Advanced DL Topics

Attention

Index the values via a differentiable operator.

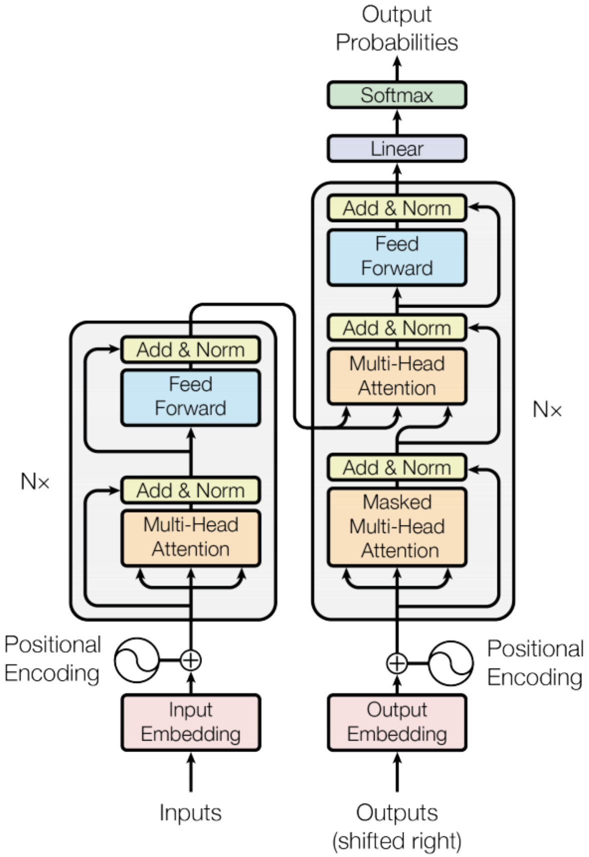
Multiply queries with keys

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



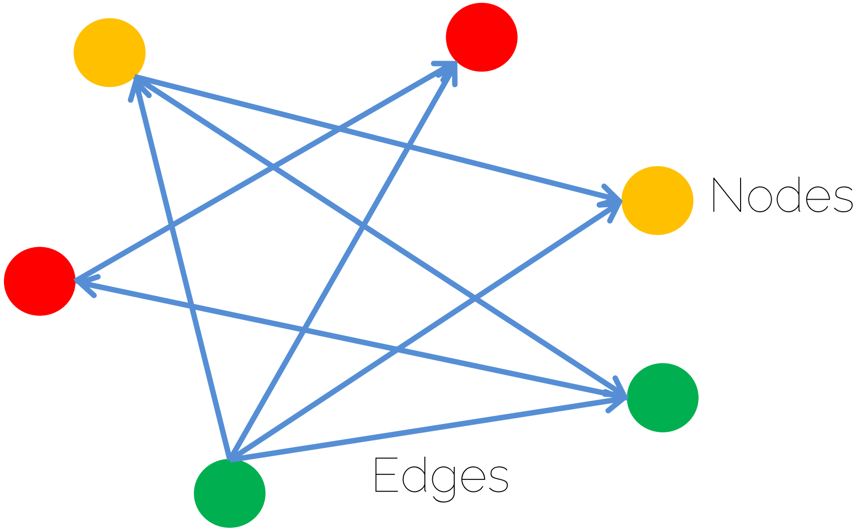
Attention Is All You Need [Vaswani et al. 17]

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

Graph Neural Networks

A graph

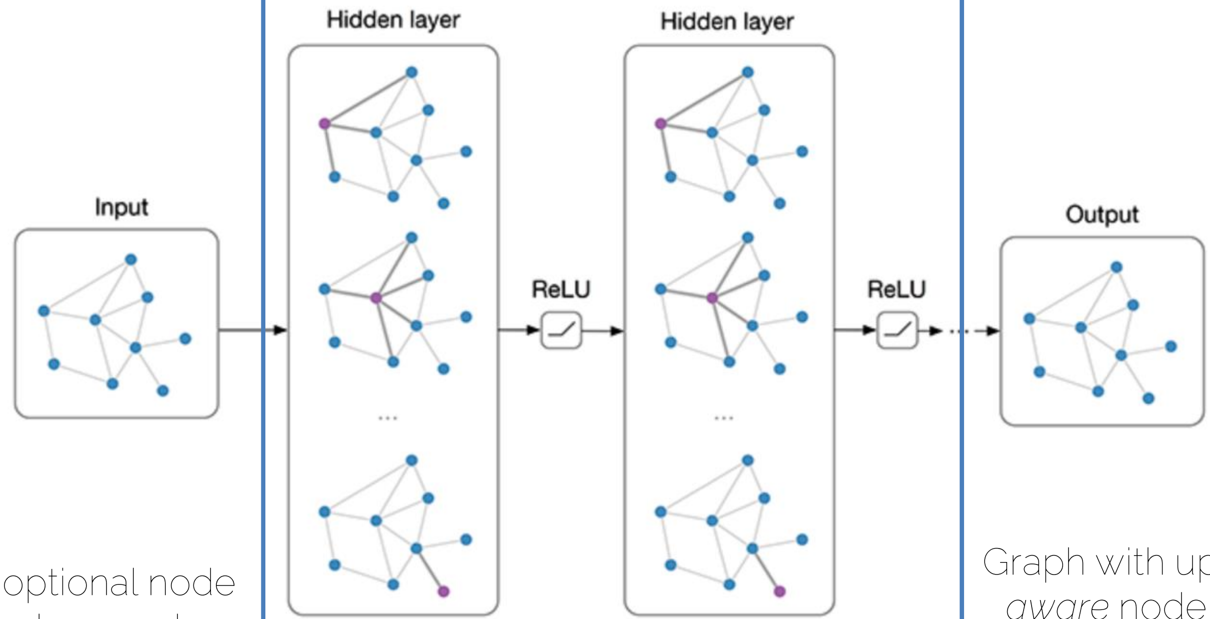
- Node: a concept
- Edge: a connection between concepts



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



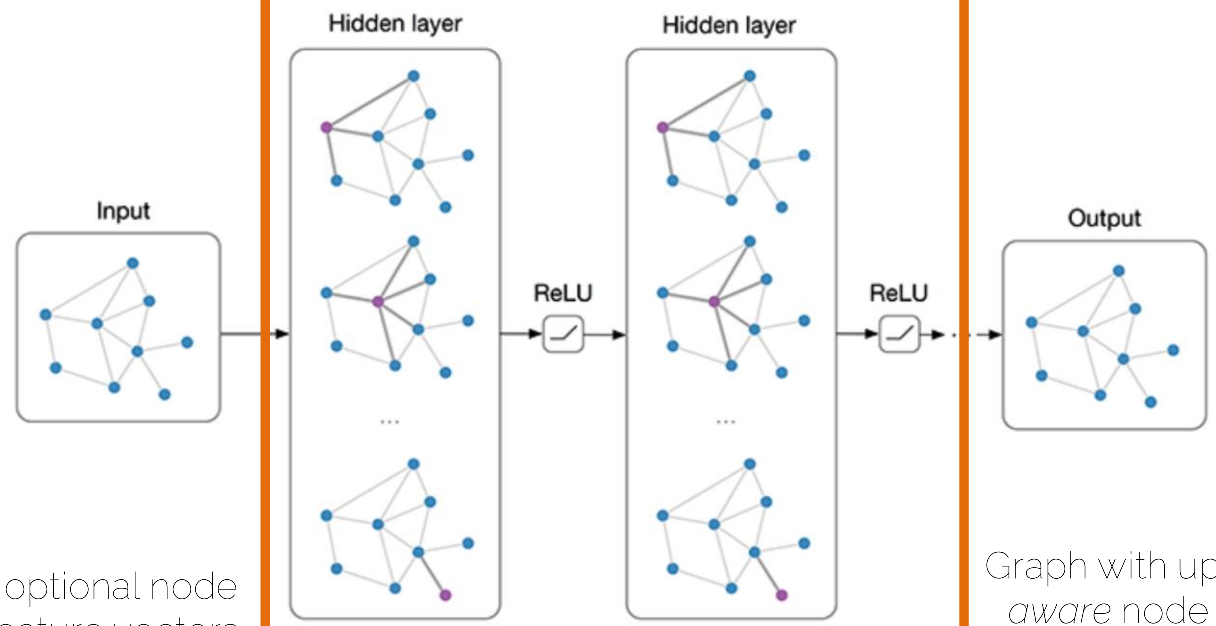
Graph with optional node and edge feature vectors

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

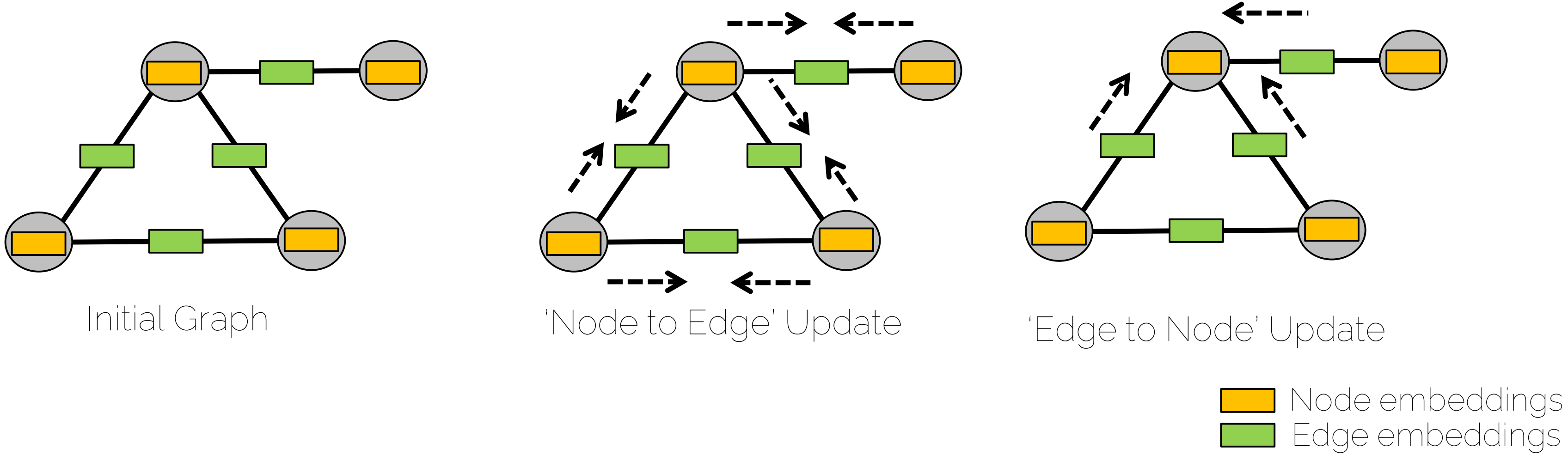
Information propagation across the graph for several iterations

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

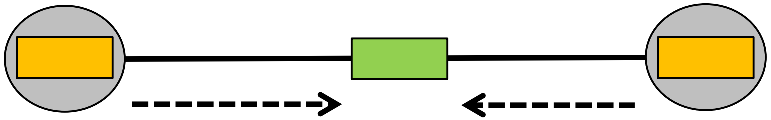
Embedding of edge (i,j) in the previous message passing step

Embedding of node j in the previous message passing step

'Node to edge' updates

- At every message passing step l, first do:

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

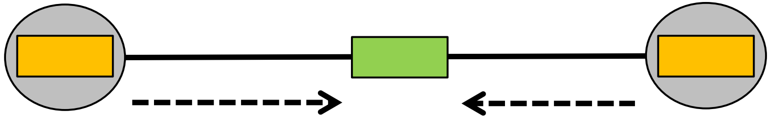


'Node to edge' updates

- At every message passing step l, first do:

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

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



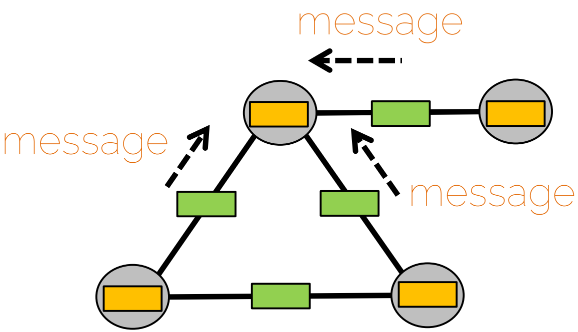
'Edge to node' updates

- 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:

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

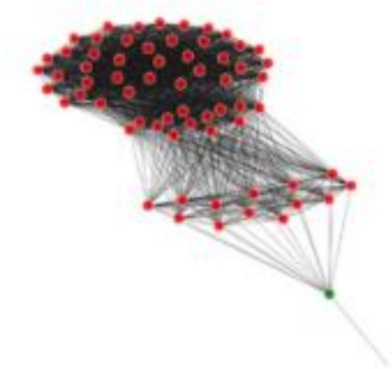
$$h_i^{(l)} = \underbrace{\mathcal{N}_v}\left(\left[m_i^{(l)}, h_i^{(l-1)}\right]\right)$$

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

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

GNN Applications

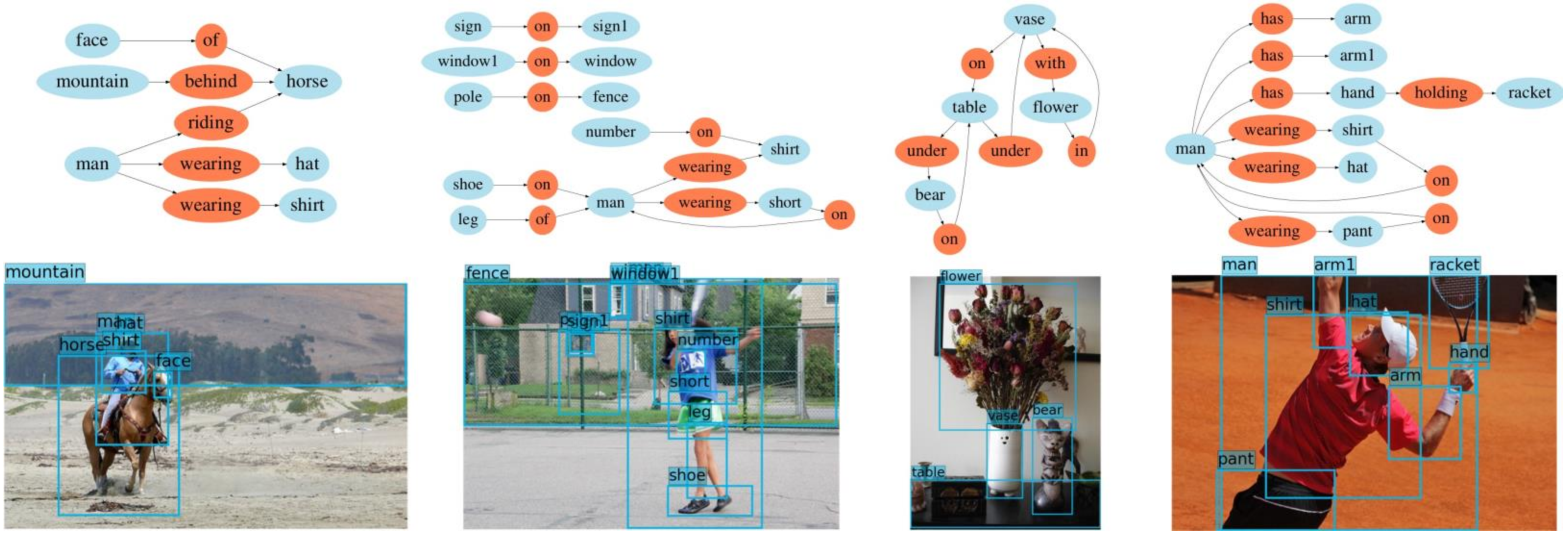
- Node or edge classification
  - identifying anomalies such as spam, fraud
  - Relationship discovery for social networks, search networks



