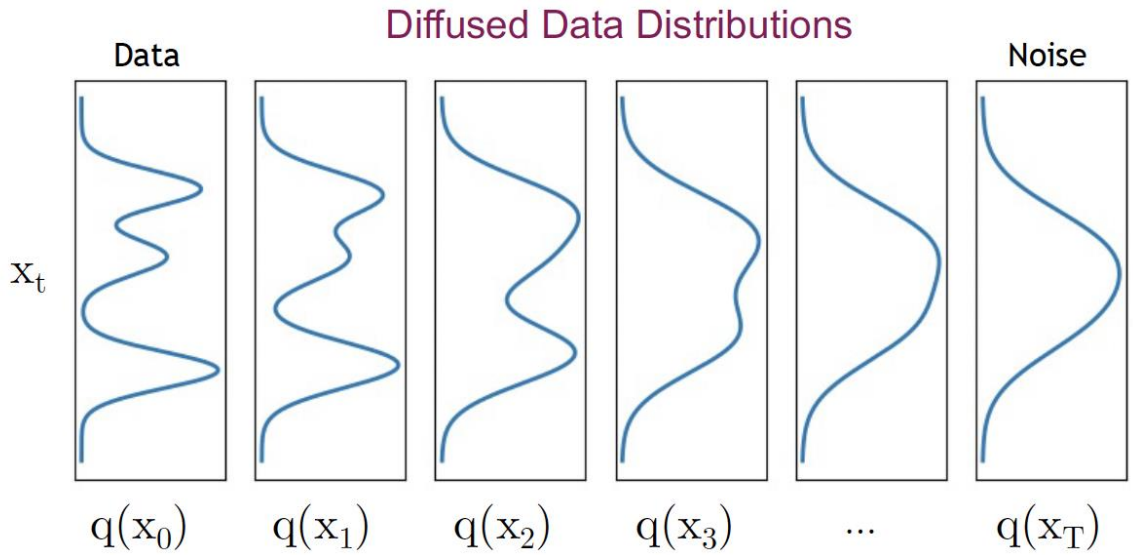
Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

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 $q(\mathbf{x}_t)$?

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

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).

