Advanced DL Topics

Attention

Index the values via a differentiable operator.

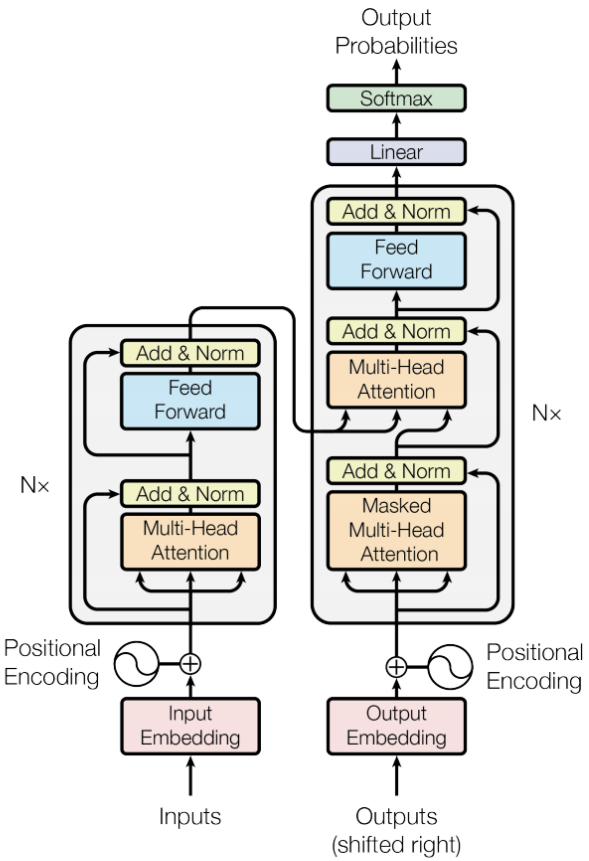
Multiply queries with keys

Get the values

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

To train them well, divide by $\sqrt{d_k}$, "probably" because for large values of the key's dimension, the dot product grows large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

Transformers



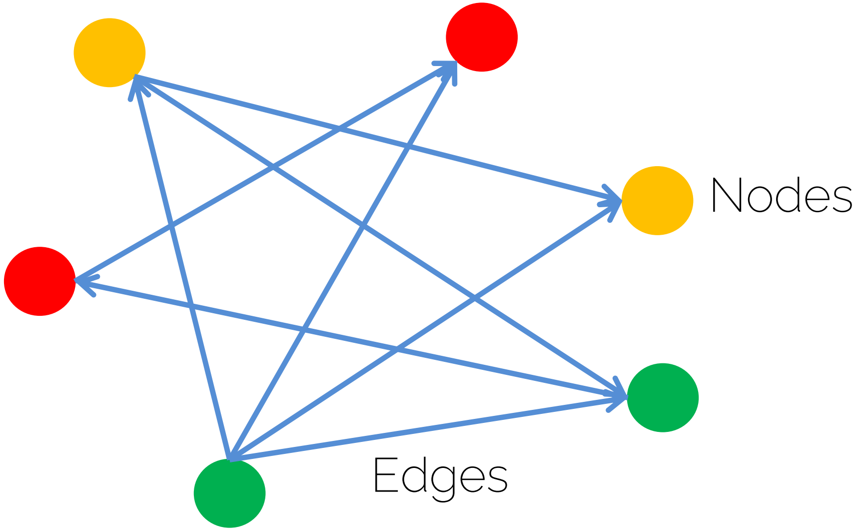
Attention Is All You Need [Vaswani et al. 17]

https://arxiv.org/pdf/1706.03762.pdf

Graph Neural Networks

A graph

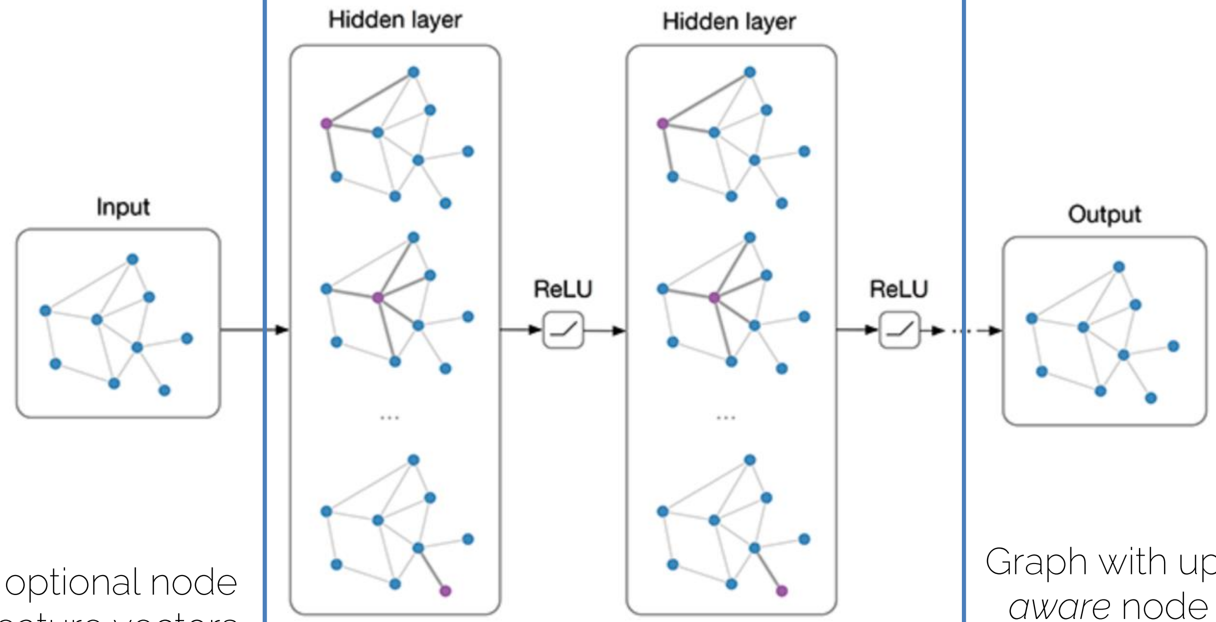
- Node: a concept
- Edge: a connection between concepts



Deep learning on graphs

- Generalizations of neural networks that can operate on graph-structured domains:
  - Scarselli et al. “The Graph Neural Network Model”, IEEE Trans. Neur. Net 2009.
  - Defferrard et al. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”, NeurIPS 2016
  - Kipf&Welling, “Semi-Supervised Classification with Graph Convolutional Networks”, ICLR 2017.
  - Gilmer et al. “Neural Message Passing for Quantum Chemistry”. ICML 2017
  - Koke&Cremers “HoloNets: Spectral Convolutions do extend to Directed Graphs”, ICLR 2024.
- Key challenges:
  - Variable sized inputs (number of nodes and edges)
  - Need **invariance to node permutations**

General Idea 1: Message passing



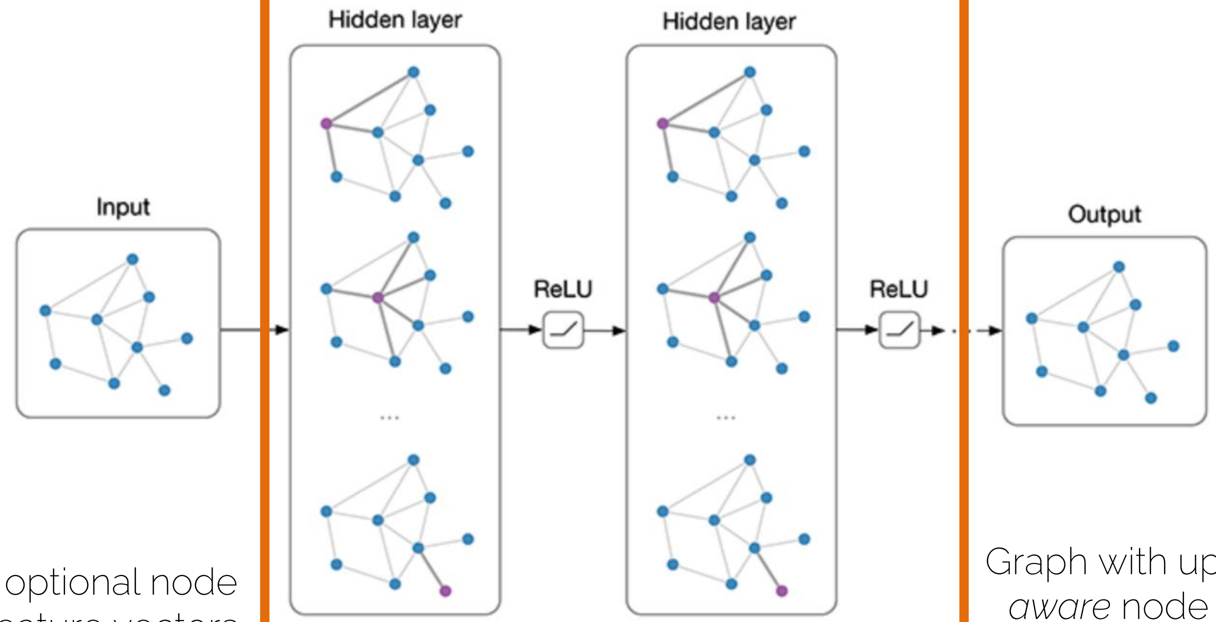
Graph with optional node and edge feature vectors

Information propagation across the graph for several iterations

Graph with updated *context-aware* node and (possibly edge) feature vector(s)

Figure credit: https://tkipf.github.io/graph-convolutional-networks/

General Idea 1: Message passing



Graph with optional node and edge feature vectors

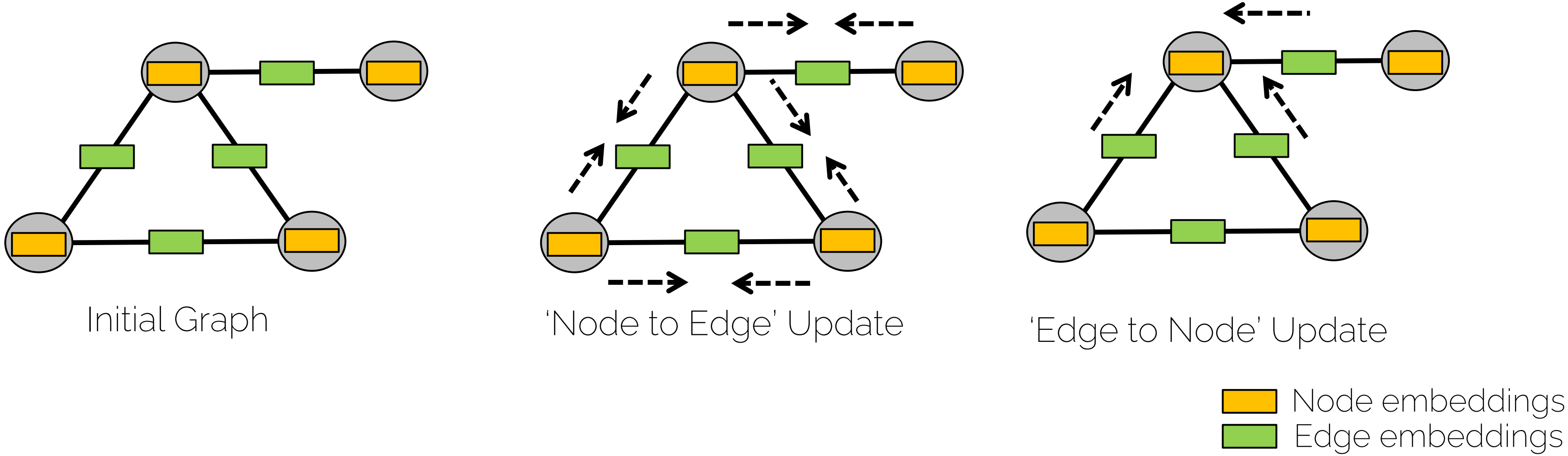
Information propagation across the graph for several iterations

Graph with updated *context-aware* node and (possibly edge) feature vector(s)

Figure credit: https://tkipf.github.io/graph-convolutional-networks/

Message Passing Networks

- We can divide the propagation process in two steps: ‘node to edge’ and ‘edge to node’ updates.



Battaglia et al. “Relational inductive biases, deep learning, and graph networks”. 2018

‘Node to edge’ updates

- At every message passing step l, first do:

$$h_{(i,j)}^{(l)} = \mathcal{N}_e \left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}] \right)$$

Embedding of node i in the previous message passing step

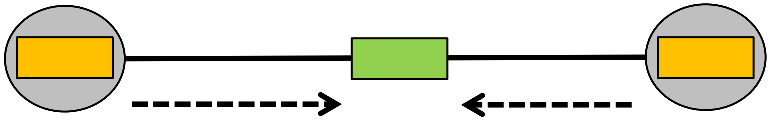
Embedding of edge (i,j) in the previous message passing step

Embedding of node j in the previous message passing step

'Node to edge' updates

- At every message passing step l, first do:

$$h_{(i,j)}^{(l)} = \mathcal{N}_e\left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}]\right)$$

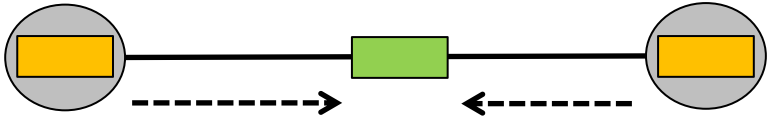


‘Node to edge’ updates

- At every message passing step l, first do:

$$h_{(i,j)}^{(l)} = \underbrace{\mathcal{N}_e}\left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}]\right)$$

Learnable function (e.g. MLP) with shared weights across the entire graph



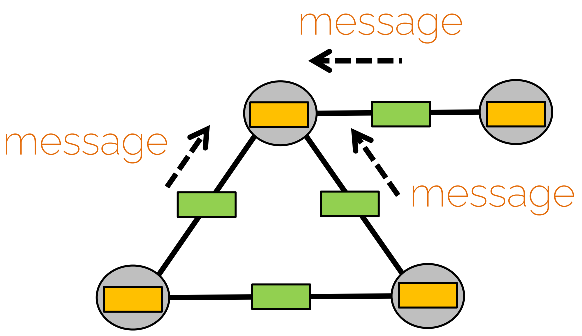
‘Edge to node’ updates

- After a round of edge updates, each edge embedding contains information about its pair of incident nodes
- Then, edge embeddings are used to update nodes:

$$m_i^{(l)} = \Phi\left(\left\{h_{(i,j)}^{(l)}\right\}_{j \in N_i}\right)$$

Order invariant operation (e.g. sum, mean, max)

Neighbors of node i



'Edge to node' updates

- After a round of edge updates, each edge embedding contains information about its pair of incident nodes
- Then, edge embeddings are used to update nodes:

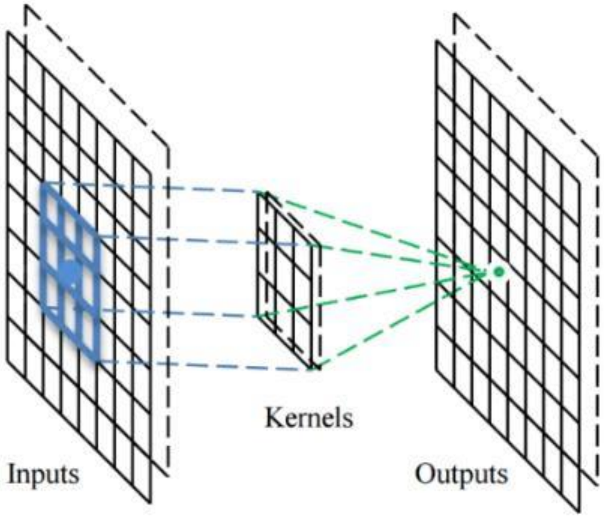
$$m_i^{(l)} = \Phi\left(\left\{h_{(i,j)}^{(l)}\right\}_{j \in N_i}\right)$$

$$h_i^{(l)} = \underbrace{\mathcal{N}_v}_{}\left(\left[m_i^{(l)}, h_i^{(l-1)}\right]\right)$$

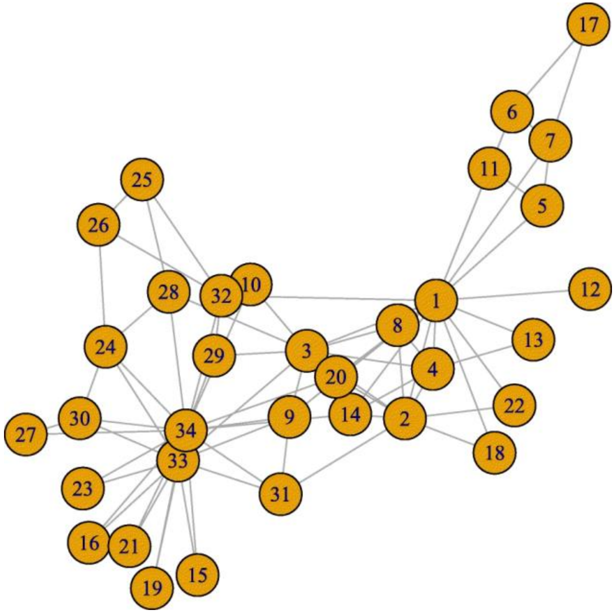
Learnable function (e.g. MLP) with shared weights across the entire graph

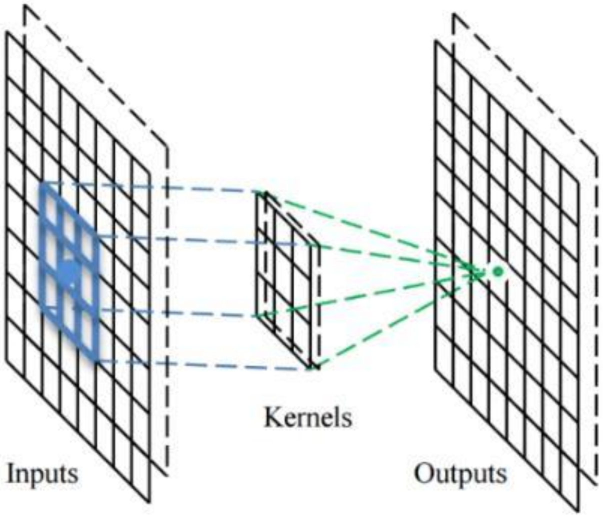
The aggregation provides each node embedding with contextual information about its neighbors

General Idea 2: Spectral Approach

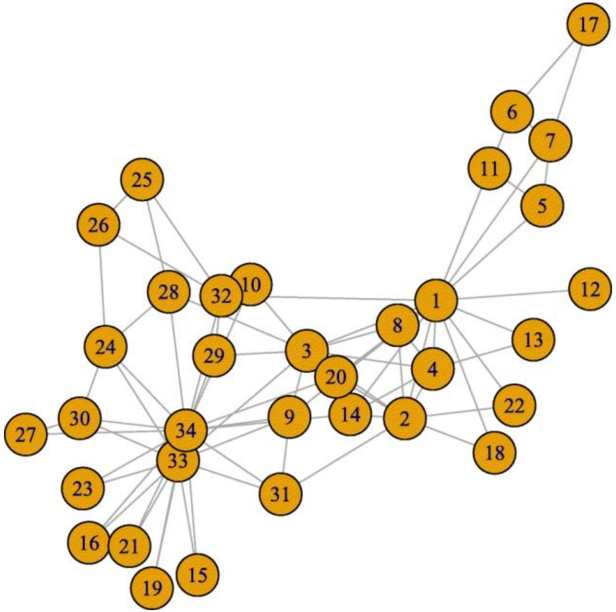


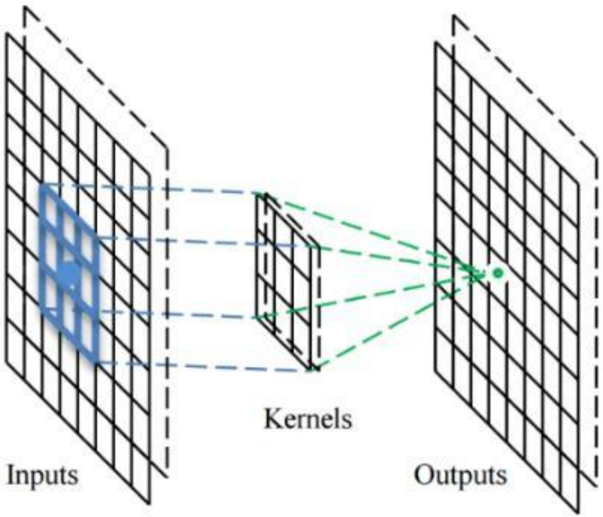
How to extend Convolutions to Graphs?





How to extend Convolutions to Graphs?





Sliding Window Interpretation

How to extend Convolutions to Graphs?

Spectral Approach!

How to extend Convolutions to Graphs?

Spectral Approach!

$$\boxed{\begin{array}{c}\text{Convolution}\\ =\\ \text{Multiplication in Fourier Domain}\end{array}}$$

How to extend Convolutions to Graphs?

Spectral Approach!

$$\boxed{\begin{array}{c}\text{Convolution}\\ =\\ \text{Multiplication in Fourier Domain}\end{array}}$$

$$\text{Conv. Filter} \sim \mathscr{F}^{-1}\left[g(k) \cdot \mathscr{F} f\right]$$

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
=
Multiplication in Fourier Domain

⇩

Fourier Transform
=
Projection onto $\{e^{ikx}\}_k$

$$[\mathscr{F} f](k) = \int e^{ikx} f(x) dx$$

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
=
Multiplication in Fourier Domain

⇓

Fourier Transform
=
Projection onto $\{e^{ikx}\}_k$

⇓

$\{e^{ikx}\}_k$ are
eigenfunctions of Δ

$$\Delta e^{ikx} = (-k^2) \cdot e^{ikx}$$

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
$=$
Multiplication in Fourier Domain

$\Downarrow$

Fourier Transform
$=$
Projection onto $\{e^{ikx}\}_k$

$\Downarrow$

$\{e^{ikx}\}_k$ are
eigenfunctions of Δ

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
=
Multiplication in Fourier Domain

⇓

Fourier Transform
=
Projection onto $\{e^{ikx}\}_k$

⇓

$\{e^{ikx}\}_k$ are
eigenfunctions of Δ

Take $\{\phi_k\}_k$
eigenvectors of Δ

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
$=$
Multiplication in Fourier Domain

$\Downarrow$

Fourier Transform
$=$
Projection onto $\{e^{ikx}\}_k$

$\Downarrow$

$\{e^{ikx}\}_k$ are
eigenfunctions of Δ

Take $\{\phi_k\}_k$
eigenvectors of Δ

$\Uparrow$

Graph Fourier Transform
$:=$
Projection onto $\{\phi_k\}_k$

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
$=$
Multiplication in Fourier Domain

Fourier Transform
$=$
Projection onto $\{e^{ikx}\}_k$

$\{e^{ikx}\}_k$ are
eigenfunctions of Δ

Graph Convolution
$:=$
Mult. in Graph Fourier Domain

Graph Fourier Transform
$:=$
Projection onto $\{\phi_k\}_k$

Take $\{\phi_k\}_k$
eigenvectors of Δ

How to extend Convolutions to Graphs?