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

GNN Applications

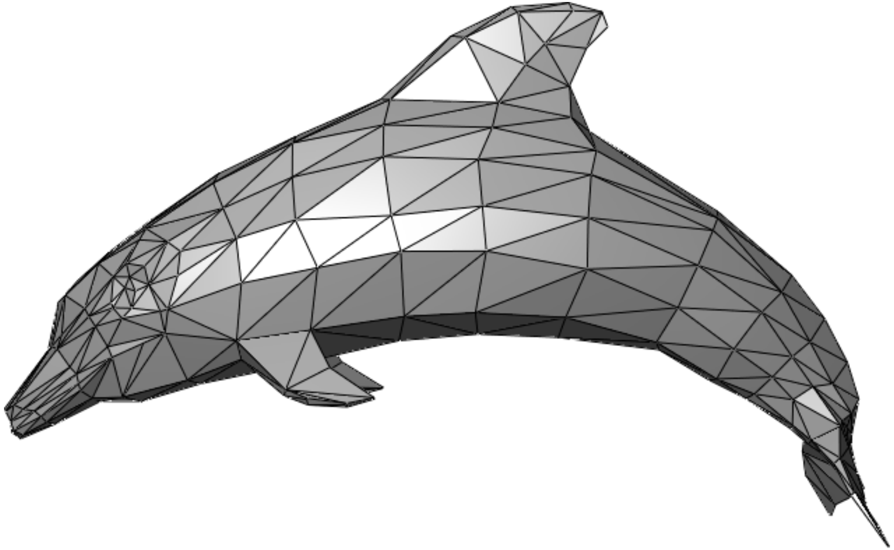
- Scene graph generation



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

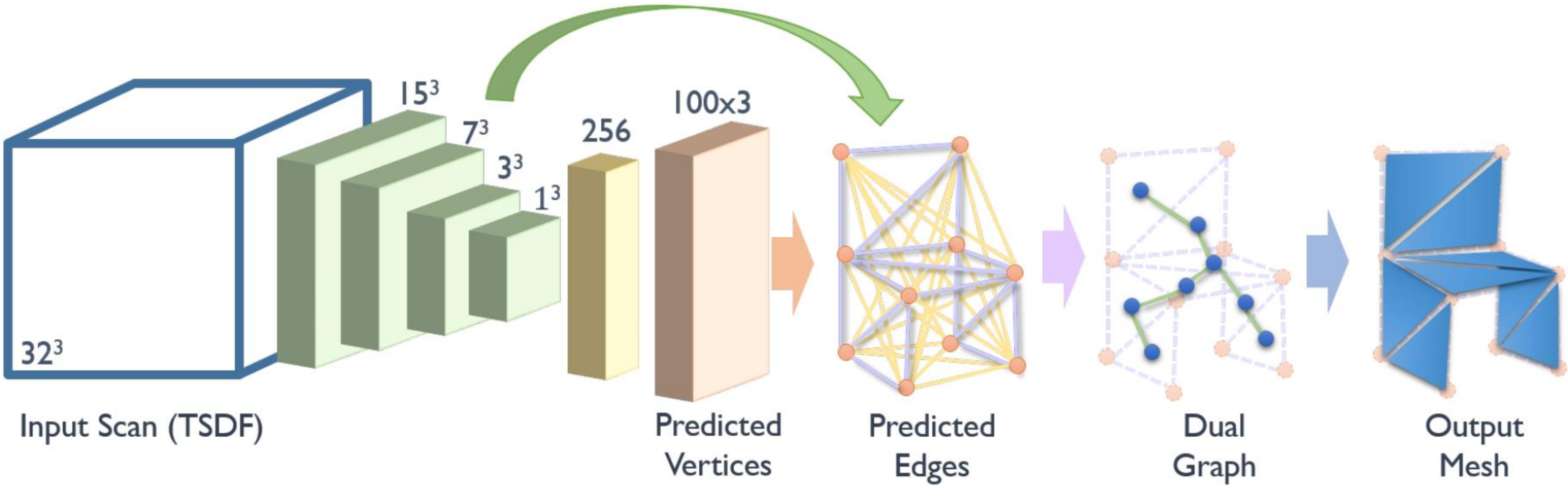
GNN Applications

- 3D Mesh Classification



GNN Applications

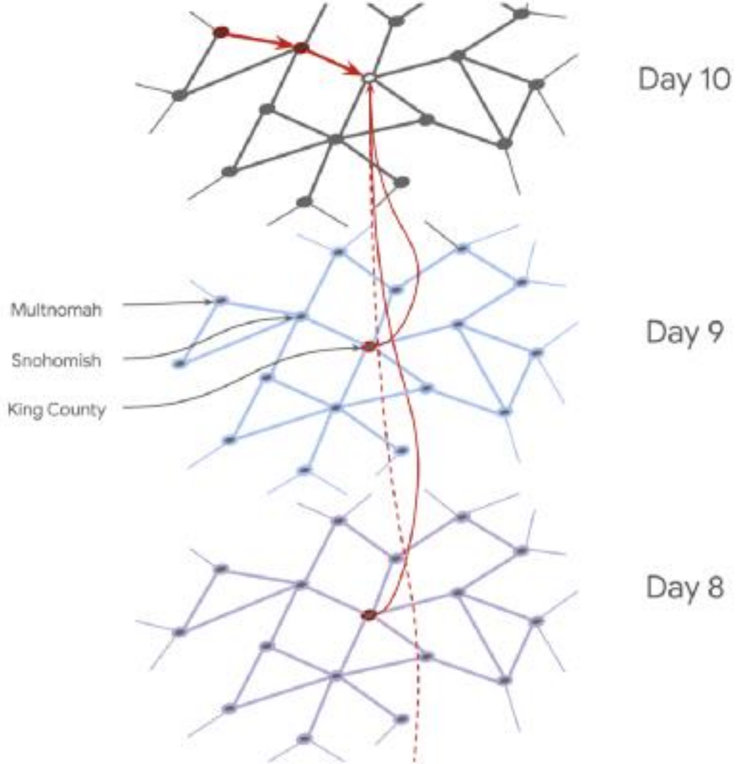
- 3D mesh generation



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

GNN Applications

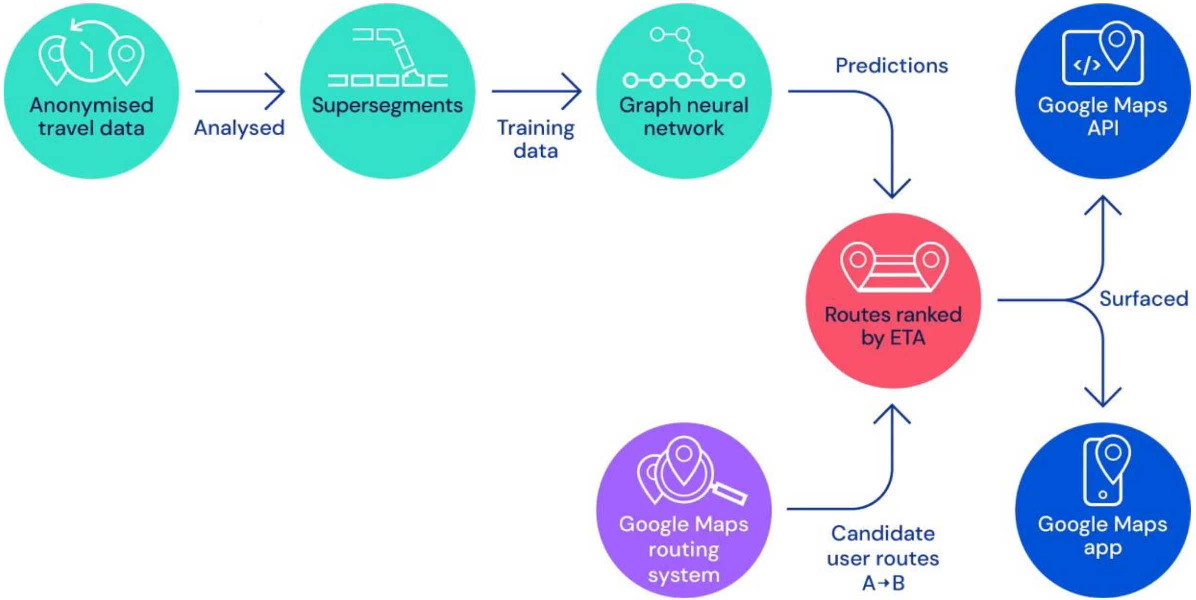
- Modeling epidemiology
  - Spatio-temporal graph



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

GNN Applications

- Traffic forecasting

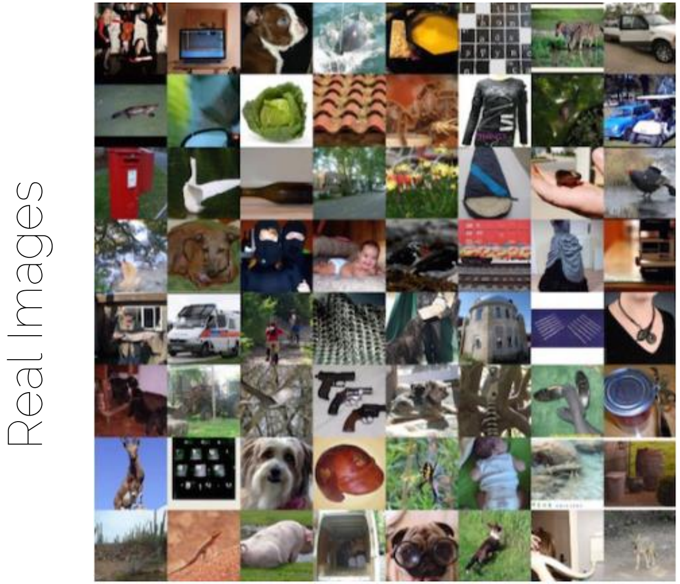


https://www.deepmind.com/blog/traffic-prediction-with-advanced-graph-neural-networks

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

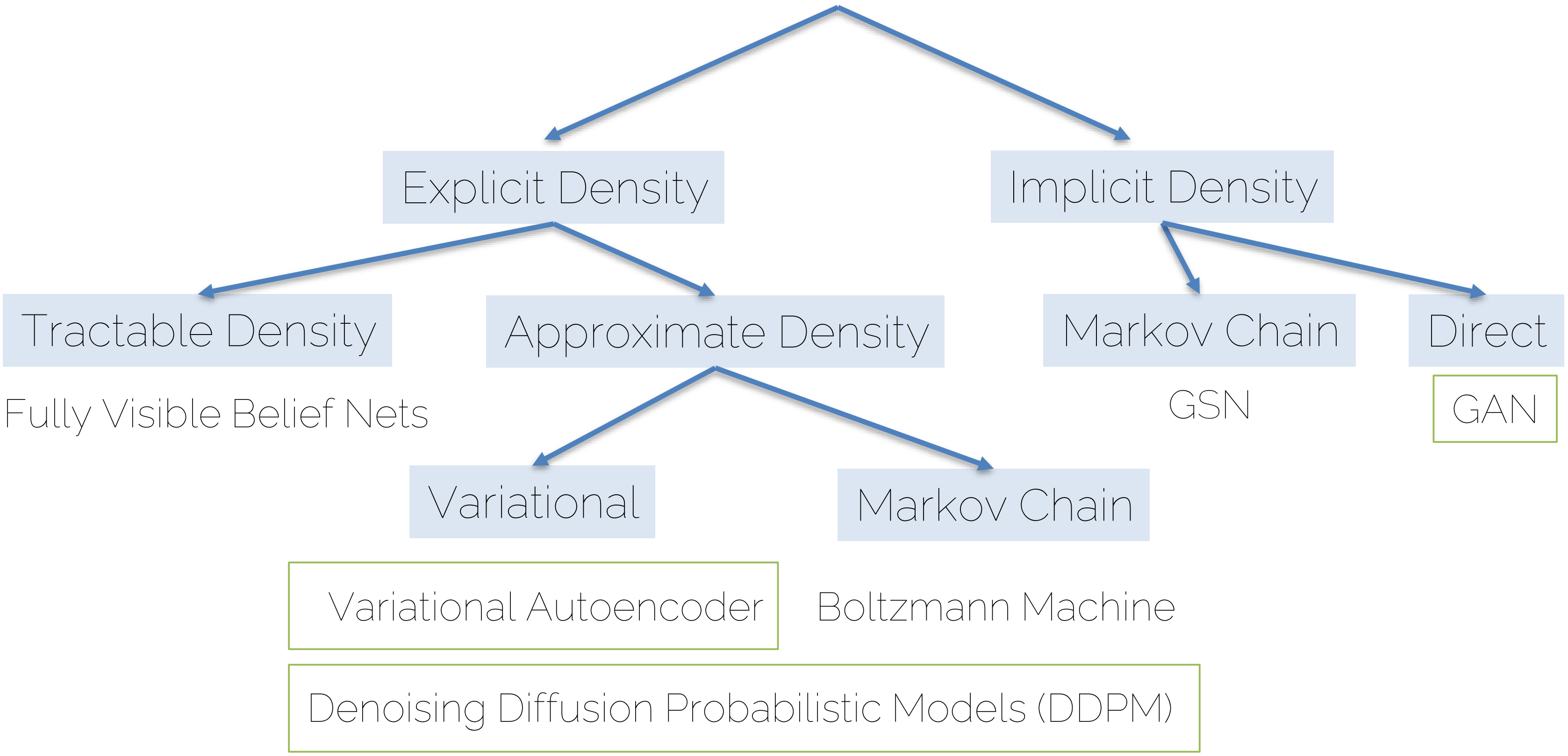


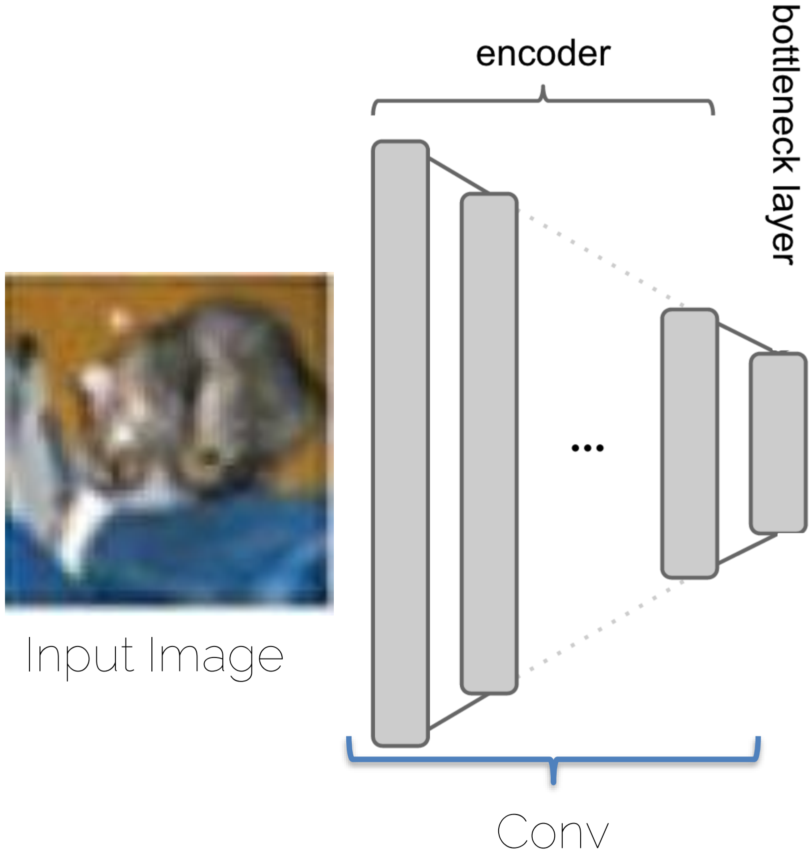
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Autoencoders & VAEs

Autoencoders

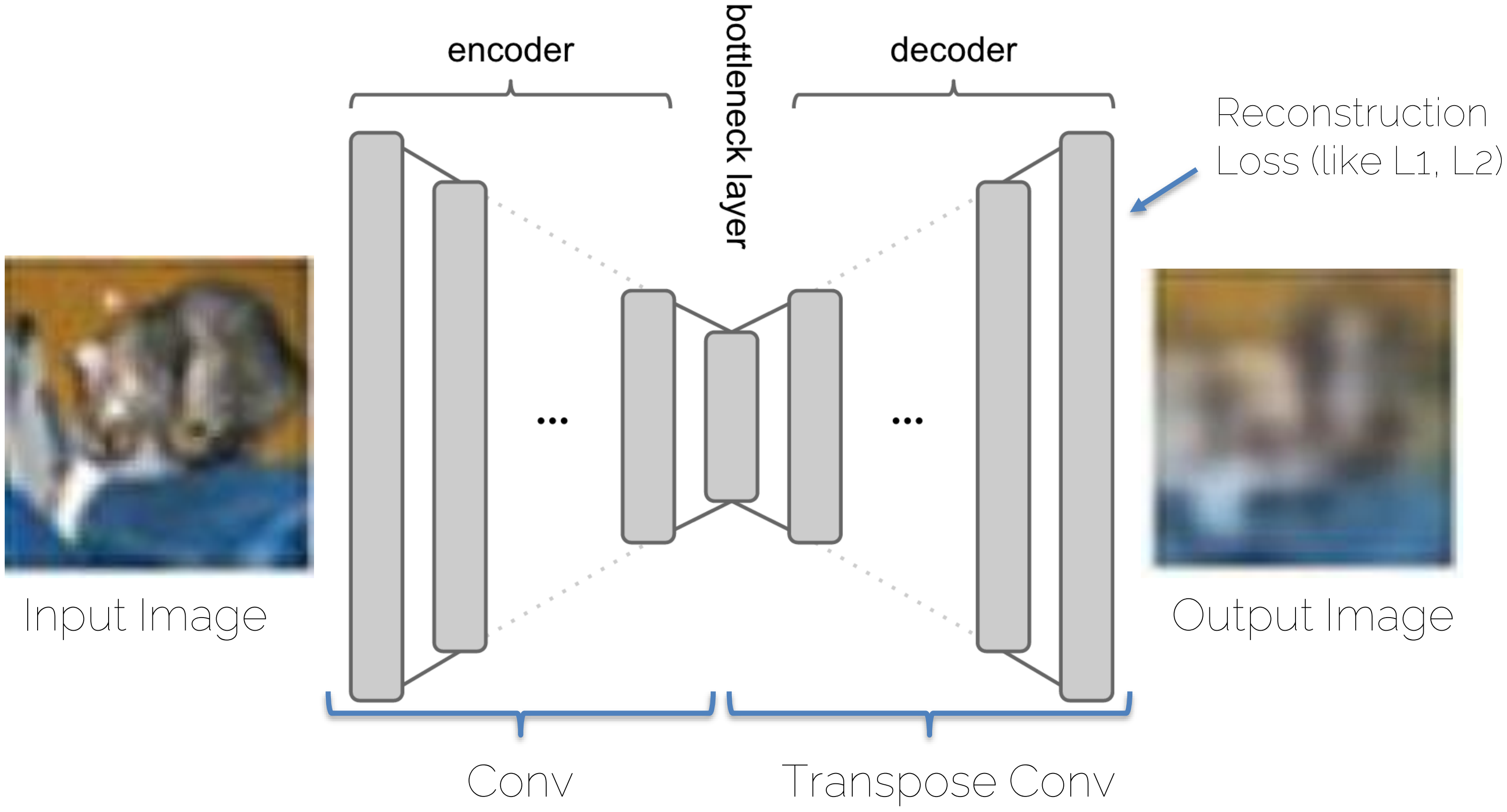
- Can be used as a basic generative models
- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