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

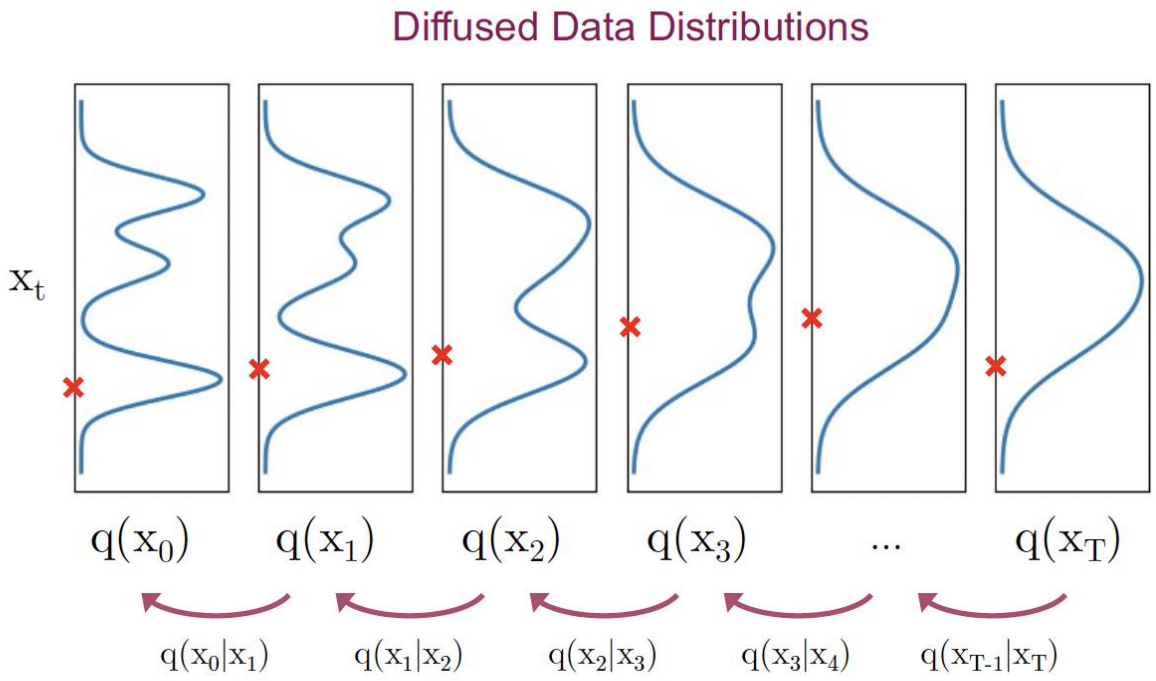
Generative Learning by Denoising

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



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.

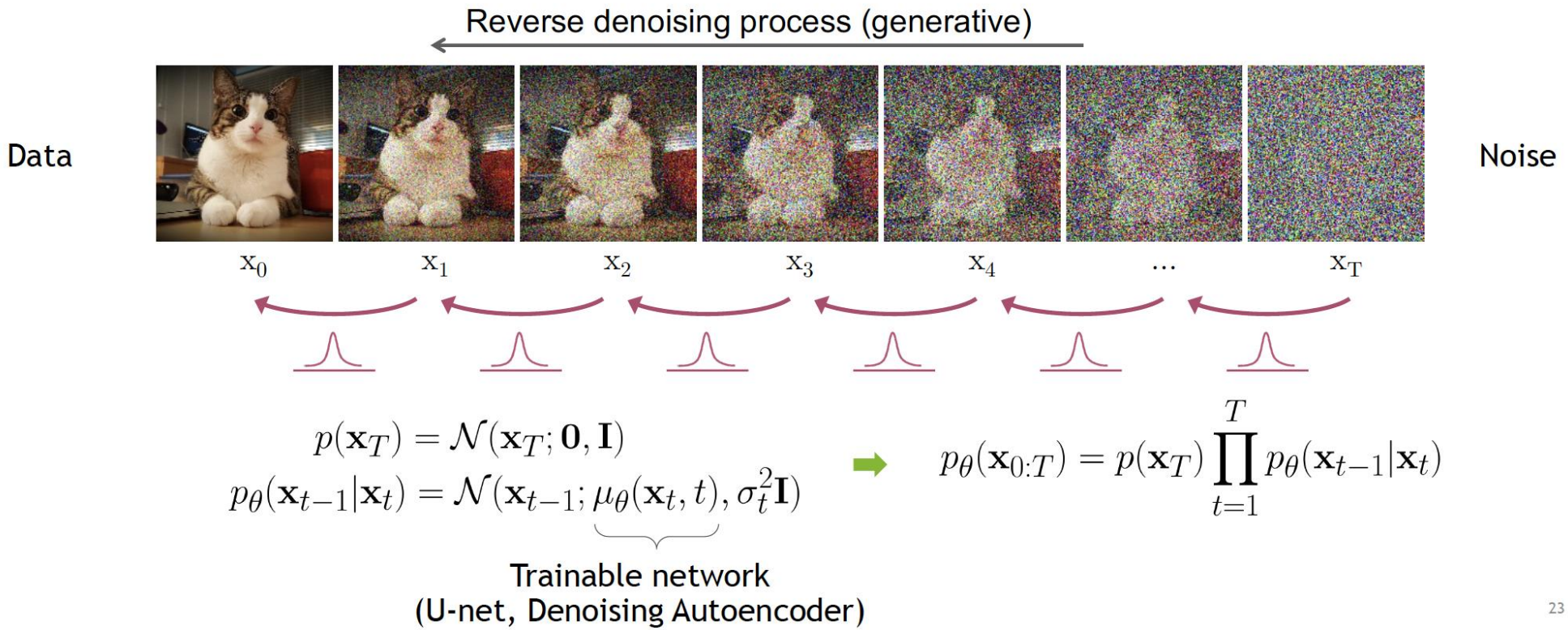
Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Normal distribution if β_t is small in each forward diffusion step.