Spectral Approach!

Convolution
=
Multiplication in Fourier Domain

Fourier Transform
=
Projection onto $\{e^{ikx}\}_k$

$\{e^{ikx}\}_k$ are
eigenfunctions of Δ

Graph Convolution
:=
Mult. in Graph Fourier Domain

Graph Fourier Transform
:=
Projection onto $\{\phi_k\}_k$

Take $\{\phi_k\}_k$
eigenvectors of Δ

Only applies to undirected graphs!

General Idea 3: Holomorophic Functional Calculus

- Key idea: Extend spectral convolutions to graphs by making use of
  - complex analysis
  - holomorphic functions & the Cauchy integral formula
  - tools from spectral theory

[Koke & Cremers "HoloNets: Spectral Convolutions do extend to Directed Graphs", ICLR 2024]

General Idea 3: Holomorophic Functional Calculus

Table 1: Results on real-world directed heterophilic datasets. OOM indicates out of memory.

Homophily	Squirrel 0.223	Chameleon 0.235	Arxiv-year 0.221	Snap-patents 0.218	Roman-Empire 0.05
MLP	28.77 ± 1.56	46.21 ± 2.99	36.70 ± 0.21	31.34 ± 0.05	64.94 ± 0.62
GCN	53.43 ± 2.01	64.82 ± 2.24	46.02 ± 0.26	51.02 ± 0.06	73.69 ± 0.74
H_2GCN	37.90 ± 2.02	59.39 ± 1.98	49.09 ± 0.10	OOM	60.11 ± 0.52
GPR-GNN	54.35 ± 0.87	62.85 ± 2.90	45.07 ± 0.21	40.19 ± 0.03	64.85 ± 0.27
LINKX	61.81 ± 1.80	68.42 ± 1.38	56.00 ± 0.17	61.95 ± 0.12	37.55 ± 0.36
FSGNN	74.10 ± 1.89	78.27 ± 1.28	50.47 ± 0.21	65.07 ± 0.03	79.92 ± 0.56
ACM-GCN	67.40 ± 2.21	74.76 ± 2.20	47.37 ± 0.59	55.14 ± 0.16	69.66 ± 0.62
GloGNN	57.88 ± 1.76	71.21 ± 1.84	54.79 ± 0.25	62.09 ± 0.27	59.63 ± 0.69
Grad. Gating	64.26 ± 2.38	71.40 ± 2.38	63.30 ± 1.84	69.50 ± 0.39	82.16 ± 0.78
DiGCN	37.74 ± 1.54	52.24 ± 3.65	OOM	OOM	52.71 ± 0.32
MagNet	39.01 ± 1.93	58.22 ± 2.87	60.29 ± 0.27	OOM	88.07 ± 0.27
DirGNN	75.13 ± 1.95	79.74 ± 1.40	63.97 ± 0.30	73.95 ± 0.05	91.3 ± 0.46
FaberNet	$\mathbf{76.71 \pm 1.92}$	$\mathbf{80.33 \pm 1.19}$	$\mathbf{64.62 \pm 1.01}$	$\mathbf{75.10 \pm 0.03}$	$\mathbf{92.24 \pm 0.43}$

[Koke & Cremers "HoloNets: Spectral Convolutions do extend to Directed Graphs", ICLR 2024]

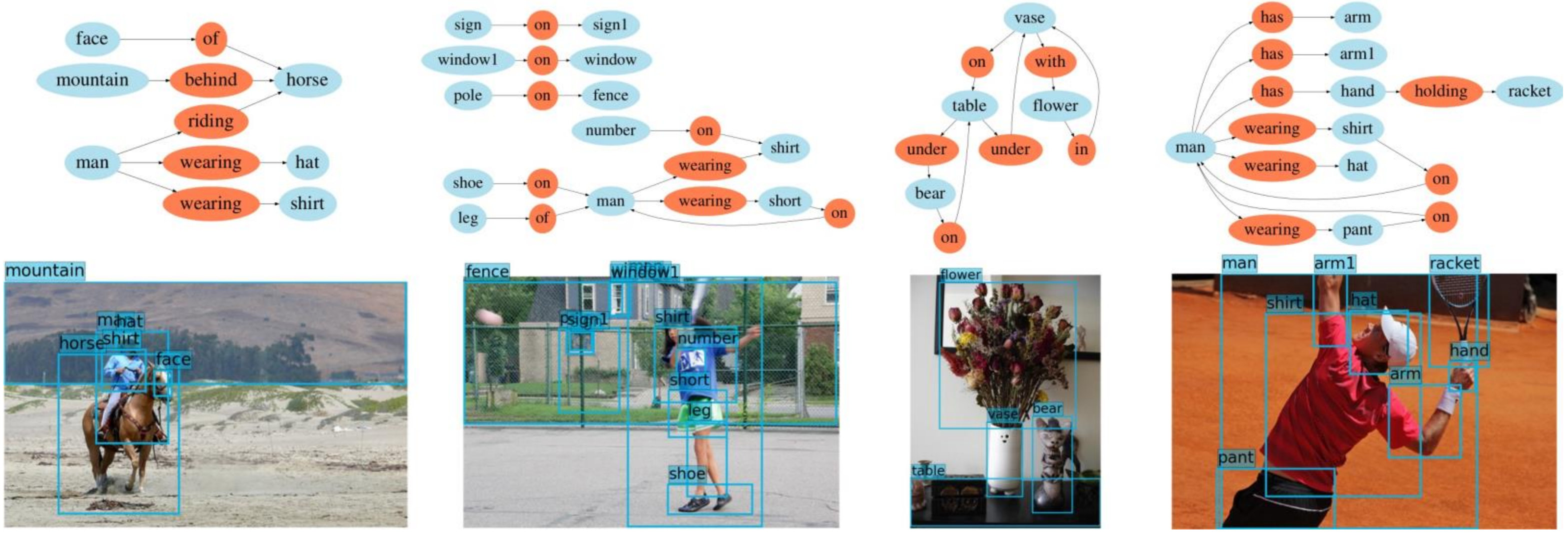
GNN Applications

- Node or edge classification
  - identifying anomalies such as spam, fraud
  - Relationship discovery for social networks, search networks

https://gm-neurips-2020.github.io/master-deck.pdf

GNN Applications

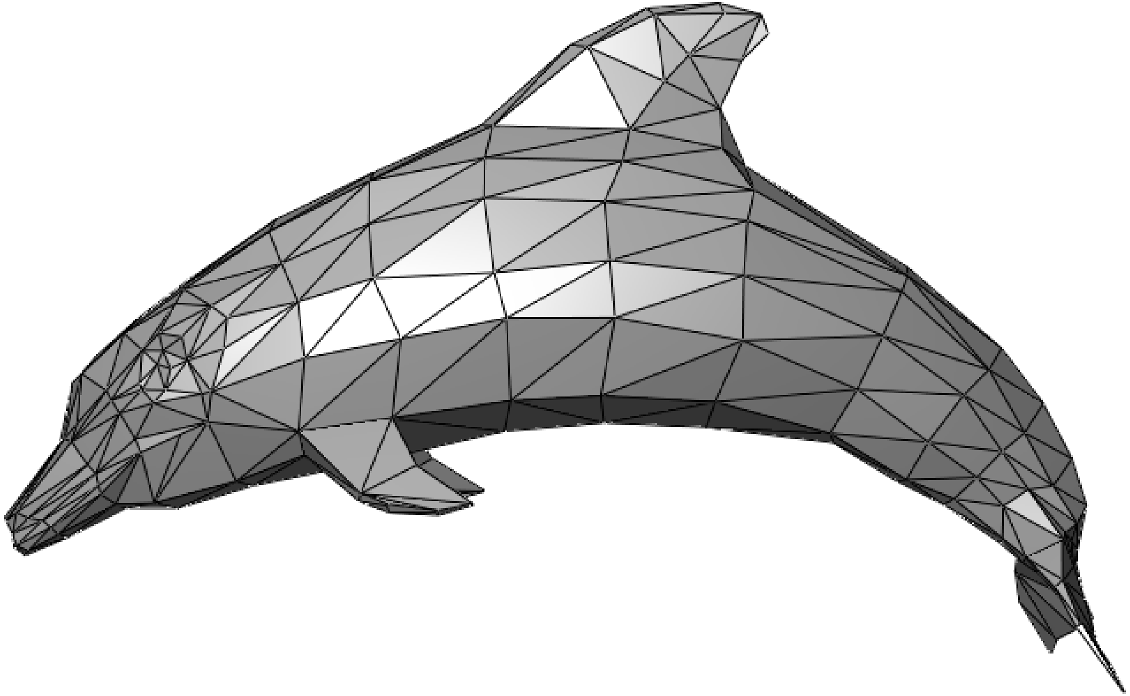
- Scene graph generation



[Xu et al. ‘17] Scene Graph Generation by Iterative Message Passing

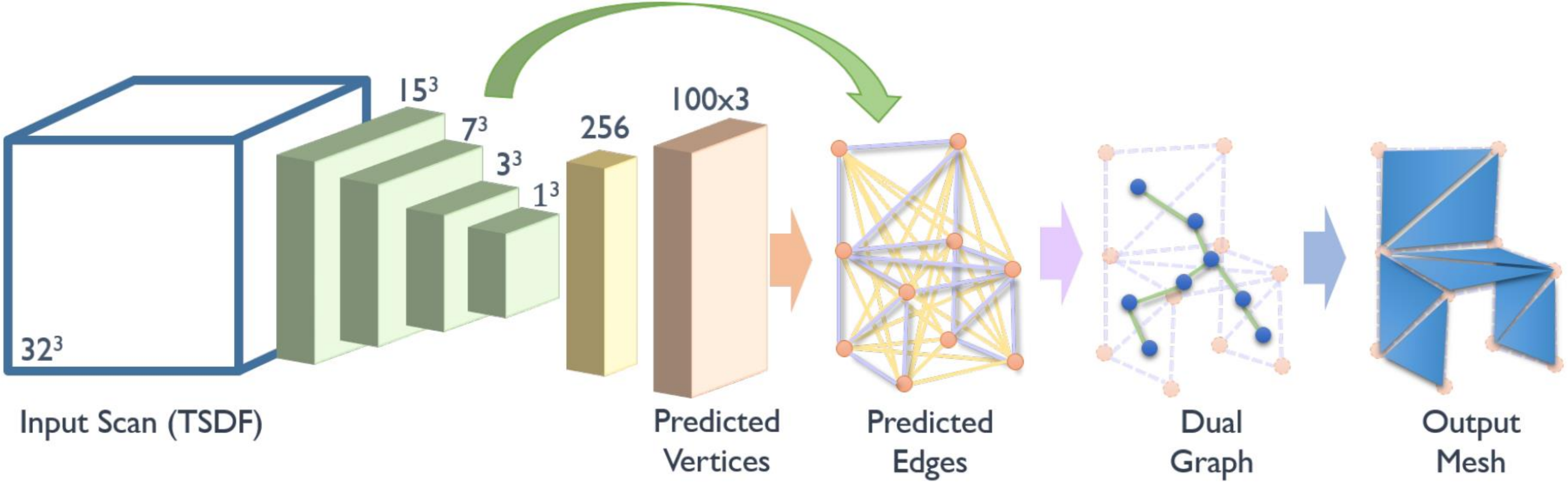
GNN Applications

- 3D Mesh Classification



GNN Applications

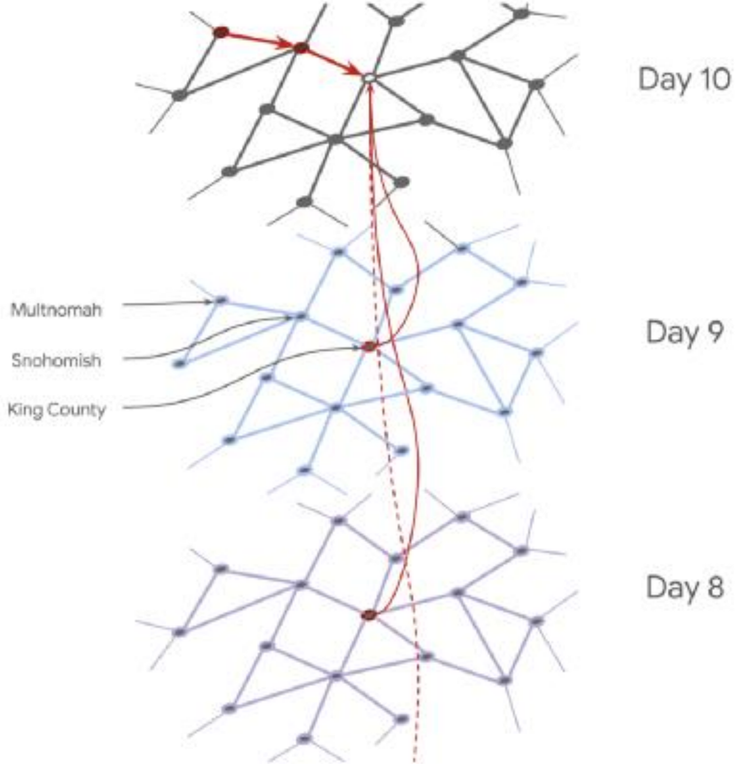
- 3D mesh generation



[Dai and Niessner, "Scan2Mesh: From Unstructured Range Scans to 3D Meshes", CVPR 2019]

GNN Applications

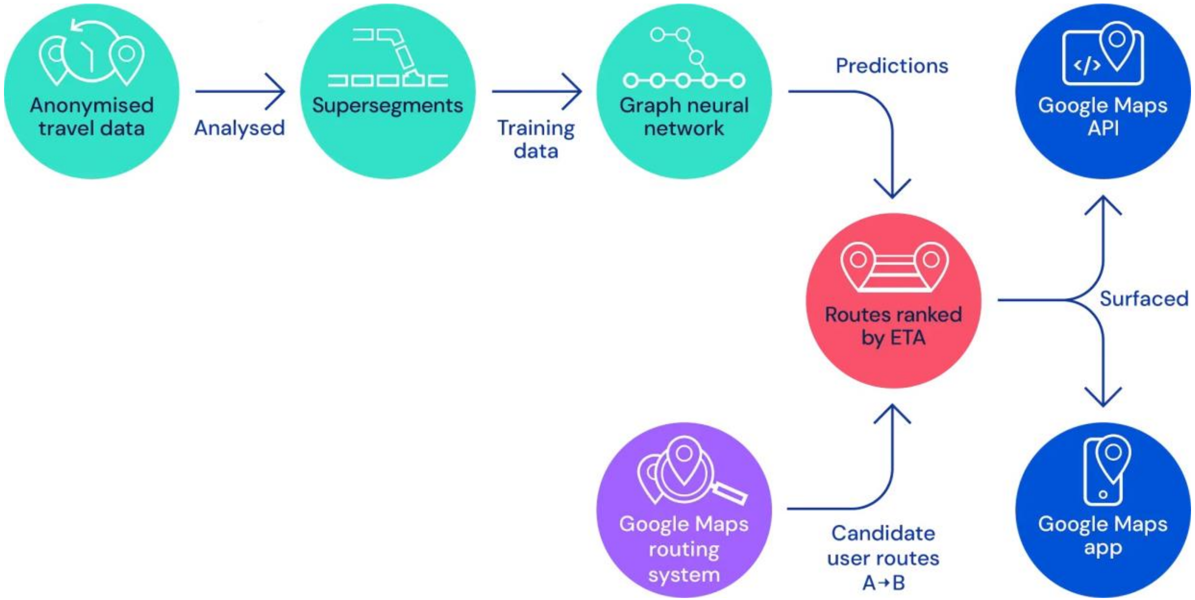
- Modeling epidemiology
  - Spatio-temporal graph



https://gm-neurips-2020.github.io/master-deck.pdf

GNN Applications

- Traffic forecasting

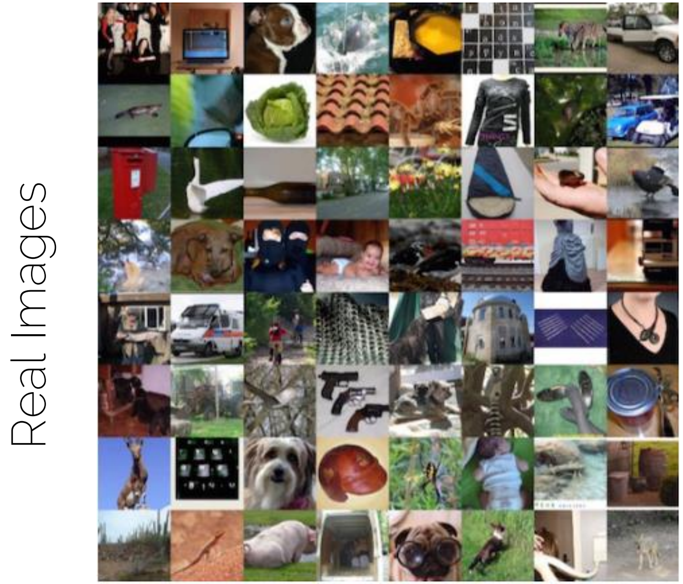


https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks

Generative Models

Generative Models

- Given training data, how to generate new samples from the same distribution



Source: https://openai.com/blog/generative-models/

Generative Models

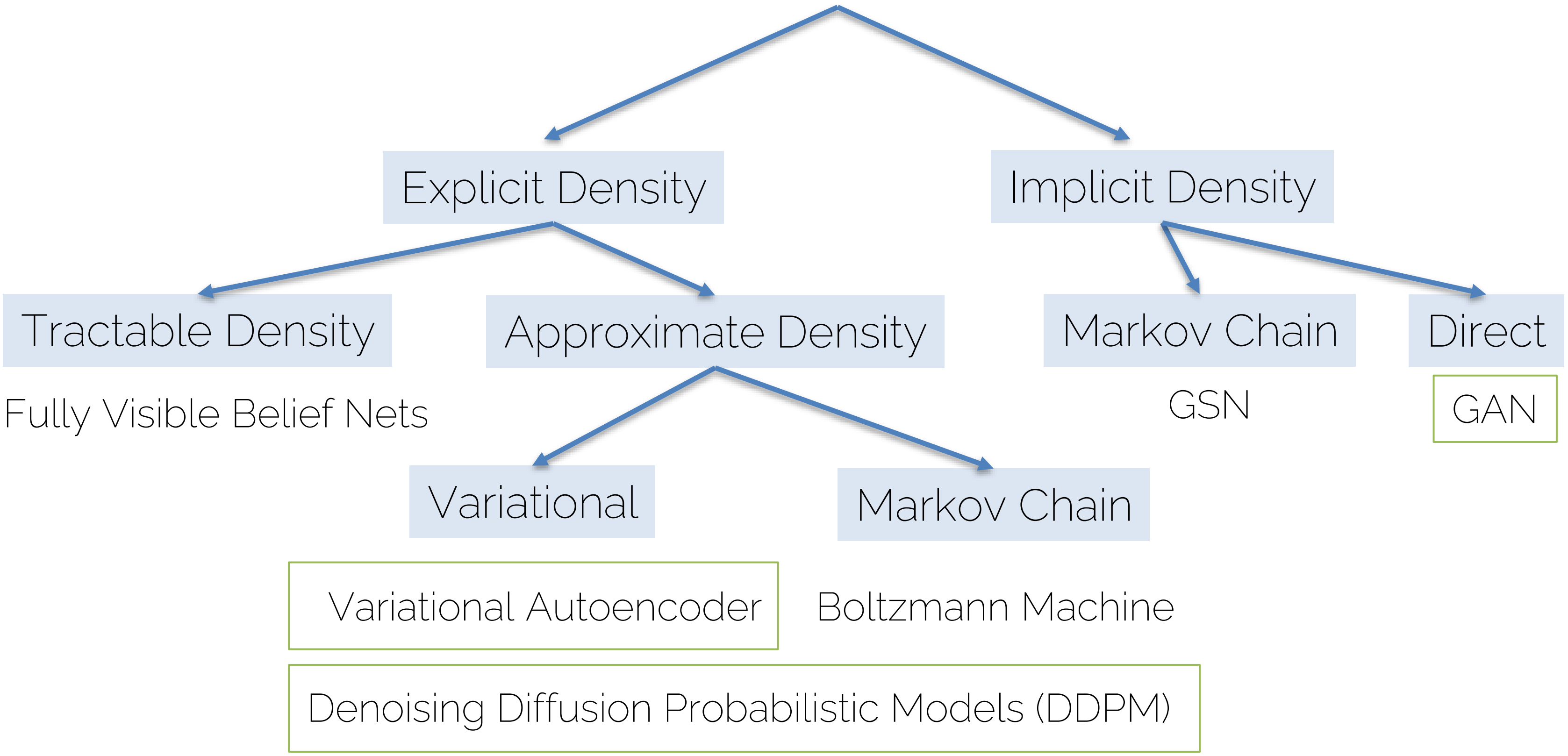


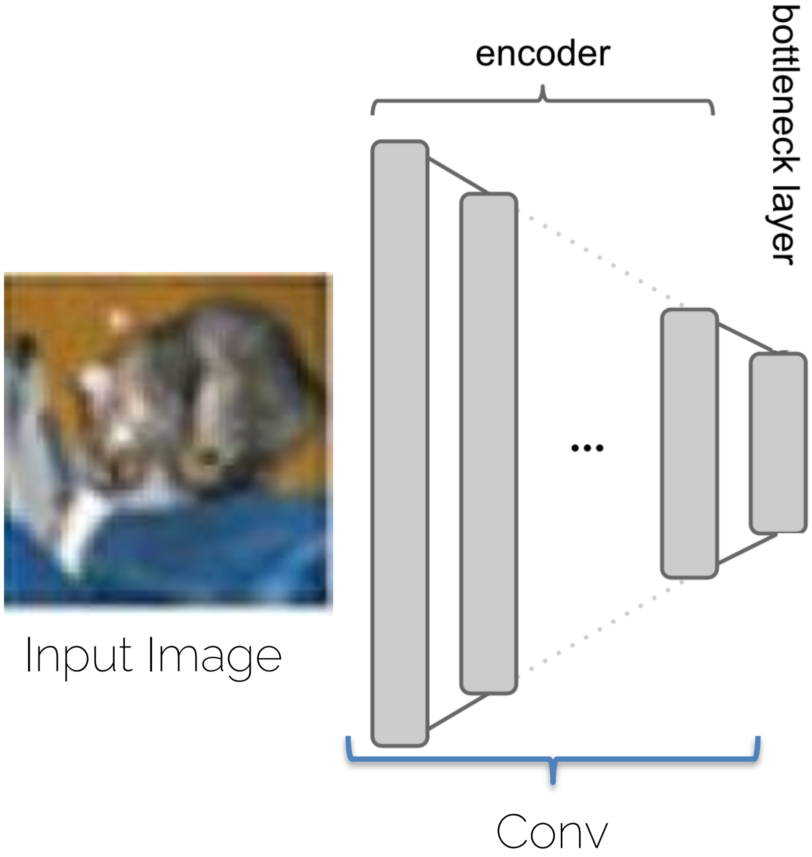
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Autoencoders