Autoencoders

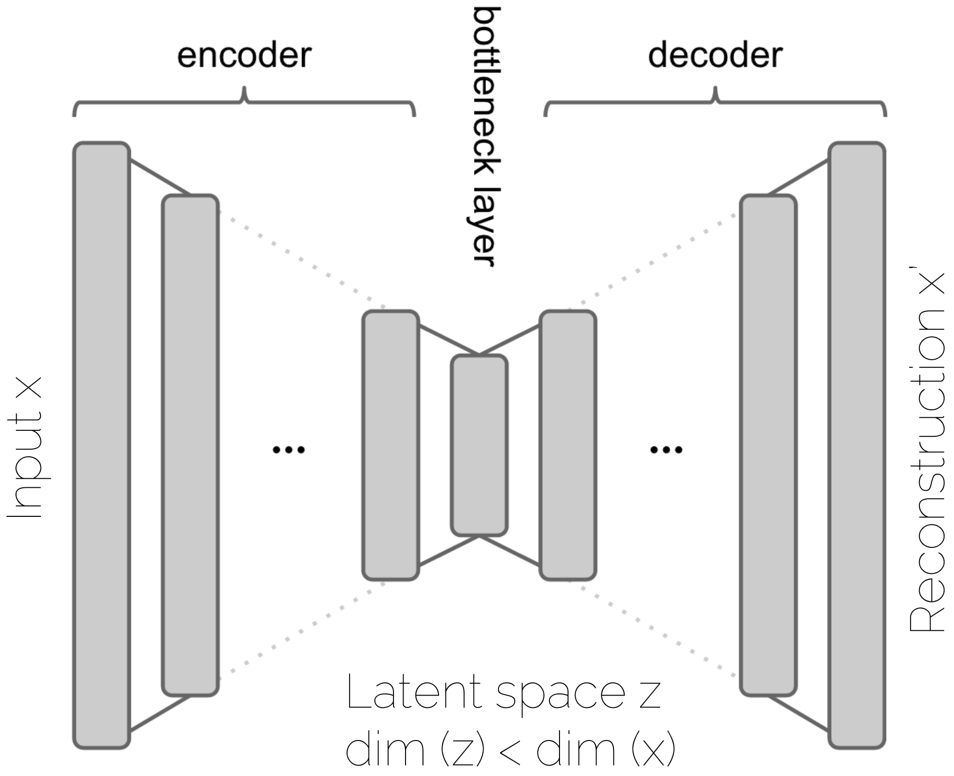


- From an input image to a feature representation (bottleneck layer)
- Encoder: a CNN in our case

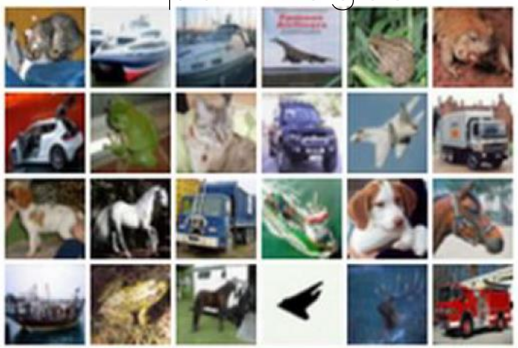
Autoencoder: training



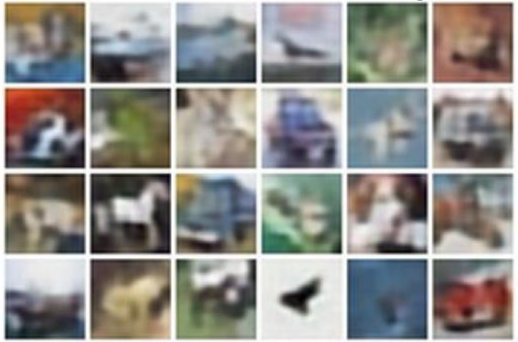
Autoencoder: training



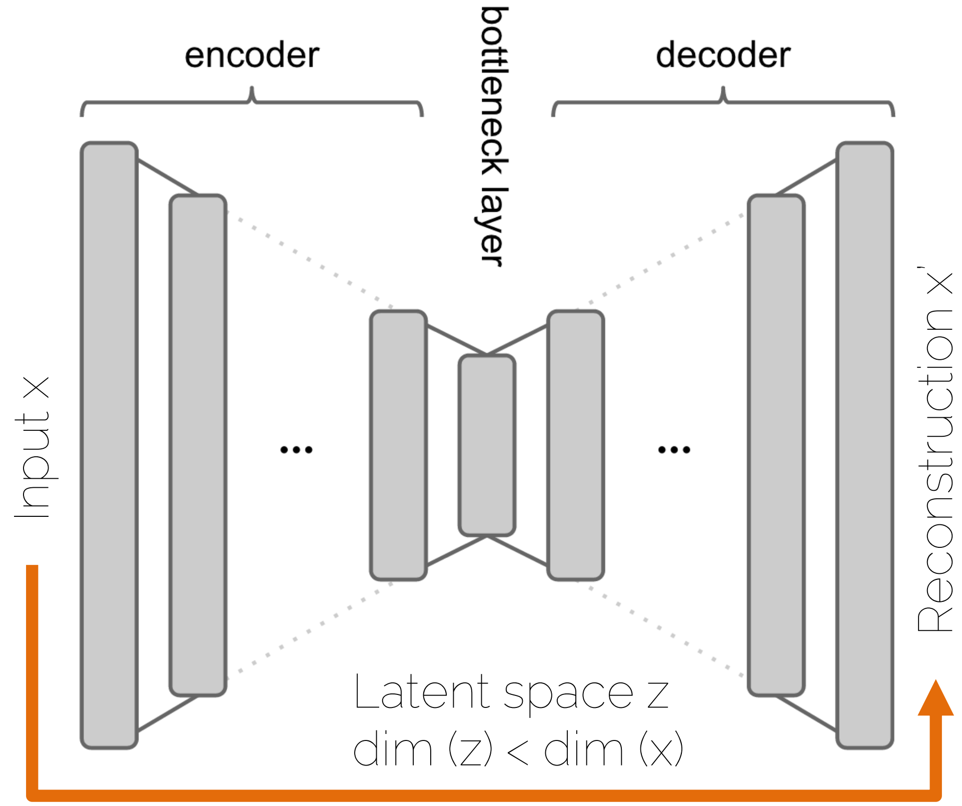
Input images



Reconstructed images

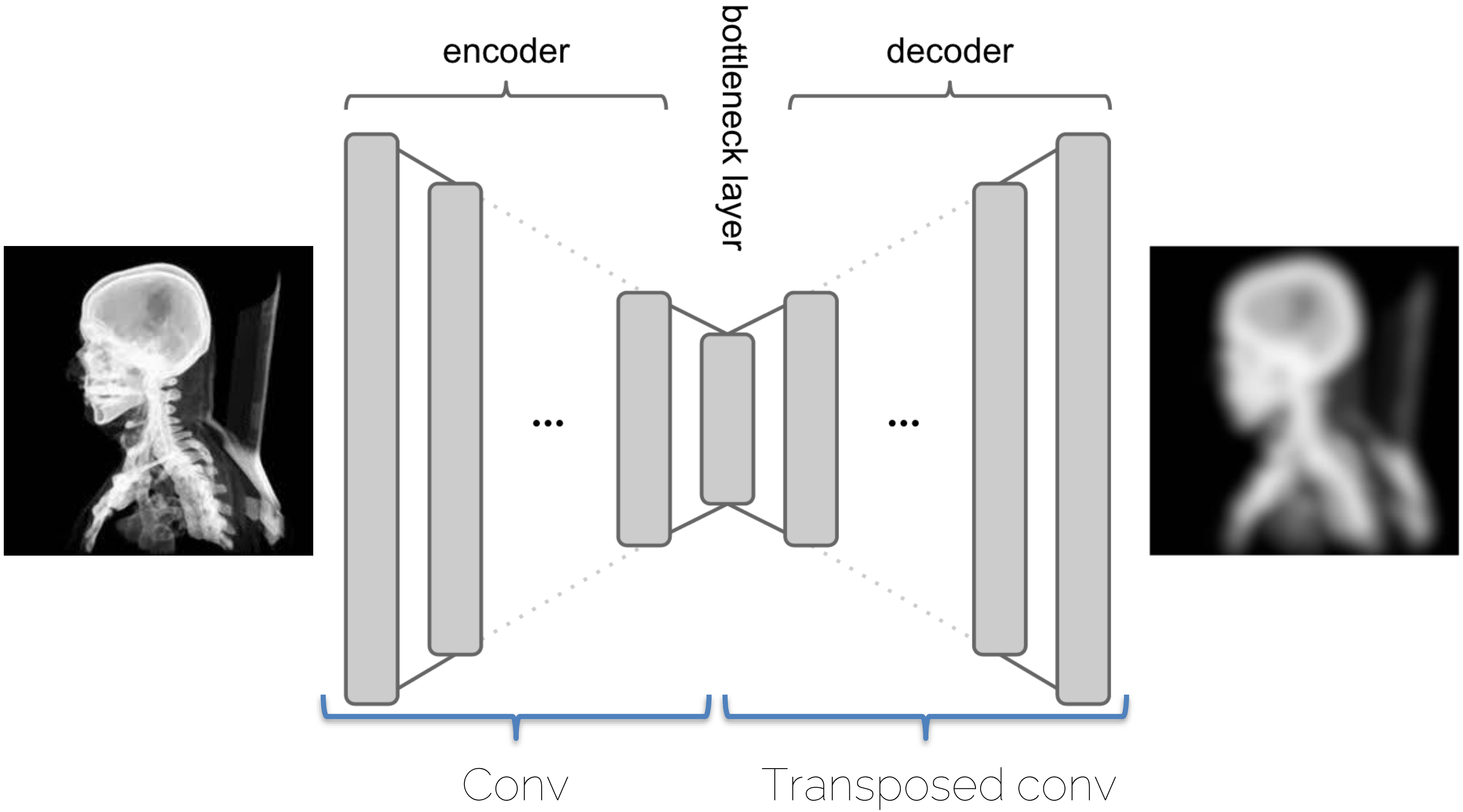


Autoencoder: training

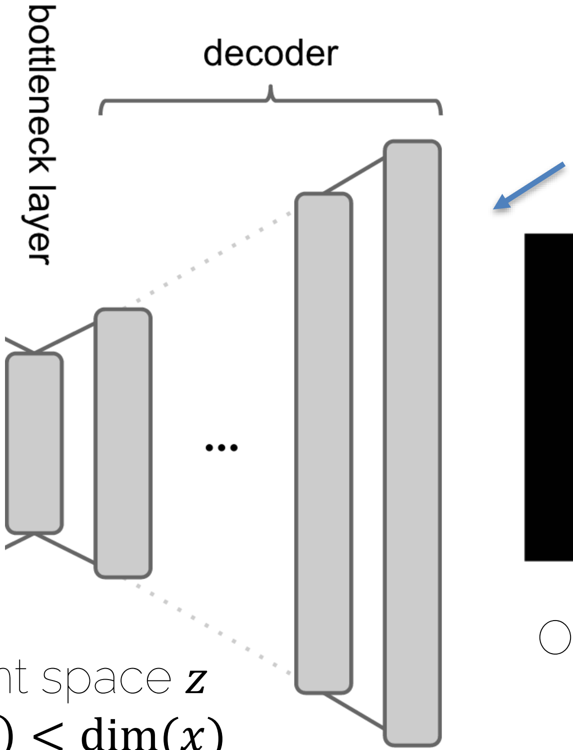


- No labels required
- We can use unlabeled data to first get its structure

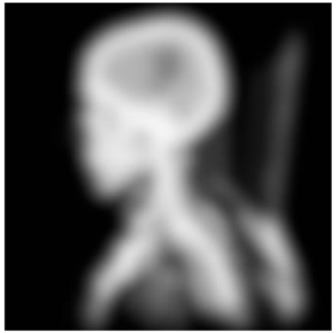
Autoencoder



Decoder as Generative Model



Reconstruction Loss (often L2)



Output Image

Test time:
-> reconstruction from 'random' vector

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

Why using autoencoders?

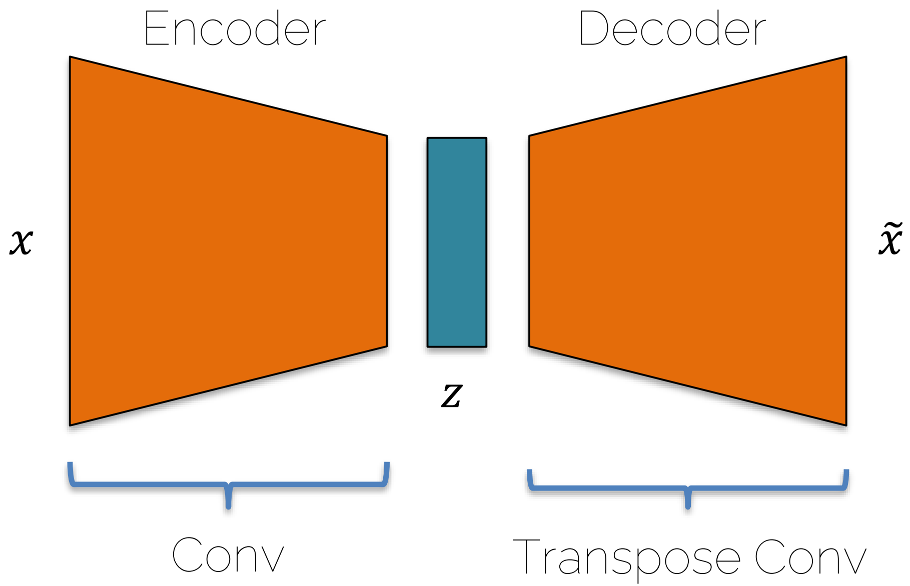
- Use 1: pre-training, as mentioned before
  - Image → same image reconstructed
  - Use the encoder as "feature extractor"

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

Variational Autoencoders

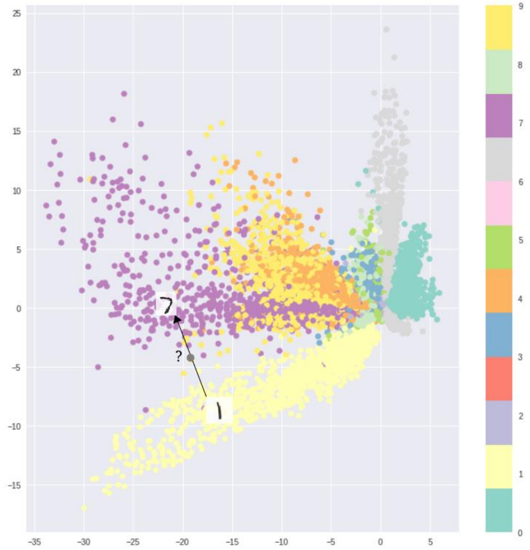
Autoencoders

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



Autoencoders

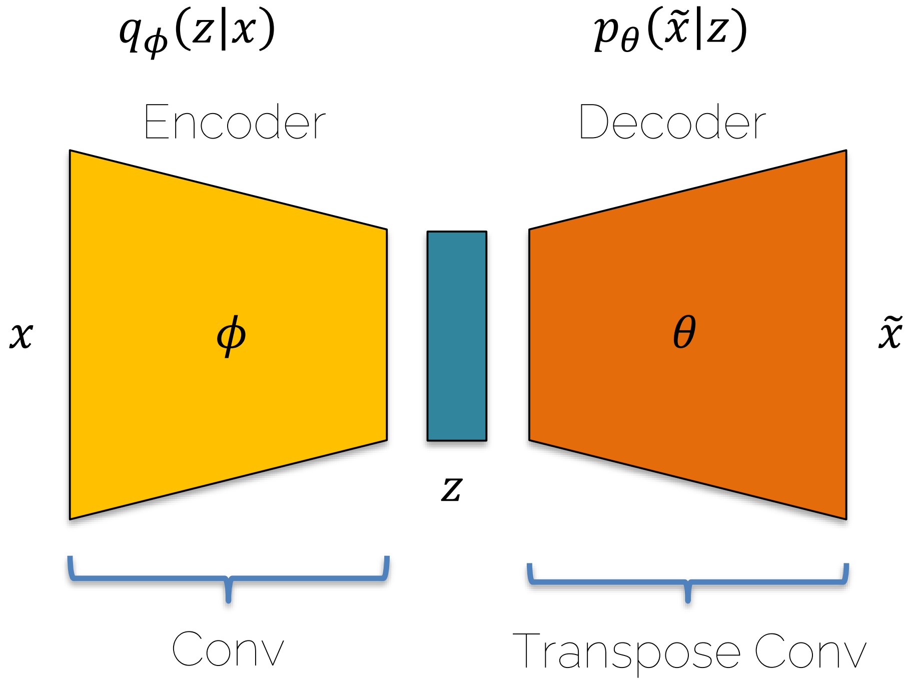
- Encode the input into a representation (bottleneck) and reconstruct it with the decoder