22

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Learning Denoising Model

Variational upper bound

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

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

Sohl-Dickstein et al. ICML 2015 and Ho et al. NeurIPS 2020 show that:

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

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

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

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

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

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:

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

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

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

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Training Objective Weighting

Trading likelihood for perceptual quality

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

The time dependent λ_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:

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

For more advanced weighting see Choi et al., Perception Prioritized Training of Diffusion Models, CVPR 2022.

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Ho et al., “Denoising Diffusion Probabilistic Models”, 2020.

The Three Terms

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

3.1 Forward process and L_T

We ignore the fact that the forward process variances β_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 $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \sigma_1^2 \mathbf{I})$:

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

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

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

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: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

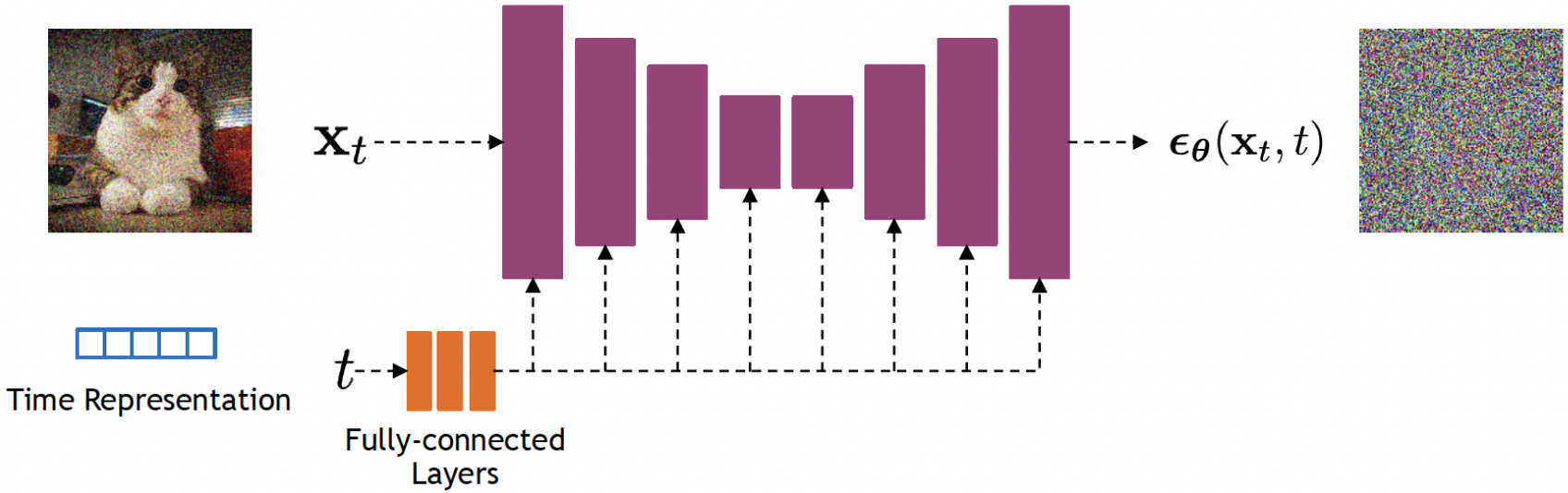
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,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

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

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 Dhariwal and Nichol NeurIPS 2021)

28

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Diffusion Parameters

Noise Schedule

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

Data

Noise

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

Above, β_t and σ_t^2 control the variance of the forward diffusion and reverse denoising processes respectively.

Often a linear schedule is used for β_t, and σ_t^2 is set equal to β_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 σ_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).