Autoencoders

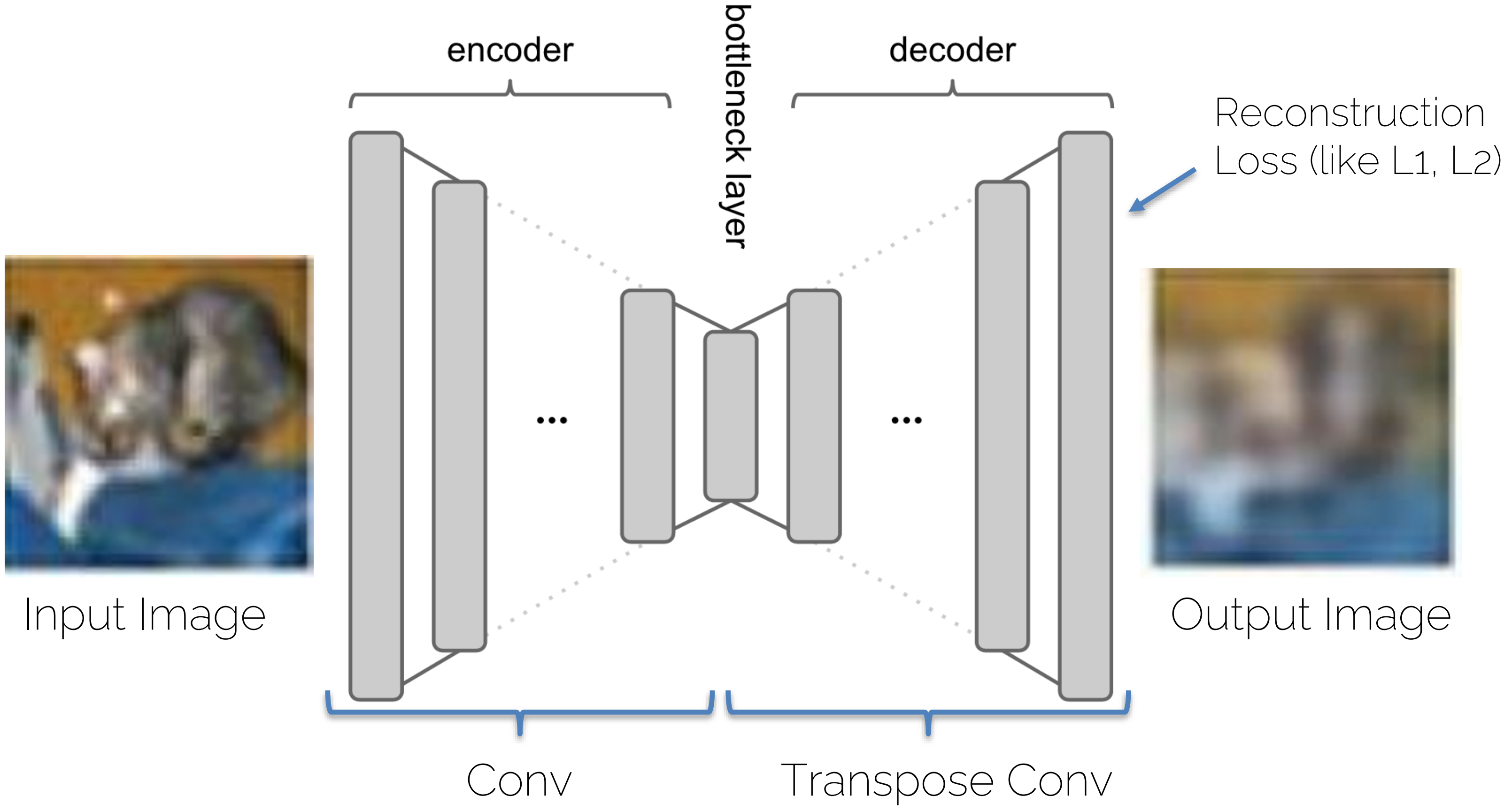
- Can be used as a basic generative models
- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

Autoencoders

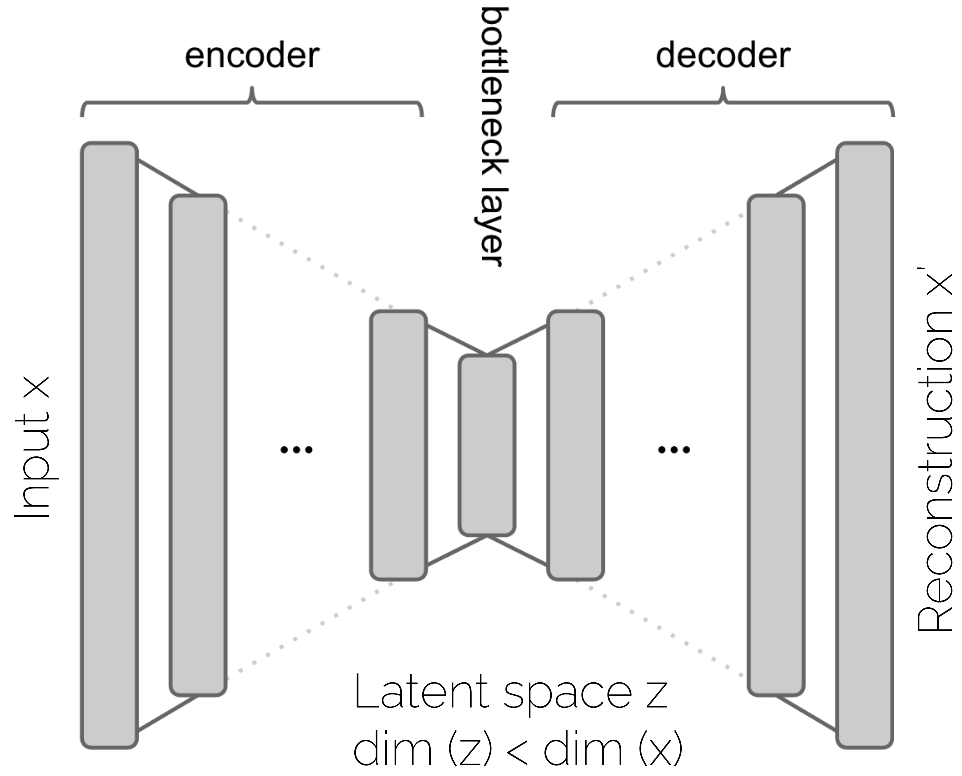


- From an input image to a feature representation (bottleneck layer)
- Encoder: a CNN in our case

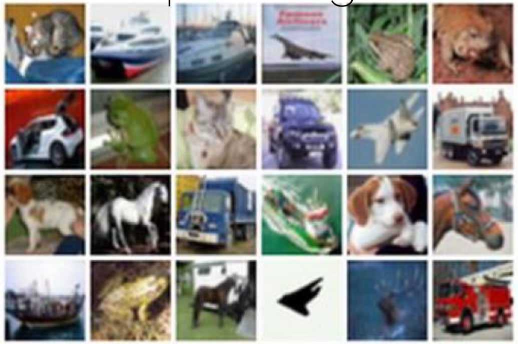
Autoencoder: training



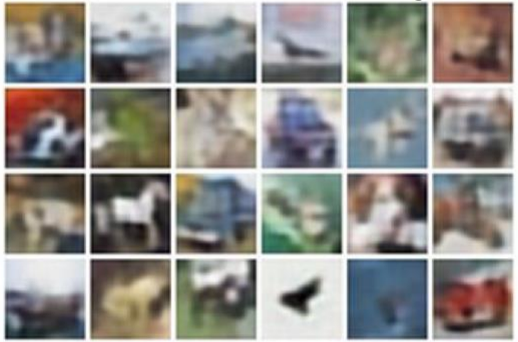
Autoencoder: training



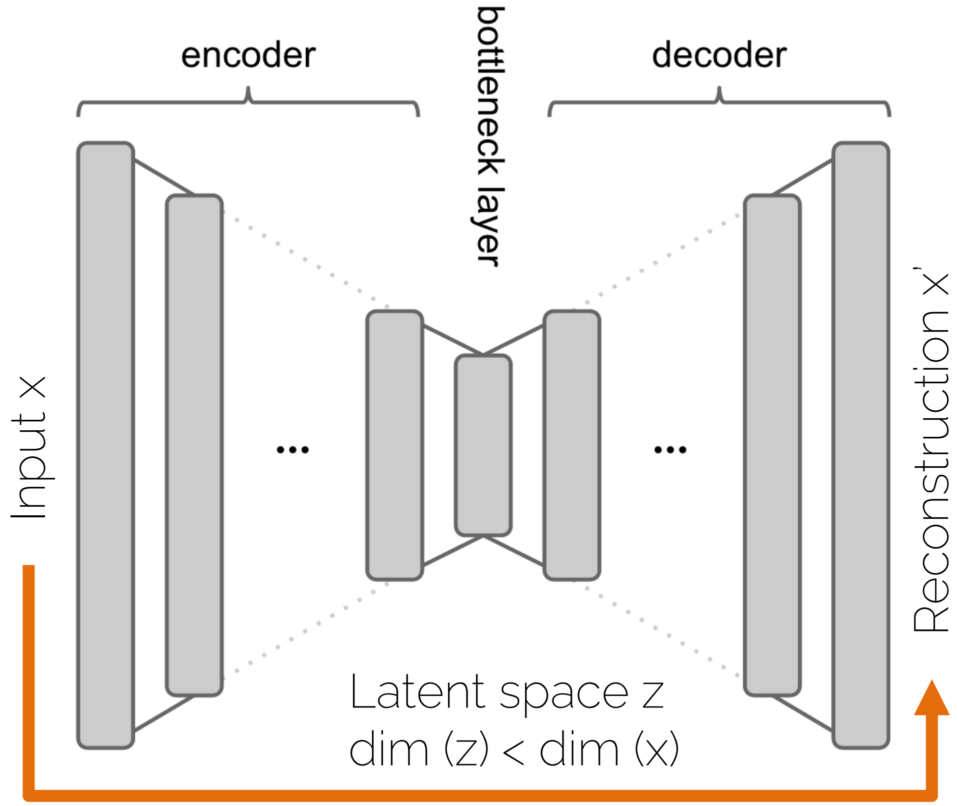
Input images



Reconstructed images



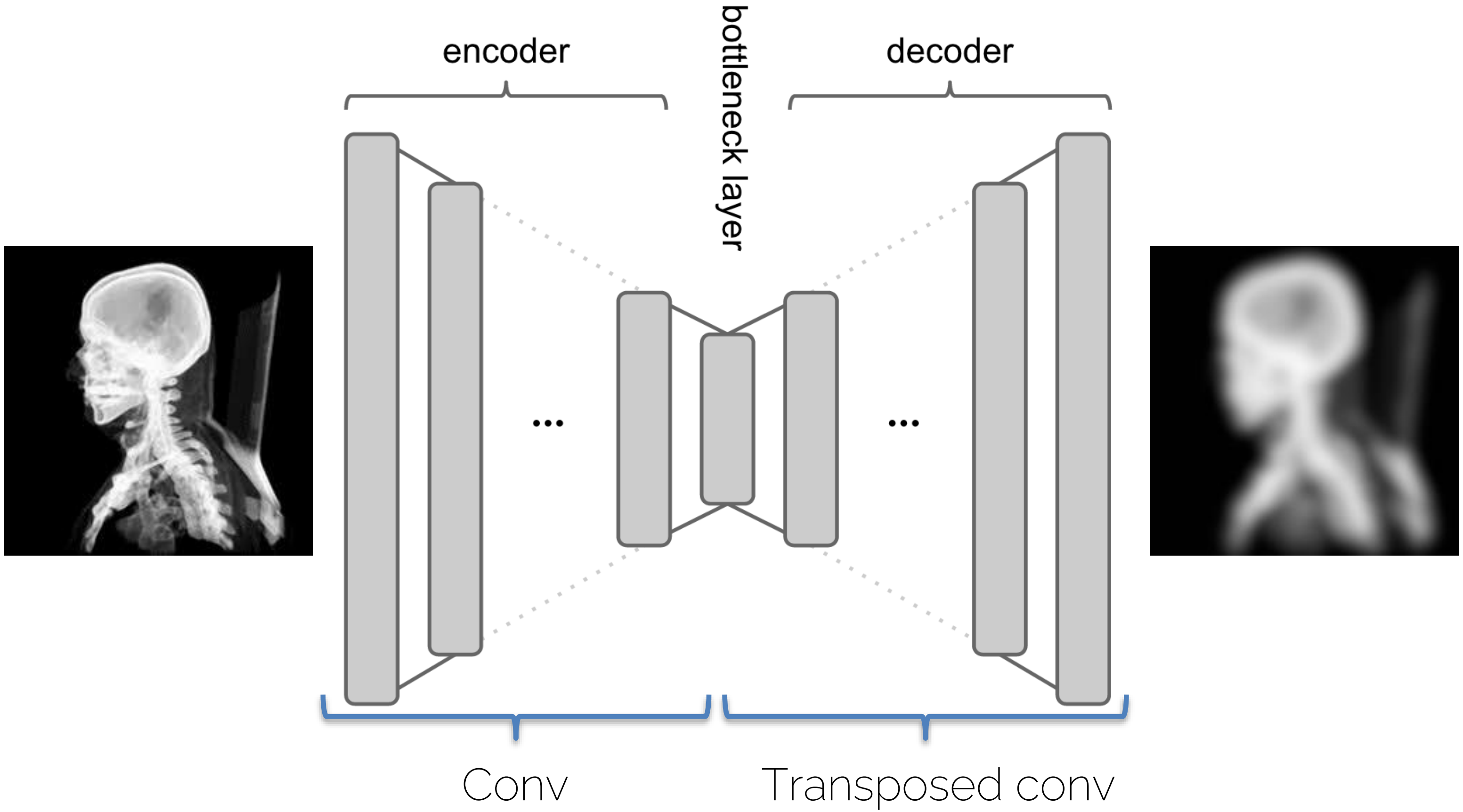
Autoencoder: training



- No labels required

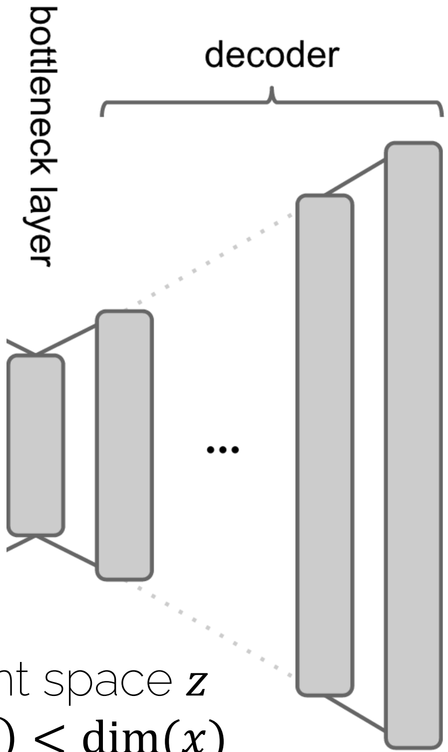
- We can use unlabeled data to first get its structure

Autoencoder



Decoder as Generative Model

Test time:
-> reconstruction from 'random' vector



Latent space z
$\dim(z) < \dim(x)$

Reconstruction Loss (often L2)

Output Image

Why using autoencoders?

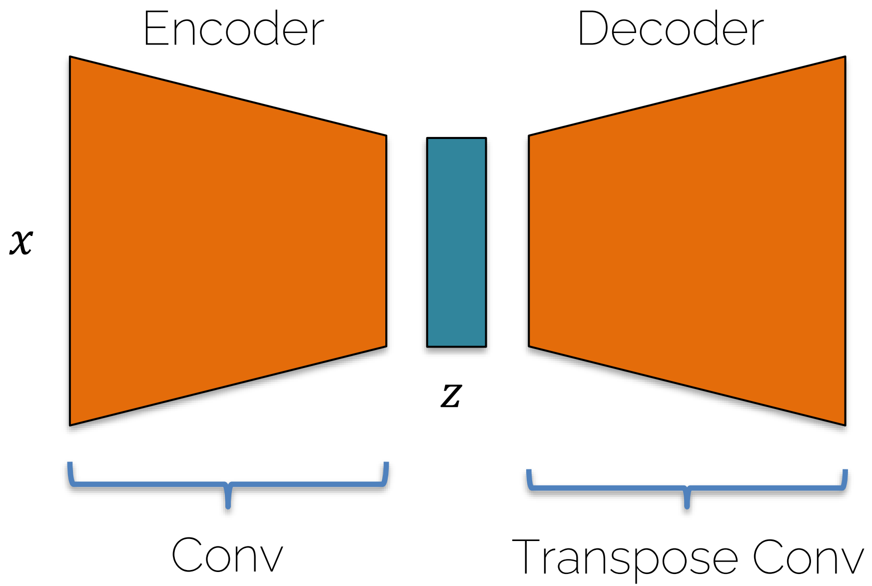
- Use 1: pre-training, as mentioned before
  - Image → same image reconstructed
  - Use the encoder as "feature extractor"

- Use 2: Use them to get pixel-wise predictions
  - Image → semantic segmentation
  - Low-resolution image → High-resolution image
  - Image → Depth map

Variational Autoencoders

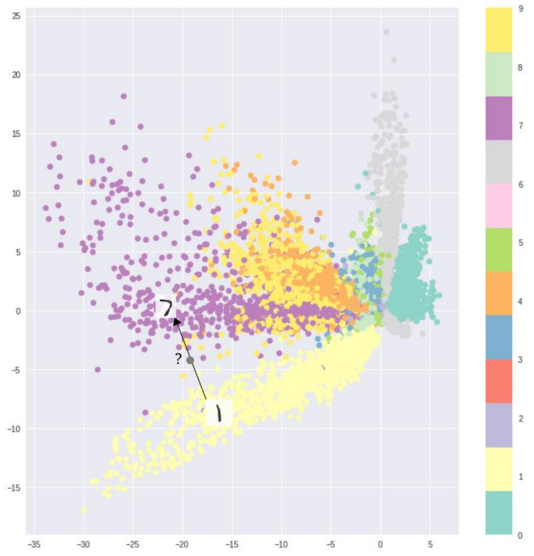
Autoencoders

- Encode the input into a representation (bottleneck) and reconstruct it with the decoder



Autoencoders

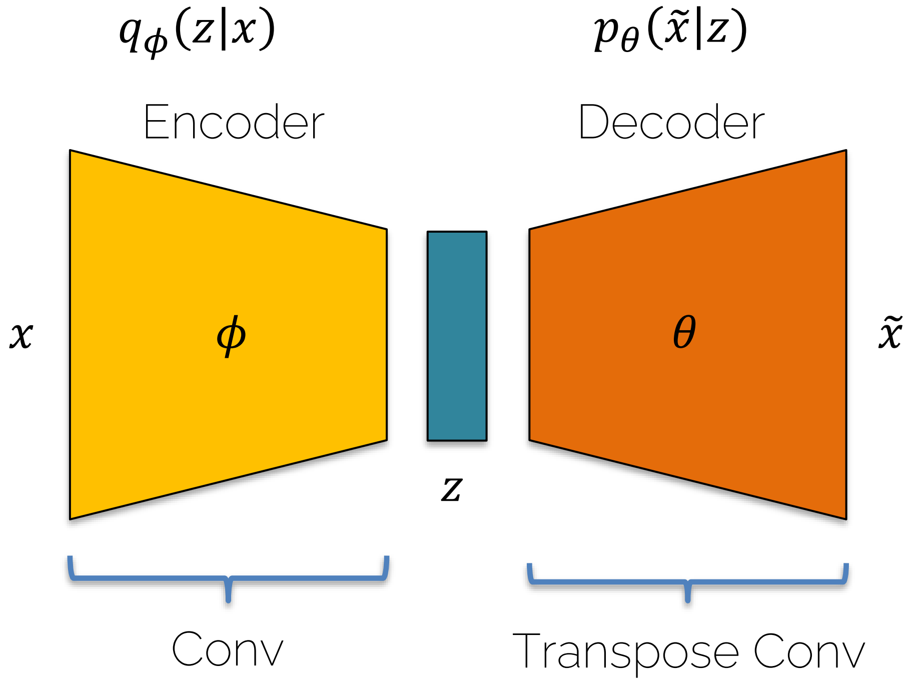
- Encode the input into a representation (bottleneck) and reconstruct it with the decoder