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

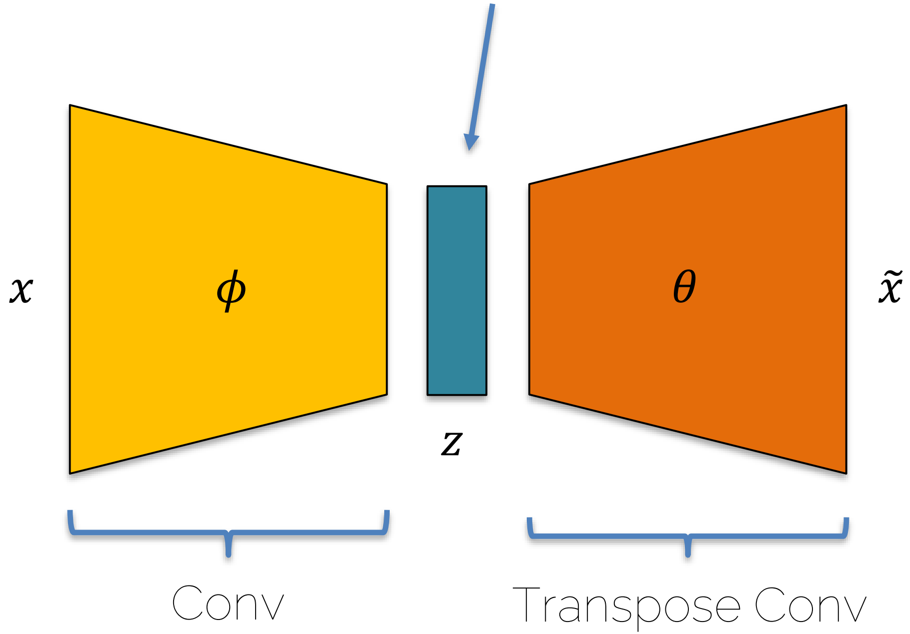
Latent space learned by autoencoder on MNIST

Variational Autoencoder



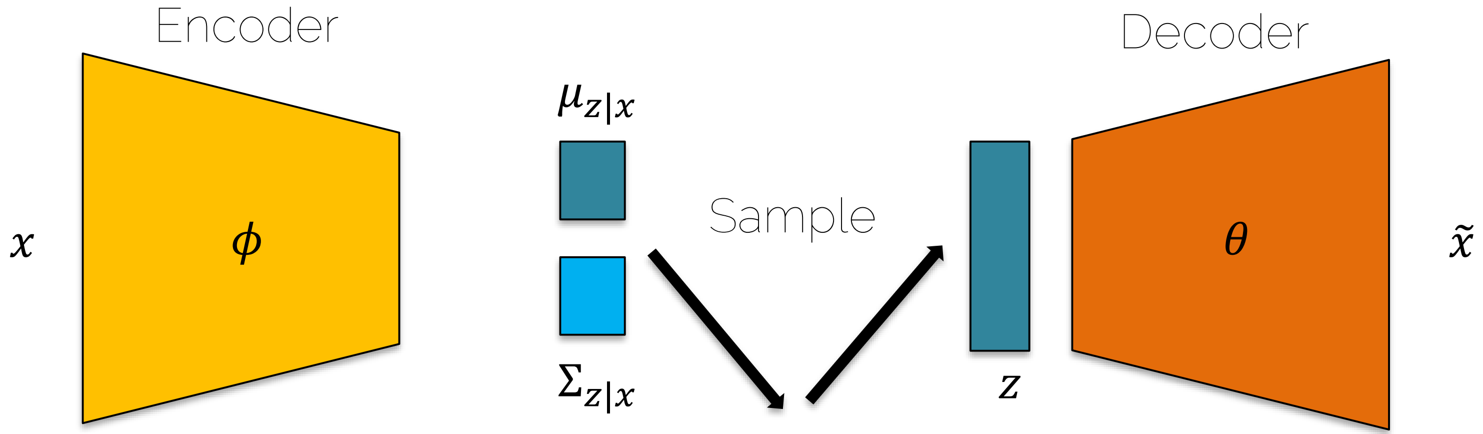
Variational Autoencoder

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



Variational Autoencoder

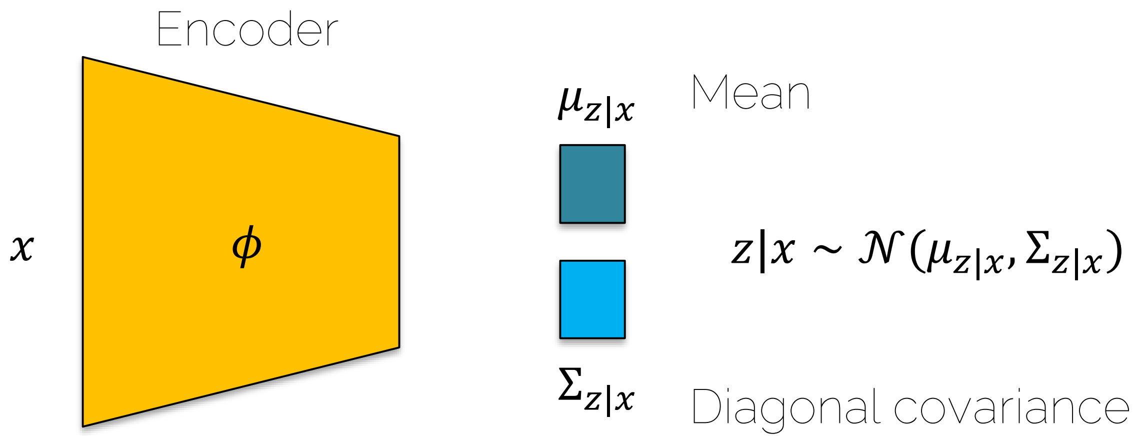
- Latent space is now a distribution
- Specifically it is a Gaussian



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

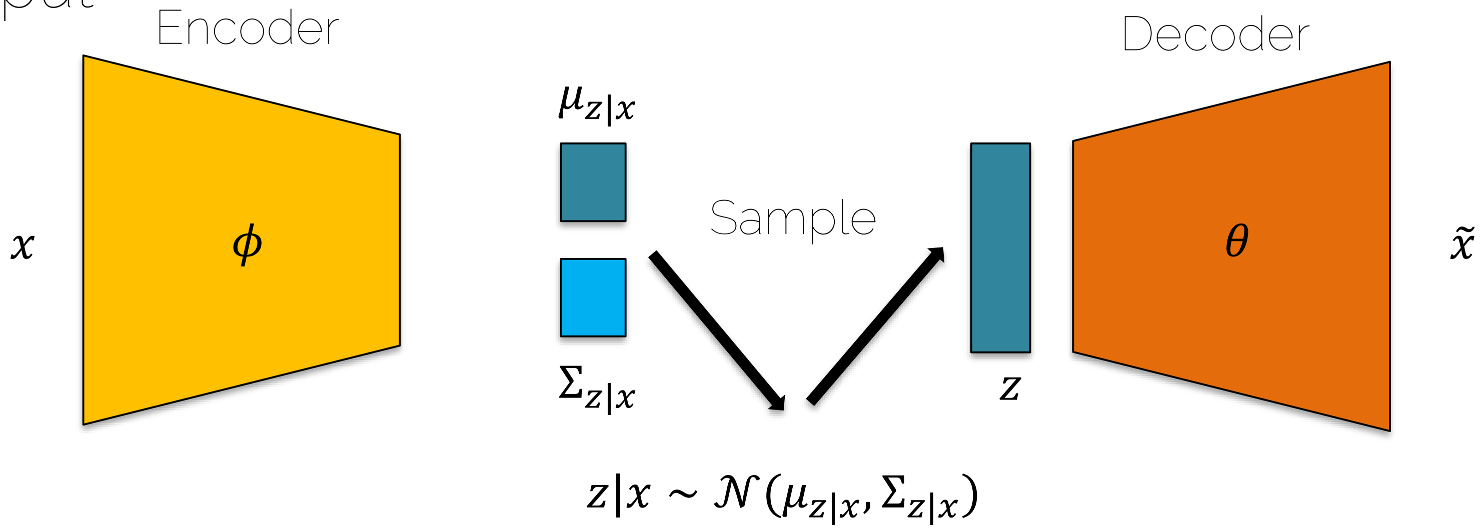
Variational Autoencoder

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



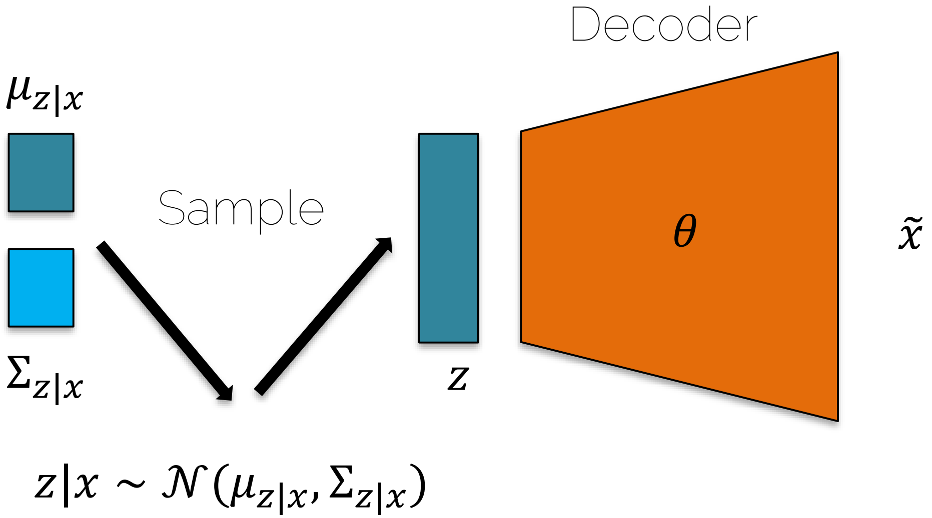
Variational Autoencoder

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



Variational Autoencoder

- Test: Sample from the latent space



Autoencoder vs VAE


Autoencoder


Variational Autoencoder


Ground Truth

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

Generating data

Degree of smile

Head pose

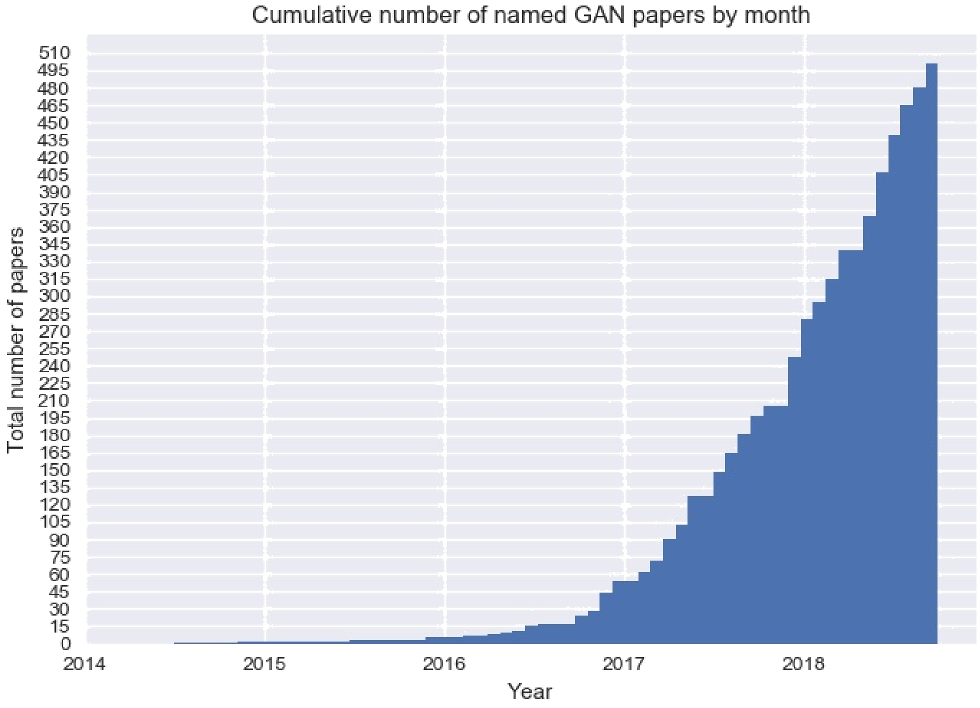
Autoencoder Overview

- 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

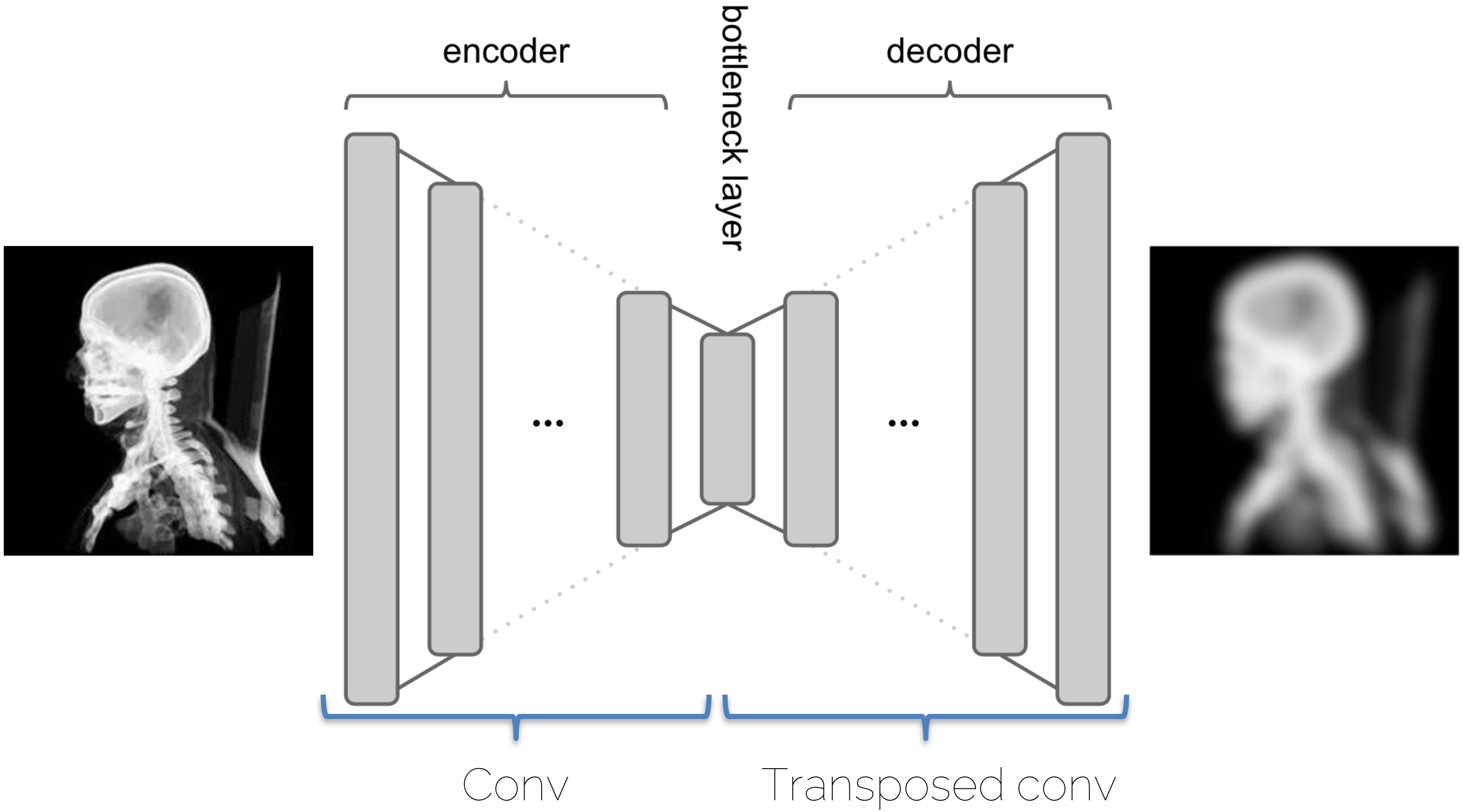
Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)



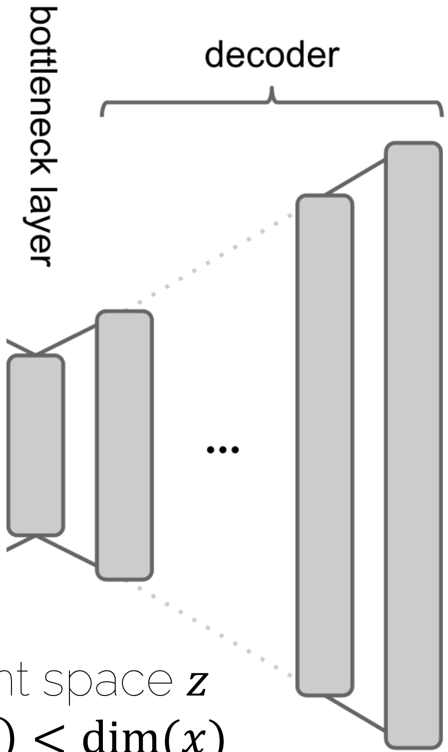
Source: https://github.com/hindupuravinash/the-gan-zoo