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

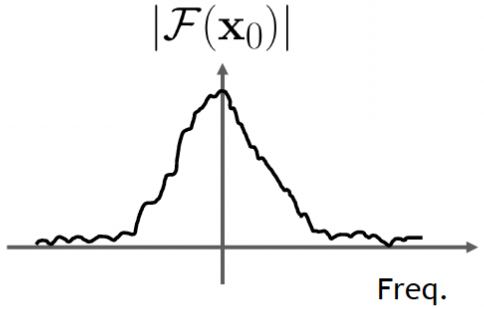
What happens to an image in the forward diffusion process?

Recall that sampling from $q(\mathbf{x}_t|\mathbf{x}_0)$ is done using $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

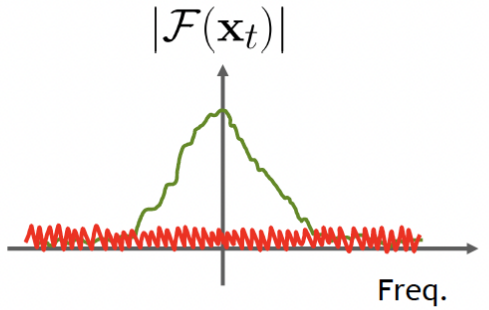
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$$

Fourier Transform

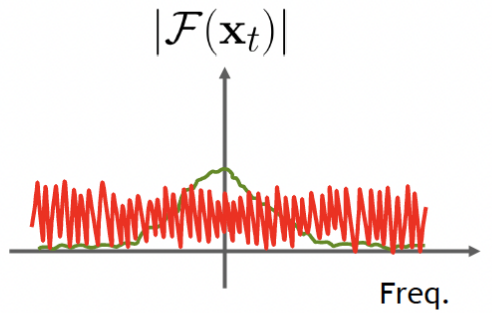
$$\mathcal{F}(\mathbf{x}_t) = \sqrt{\bar{\alpha}_t}\ \mathcal{F}(\mathbf{x}_0) + \sqrt{(1-\bar{\alpha}_t)}\ \mathcal{F}(\epsilon)$$



Small t
$\bar{\alpha}_t \sim 1$



Large t
$\bar{\alpha}_t \sim 0$

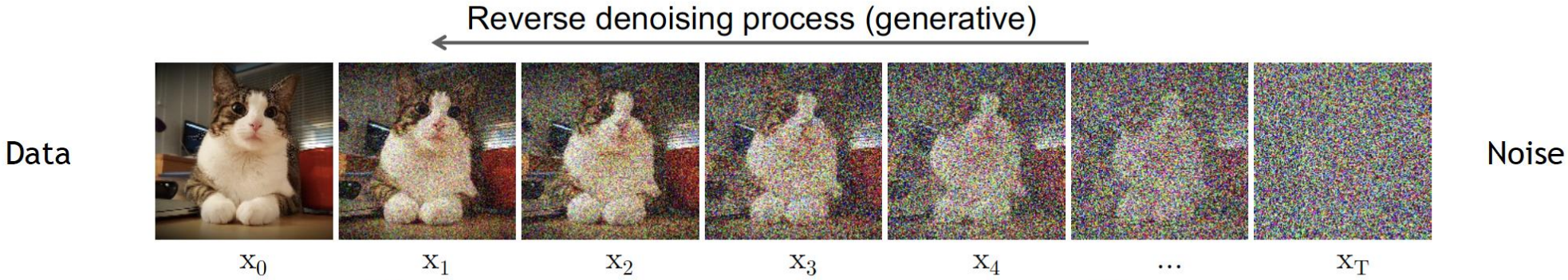


In the forward diffusion, the high frequency content is perturbed faster.

30

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Content-Detail Tradeoff



The weighting of the training objective for different timesteps is important!