Latent space learned
by autoencoder on MNIST

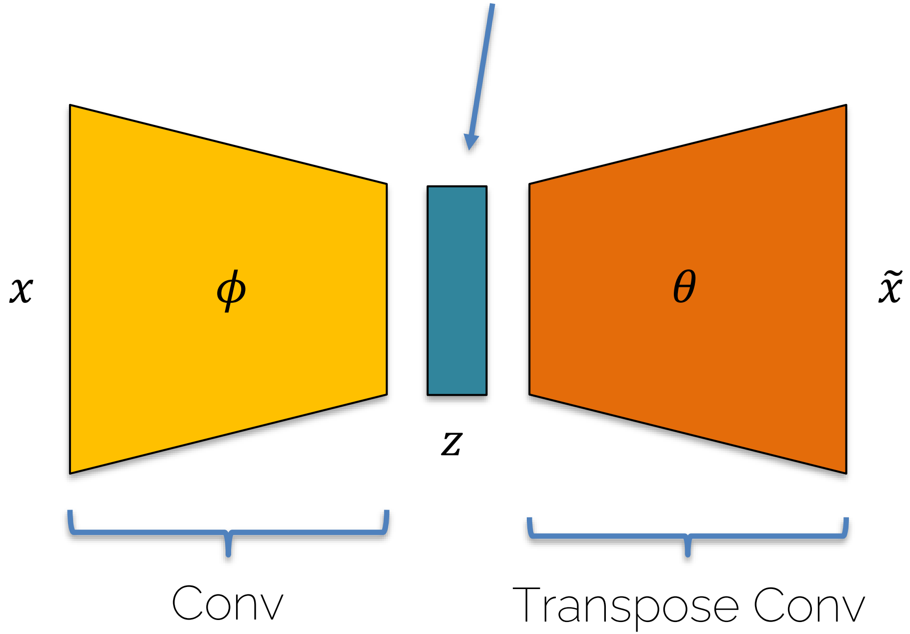
Source: https://bit.ly/37ctFMS

Variational Autoencoder



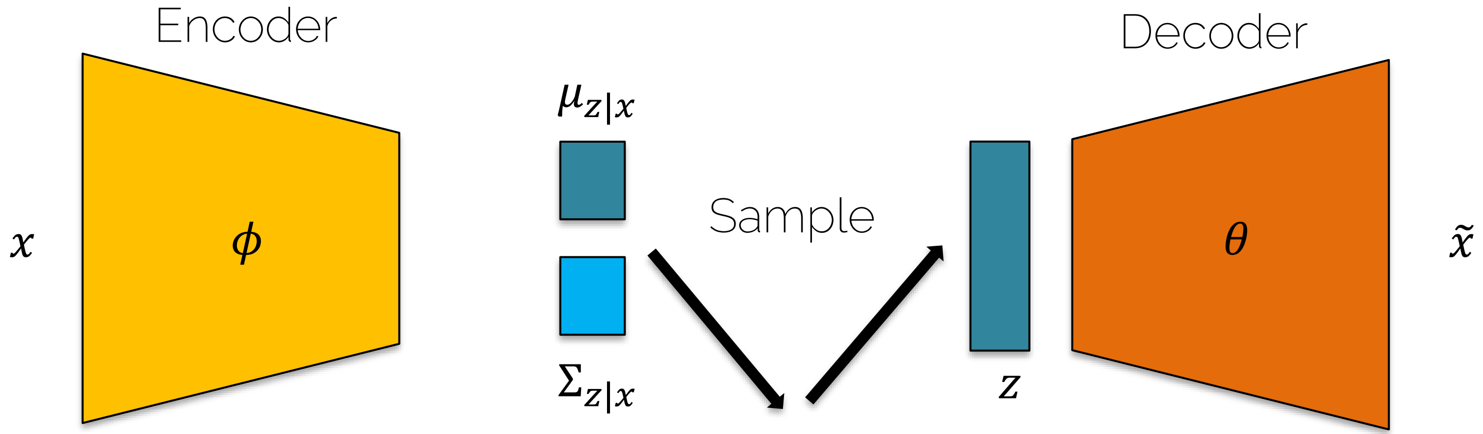
Variational Autoencoder

Goal: Sample from the latent distribution to generate new outputs!



Variational Autoencoder

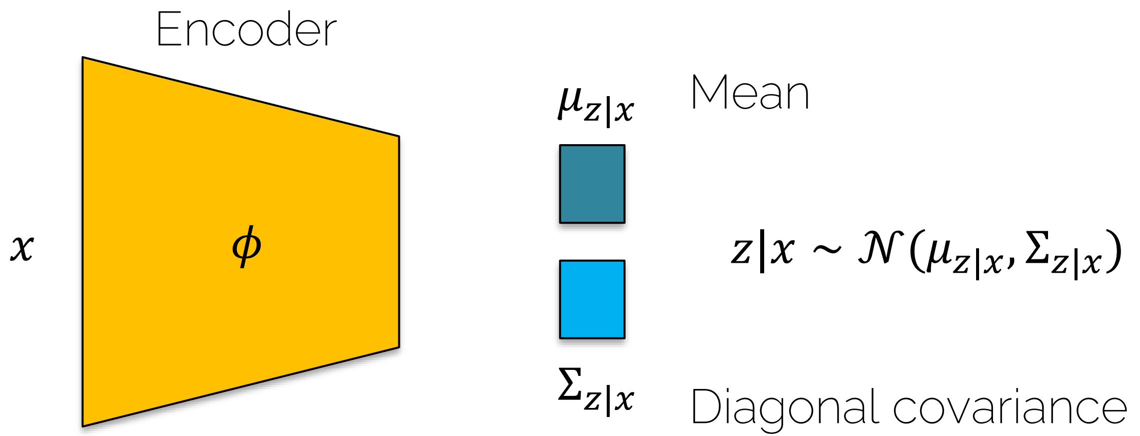
- Latent space is now a distribution
- Specifically it is a Gaussian



$$z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$$

Variational Autoencoder

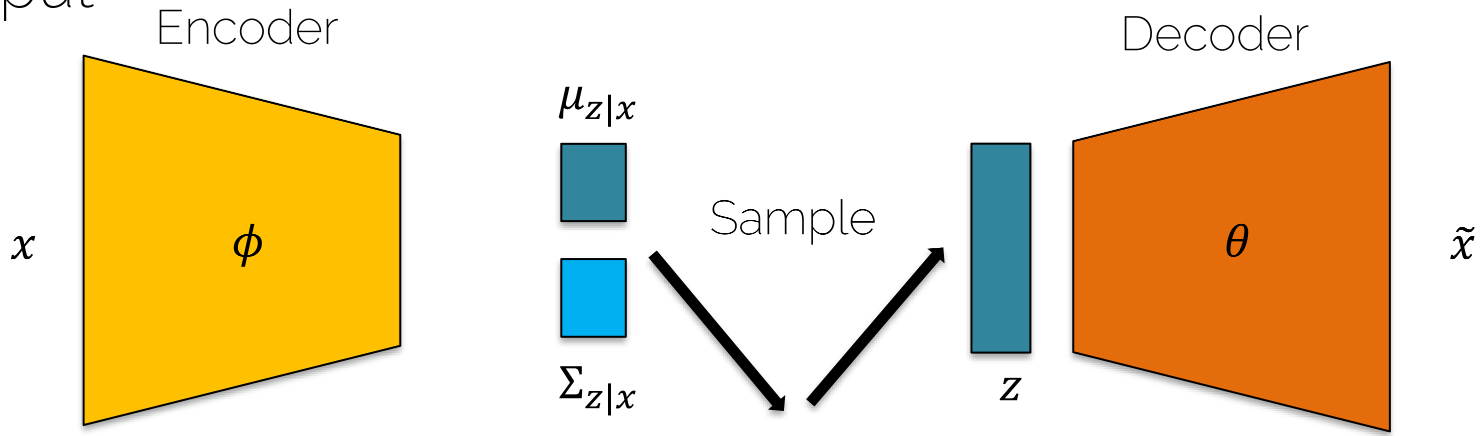
- Latent space is now a distribution
- Specifically it is a Gaussian



$$z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$$

Variational Autoencoder

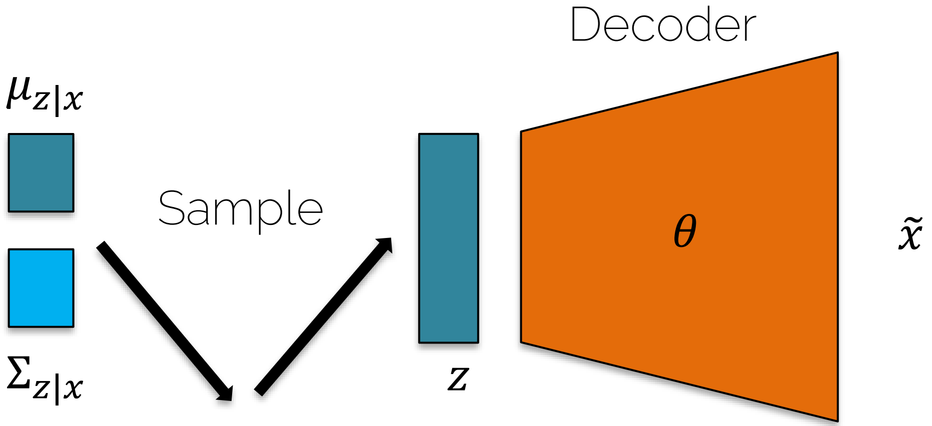
- Training: loss makes sure the latent space is close to a Gaussian and the reconstructed output is close to the input



$$z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$$

Variational Autoencoder

- Test: Sample from the latent space



$$z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$$

Autoencoder vs VAE

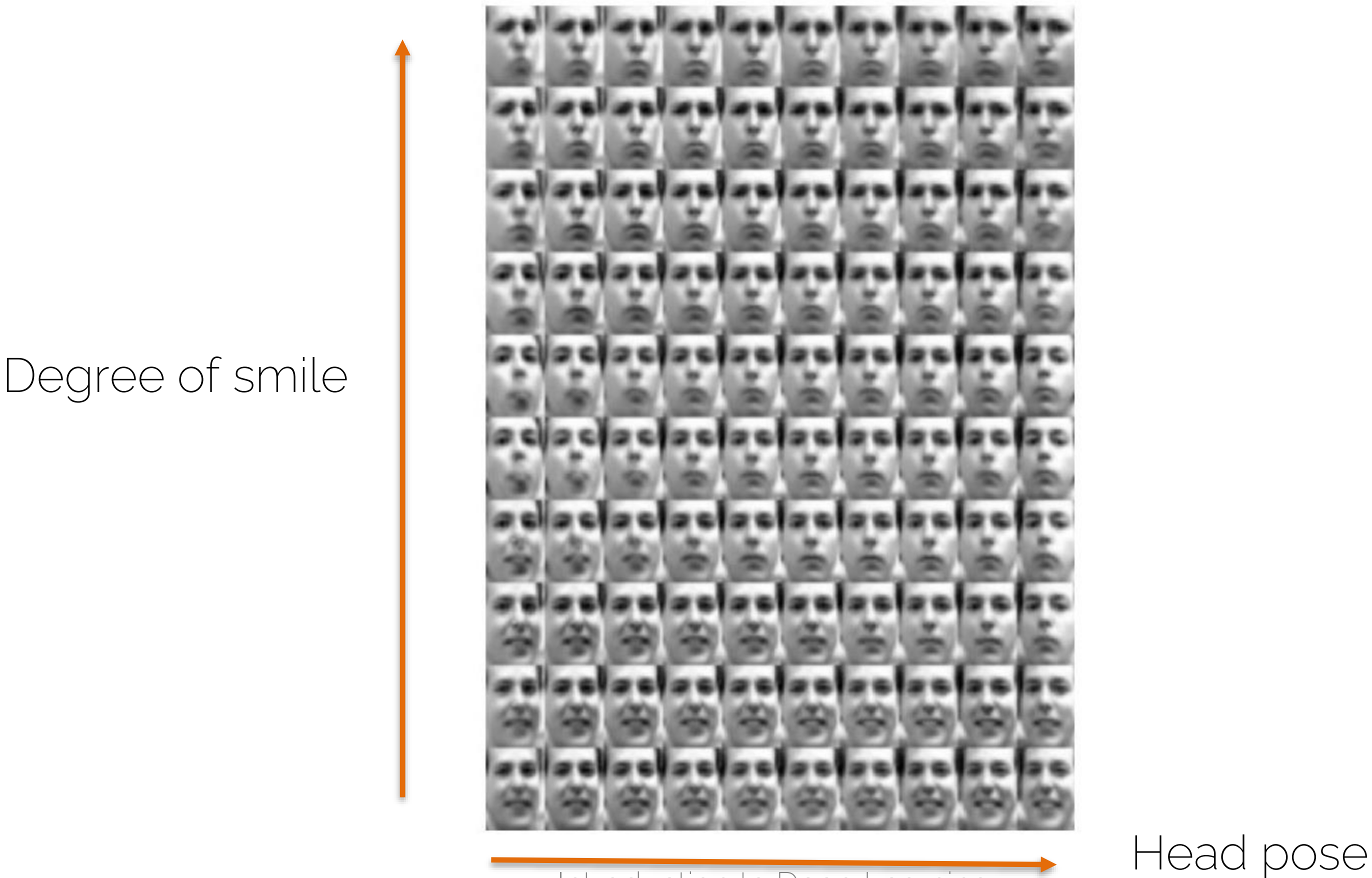
Autoencoder

Variational Autoencoder

Ground Truth

Source: https://github.com/kvfrans/variational-autoencoder

Generating data



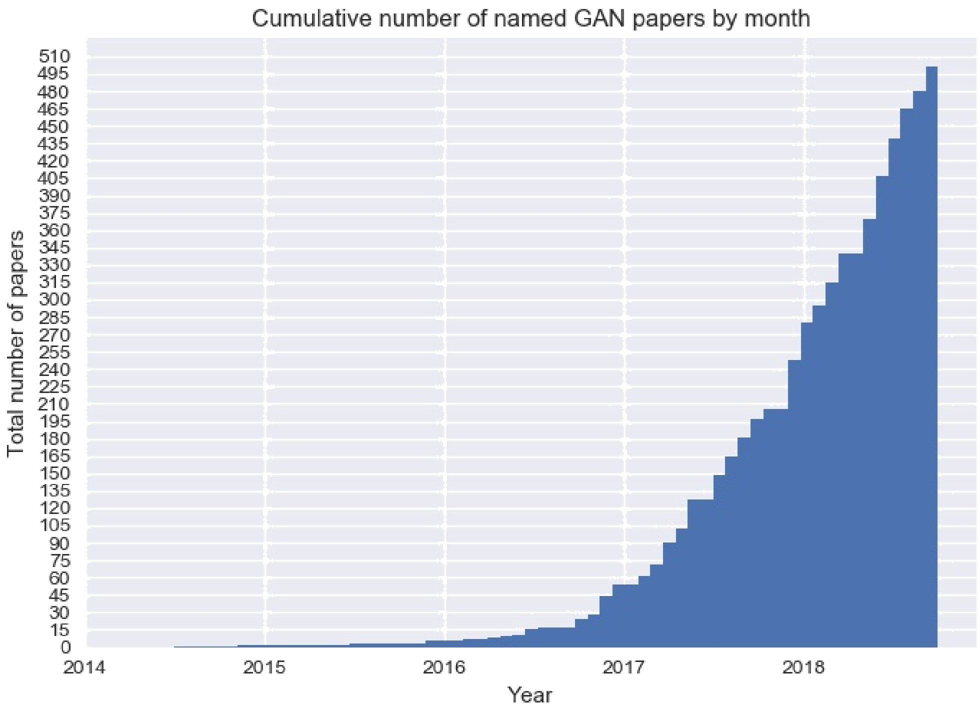
Autoencoder Overview

- Autoencoders (AE)
  - Reconstruct input
  - Unsupervised learning

- Variational Autoencoders (VAE)
  - Probability distribution in latent space (e.g., Gaussian)
  - Interpretable latent space (head pose, smile)
  - Sample from model to generate output

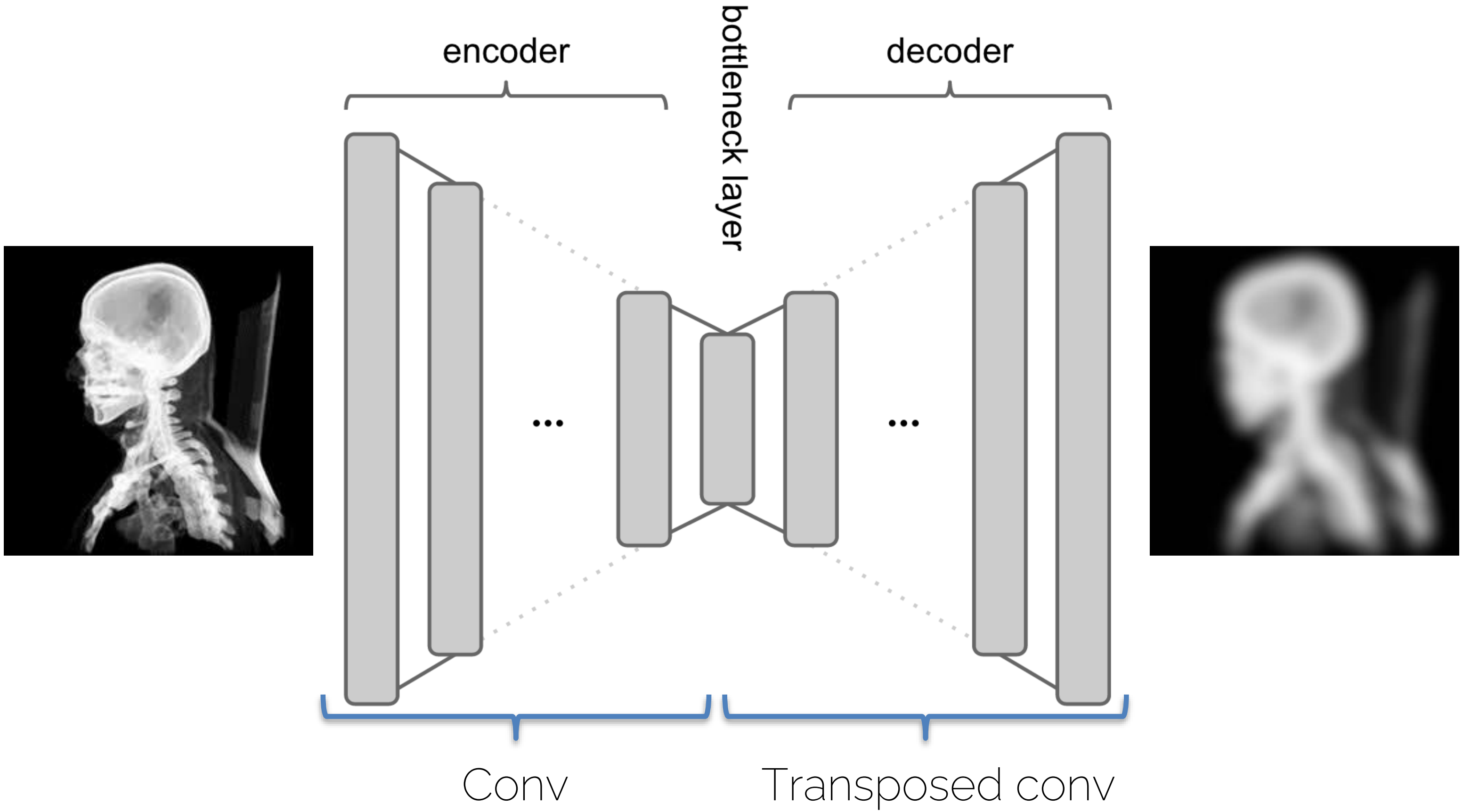
Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)



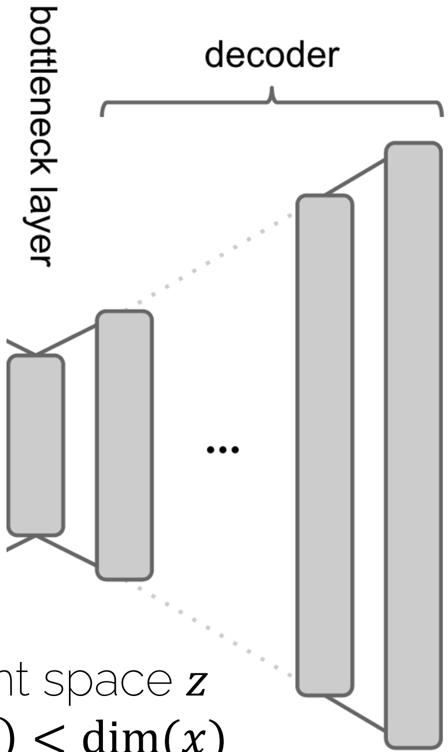
Source: https://github.com/hindupuravinash/the-gan-zoo

Autoencoder



Decoder as Generative Model

Test time:
-> reconstruction from 'random' vector

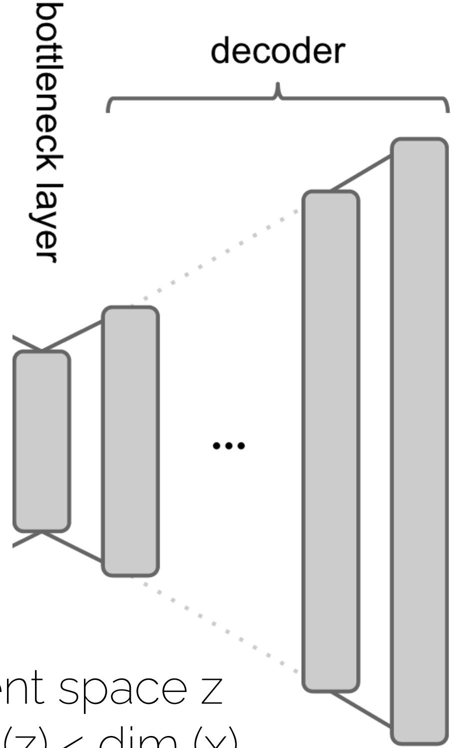


Latent space z
$\dim(z) < \dim(x)$

Reconstruction Loss (often L2)