Autoencoder



Decoder as Generative Model

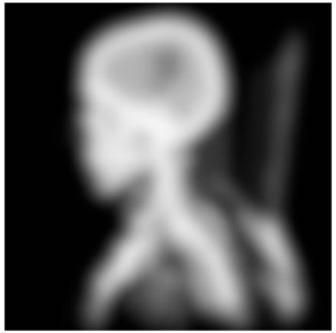
Test time:
-> reconstruction from 'random' vector



Latent space z

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

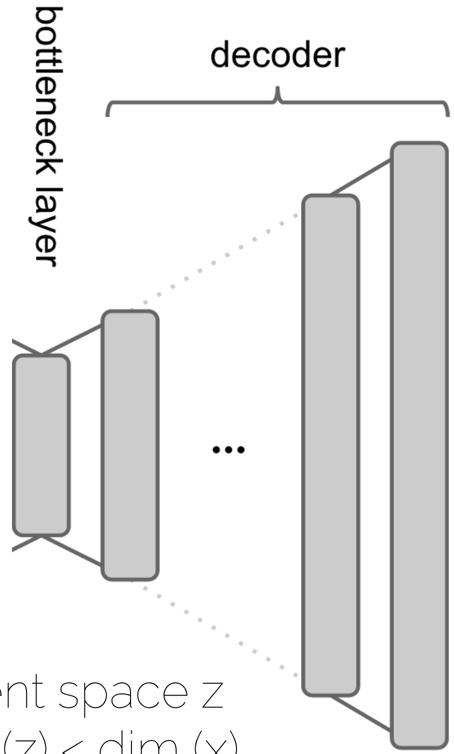
Reconstruction Loss (often L2)



Output Image

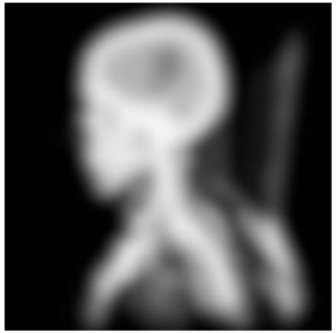
Decoder as Generative Model

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



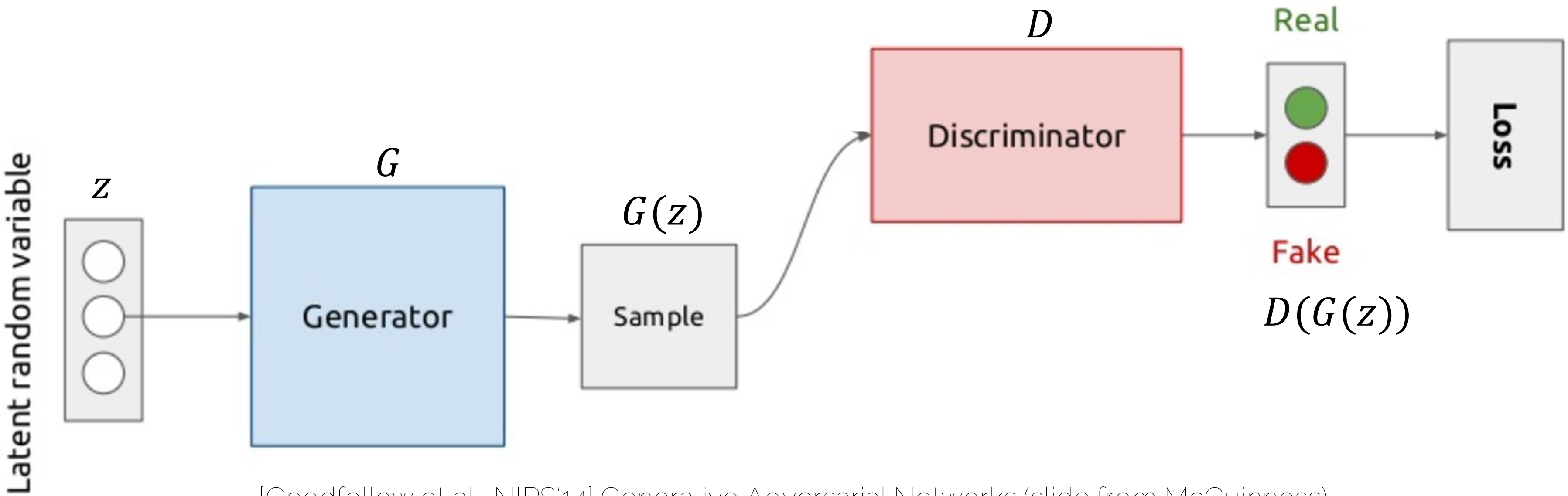
Latent space z
dim (z) < dim (x)

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



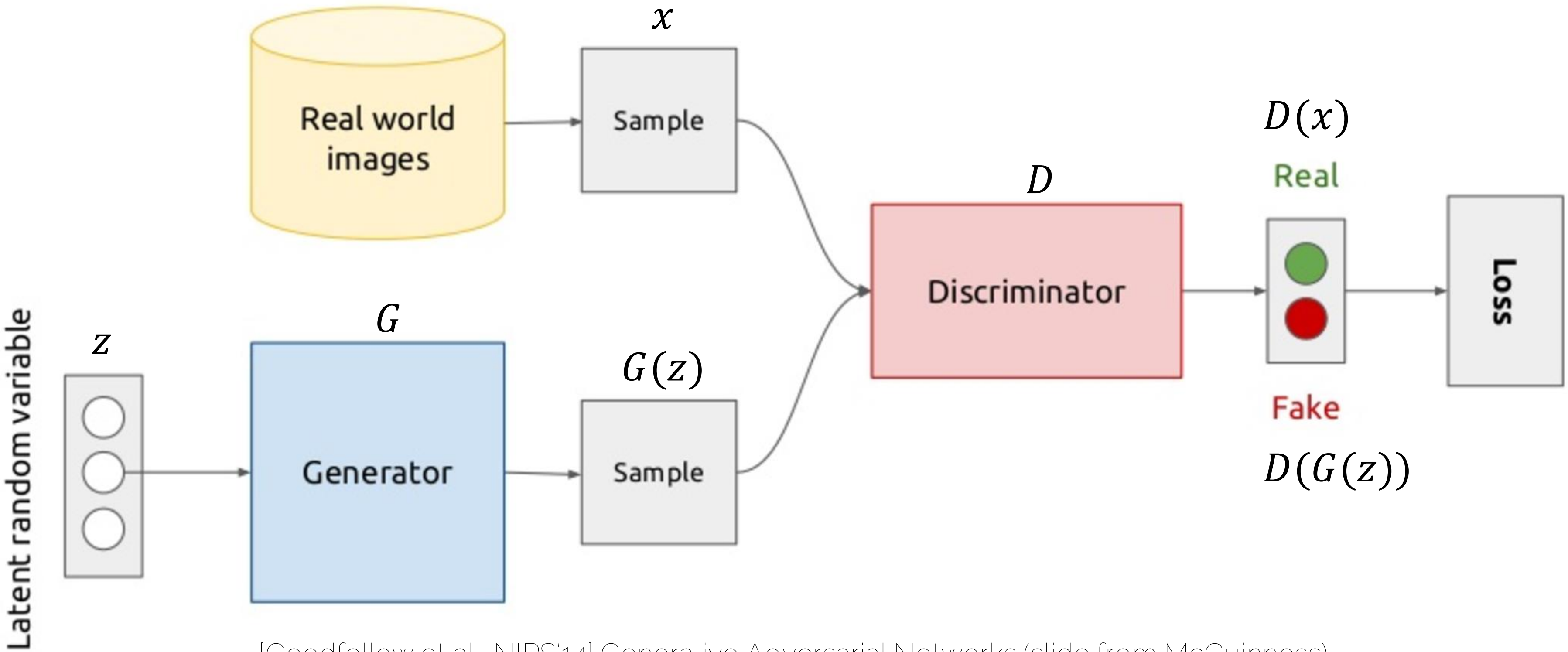
Instead of L2, can we "learn" a loss function?

Generative Adversarial Networks (GANs)



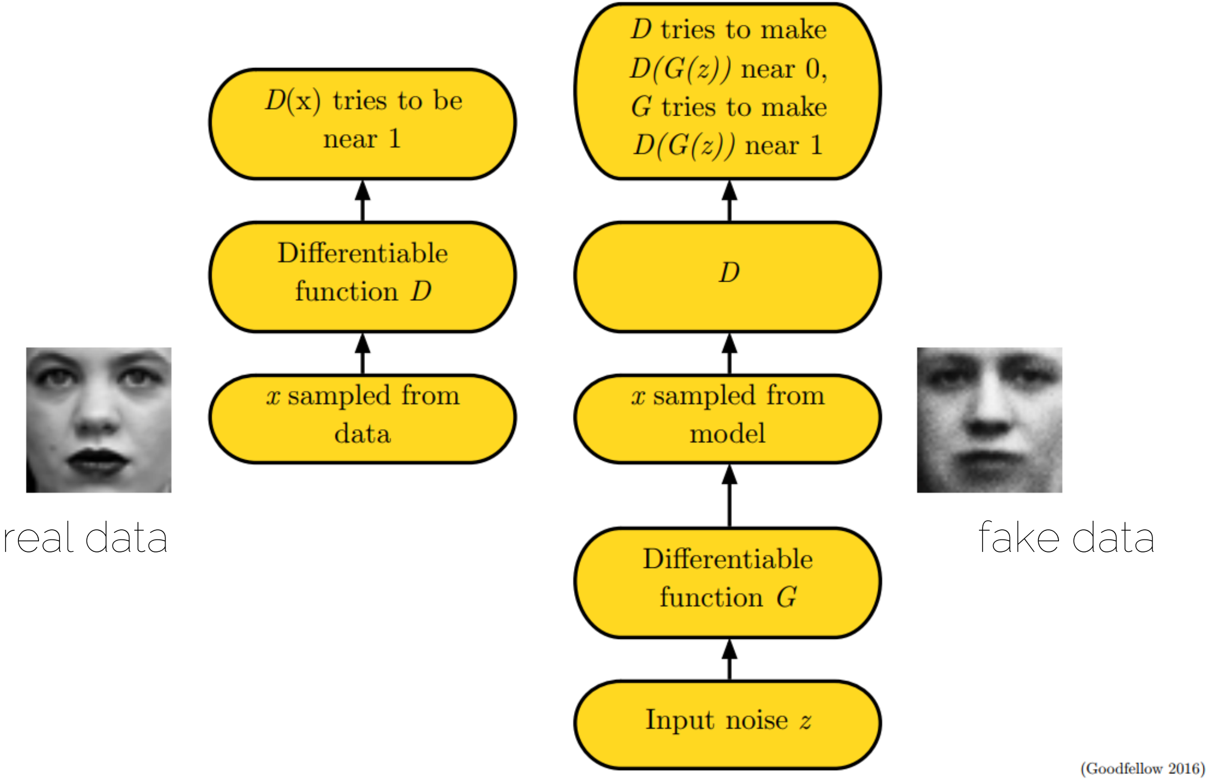
[Goodfellow et al., NIPS'14] Generative Adversarial Networks (slide from McGuinness)

Generative Adversarial Networks (GANs)



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

Generative Adversarial Networks (GANs)



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

GANs: Loss Functions

- Discriminator loss

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

- 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

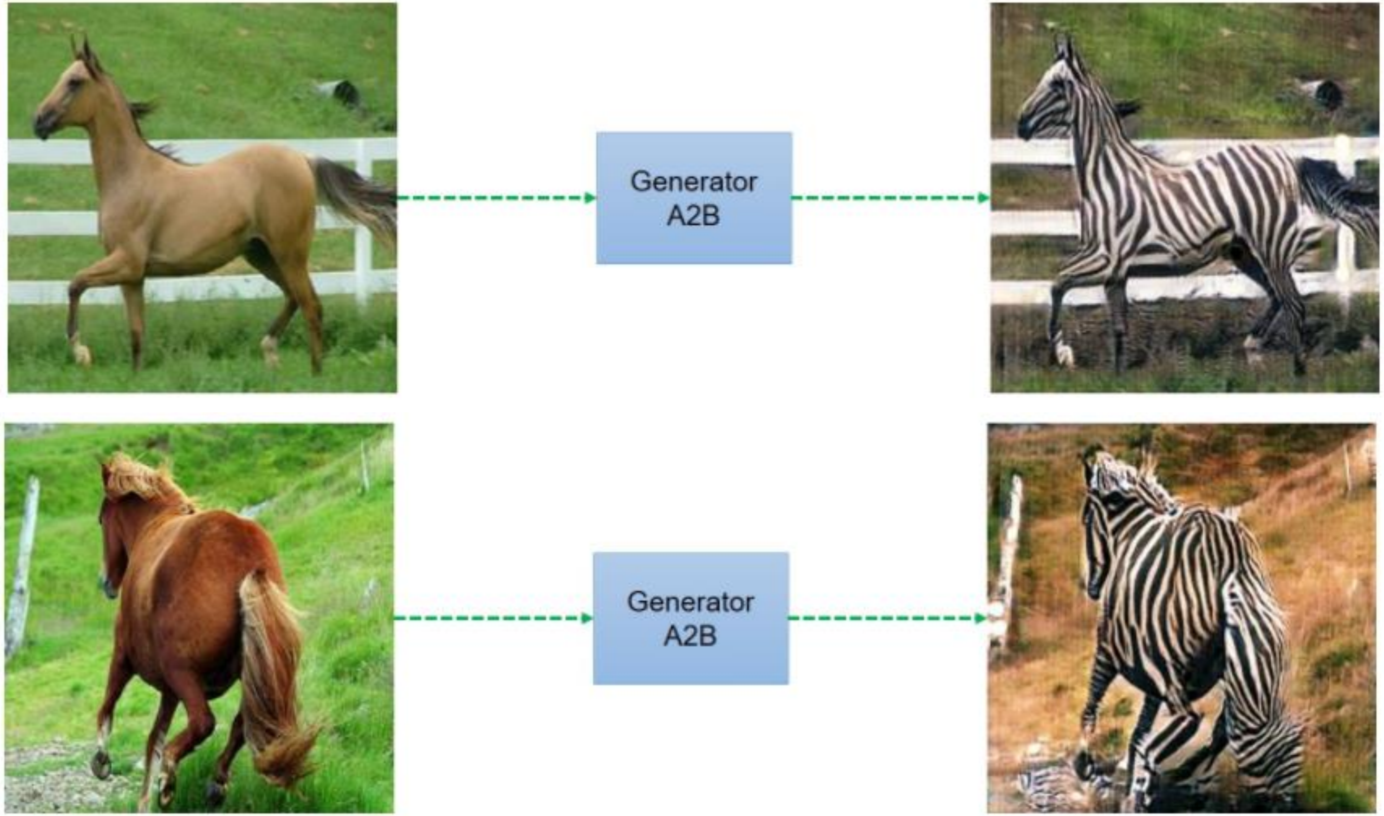
BigGAN: HD Image Generation

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

StyleGAN: Face Image Generation

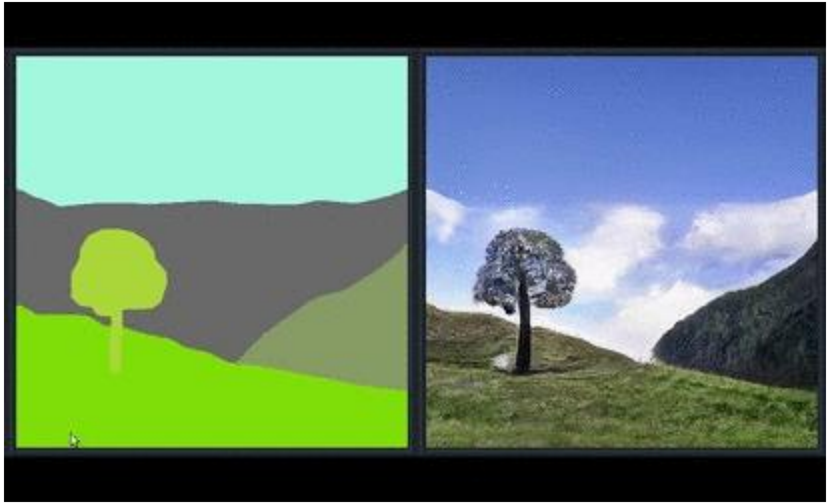
[Karras et al., '18] StyleGAN : A Style-Based Generator Architecture for Generative Adversarial Networks
[Karras et al., '19] StyleGAN2 : Analyzing and Improving the Image Quality of StyleGAN