31

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Latent Diffusion Models (Stable Diffusion)

Main differences:

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

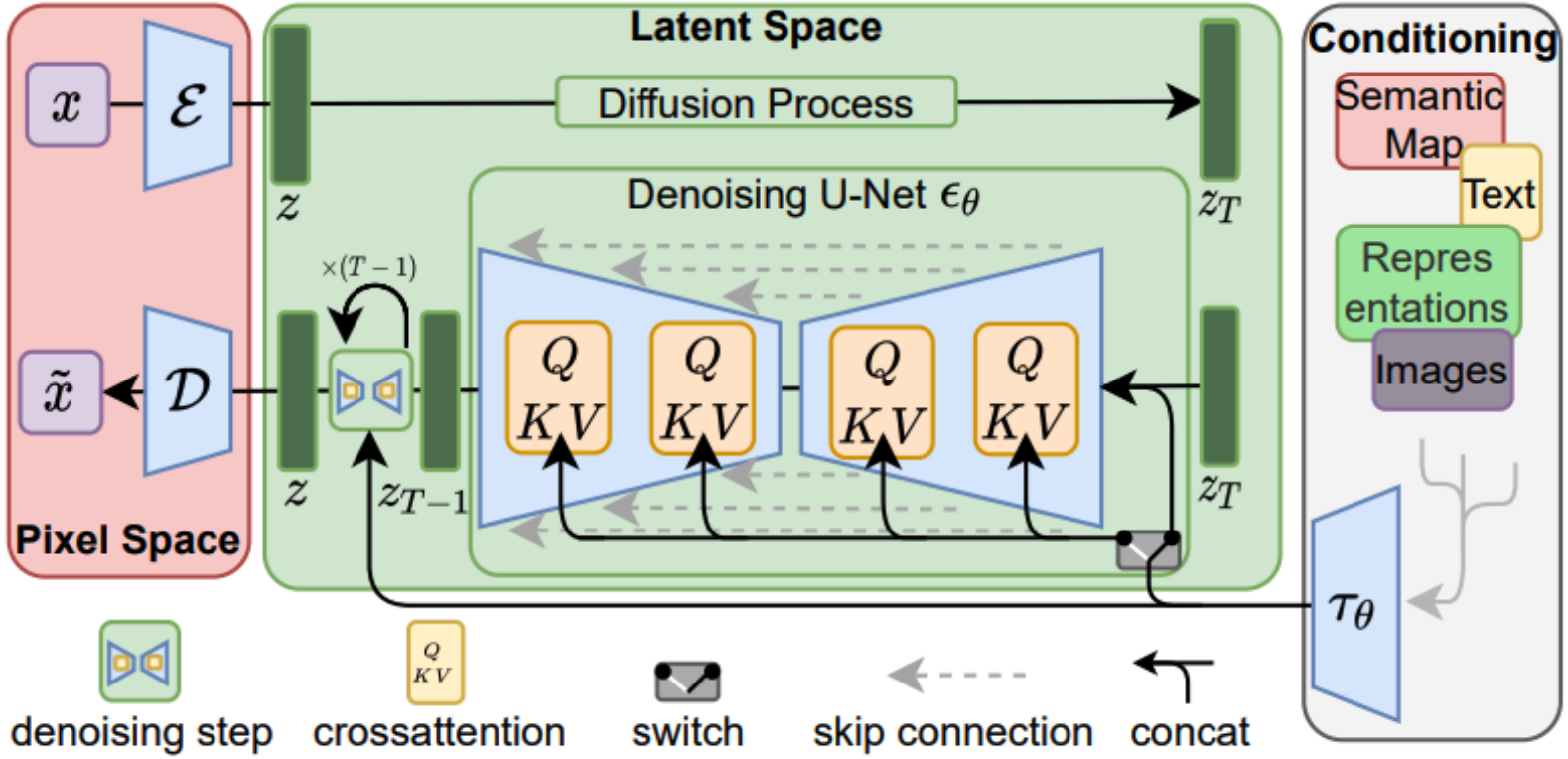


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3

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

Slide: https://deeplearning.cs.cmu.edu/F23/document/slides/lec23.diffusion.updated.pdf