Output Image

Decoder as Generative Model



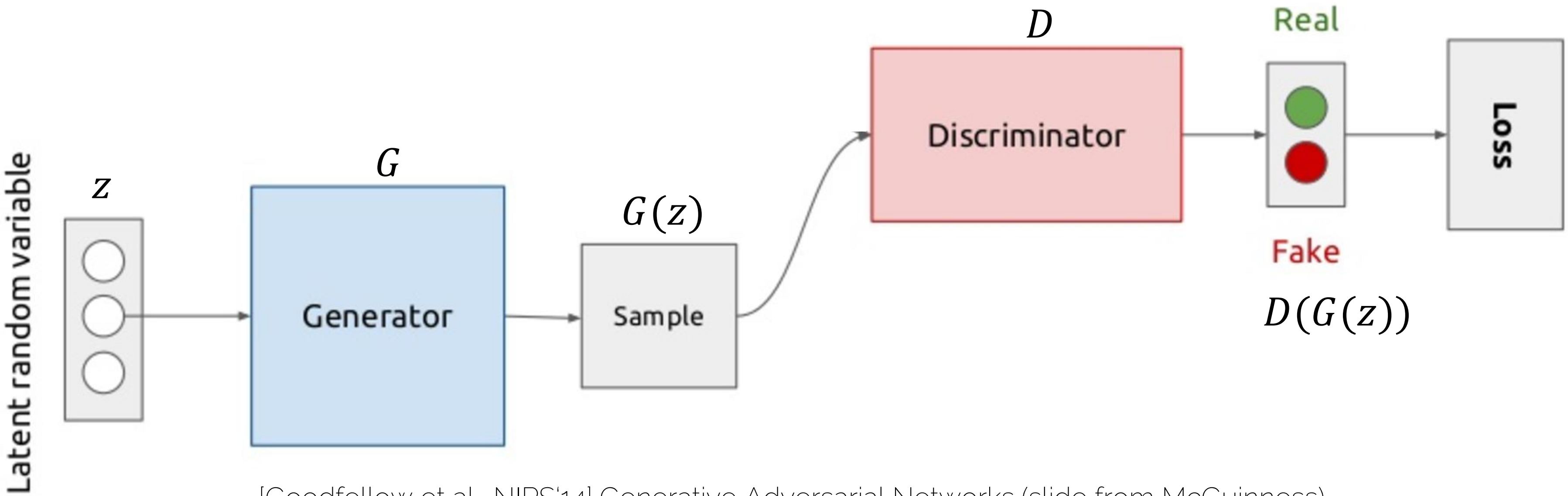
"Test time":
-> reconstruction from 'random' vector

Latent space z
dim (z) < dim (x)

Reconstruction Loss
Often L2, i.e., sum of squared dist.
-> L2 distributes error equally
-> mean is opt.
-> res. Is blurry

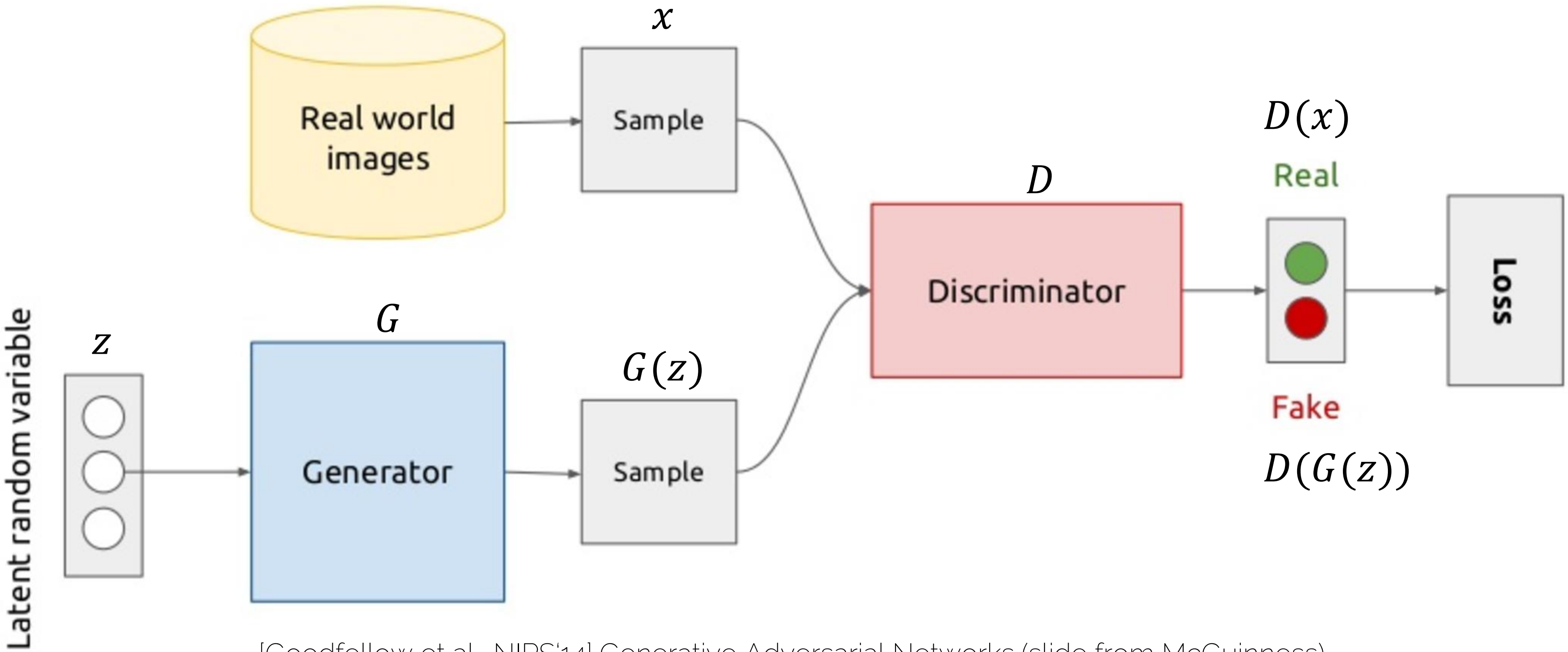
Instead of L2, can we "learn" a loss function?

Generative Adversarial Networks (GANs)



[Goodfellow et al., NIPS'14] Generative Adversarial Networks (slide from McGuinness)

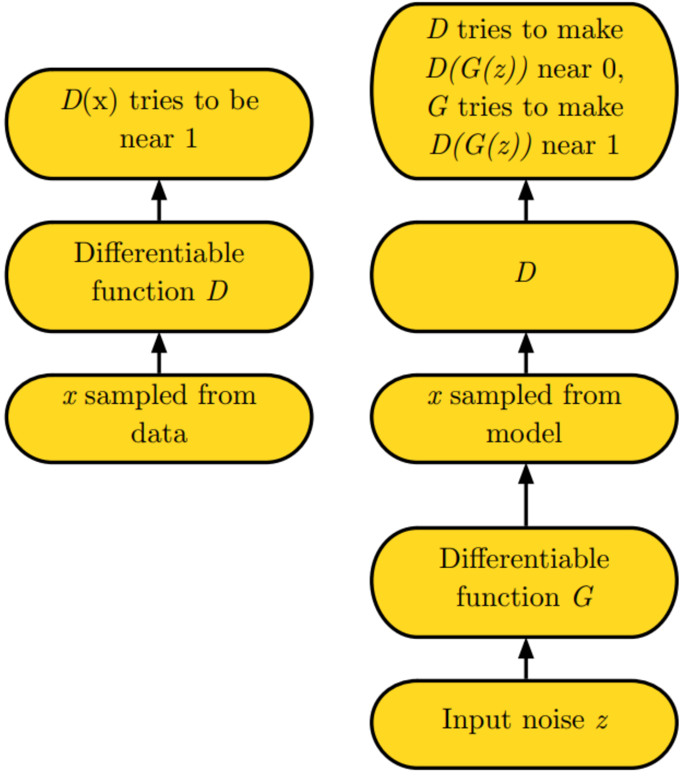
Generative Adversarial Networks (GANs)



[Goodfellow et al., NIPS'14] Generative Adversarial Networks (slide from McGuinness)

Generative Adversarial Networks (GANs)

real data



fake data

(Goodfellow 2016)

[Goodfellow, NIPS'16] Tutorial: Generative Adversarial Networks

GANs: Loss Functions

- Discriminator loss

$$J^{(D)} = \underbrace{-\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)}_{\text{binary cross entropy}}$$

- Generator loss

$$J^{(G)} = -J^{(D)}$$

- Minimax Game:
  - G minimizes probability that D is correct
  - Equilibrium is saddle point of discriminator loss
    - D provides supervision (i.e., gradients) for G

[Goodfellow et al., NIPS'14] Generative Adversarial Networks

GAN Applications

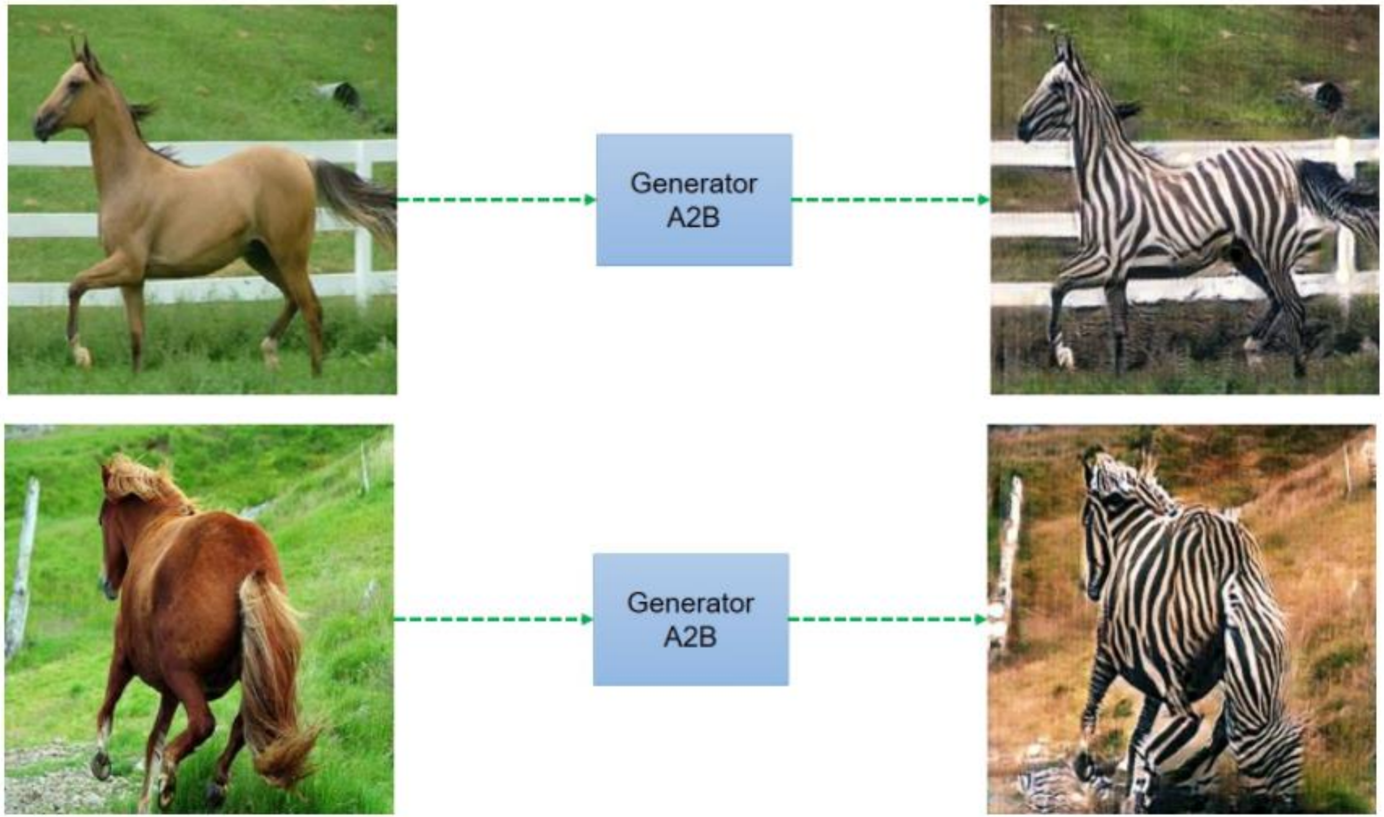
BigGAN: HD Image Generation

[Brock et al., ICLR'18] BigGAN : Large Scale GAN Training for High Fidelity Natural Image Synthesis

StyleGAN: Face Image Generation

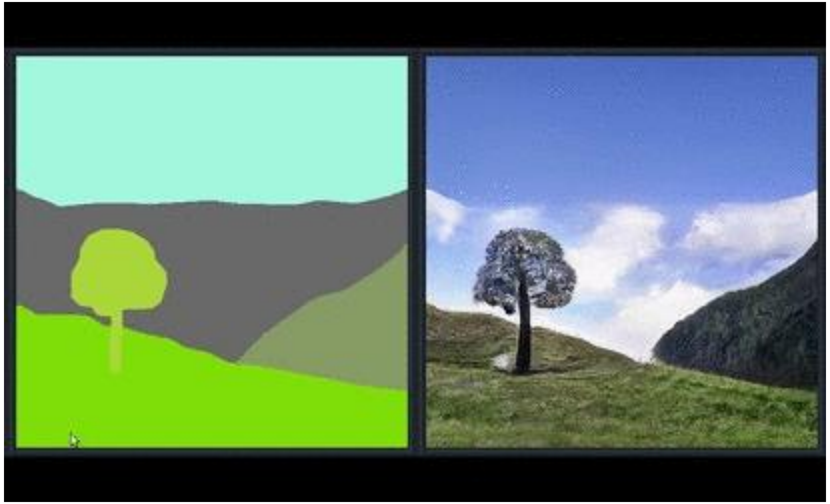
[Karras et al., '18] StyleGAN : A Style-Based Generator Architecture for Generative Adversarial Networks
[Karras et al., '19] StyleGAN2 : Analyzing and Improving the Image Quality of StyleGAN

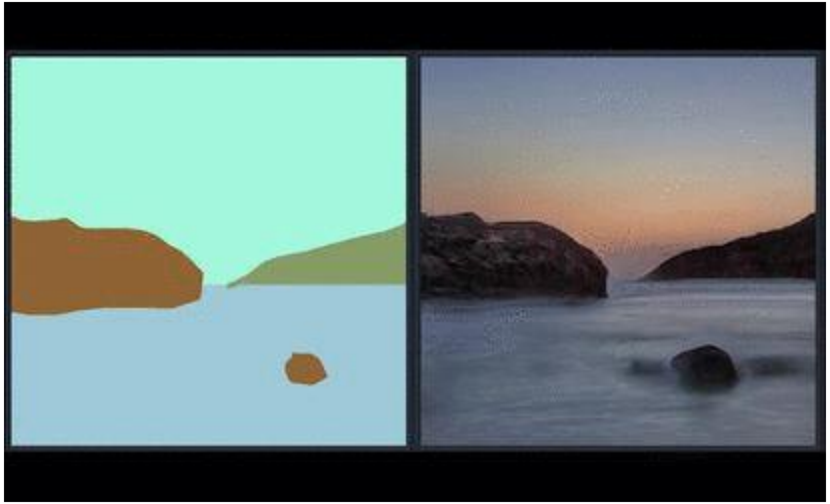
Cycle GAN: Unpaired Image-to-Image Translation



[Zhu et al., ICCV'17] Cycle GAN : Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

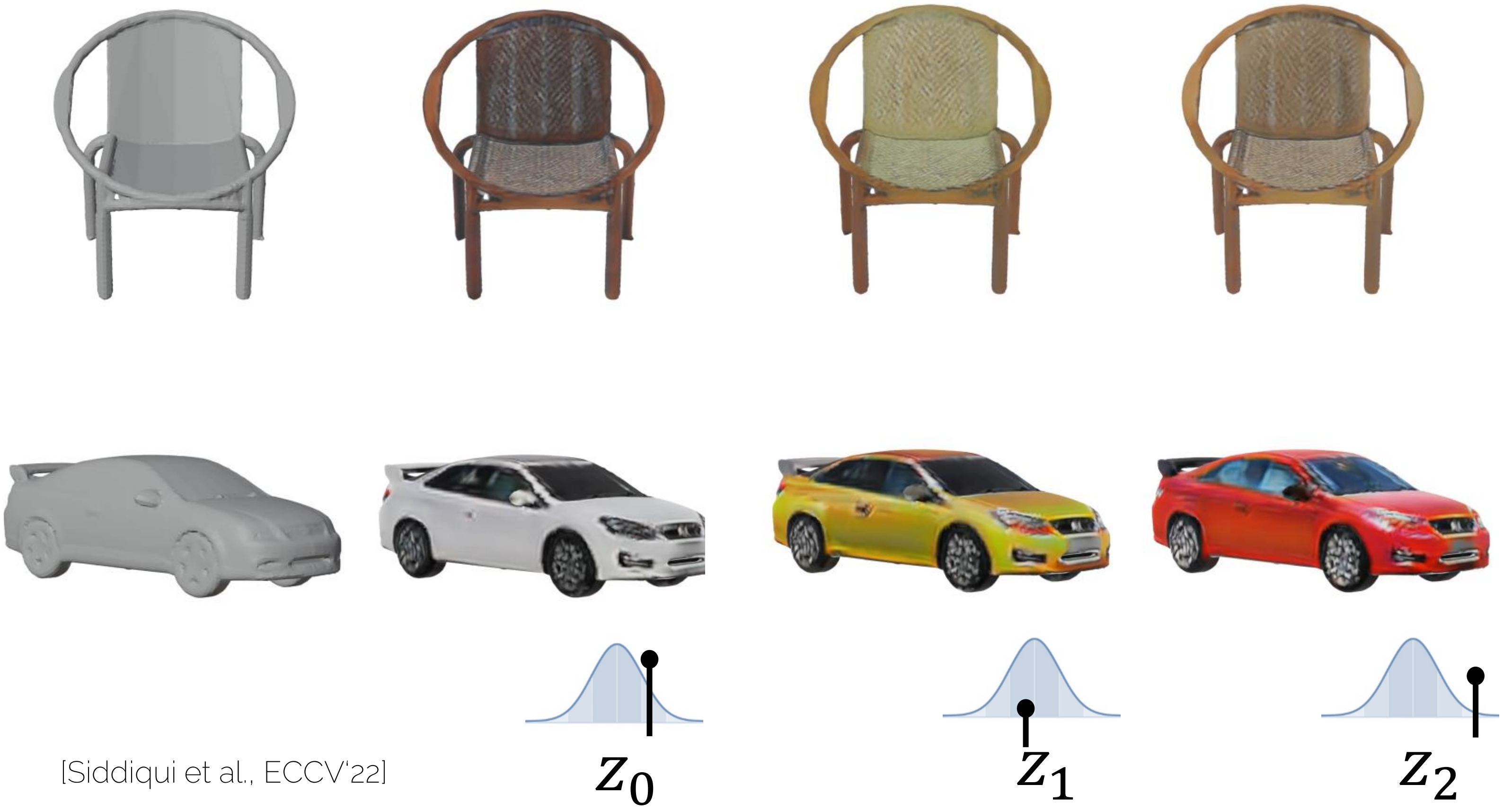
SPADE: GAN-Based Image Editing





[Park et al., CVPR'19] SPADE : Semantic Image Synthesis with Spatially-Adaptive Normalization

Texturify: 3D Texture Generation



[Siddiqui et al., ECCV'22]

Diffusion

Diffusion – Search Interest

Source: Google Trends

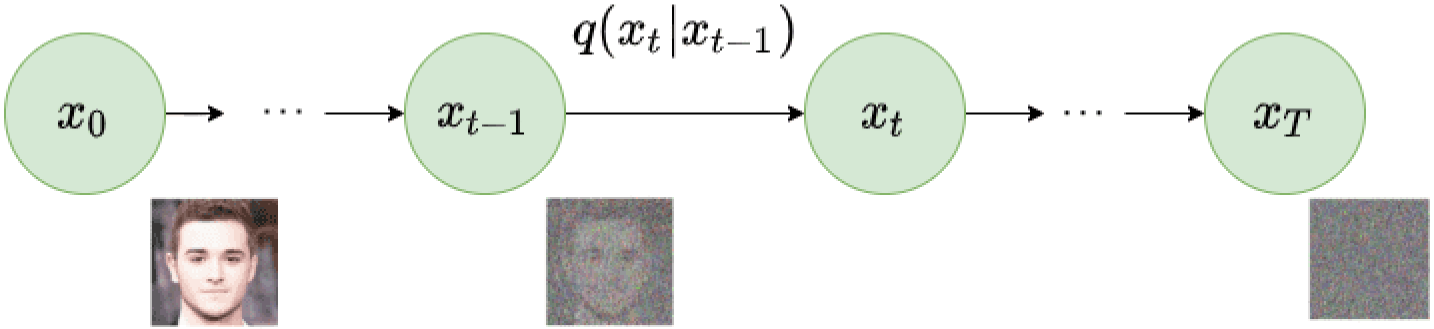
Diffusion Models

- Class of generative models
- Achieved state-of-the-art image generation (DALLE-2, Imagen, StableDiffusion)
- What is diffusion?

Diffusion Process

- Gradually add noise to input image x_0 in a series of T time steps

- Neural network trained to recover original data



[Ho et al. '20] Denoising Diffusion Probabilistic Models

Forward Diffusion

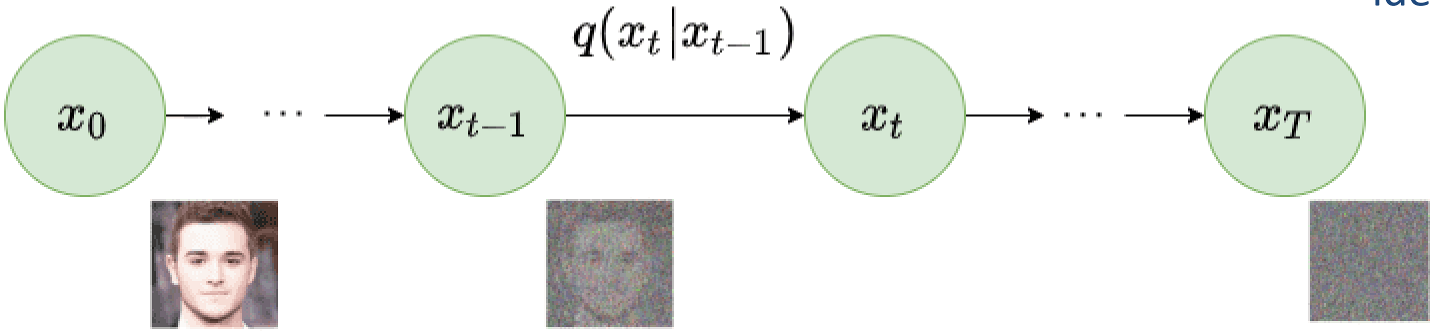
- Markov chain of T steps
  - Each step depends only on previous
- Adds noise to x_0 sampled from real distribution $q(x)$

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

mean

variance

identity matrix



[Ho et al. '20] Denoising Diffusion Probabilistic Models

Forward Diffusion

- Go from $\boldsymbol{x_0}$ to $\boldsymbol{x_T}$:

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

- Efficiency?