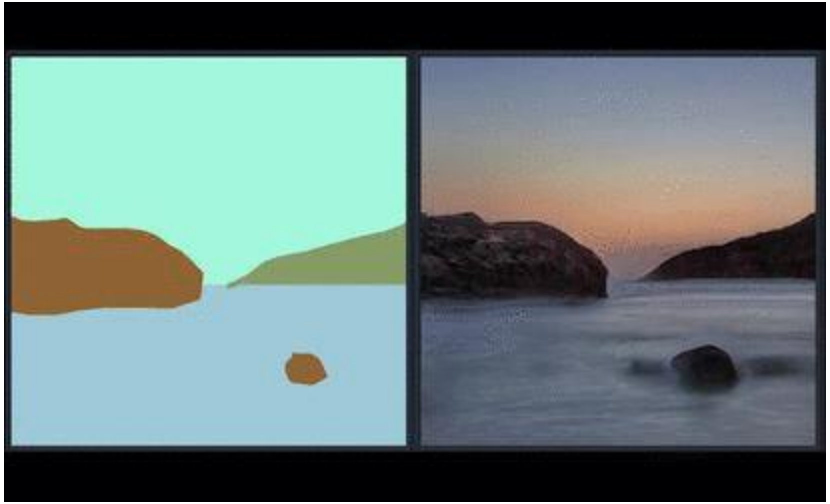
Cycle GAN: Unpaired Image-to-Image Translation



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

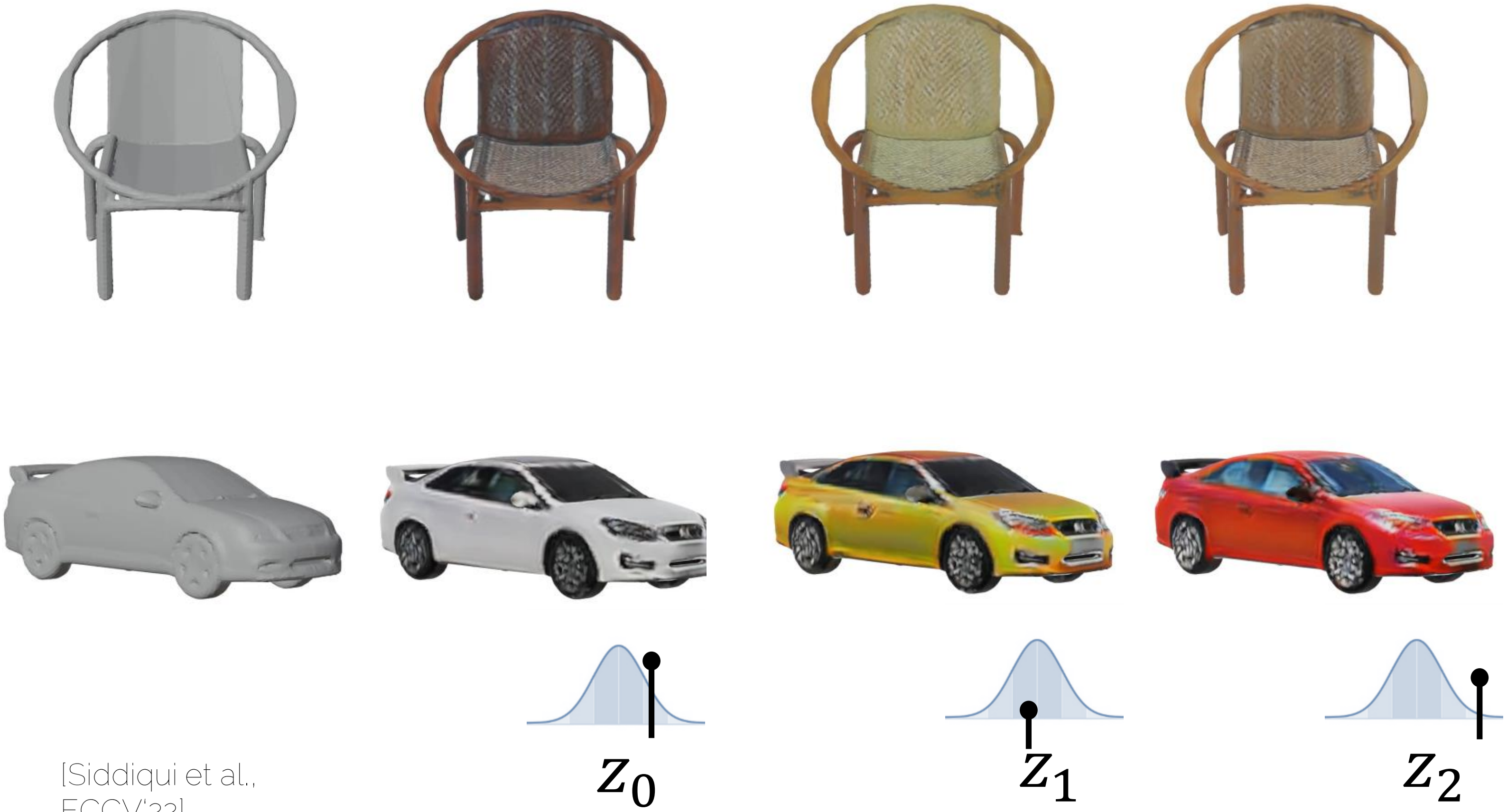
SPADE: GAN-Based Image Editing





[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

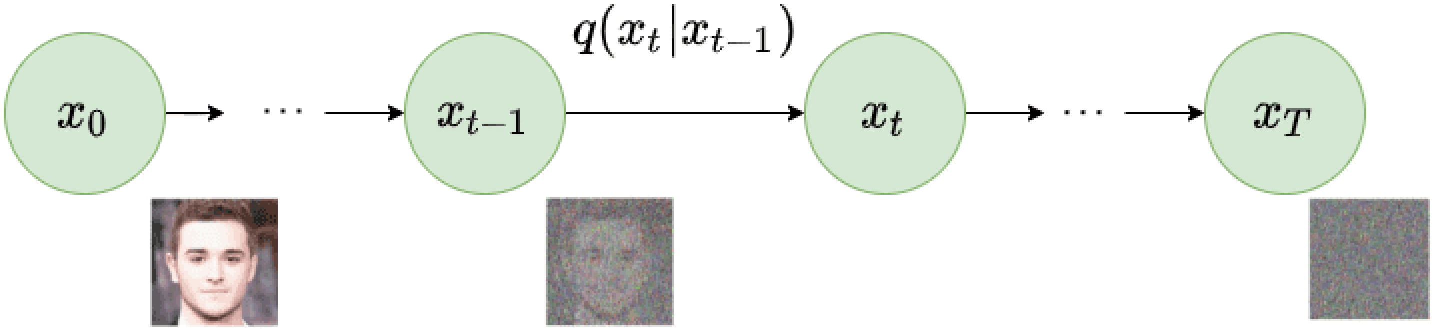
Diffusion Models

- 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

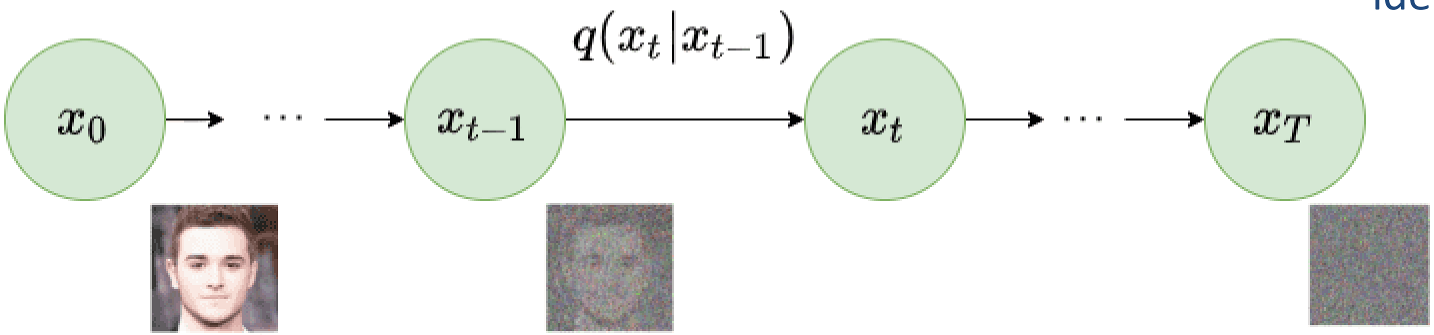
- Markov chain of $\boldsymbol{T}$ steps
  - Each step depends only on previous
- Adds noise to $\boldsymbol{x_0}$ sampled from real distribution $\boldsymbol{q(x)}$

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

mean → $\boldsymbol{\mu}_t$

variance → $\boldsymbol{\Sigma}_t$

identity matrix → $\mathbf{I}$



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

Forward Diffusion

- Go from x_0 to 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

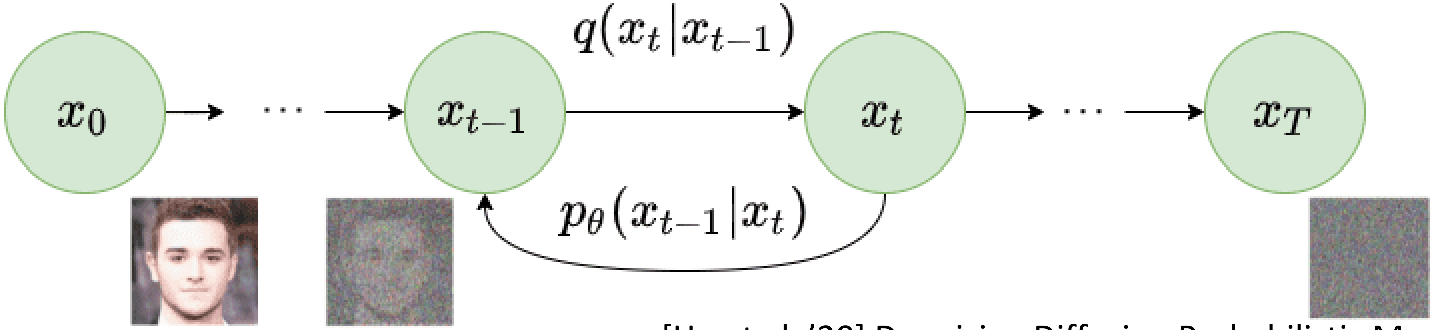
- How to model reverse process?

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

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

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

Training a Diffusion Model

- $L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$
- 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

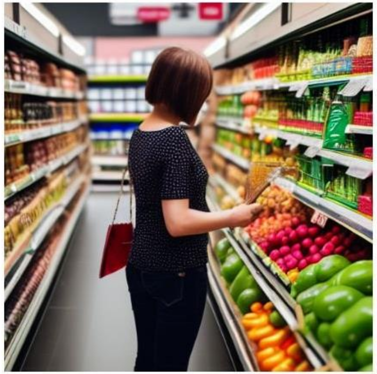
- Input and output dimensions must match
- Highly flexible to architecture design
- Commonly implemented with U-Net architectures

Applications for Diffusion Models

- Text-to-image

Oil Painting

Digital Illustration

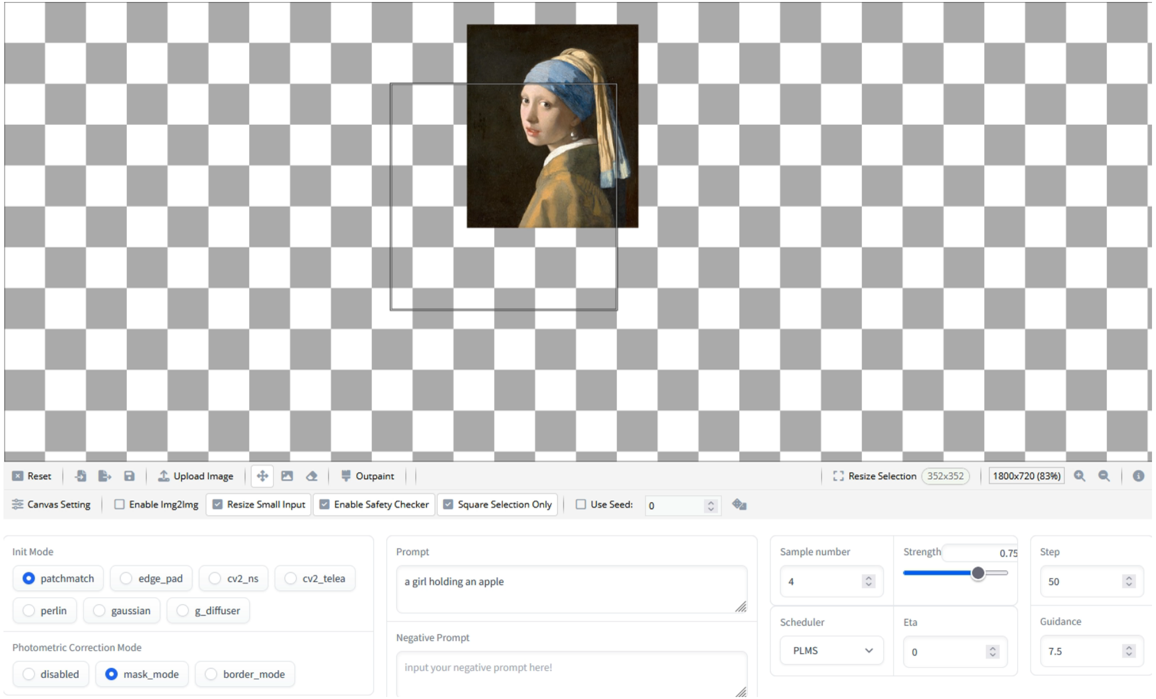


Hyperrealistic

Cartoon

Applications for Diffusion Models

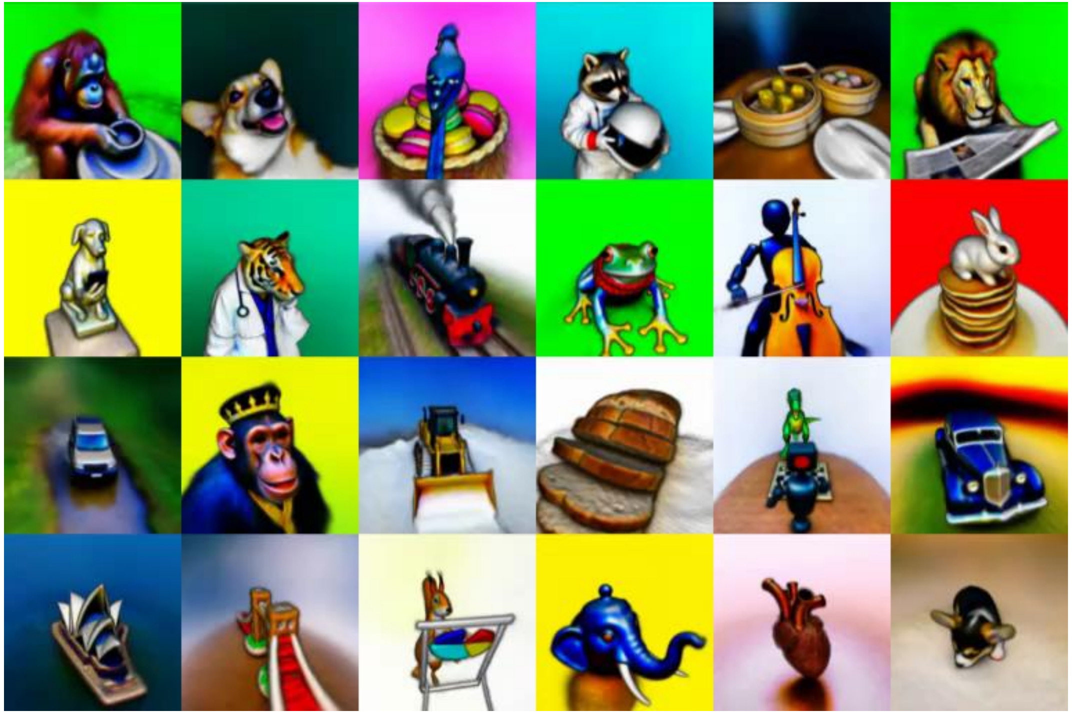
- Image inpainting & outpainting



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

Applications for Diffusion Models

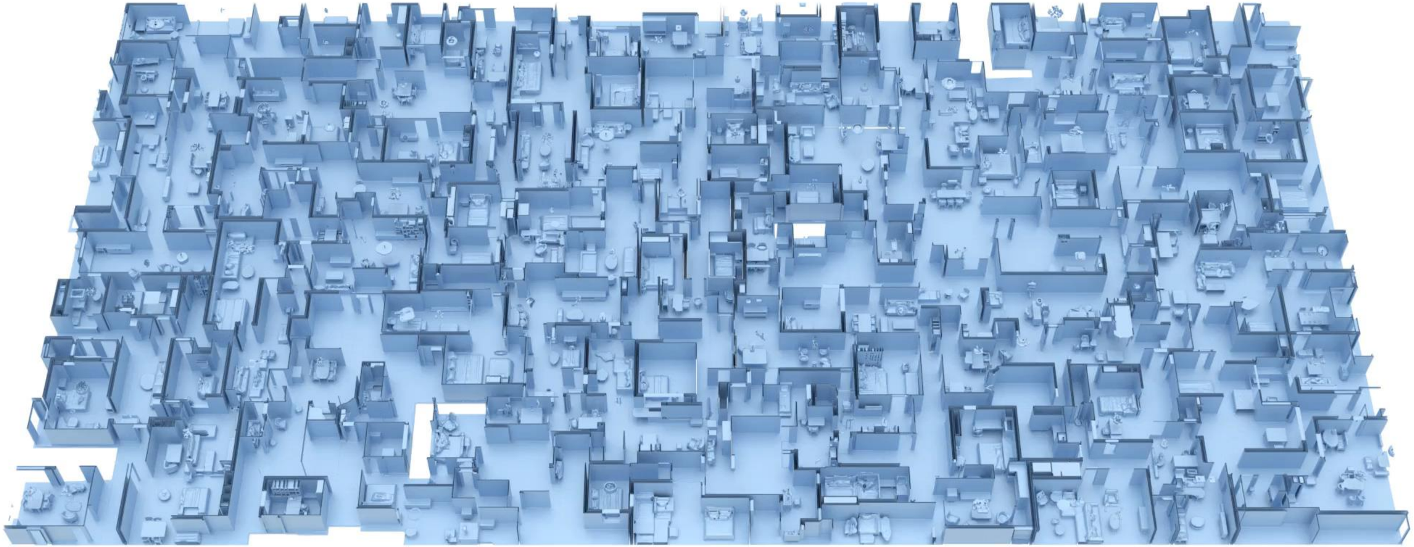
- Text-to-3D Neural Radiance Fields



https://dreamfusion3d.github.io/

Applications for Diffusion Models

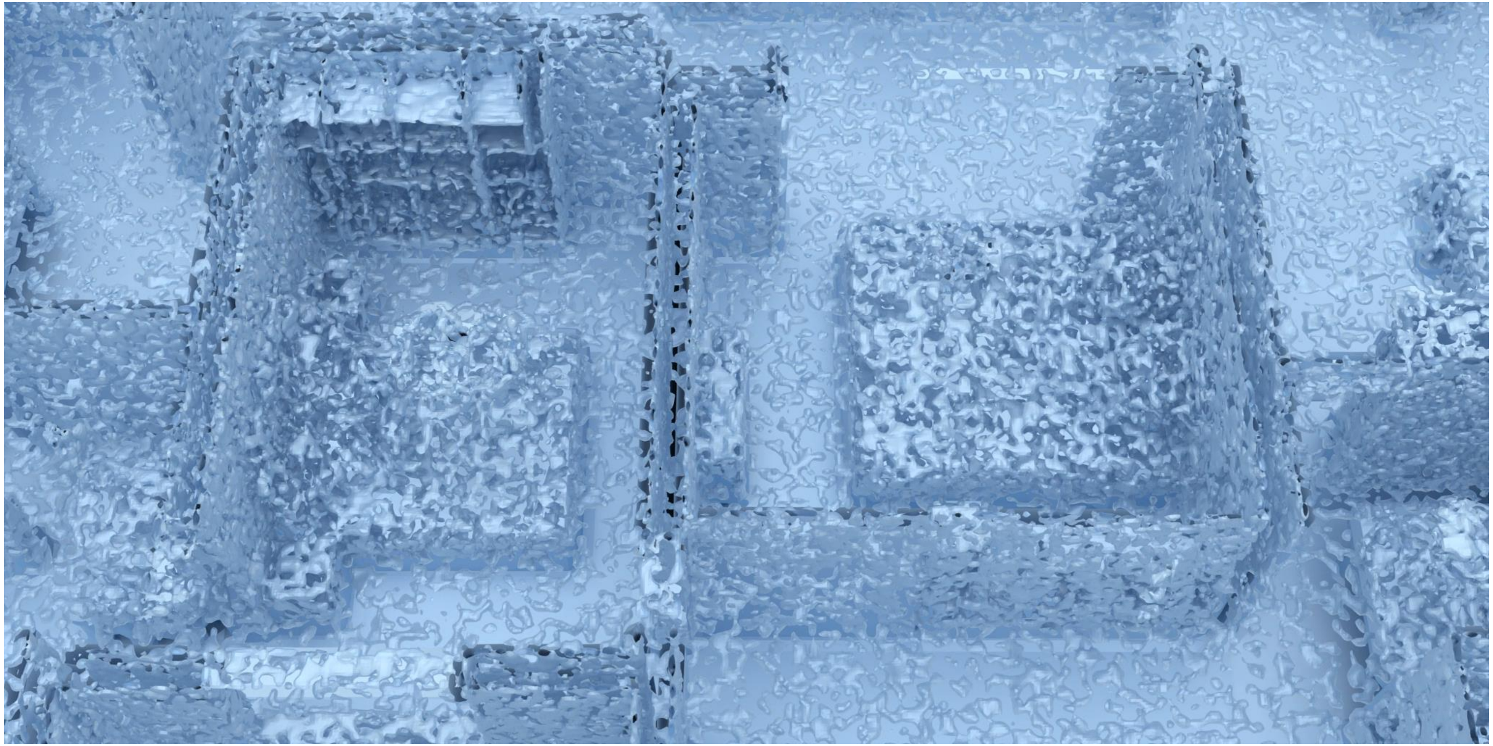
- 3D Scene Generation



[Meng et al.] LT3SD (CVPR'25)

Applications for Diffusion Models

- 3D Scene Generation



[Meng et al.] LT3SD (CVPR'25)

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

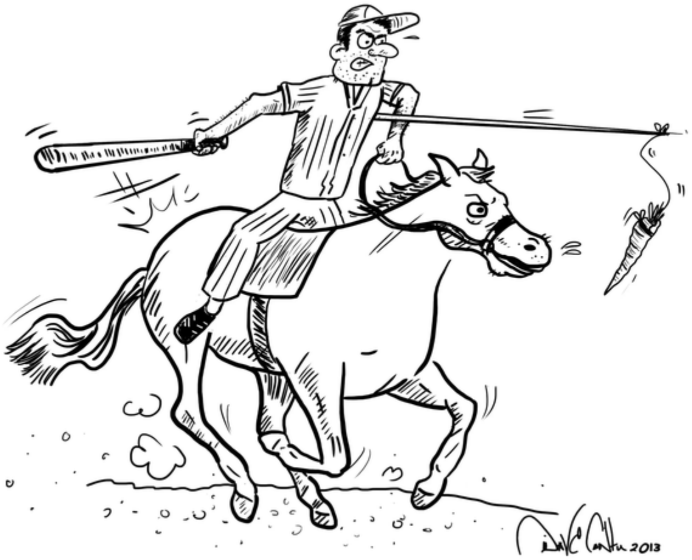
Reinforcement Learning

Sequential data

Learning by interaction with the environment

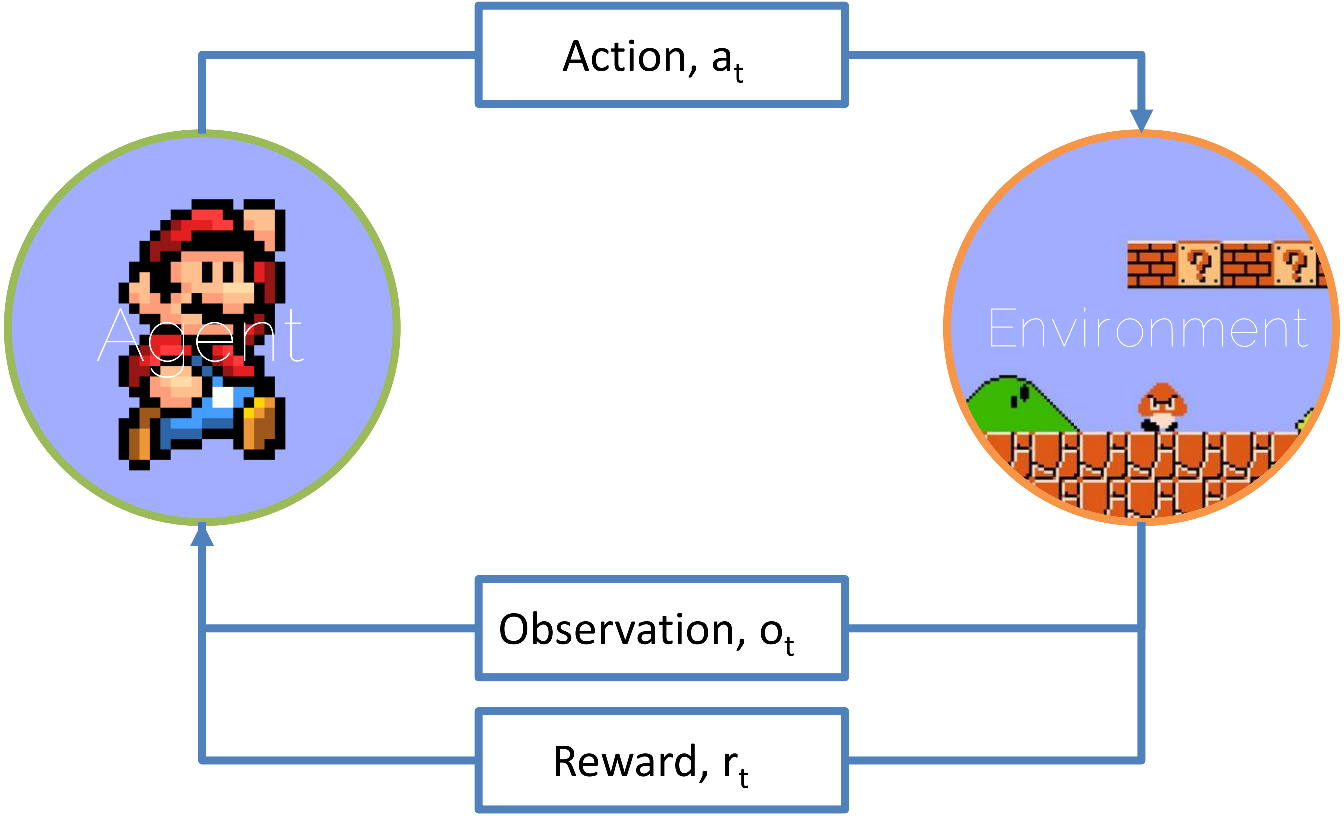
In a Nutshell

- 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:

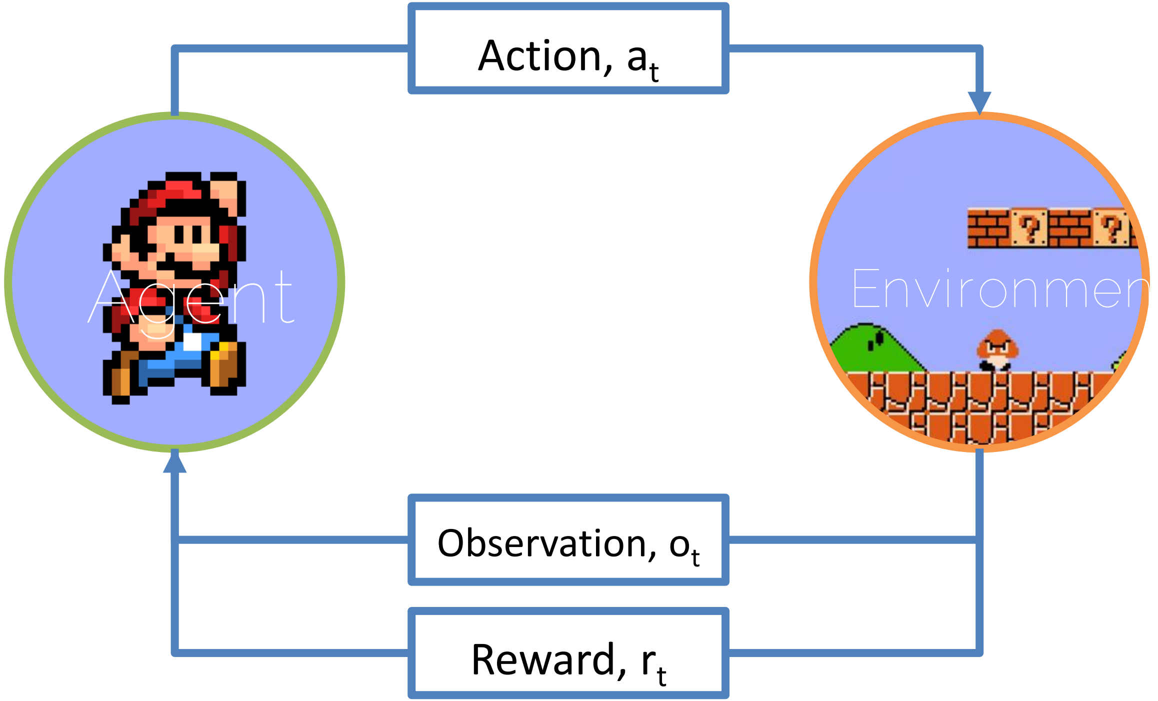
$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \dots s_t]$$

- "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

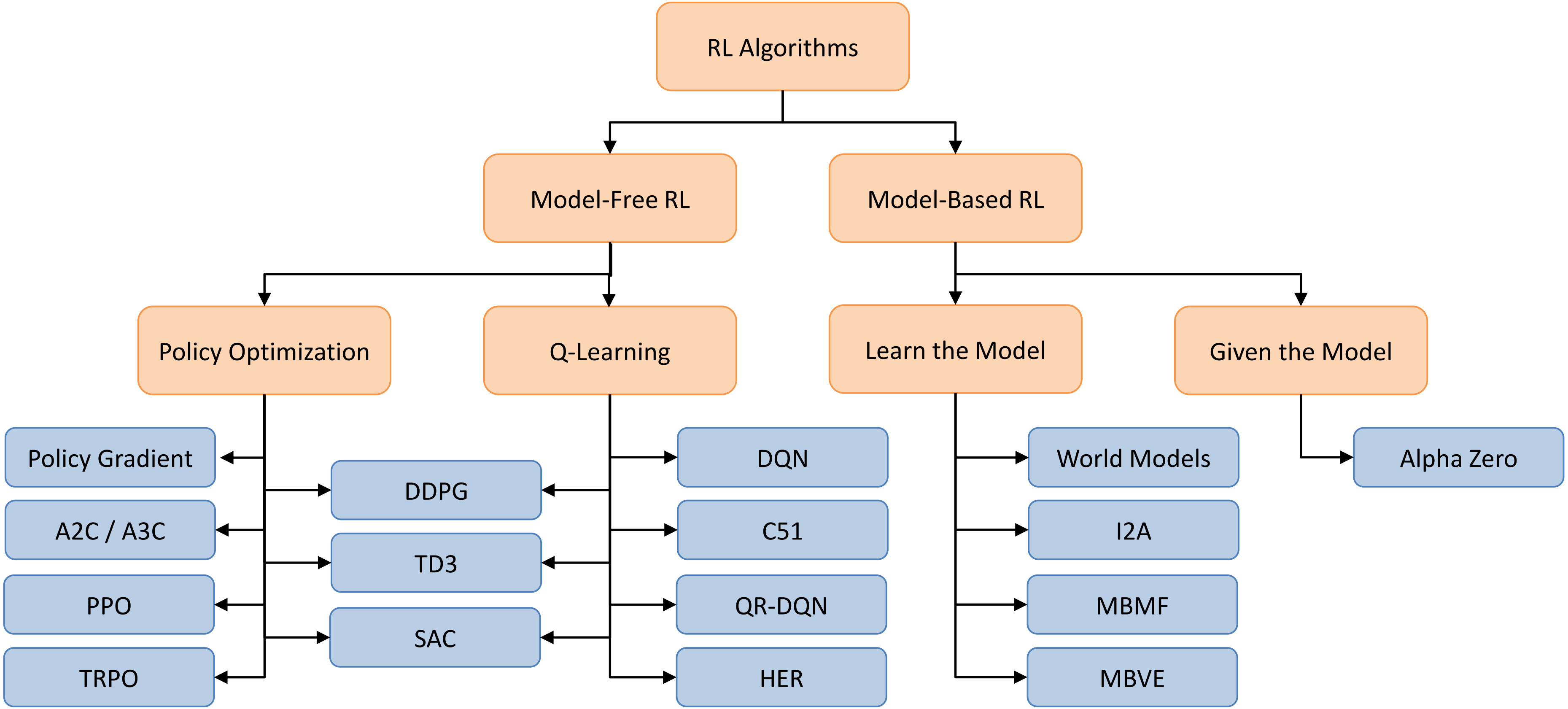
Components of an RL Agent

- 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

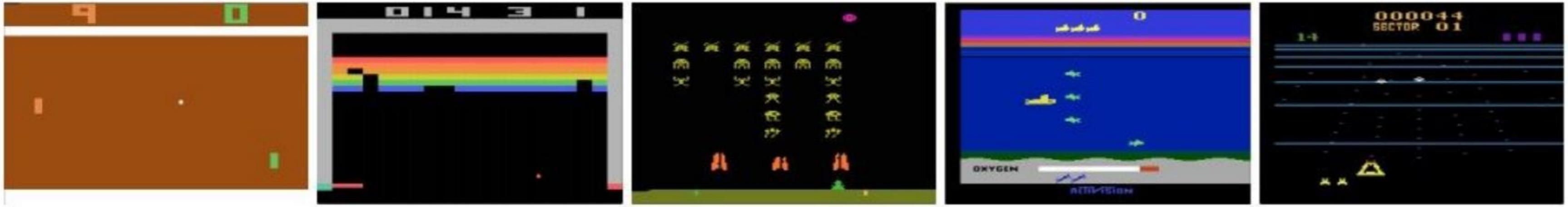
Source: UCL Reinforcement Learning

Taxonomy of RL Algorithms



Source: spinningup.openai.com

RL Milestones: Playing Atari

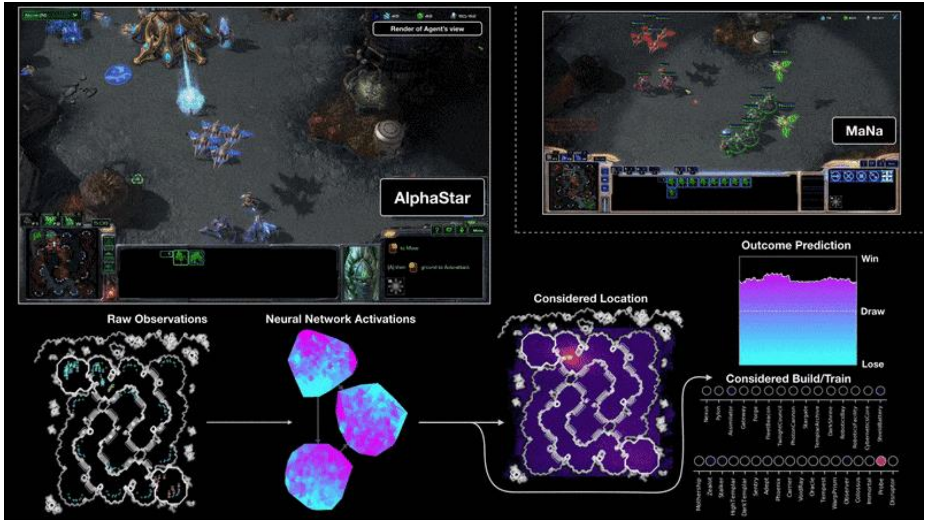


- Mnih et al. 2013, first appearance of DQN
- Successfully learned to play different Atari games like Pong, Breakout, Space Invaders, Seaquest and Beam Rider