Reparameterization

- Define $\alpha_t = 1 - \beta_t,\ \overline{\alpha_t} = \prod_{s=0}^{t} \alpha_s\ ,\ \ \epsilon_0, \dots, \epsilon_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$$
\begin{aligned}
x_t &= \sqrt{1-\beta_t}\,x_{t-1} + \sqrt{\beta_t}\,\epsilon_{t-1} \\
&= \sqrt{\alpha_t}\,x_{t-1} + \sqrt{1-\alpha_t}\,\epsilon_{t-1} \\
&= \sqrt{\alpha_t \alpha_{t-1}}\,x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\,\epsilon_{t-2} \\
&= \sqrt{\overline{\alpha_t}}\,x_0 + \sqrt{1-\overline{\alpha_t}}\,\epsilon_0
\end{aligned}
$$

$$x_t \sim q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\,x_0, (1-\overline{\alpha_t})\mathbf{I})$$

Reverse Diffusion

- $x_{T \to \infty}$ becomes a Gaussian distribution

- Reverse distribution $q(x_{t-1}|x_t)$
  - Sample $x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and run reverse process
  - Generates a novel data point from original distribution

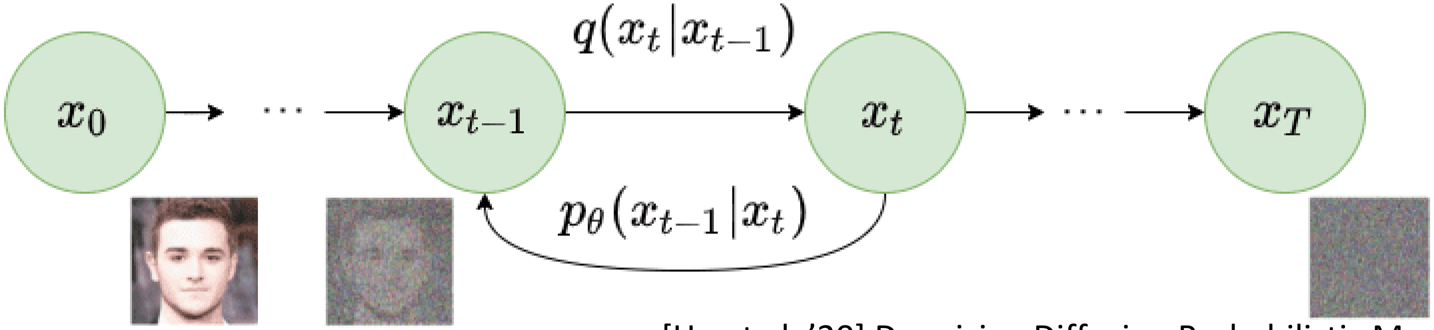
- How to model reverse process?

Approximate Reverse Process

- Approximate $q(x_{t-1}|x_t)$ with parameterized model p_θ (e.g., deep network)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

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



[Ho et al. '20] Denoising Diffusion Probabilistic Models

Training a Diffusion Model

- Optimize negative log-likelihood of training data

$$
\begin{aligned}
&L_{VLB} \\
&= \mathbb{E}_q[\underbrace{D_{KL}(q(x_T|x_0||p_\theta(x_T))}_{L_T} \\
&+ \sum_{t=2}^{T} \underbrace{D_{KL}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{- \log p_\theta(x_0|x_1)}_{L_0}]
\end{aligned}
$$

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

Training a Diffusion Model

- $L_{t-1} = D_{KL}\big(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)\big)$
- Comparing two Gaussian distributions
- $L_{t-1} \propto ||\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)||^2$

- Predicts diffusion posterior mean

Diffusion Model Architecture

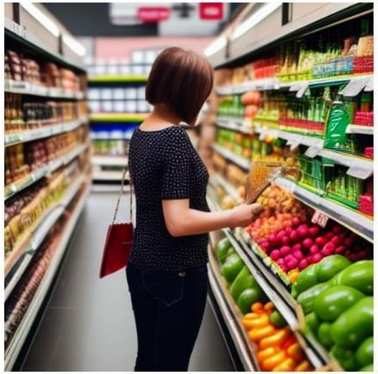
- Input and output dimensions must match
- Highly flexible to architecture design
- Commonly implemented with U-Net architectures

Applications for Diffusion Models

- Text-to-image

Oil Painting

Digital Illustration

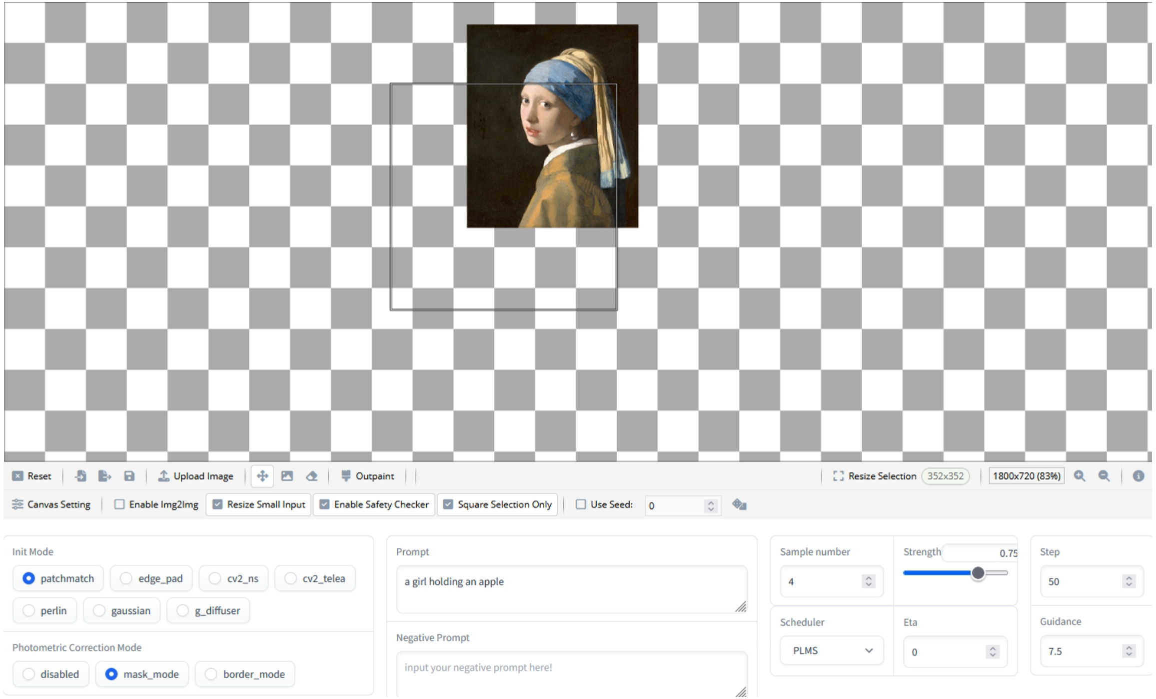


Hyperrealistic

Cartoon

Applications for Diffusion Models

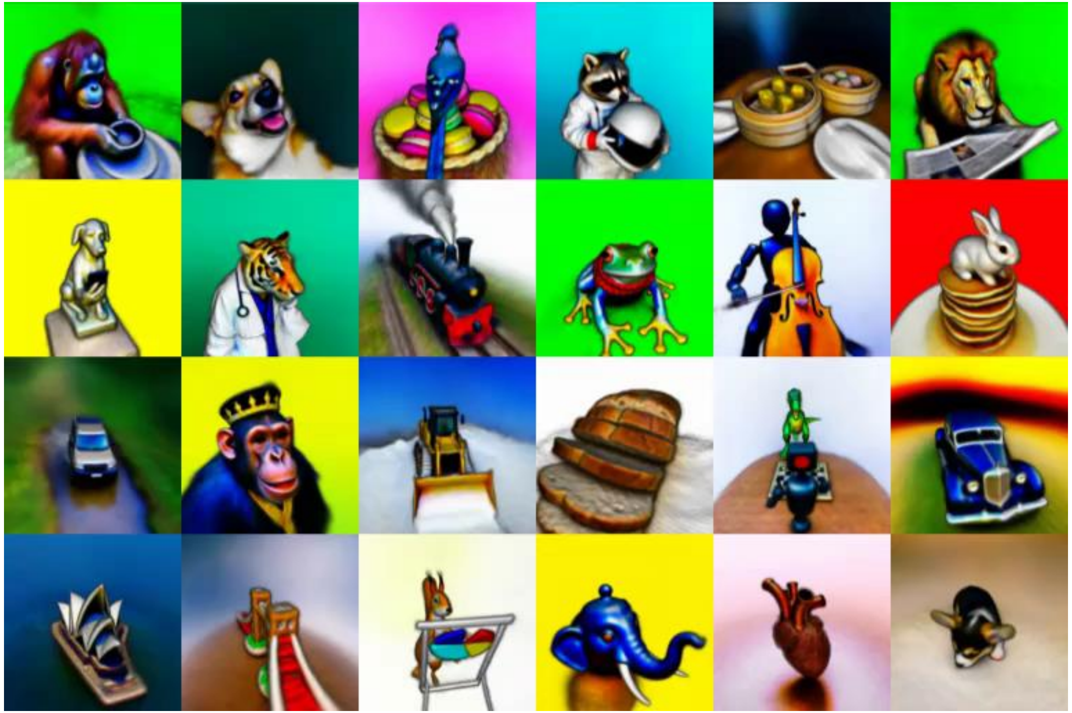
- Image inpainting & outpainting



https://github.com/lkwq007/stablediffusion-infinity

Applications for Diffusion Models

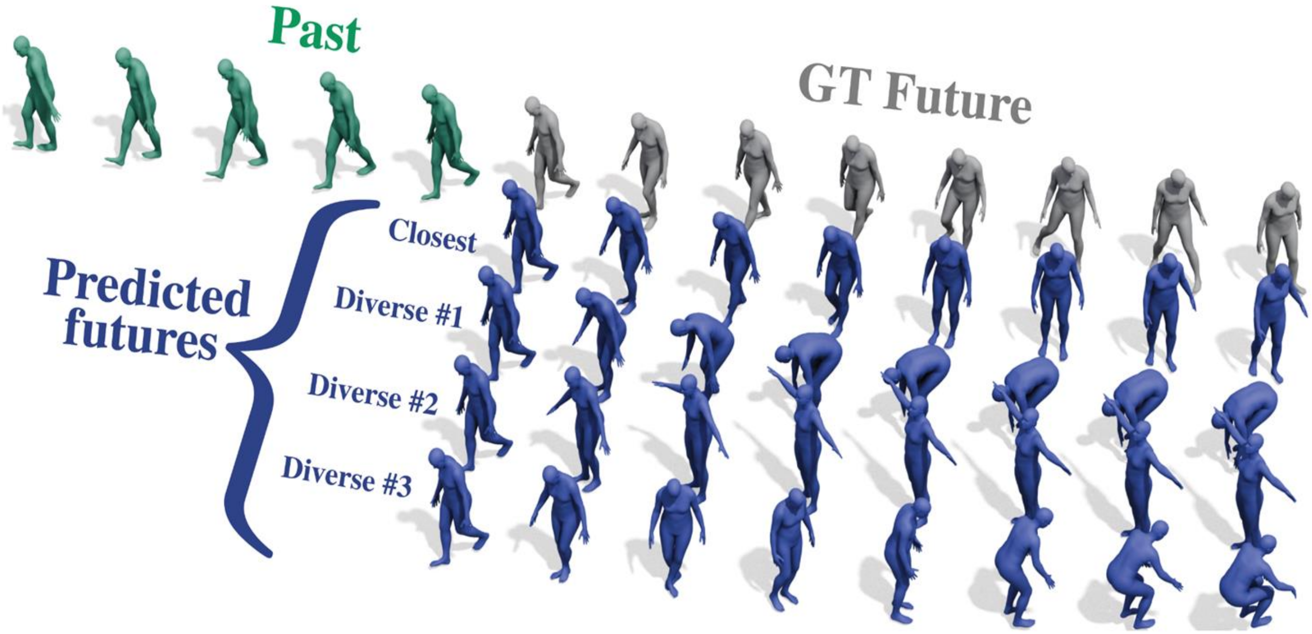
- Text-to-3D Neural Radiance Fields



https://dreamfusion3d.github.io/

Realistic 3D Human Motion Generation with Anisotropic Diffusion

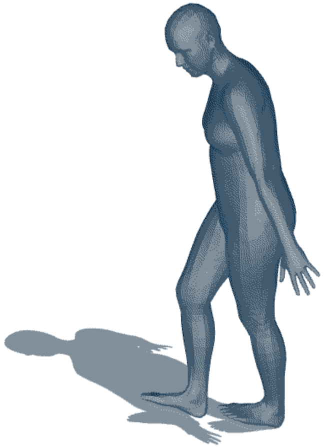
Cecilia Curreli[1,2] Dominik Muhle[1,2] Abhishek Saroha[1,2]
Zhenzhang Ye[1] Riccardo Marin[1,2] Daniel Cremers[1,2]
[1]Technical University of Munich [2]Munich Center for Machine Learning



3D Human Motion Prediction

Input: body joint positions up to time T

Past $\in \mathbb{R}^{P \times J \times 3}$



Output: body joint positions from time T

Future $\in \mathbb{R}^{F \times J \times 3}$

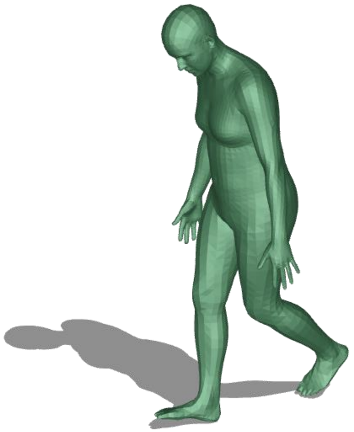
3D Human Motion Prediction

Input: body joint positions up to time T

Past $\in \mathbb{R}^{P \times J \times 3}$

Output: body joint positions from time T

Future $\in \mathbb{R}^{F \times J \times 3}$



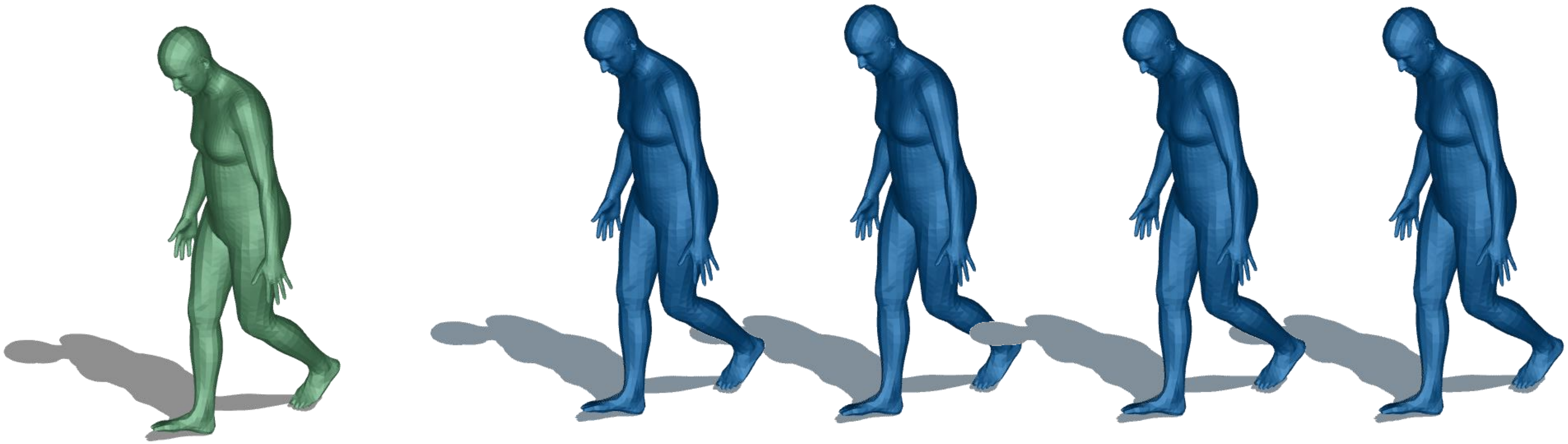
3D Human Motion Prediction

Input: body joint positions up to time T

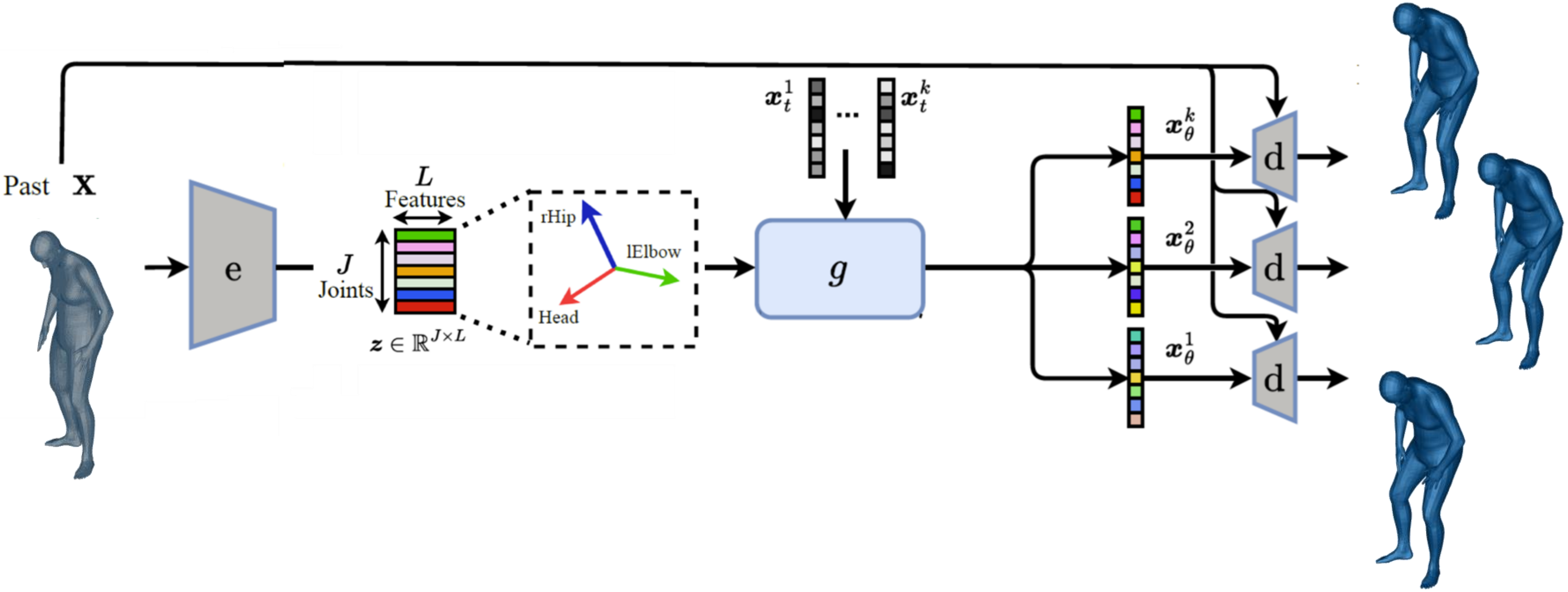
Past $\in \mathbb{R}^{P \times J \times 3}$

Output: body joint positions from time T

Future $\in \mathbb{R}^{F \times J \times 3}$

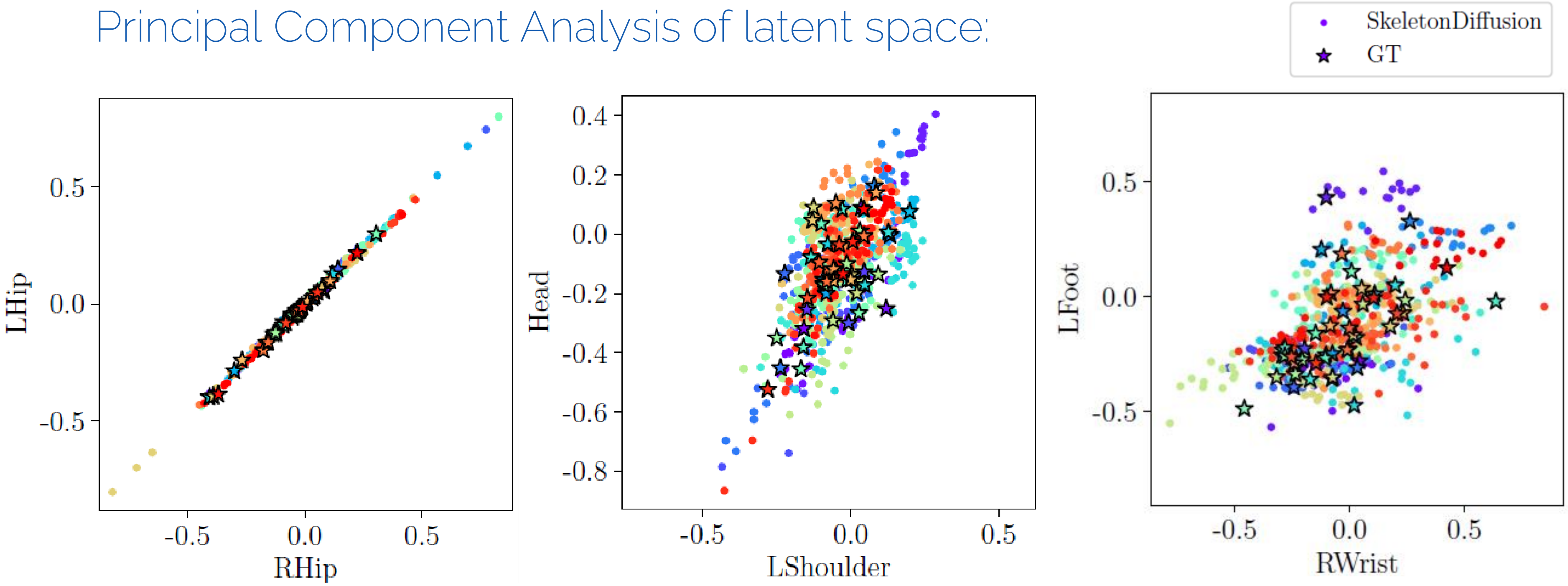


3D Human Motion Prediction

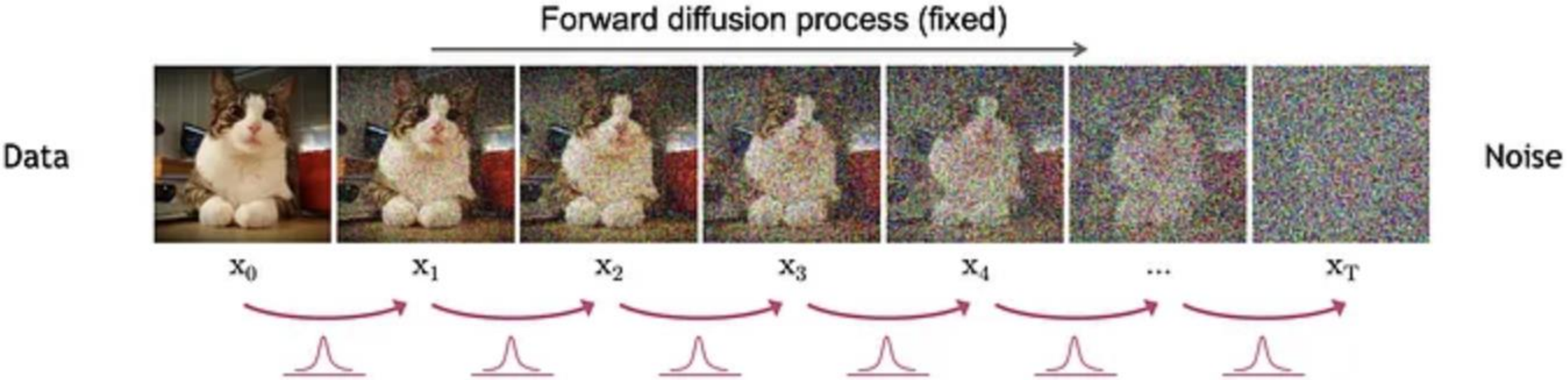


Anisotropies in Latent Space

Principal Component Analysis of latent space:



Recap: Isotropic Diffusion



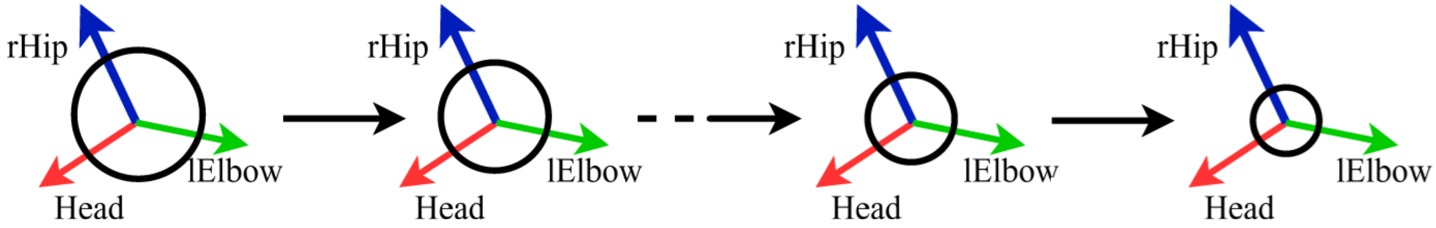
$$\mathcal{N}(\boldsymbol{x}_t; \sqrt{\alpha_t}\boldsymbol{x}_{t-1}, \boldsymbol{\Sigma}_t)$$

$$\boldsymbol{\Sigma}_t = (1 - \alpha_t)\mathbb{I}$$

$$\boldsymbol{x}_t = \sqrt{\alpha_t}\boldsymbol{x}_{t-1} + (1 - \alpha_t)\epsilon \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$$

Isotropic vs. Anisotropic Diffusion

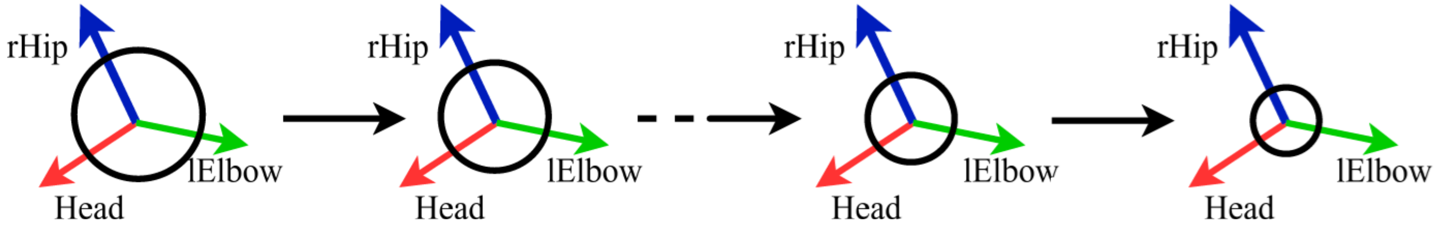
Isotr



$$\mathbf{\Sigma}_t = (1 - \alpha_t)\mathbb{I}$$

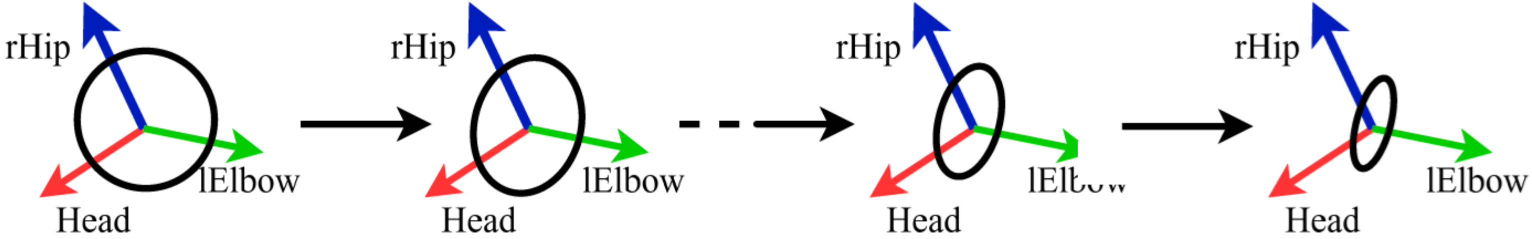
Isotropic vs. Isotropic Diffusion

Isotr



$$\boldsymbol{\Sigma}_t = (1 - \alpha_t)\mathbb{I}$$

Ours:



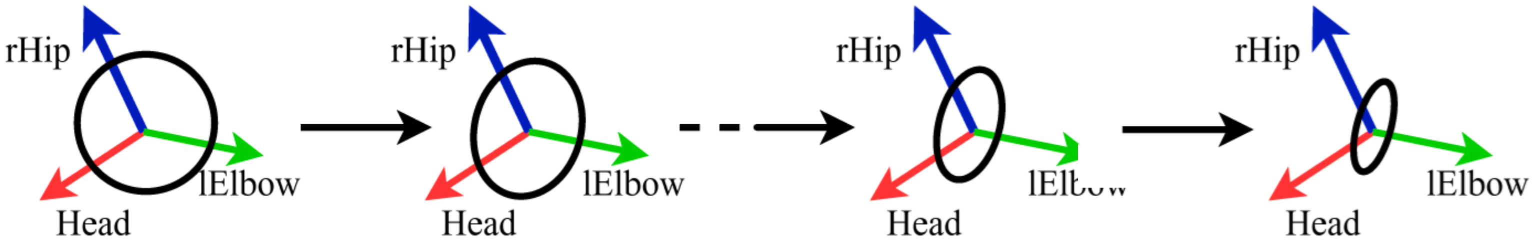
$$\boldsymbol{\Sigma}_t = (1 - \alpha_t)\gamma_t \boldsymbol{\Sigma}_N + (1 - \alpha_t)(1 - \gamma_t)\mathbb{I}$$

Isotropic vs. Anisotropic Diffusion

Ours:

$$\boldsymbol{\Sigma}_N = \frac{\mathbf{A} - \lambda_{\min}(\mathbf{A})\mathbb{I}}{\lambda_{\max}(\mathbf{A}) - \lambda_{\min}(\mathbf{A})}$$

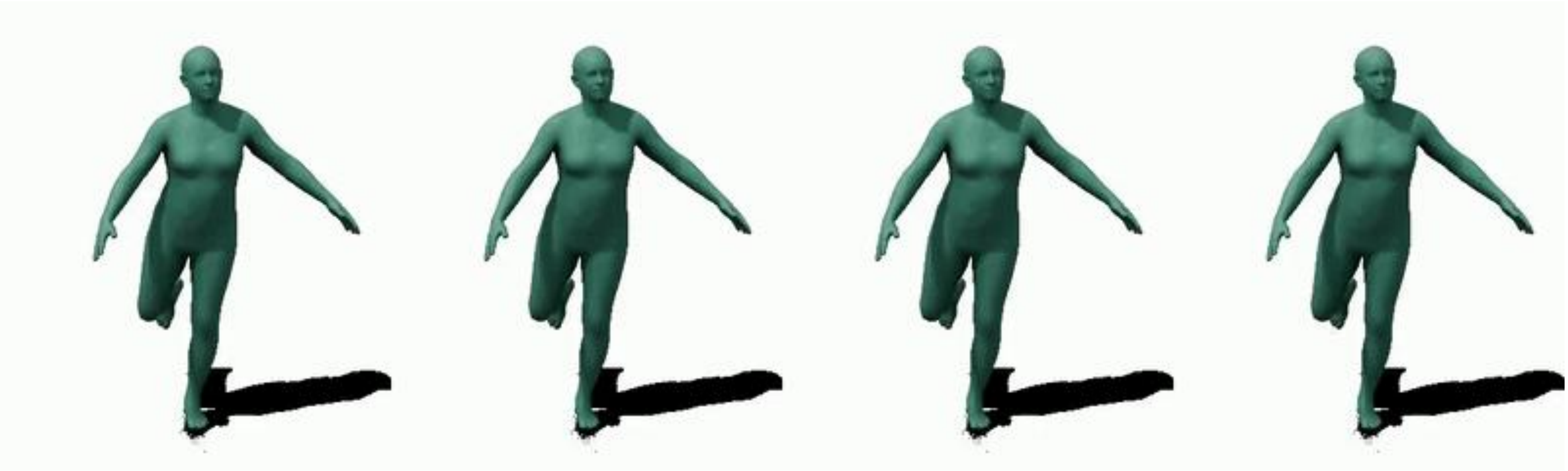
A: adjacency matrix of the skeleton graph



$$\boldsymbol{\Sigma}_t = (1-\alpha_t)\gamma_t\boldsymbol{\Sigma}_N + (1-\alpha_t)(1-\gamma_t)\mathbb{I}$$

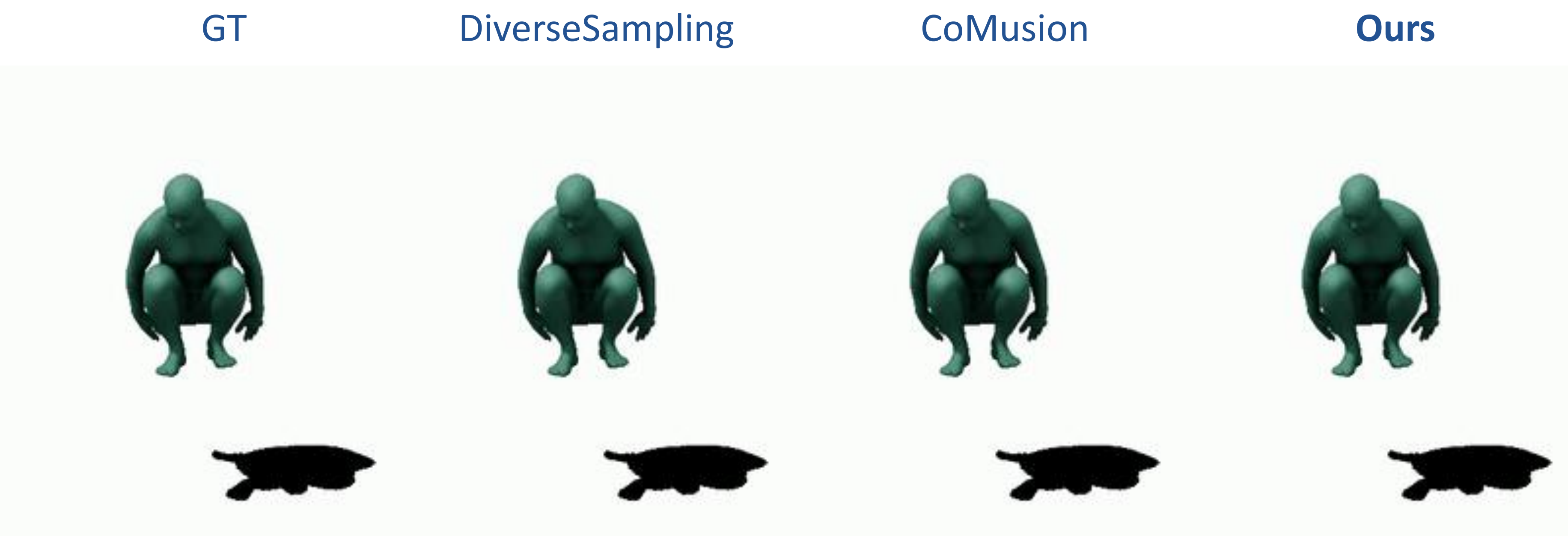
Comparison to Baselines

GT	DiverseSampling	CoMusion	**Ours**



Diverse yet realistic

Comparison to Baselines



Future coherent with past motion

Reinforcement Learning

Learning Paradigms in ML

Supervised Learning
E.g., classification, regression

Labeled data

Find mapping from input to label

Unsupervised Learning
E.g., clustering, anomaly detection

Unlabeled data

Find structure in data

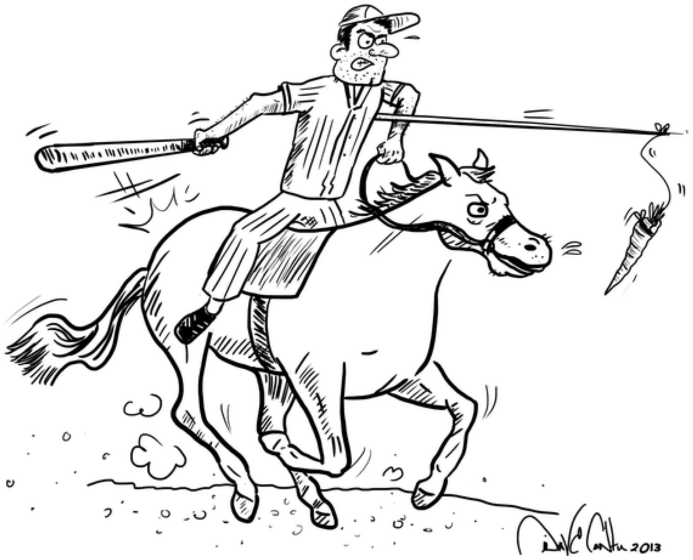
Reinforcement Learning

Sequential data

Learning by interaction with the environment

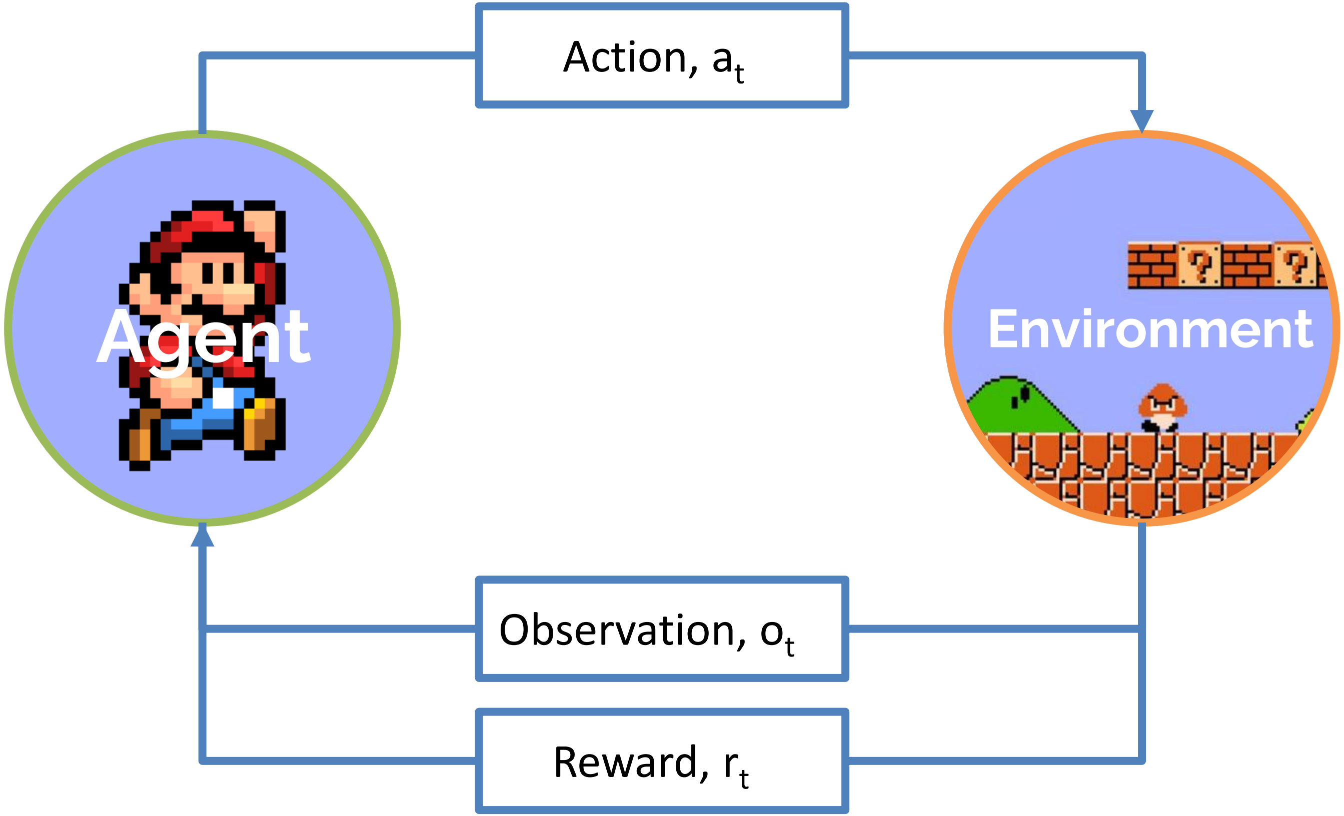
In a Nutshell

- RL-agent is trained using the "carrot and stick" approach
- Good behavior is encouraged by rewards
- Bad behavior is discouraged by punishment



Source: quora.com

Agent and Environment



Characteristics of RL

- Sequential, non i.i.d. data (time matters)
- Actions have an effect on the environment
  -> Change future input
- No supervisor, target is approximated by the reward signal

History and State

- The agent makes decisions based on the **history h** of observations, actions and rewards up to time-step t

$$h_t = o_1, a_1, r_1, \dots, a_{t-1}, r_{t-1}, o_t$$

- The **state s** contains all the necessary information from **h** -> **s** is a function of **h**

$$s_t = f(h_t)$$

 Source: UCL Reinforcement Learning

Markov Assumption

- Problem: History grows linearly over time
- Solution: Markov Assumption
- A state S_t is Markov if and only if:

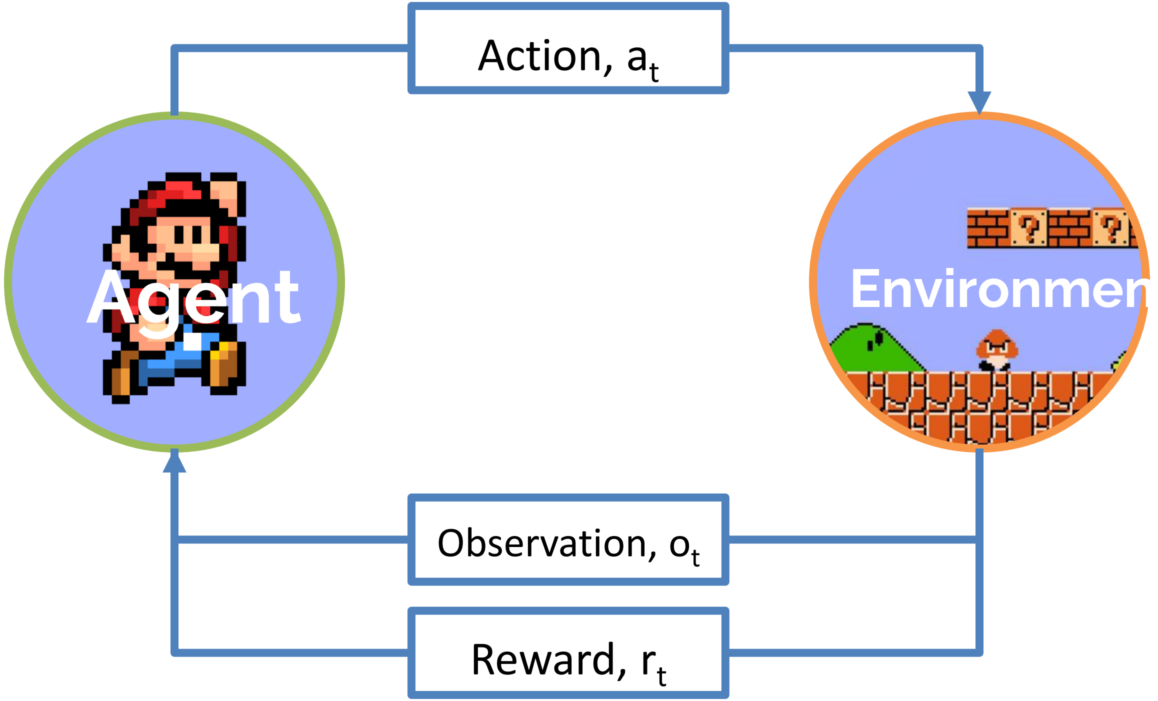
$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \dots s_t]$$

- "The future is independent of the past given the present"

Source: UCL Reinforcement Learning

Agent and Environment

- Reward and next state are functions of current observation o_t and action a_t only



Mathematical Formulation

- The RL problem is a Markov Decision Process (MDP) defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: Set of possible states
$\mathcal{A}$: Set of possible actions
$\mathcal{R}$: Distribution of reward given (state, action) pair
$\mathbb{P}$: Transition probability of a (state, action) pair
γ : Discount factor (discounts future rewards)