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

RL Milestones: AlphaZero (StarCraft)

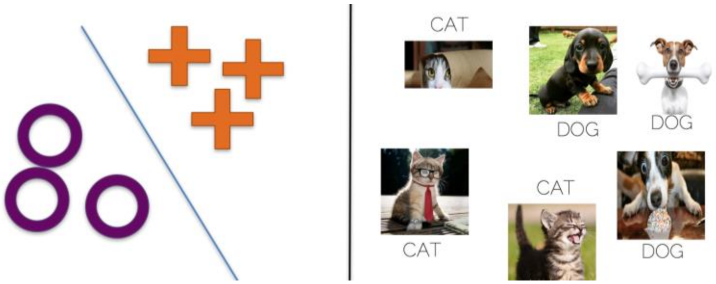
- Model: Transformer network with a LSTM core
- Trained on 200 years of StarCraft play for 14 days
- 16 Google v3 TPUs
- December 2018: Beats MaNa, a professional StarCraft player (world rank 13)



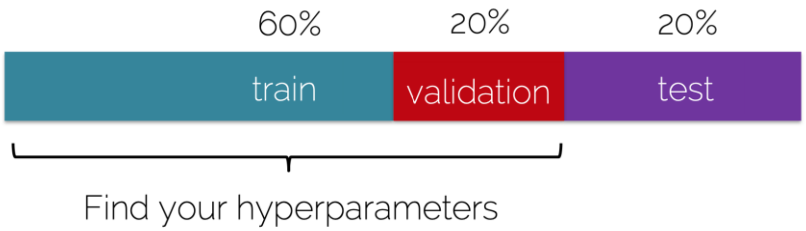
I2DL Summary

Machine Learning Basics

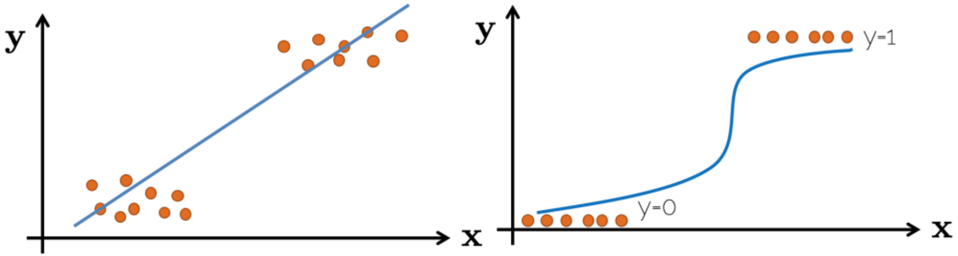
- Unsupervised vs Supervised Learning



- Data splitting

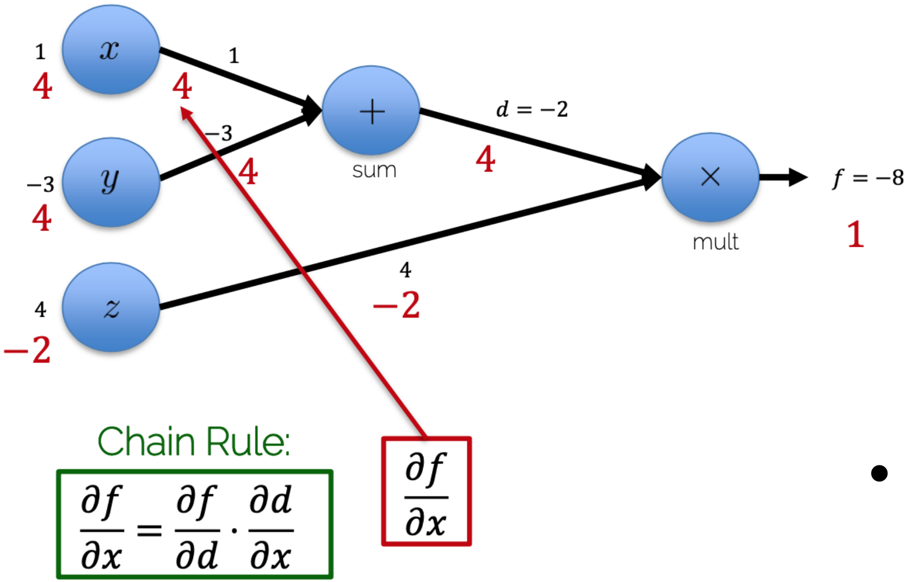


- Linear vs logistic regression

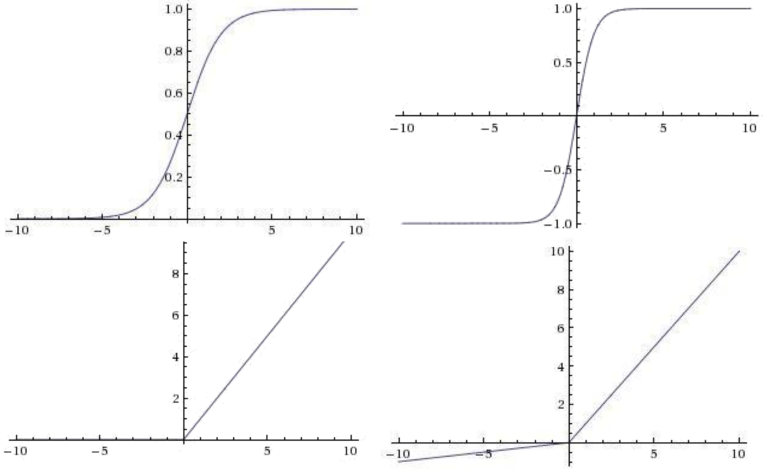


Intro to Neural Networks

- Backpropagation



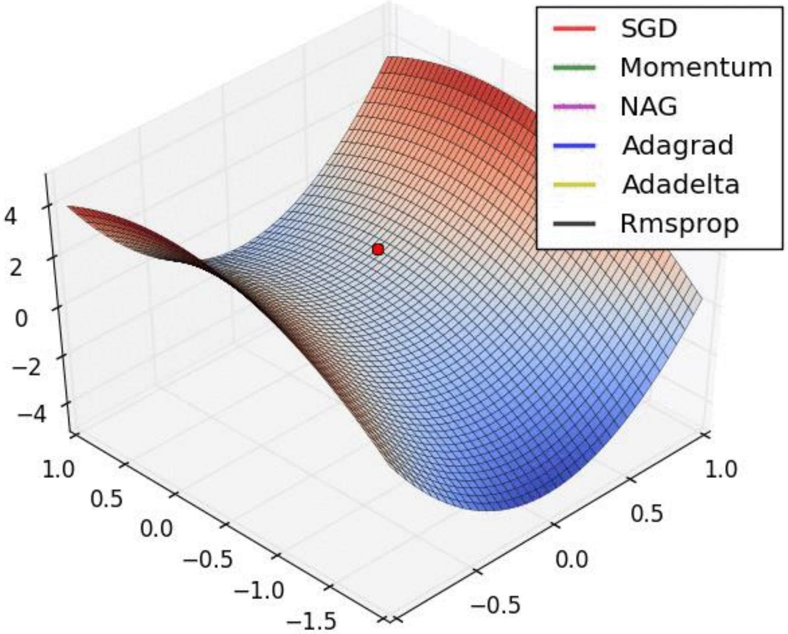
- Activation functions



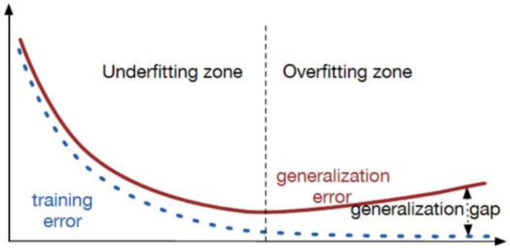
- Loss functions
  - Comparison & effects

Training Neural Networks

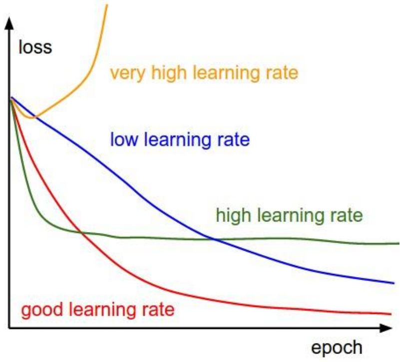
- Gradient Descent/ SGD



- Regularization



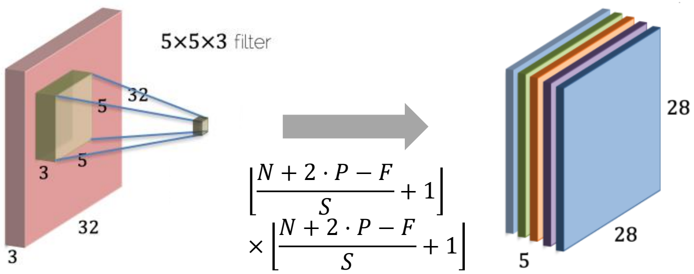
- Parameter & interpretation



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

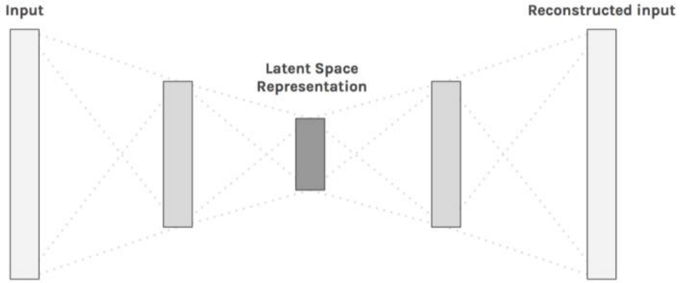
Typology of Neural Networks

- CNNs

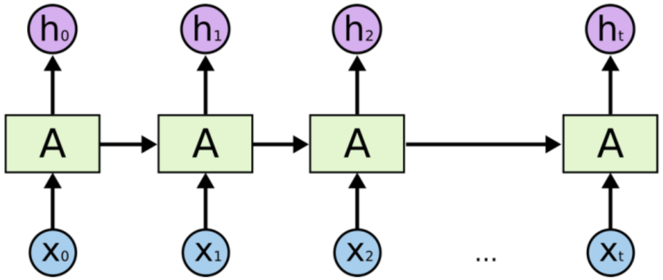


$$\left\lfloor \frac{N + 2 \cdot P - F}{S} + 1 \right\rfloor \times \left\lfloor \frac{N + 2 \cdot P - F}{S} + 1 \right\rfloor$$

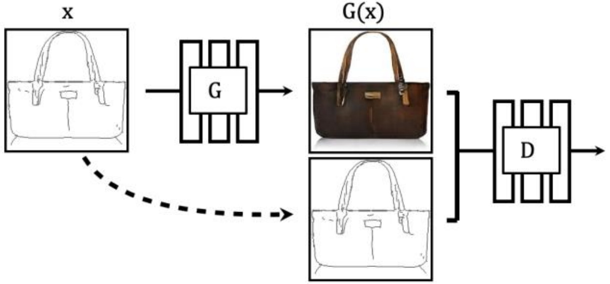
- Autoencoder



- RNNs

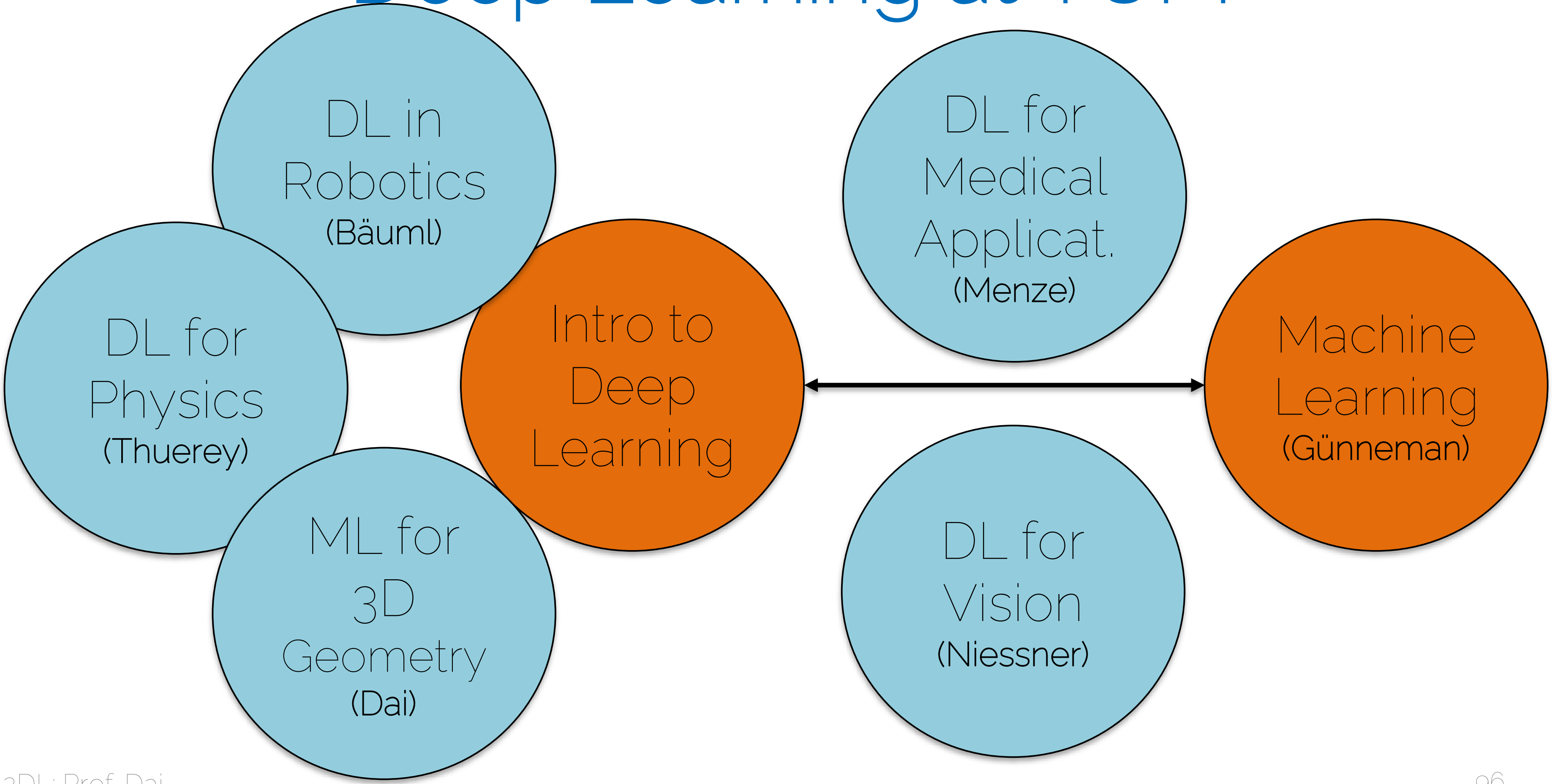


- GANs



Other DL Courses

Deep Learning at TUM



Deep Learning at TUM

- Keep expanding the courses on Deep Learning
- This Introduction to Deep Learning course is the basis for a series of Advanced DL lectures on different topics
- Advanced topics are only for Master students
  - Preparation for MA theses, etc.

Advanced DL for Computer Vision

- Deep Learning for Vision (Profs. Niessner)

- Syllabus
  - Advanced architectures, e.g., Siamese neural networks
  - Variational Autoencoders
  - Generative models, e.g. GAN,
  - Multi-dimensional CNN
  - Graph neural networks
  - Domain adaptation

Advanced DL for Computer Vision

- Deep Learning for Vision
  - 2 V + 3 P
  - **Must** have attended the Intro to DL
  - Practical part is a project that will last the whole semester
  - Please do not sign up unless you are willing to spend a lot of time on the project!

ML for 3D Geometry

- Lectures + Practical Project
  - Geometric foundations
  - Shape descriptors, similarity, segmentation
  - Shape modeling, reconstruction, synthesis
  - Deep learning for multi-view, volumetric, point cloud, and graph data

  - Prof. Dai

Exam

- 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