Source: Stanford cs231n

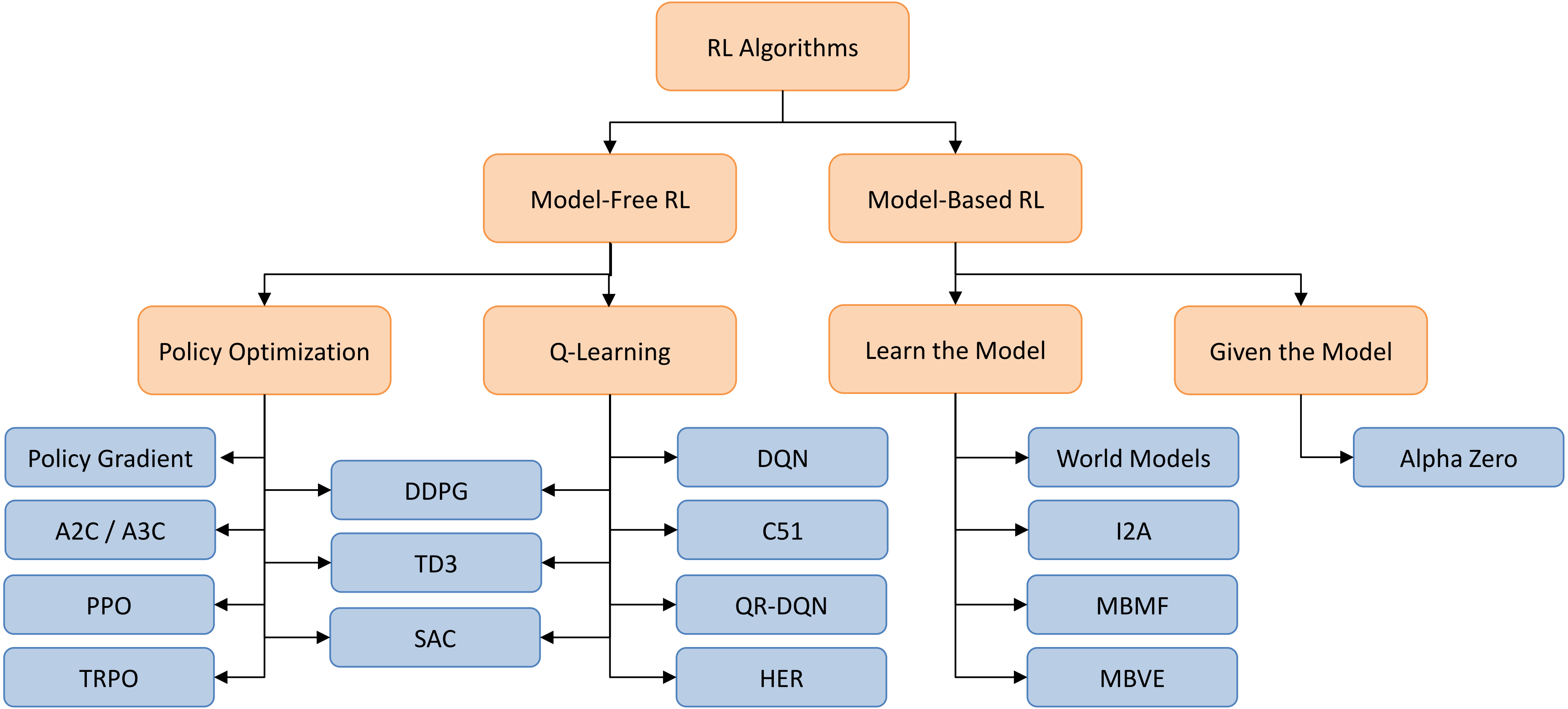
Components of an RL Agent

- Policy π : Behavior of the agent
  -> Mapping from state to action: $a = \pi(s)$

- Value-, Q-Function: How good is a state or (state, action) pair
  -> Expected future reward

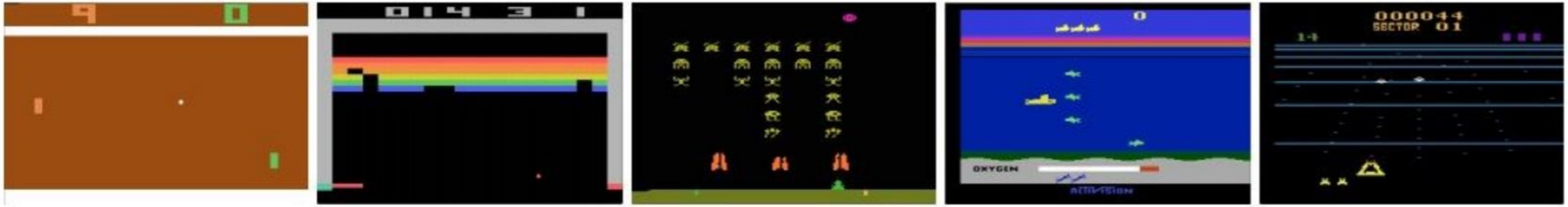
Source: UCL Reinforcement Learning

Taxonomy of RL Algorithms



Source: spinningup.openai.com

RL Milestones: Playing Atari

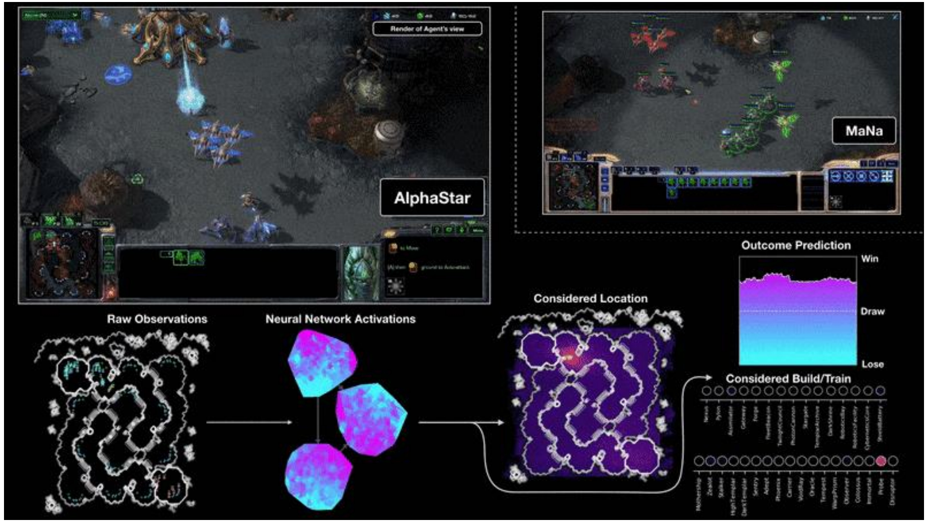


- Mnih et al. 2013, first appearance of DQN
- Successfully learned to play different Atari games like Pong, Breakout, Space Invaders, Seaquest and Beam Rider

[Mnih et al., NIPS'13] Playing Atari with Deep Reinforcement Learning

RL Milestones: AlphaZero (StarCraft)

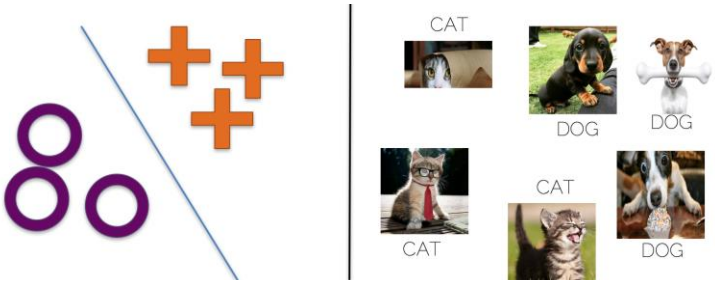
- Model: Transformer network with a LSTM core
- Trained on 200 years of StarCraft play for 14 days
- 16 Google v3 TPUs
- December 2018: Beats MaNa, a professional StarCraft player (world rank 13)



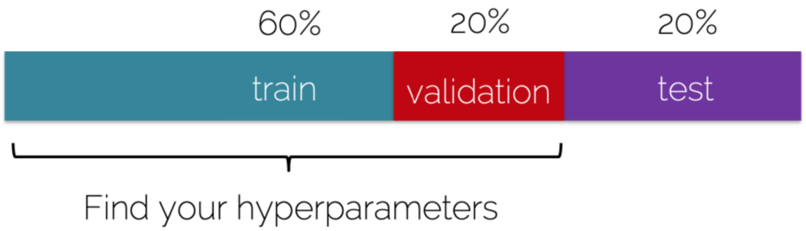
I2DL Summary

Machine Learning Basics

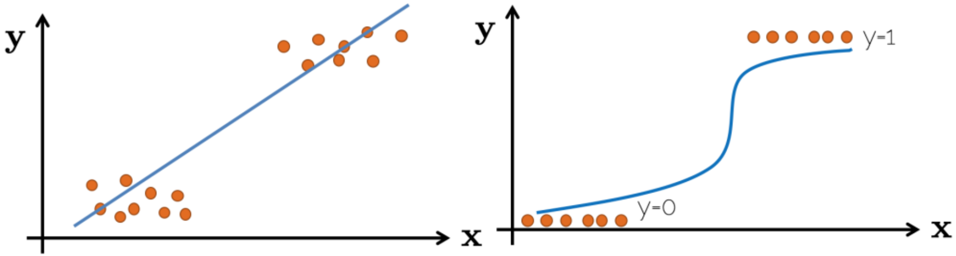
- Unsupervised vs Supervised Learning



- Data splitting

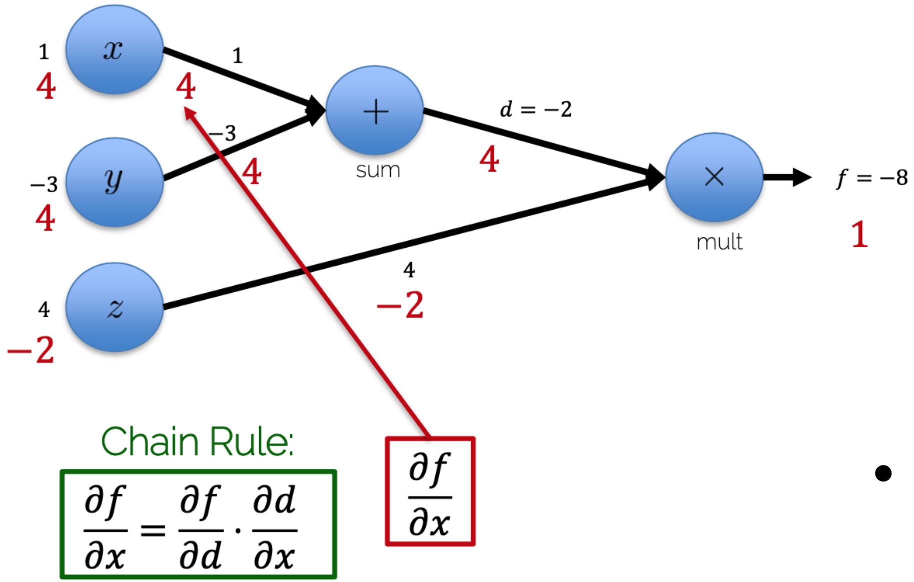


- Linear vs logistic regression

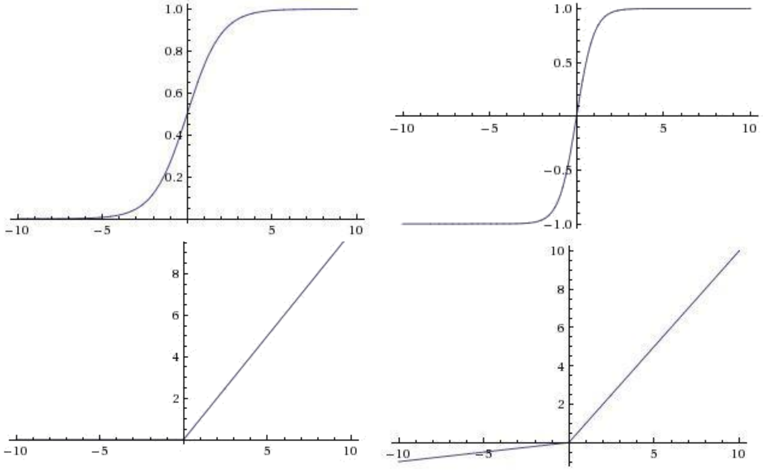


Intro to Neural Networks

- Backpropagation



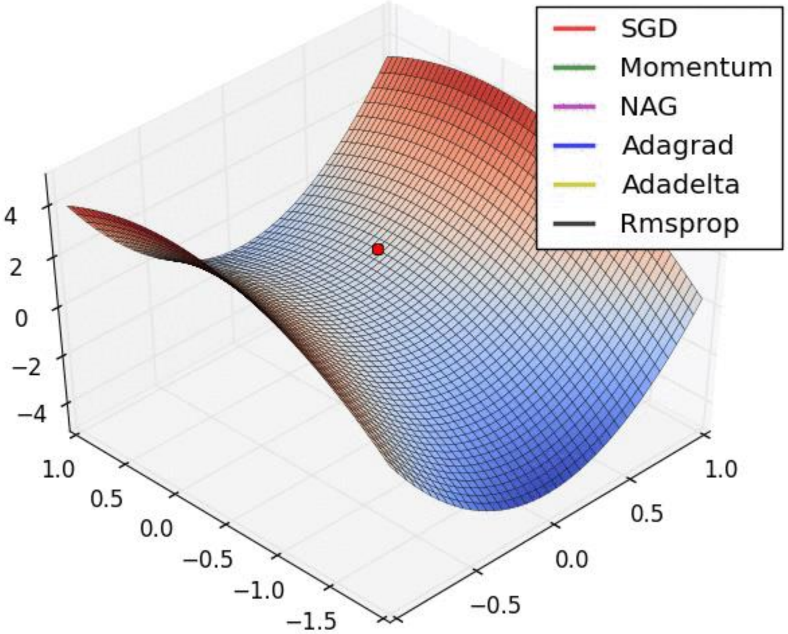
- Activation functions



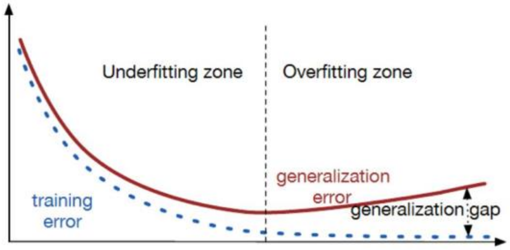
- Loss functions
  – Comparison & effects

Training Neural Networks

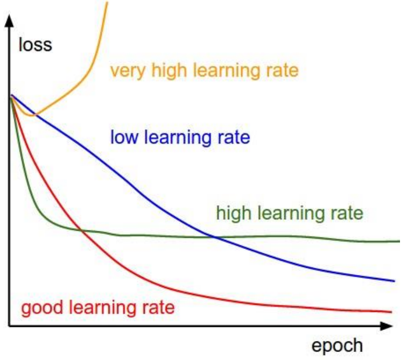
- Gradient Descent/ SGD



- Regularization



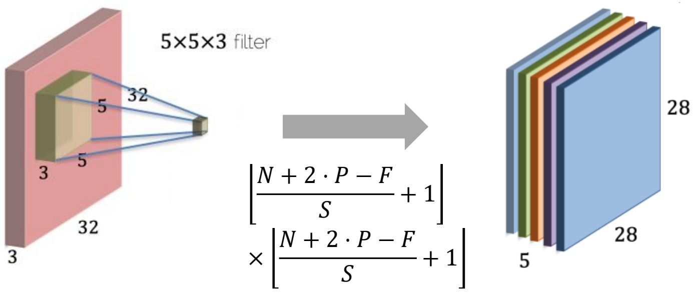
- Parameter & interpretation



Source: http://ruder.io/optimizing-gradient-descent/, https://srdas.github.io/DLBook/ImprovingModelGeneralization.html, http://cs231n.github.io/neural-networks-3/

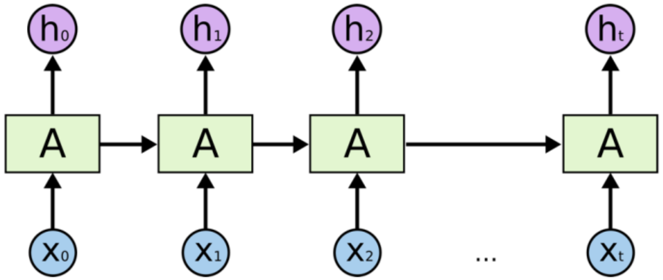
Typology of Neural Networks

- CNNs

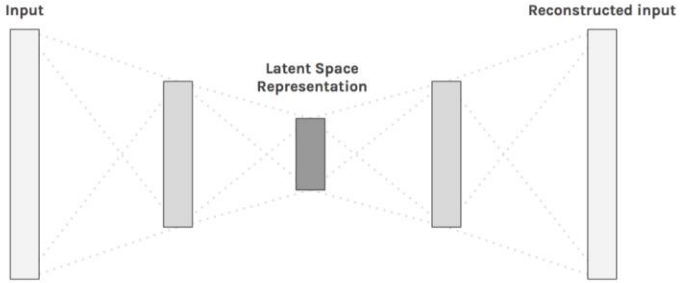


$$\left\lfloor \frac{N + 2 \cdot P - F}{S} + 1 \right\rfloor \times \left\lfloor \frac{N + 2 \cdot P - F}{S} + 1 \right\rfloor$$

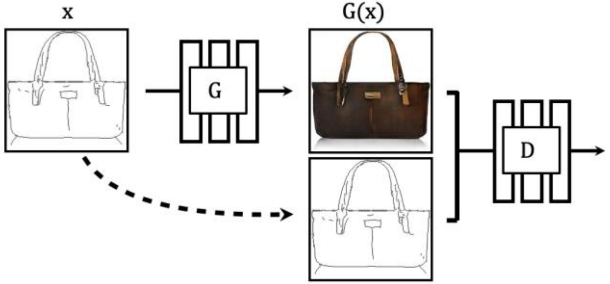
- RNNs



- Autoencoder



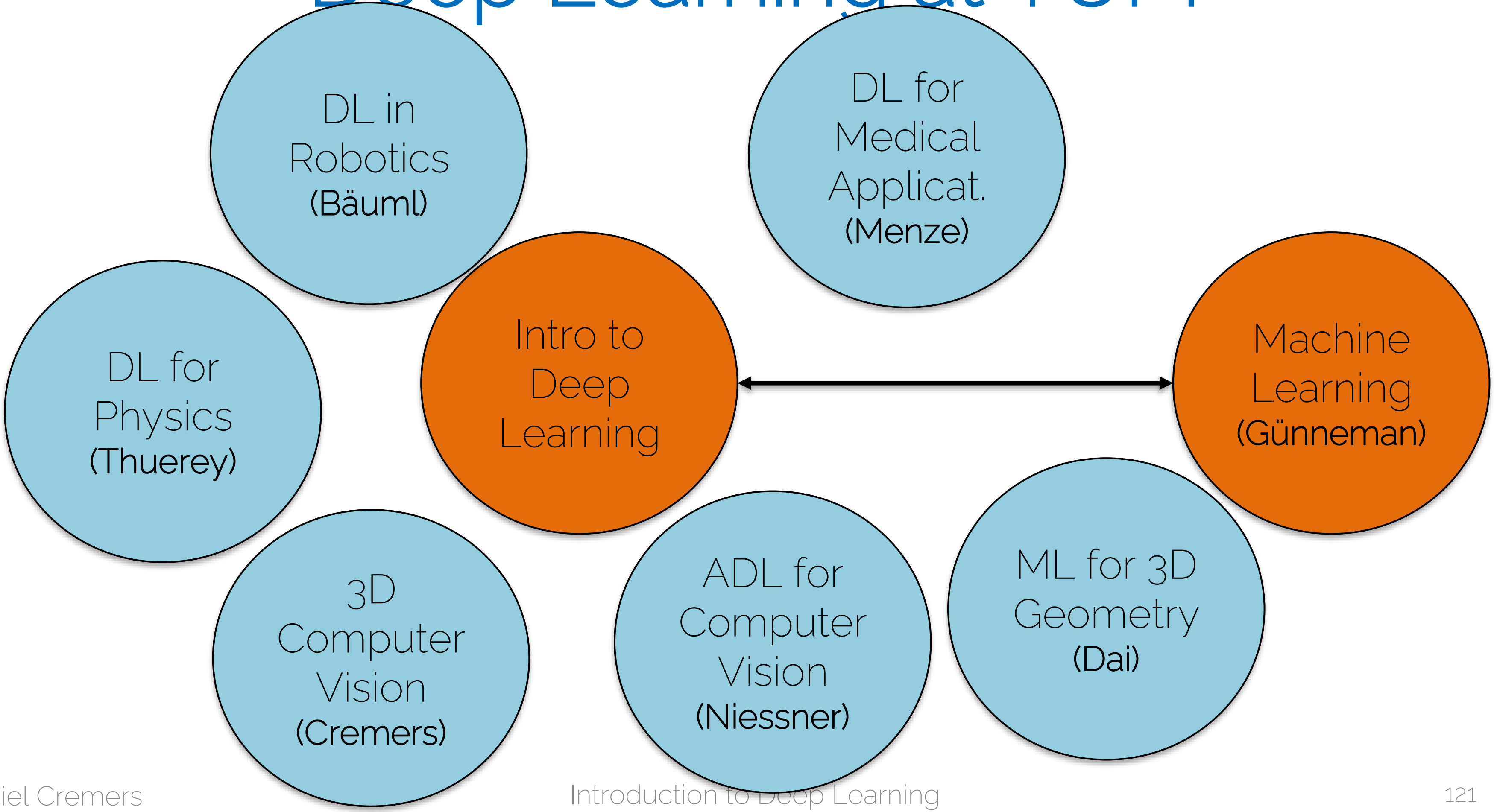
- GANs



real or fake pair?

Other DL Courses

Deep Learning at TUM



Next Dates and Exam

- Guest Lecture by Ben Poole!
  - Monday July 17th at 7pm (CEST)
  - Join via Live Stream: https://www.youtube.com/watch?v=xk-TibnYEDA

- Exam
  - There will NOT be a retake exam
  - Neither cheat sheet nor calculator during the exam

Good Luck
in the Exam ☺

References for Further Reading

- https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
- https://phillipi.github.io/pix2pix/
- http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf