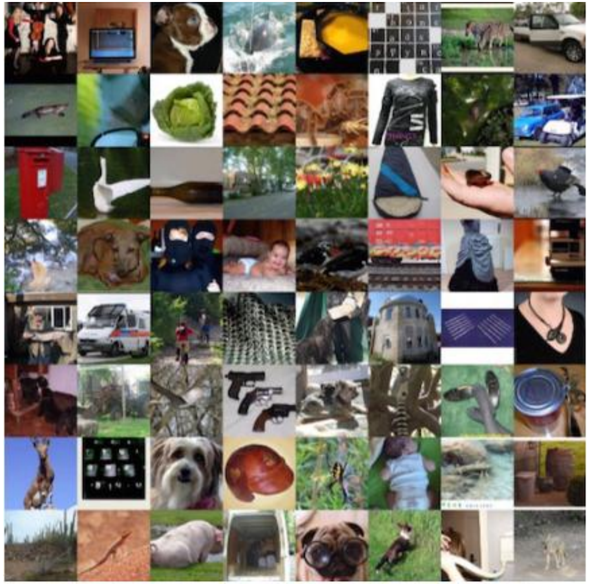
Generative Models

Generative Models

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

Real Images



Generated Images

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

Generative Models

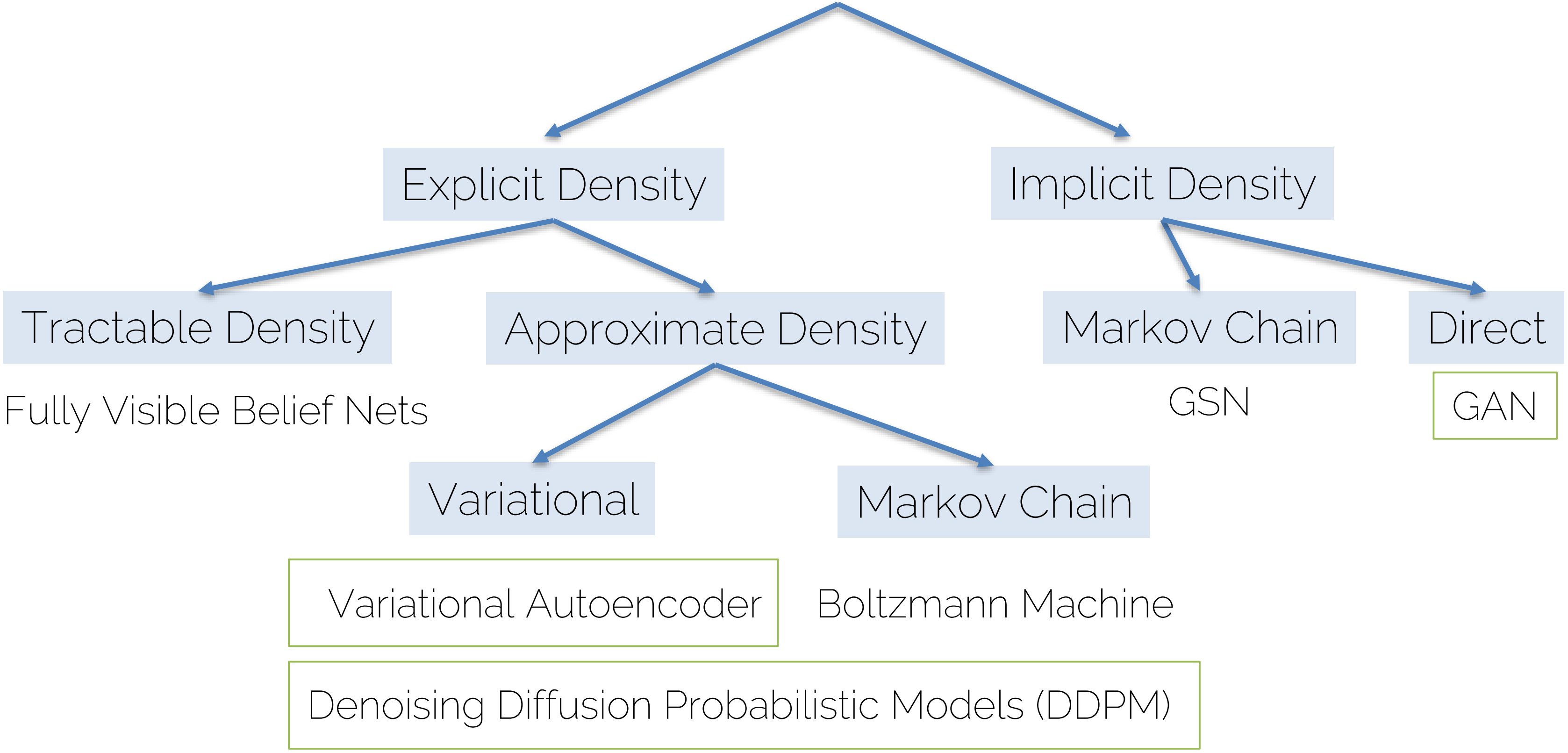


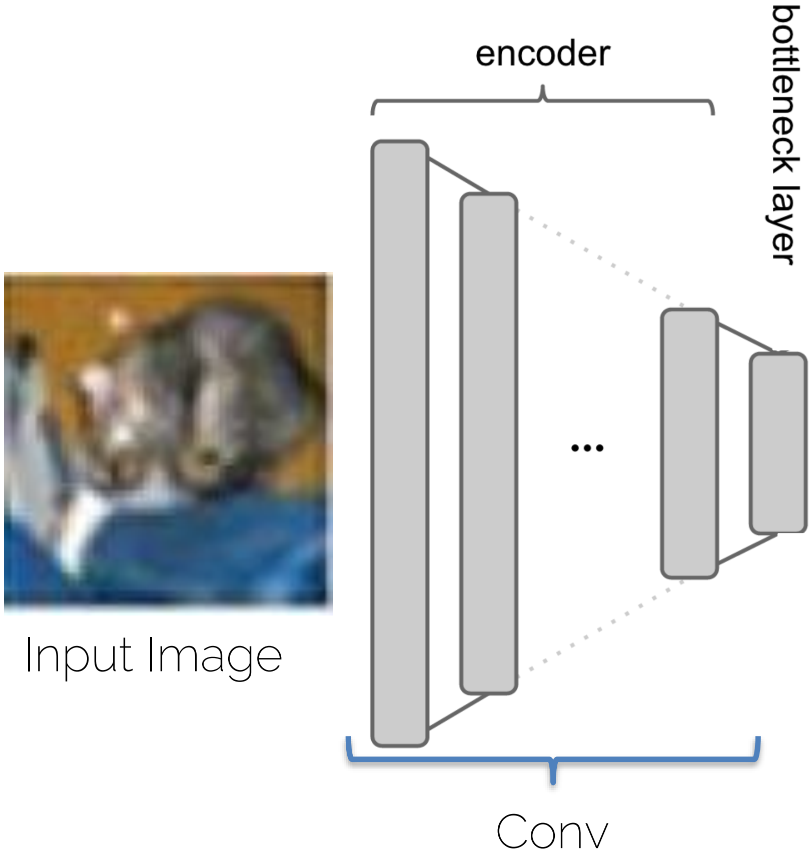
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Autoencoders & VAEs

Autoencoders

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

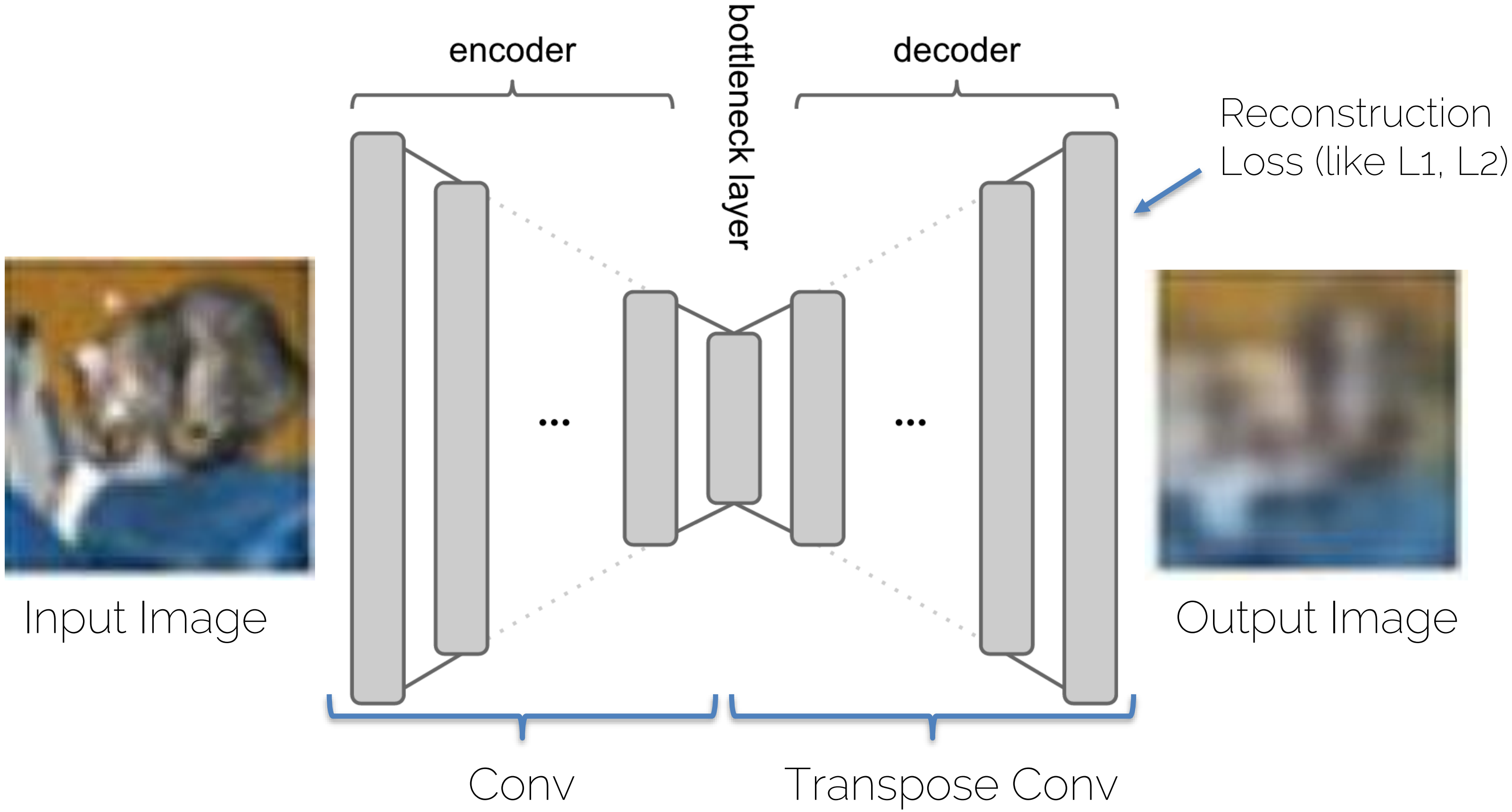
Autoencoders



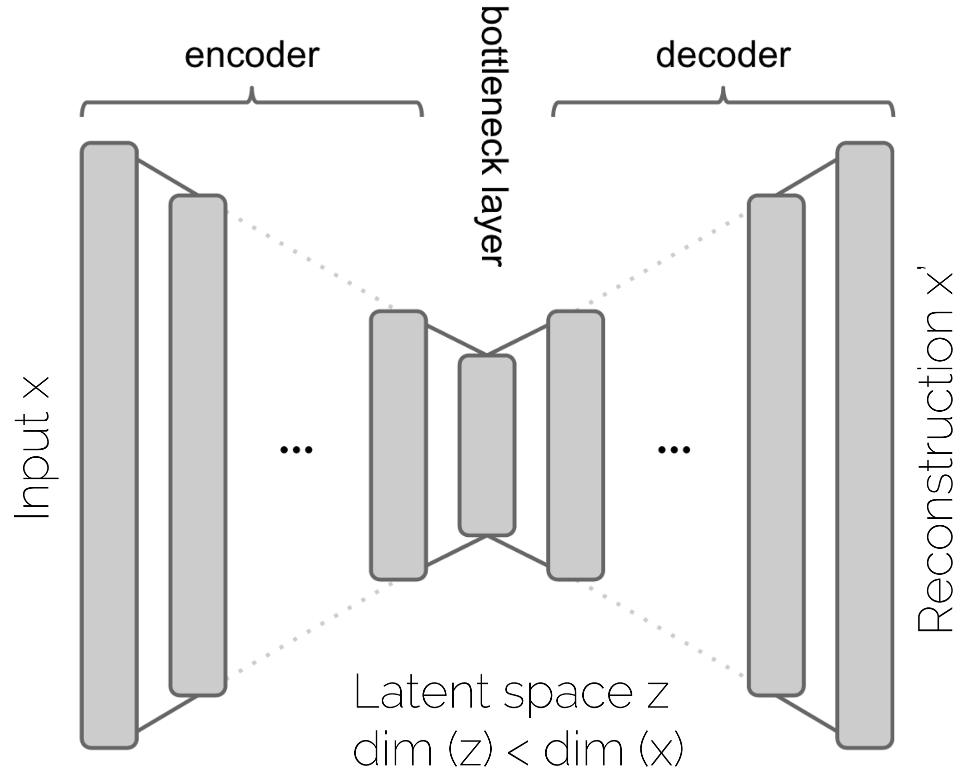
- From an input image to a feature representation (bottleneck layer)

- Encoder: a CNN in our case

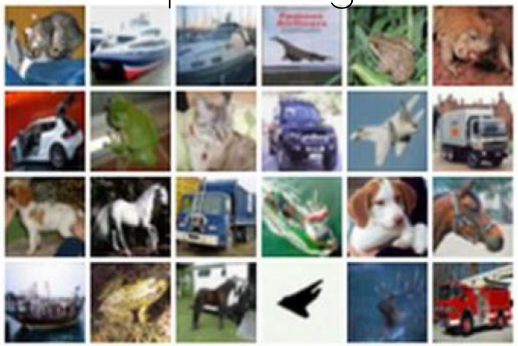
Autoencoder: training



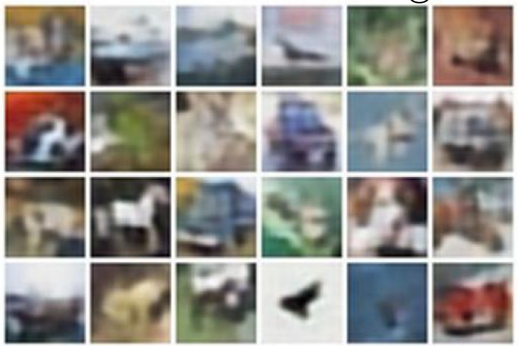
Autoencoder: training



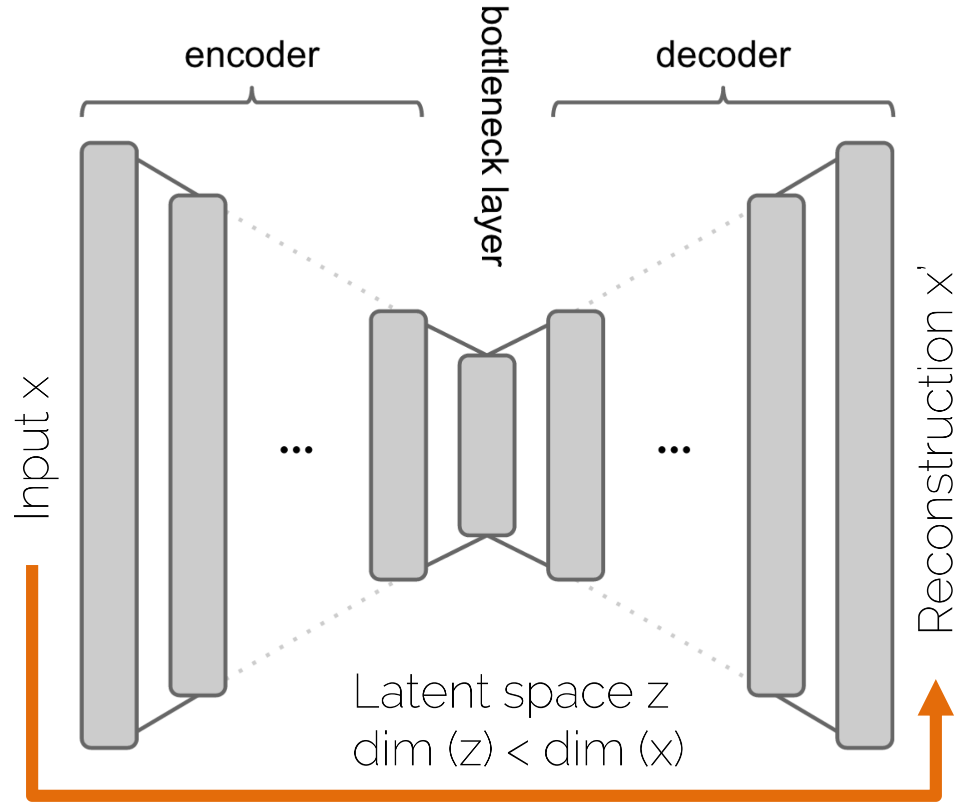
Input images



Reconstructed images



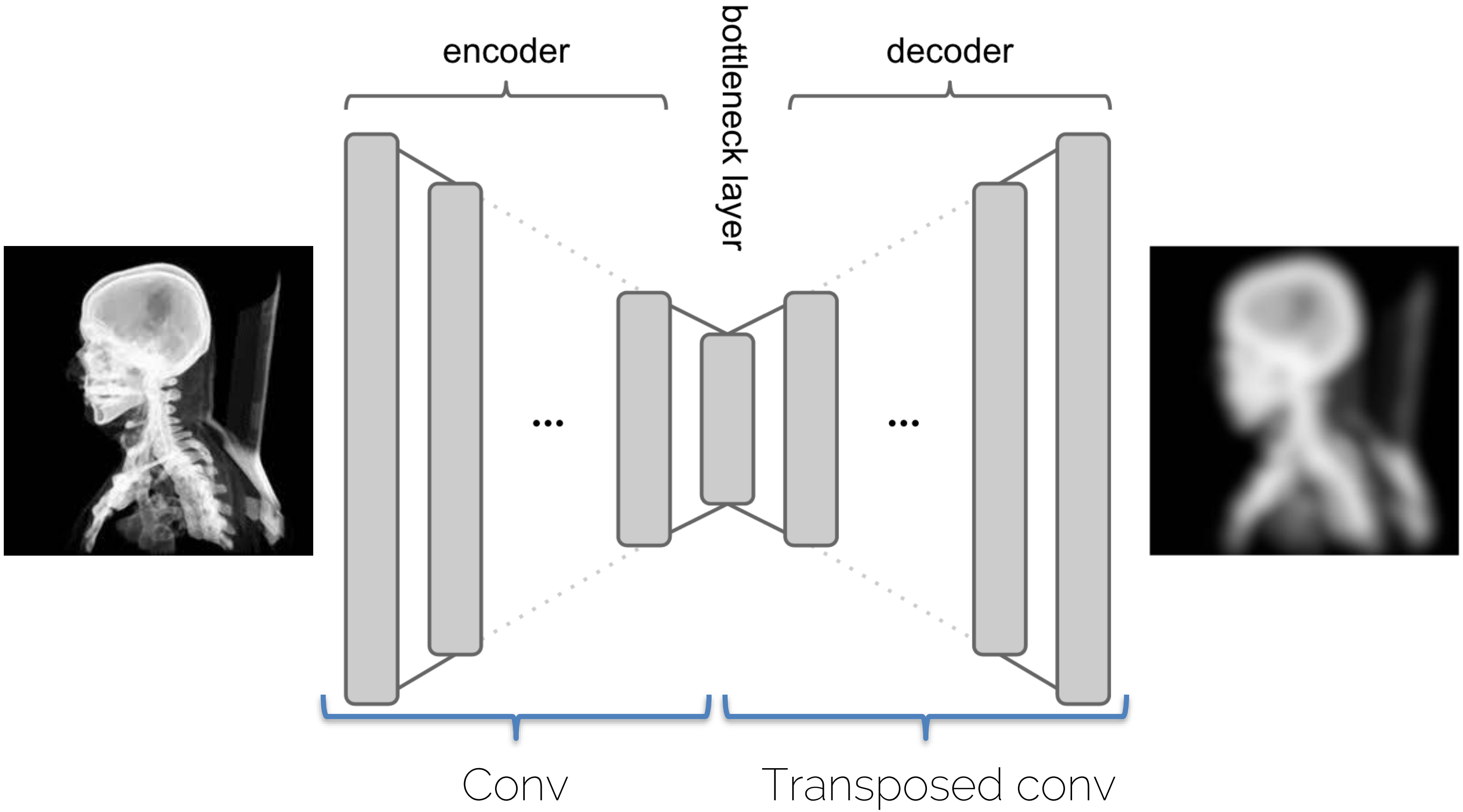
Autoencoder: training



- No labels required

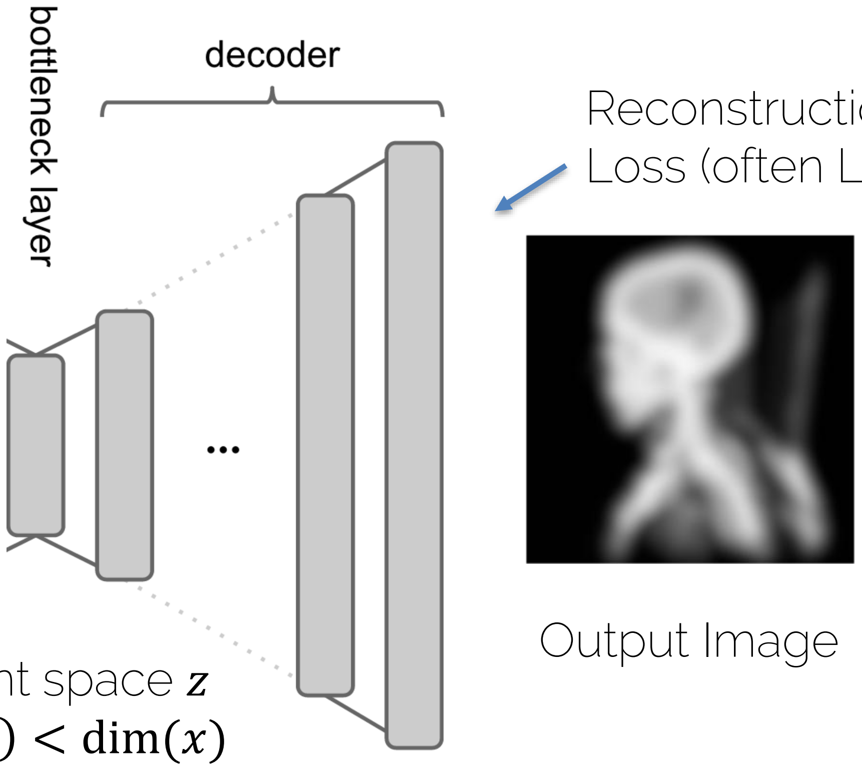
- We can use unlabeled data to first get its structure

Autoencoder



Decoder as Generative Model

Test time:
-> reconstruction from 'random' vector



Reconstruction Loss (often L2)

Output Image

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

Why using autoencoders?

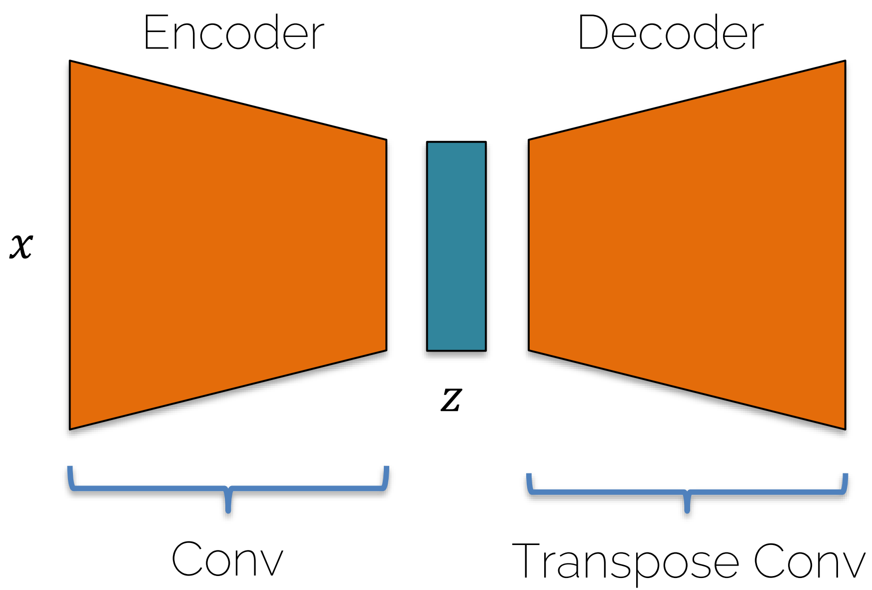
- Use 1: pre-training, as mentioned before
  - Image → same image reconstructed
  - Use the encoder as "feature extractor"

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

Variational Autoencoders

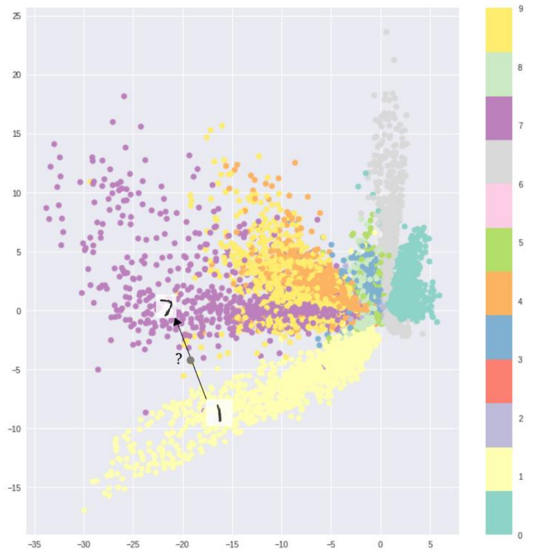
Autoencoders

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



Autoencoders

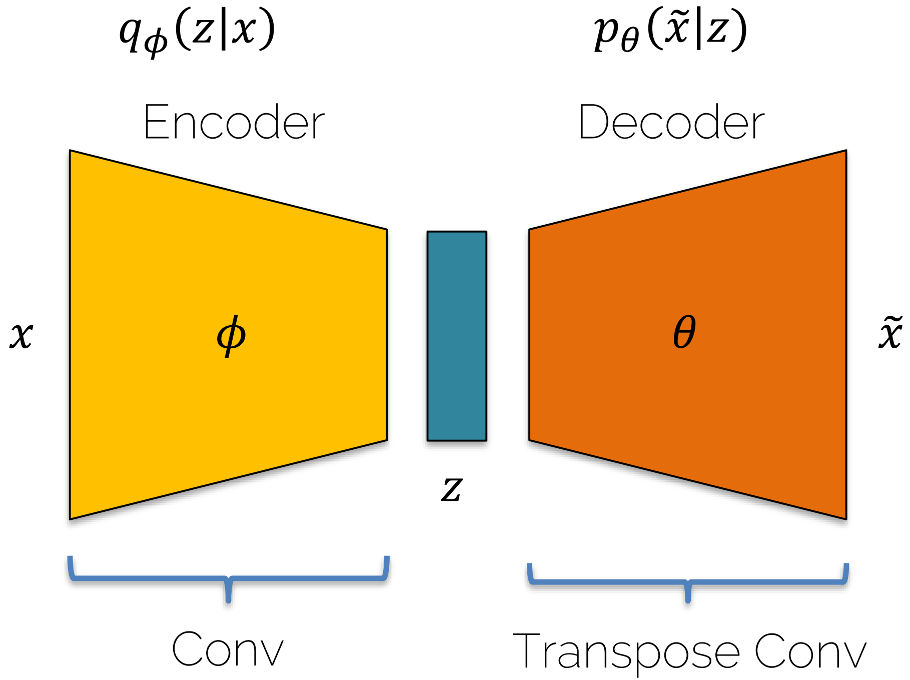
- Encode the input into a representation (bottleneck) and reconstruct it with the decoder



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

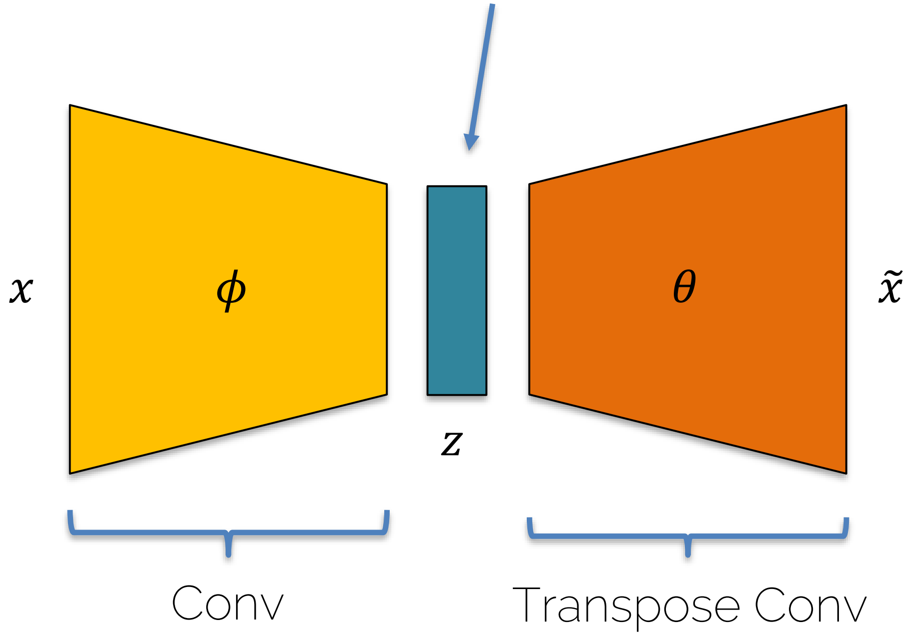
Latent space learned
by autoencoder on MNIST

Variational Autoencoder



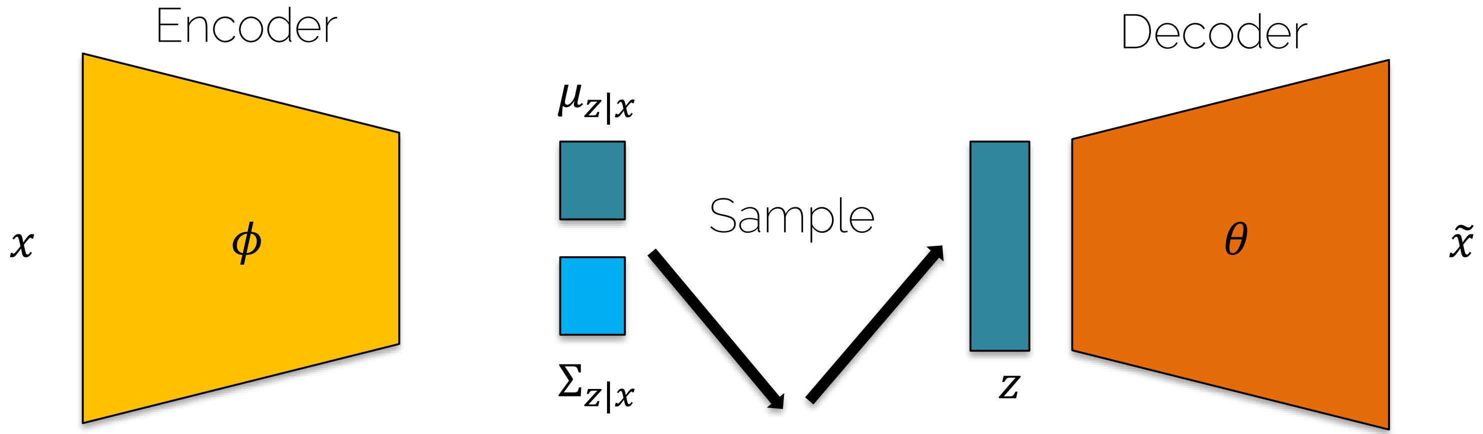
Variational Autoencoder

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



Variational Autoencoder

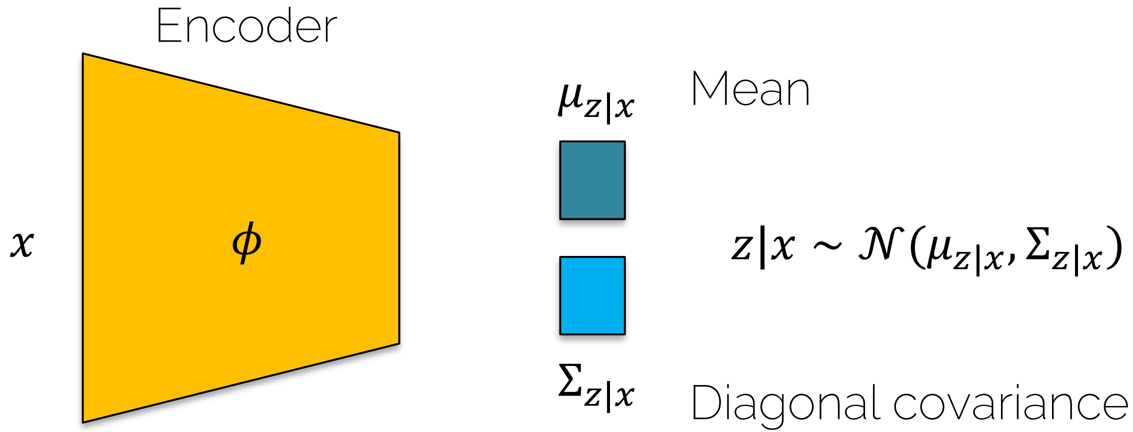
- Latent space is now a distribution
- Specifically it is a Gaussian



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

Variational Autoencoder

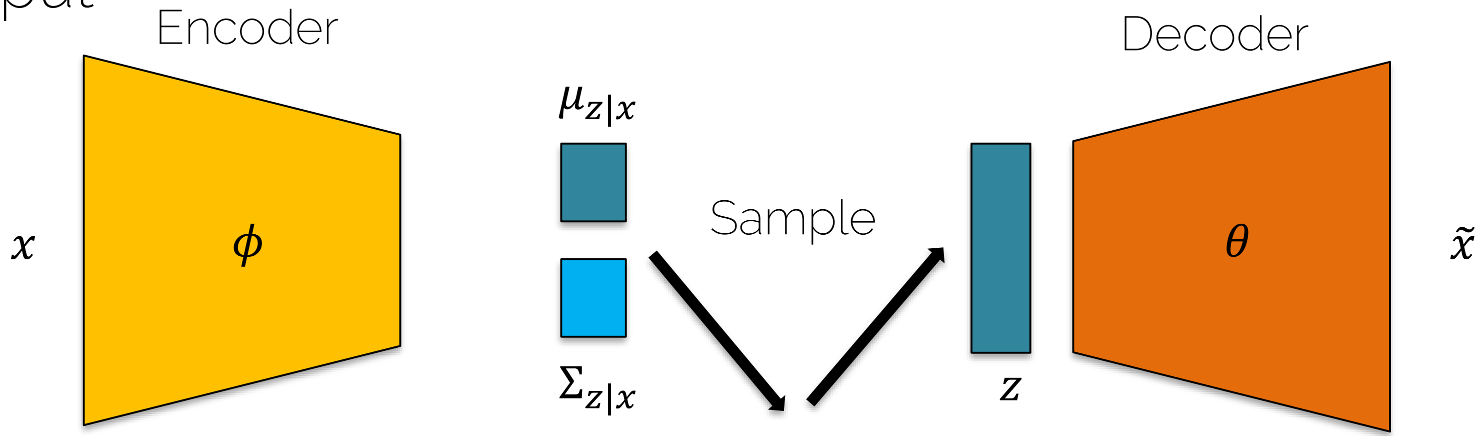
- Latent space is now a distribution
- Specifically it is a Gaussian



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

Variational Autoencoder

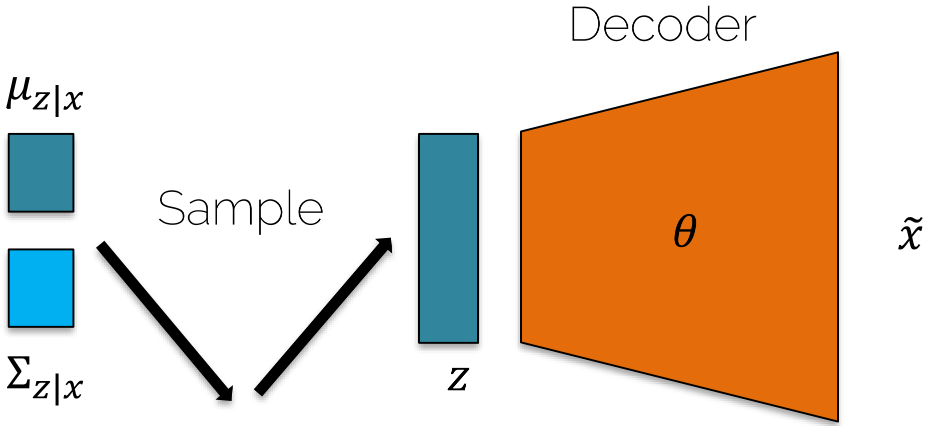
- Training: loss makes sure the latent space is close to a Gaussian and the reconstructed output is close to the input



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

Variational Autoencoder

- Test: Sample from the latent space



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

Autoencoder vs VAE

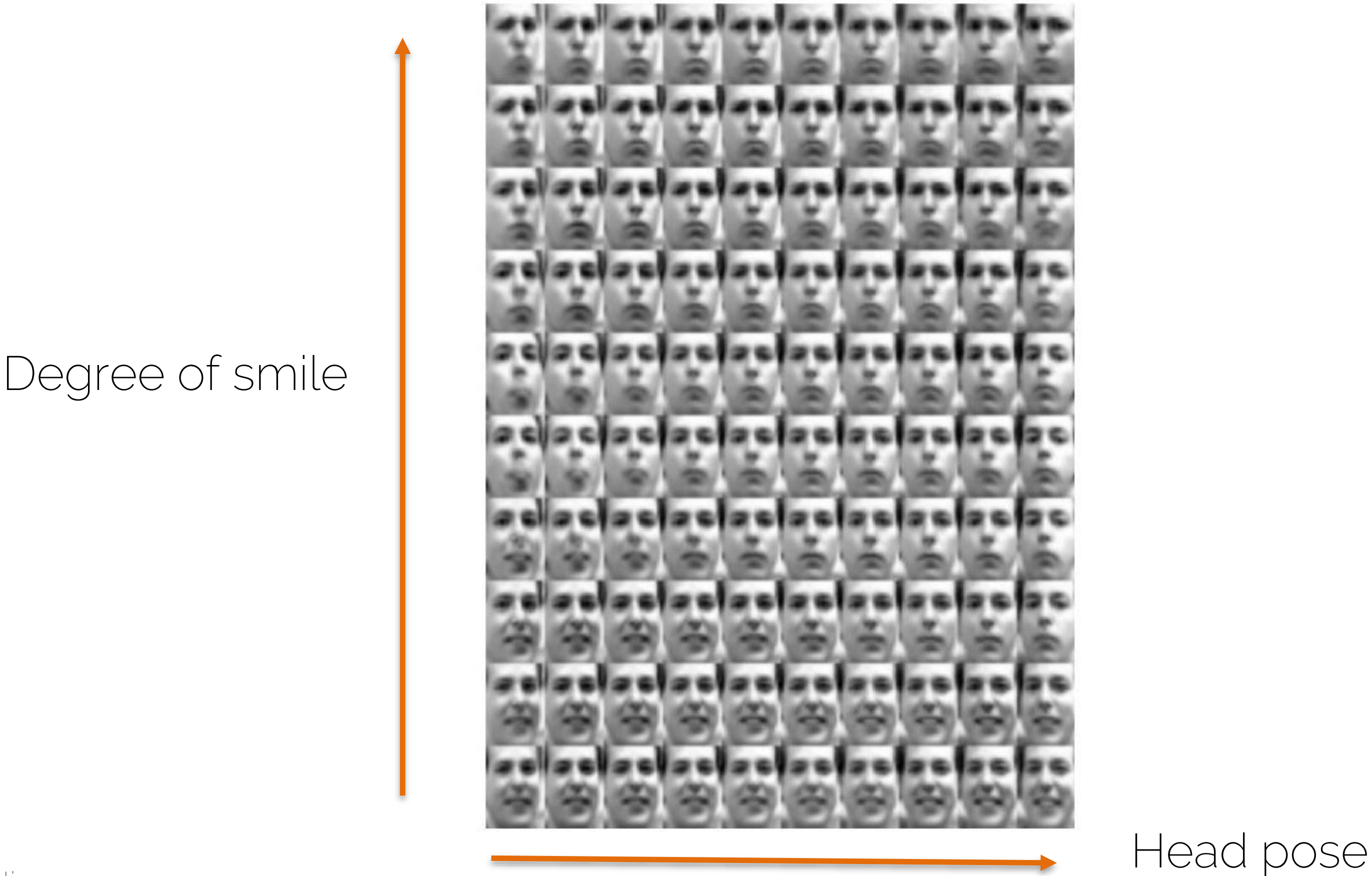
Autoencoder

Variational Autoencoder

Ground Truth

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

Generating data



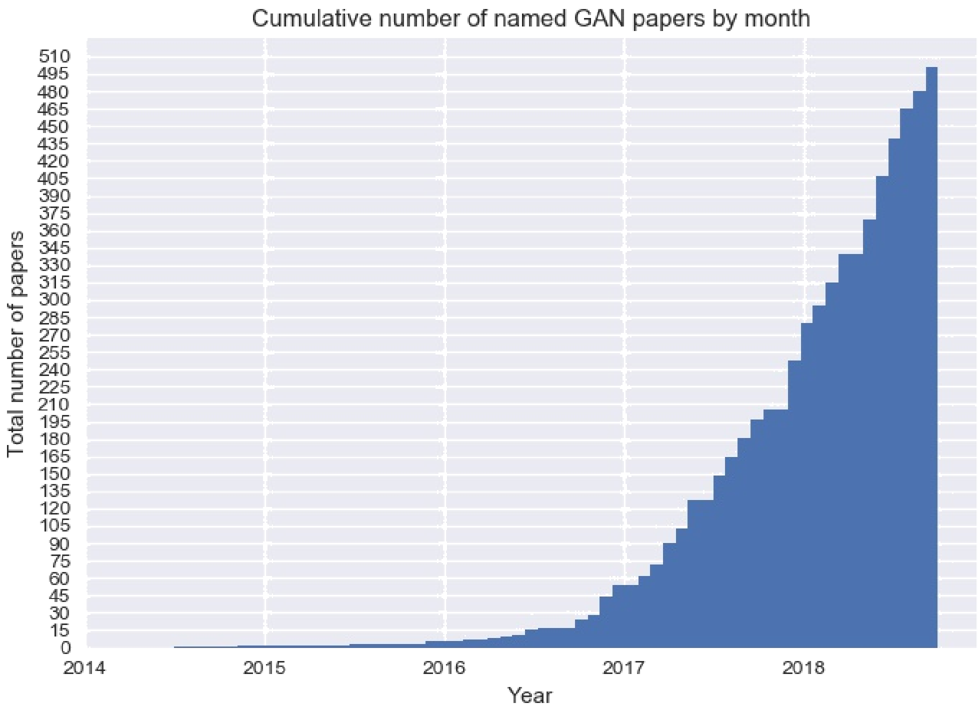
Autoencoder Overview

- 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

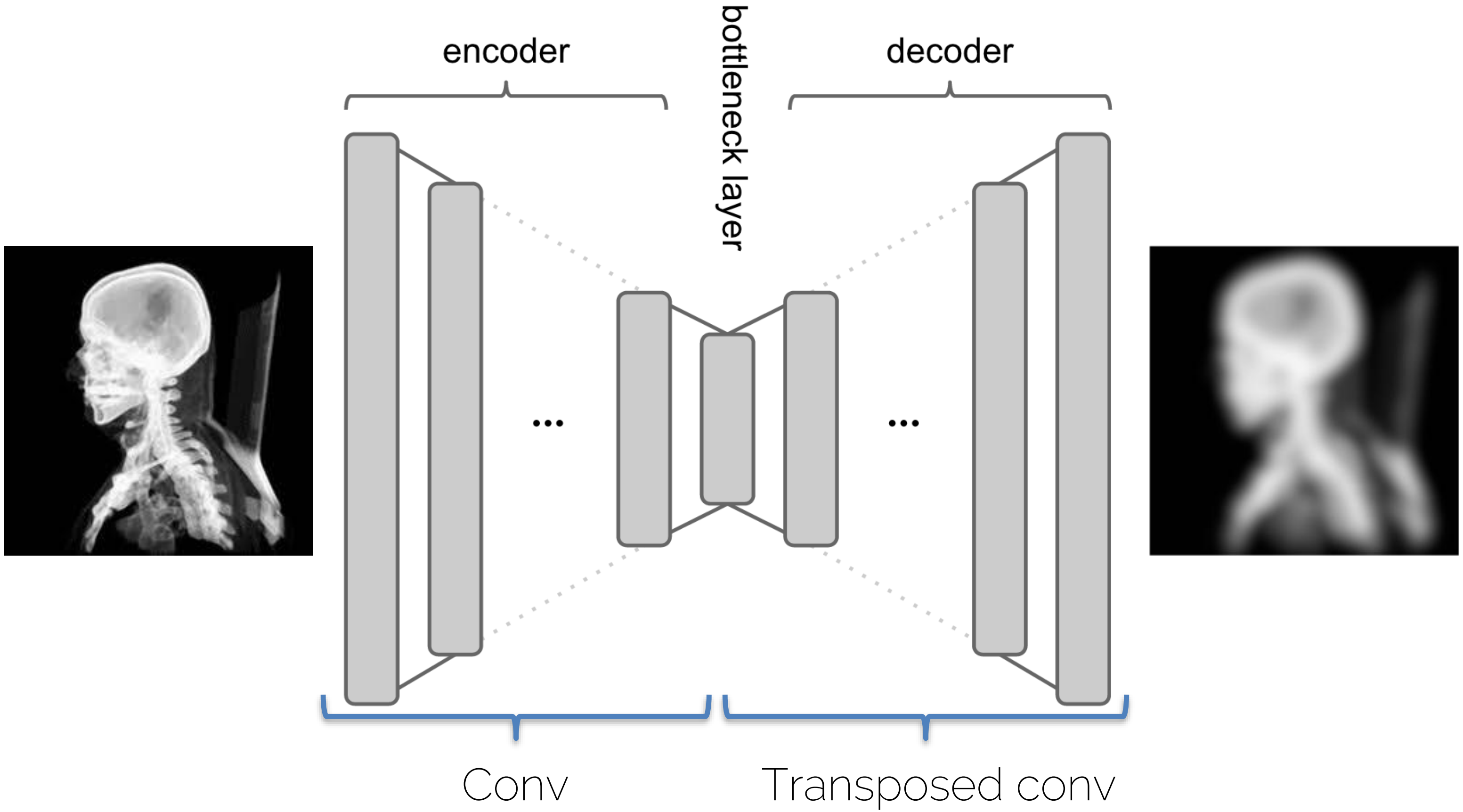
Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)



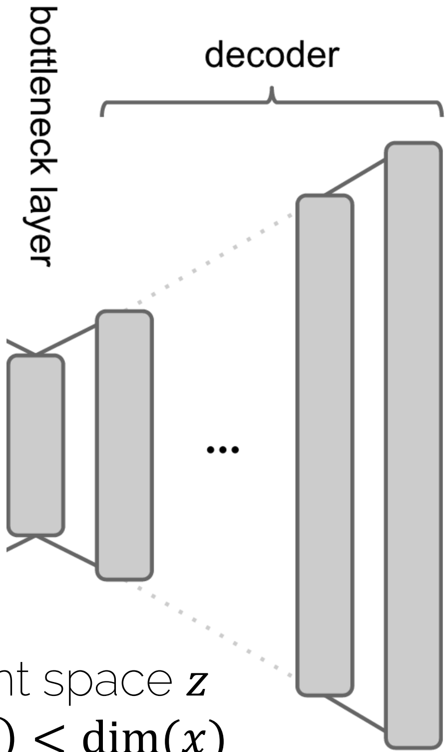
Source: https://github.com/hindupuravinash/the-gan-zoo

Autoencoder



Decoder as Generative Model

Test time:
-> reconstruction from
'random' vector



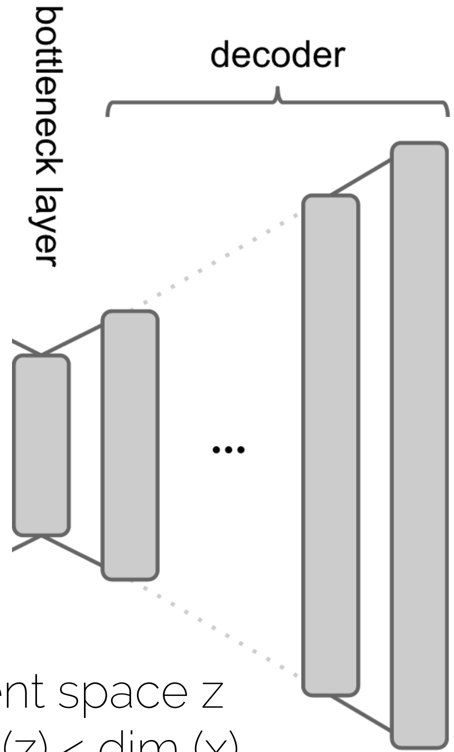
Reconstruction
Loss (often L2)

Output Image

Latent space $\boldsymbol{z}$
$\dim(z) < \dim(x)$

Decoder as Generative Model

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

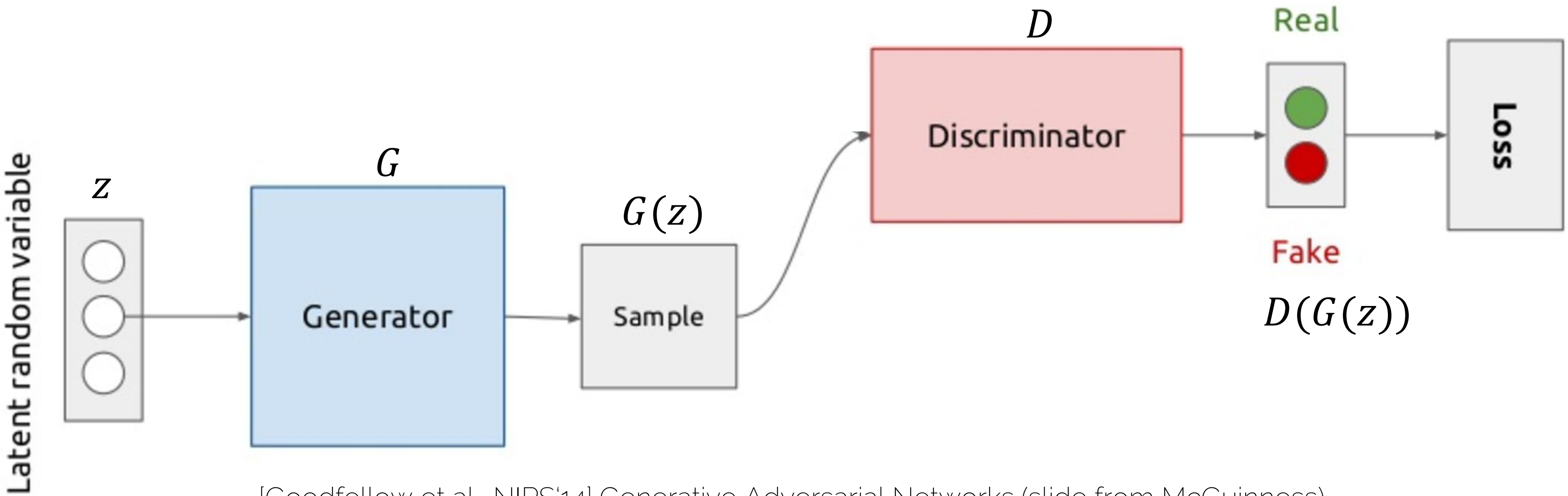


Latent space z
dim (z) < dim (x)

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

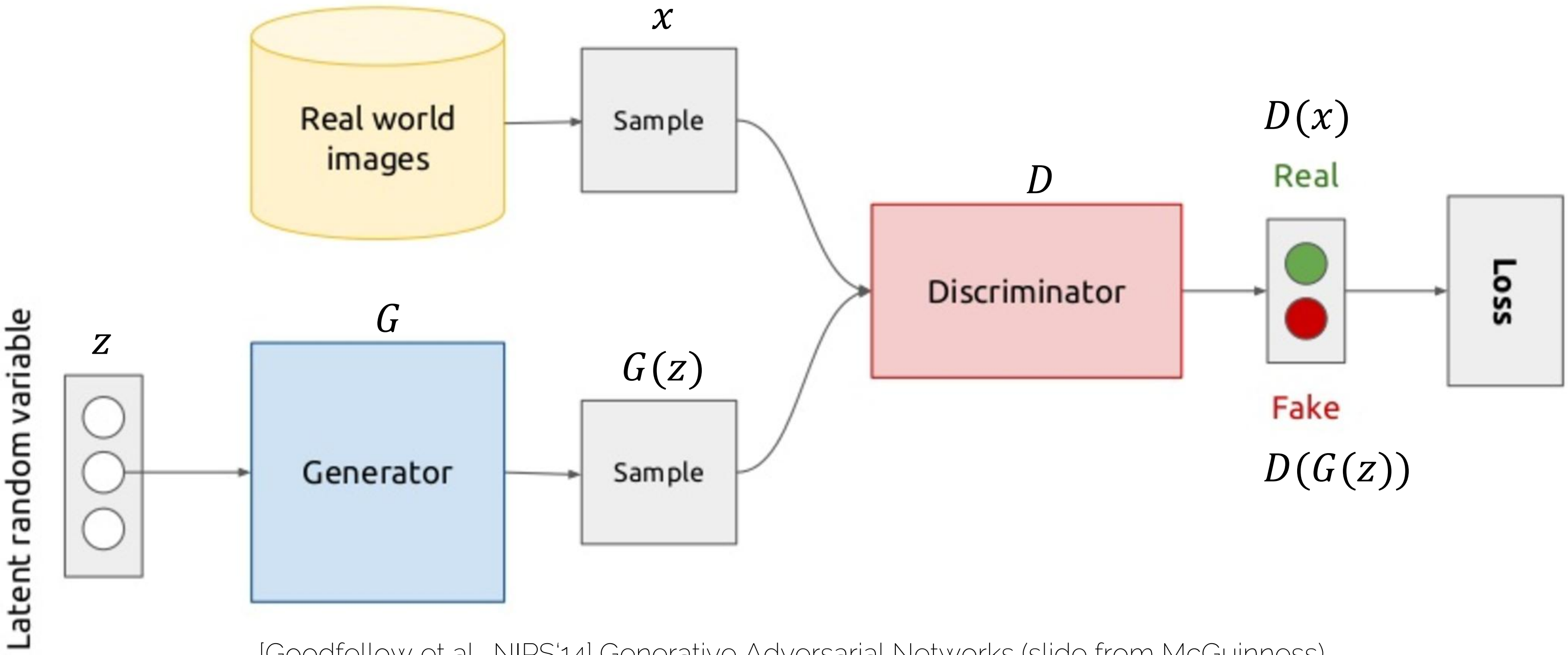
Instead of L2, can we "learn" a loss function?

Generative Adversarial Networks (GANs)



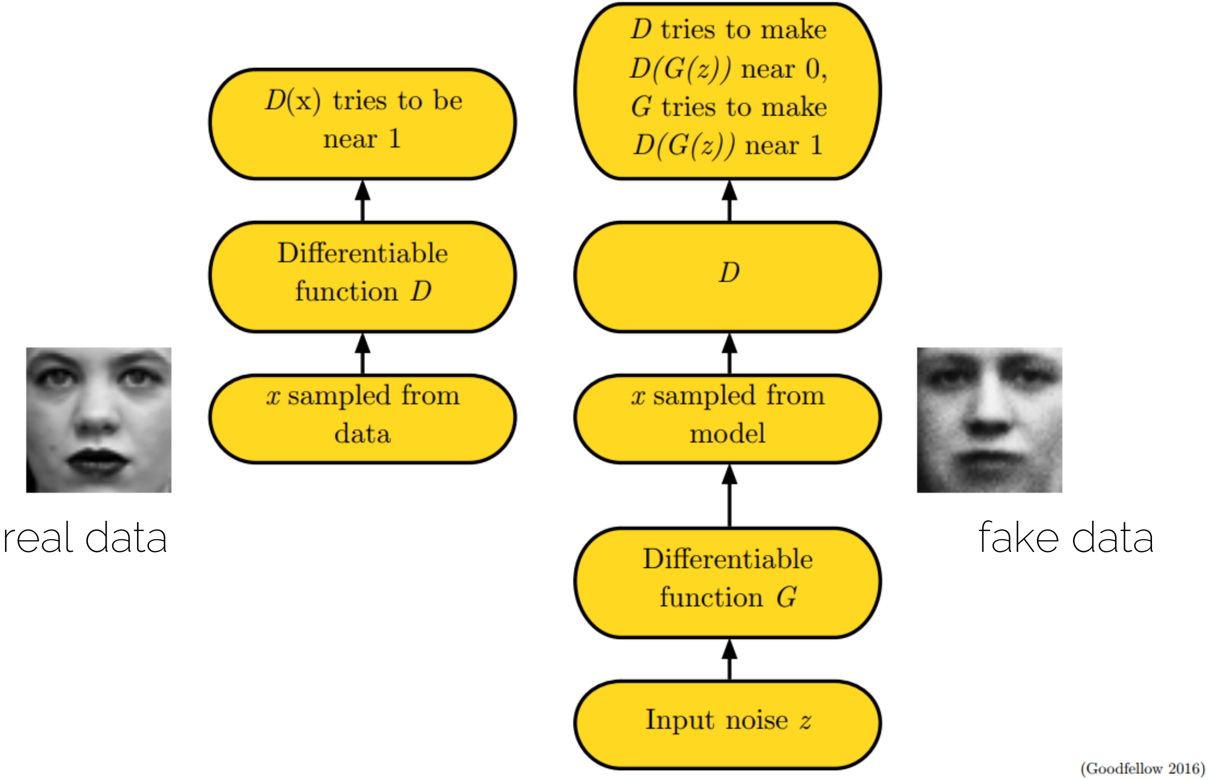
[Goodfellow et al., NIPS'14] Generative Adversarial Networks (slide from McGuinness)

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

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

- 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

GANs: Loss Functions

Discriminator loss

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

Generator loss

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

- Heuristic Method (often used in practice)
  - G maximizes the log-probability of D being mistaken
  - G can still learn even when D rejects all generator samples

[Goodfellow et al. 14/16] GANs

Alternating Gradient Updates

- Step 1: Fix G, and perform gradient step to

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

- Step 2: Fix D, and perform gradient step to

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

Vanilla GAN

for number of training iterations **do**

  for k steps **do**

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

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

  end for

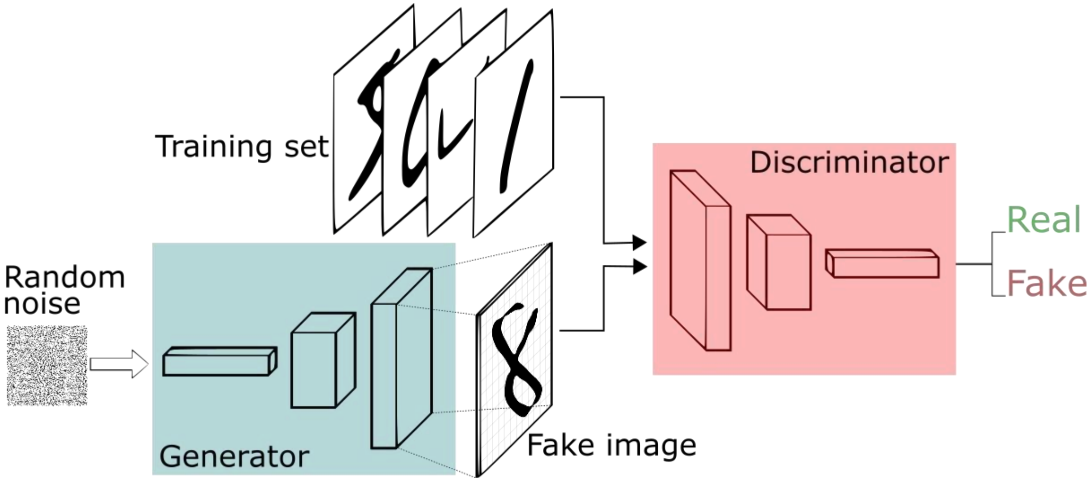
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

end for

https://papers.nips.cc/paper/5423-generative-adversarial-nets

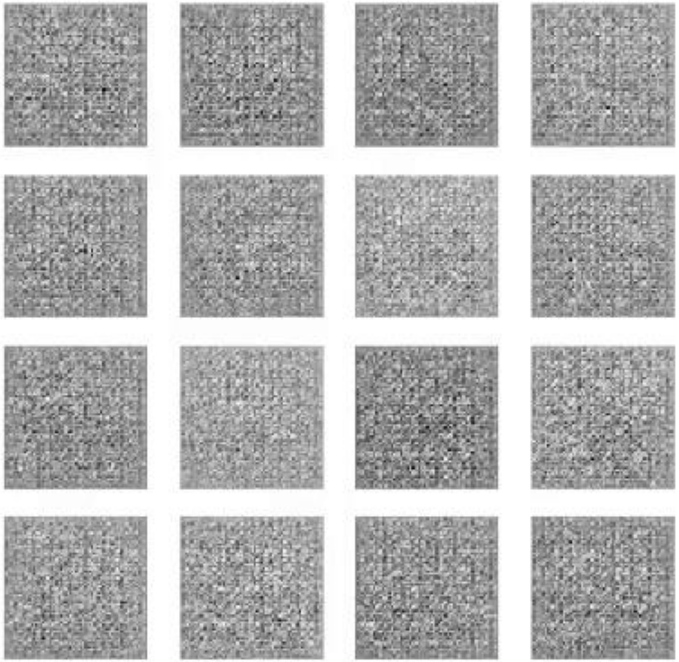
Putting it all Together



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Vitaly Bondar: Generative Models Lecture

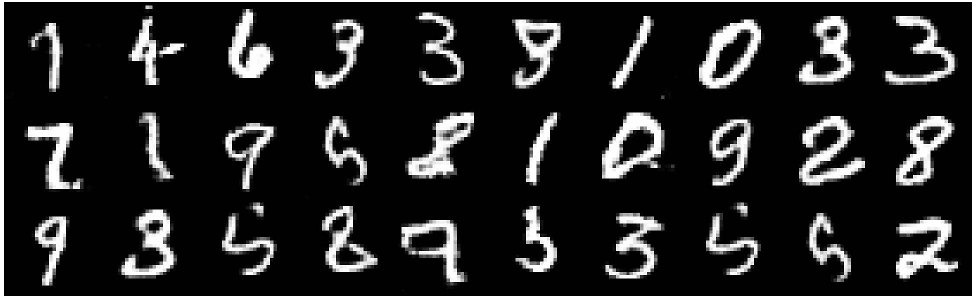
Training a GAN

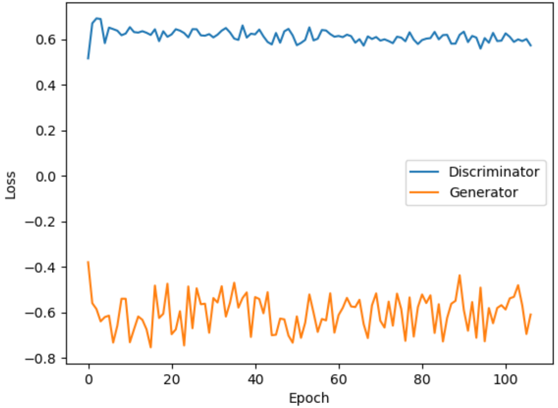


https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f

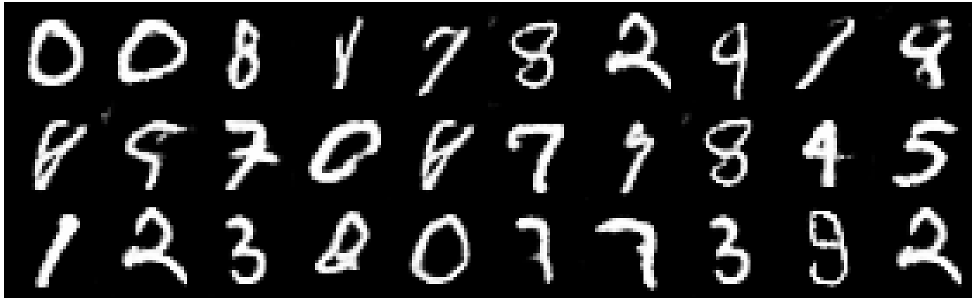
GANs: Loss Functions

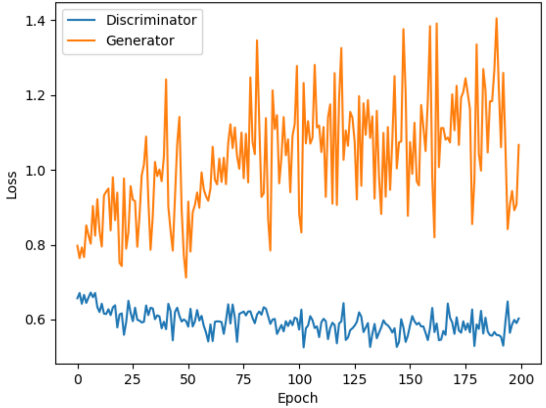
Minimax





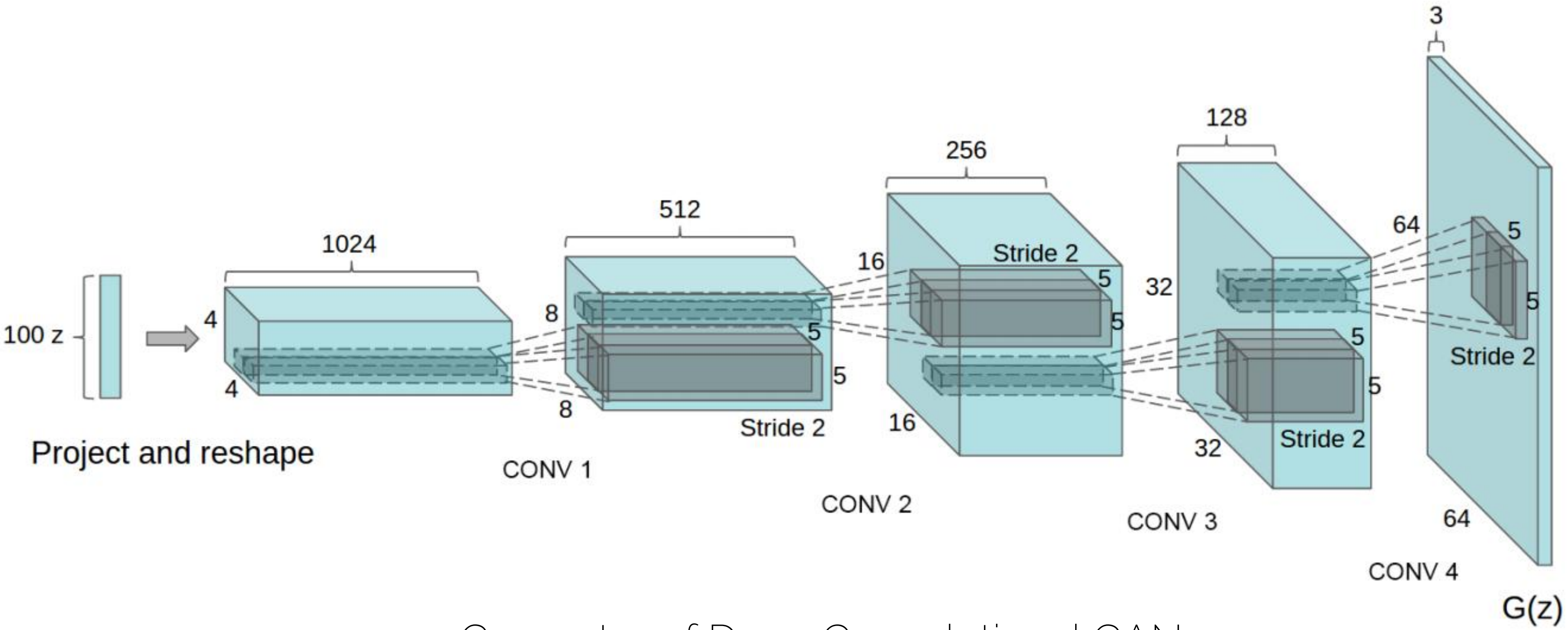
Heuristic





[Goodfellow et al. 14/16] GANs

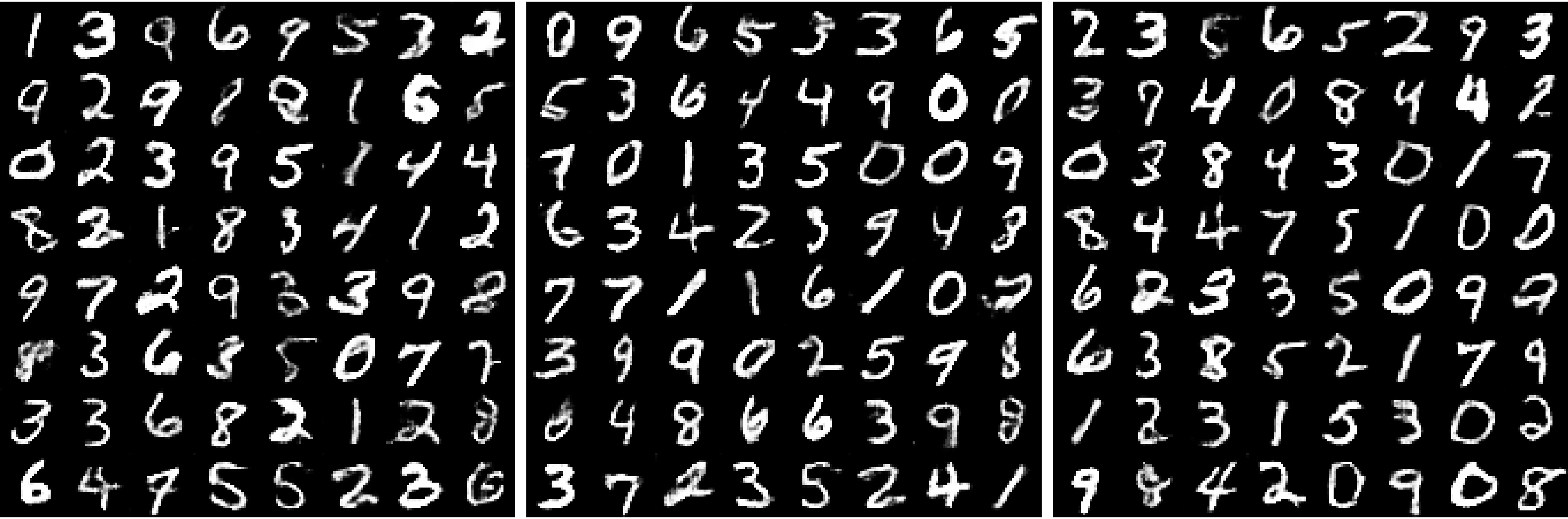
DCGAN: Generator



Generator of Deep Convolutional GANs

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

DCGAN: Results



Results on MNIST

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

DCGAN: Results

Results on CelebA (200k relatively well aligned portrait photos)

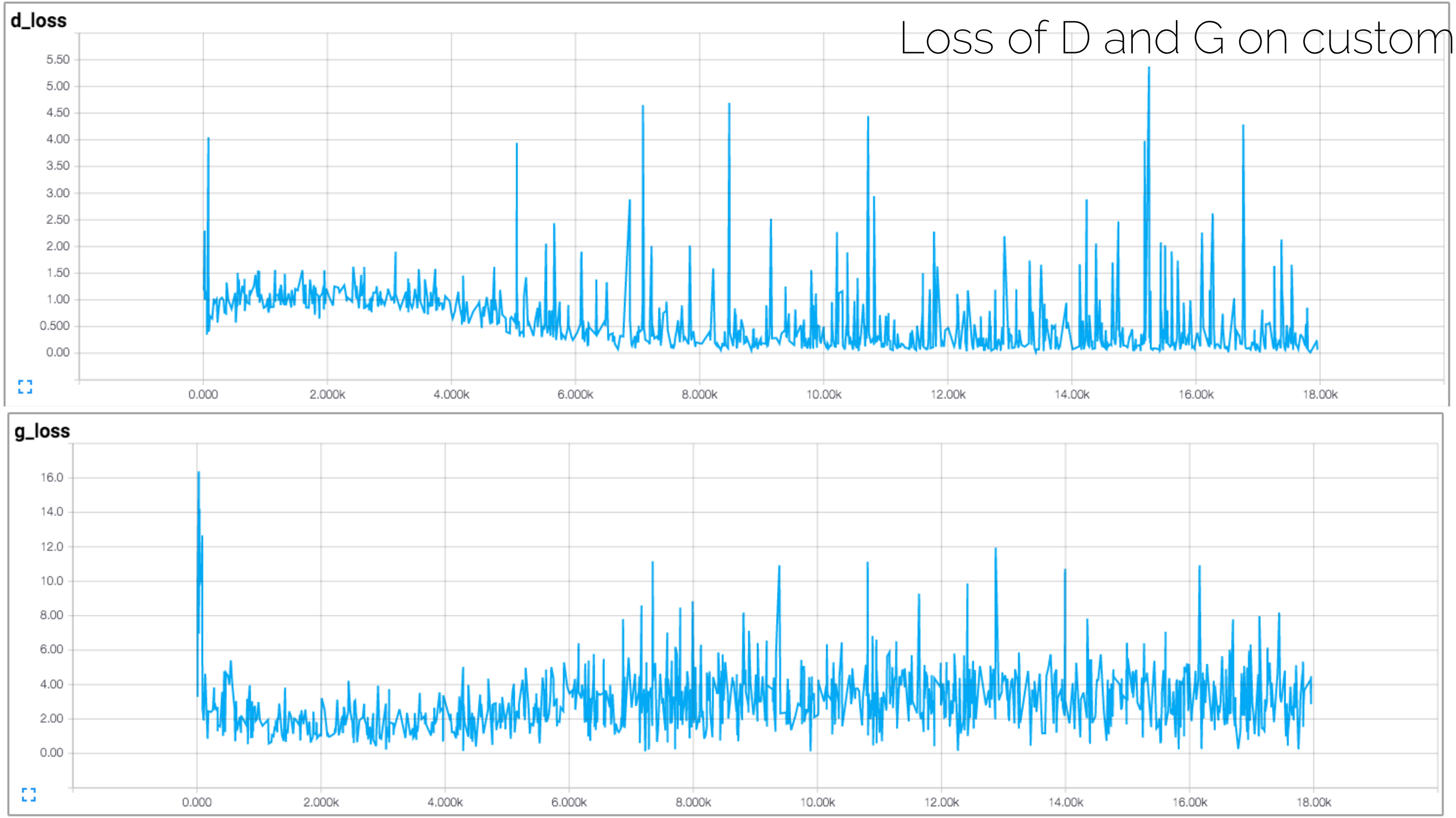
DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

DCGAN: Results

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

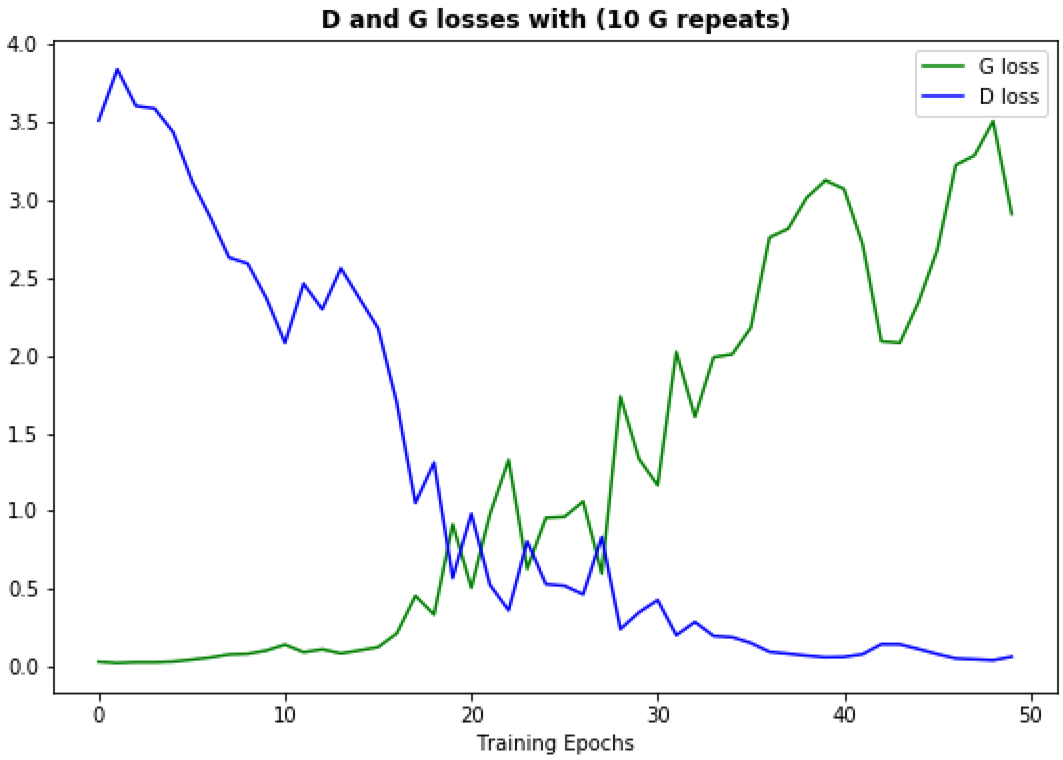
DCGAN: Results

Loss of D and G on custom dataset



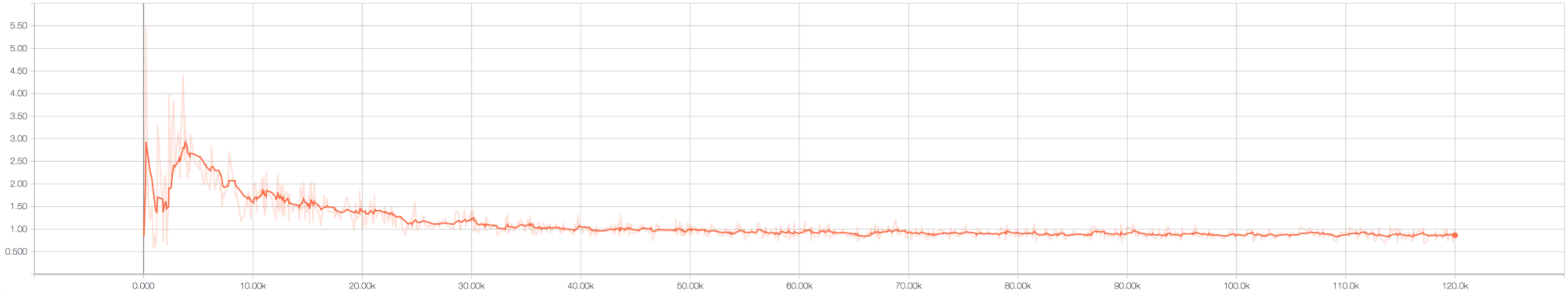
DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

“Bad” Training Curves

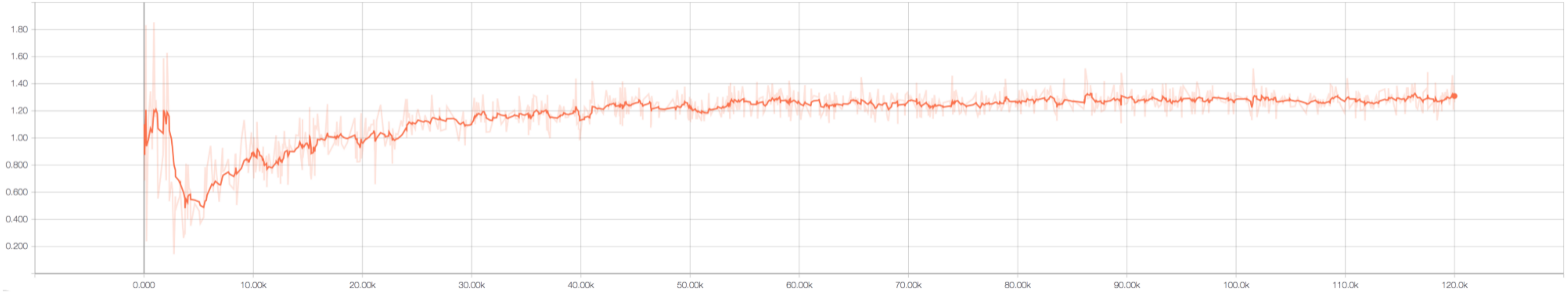


https://stackoverflow.com/questions/44313306/dcgans-discriminator-getting-too-strong-too-quickly-to-allow-generator-to-learn

"Good" Training Curves



Generator's Error through Time



Discriminator's Error through Time

https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f

GAN Applications

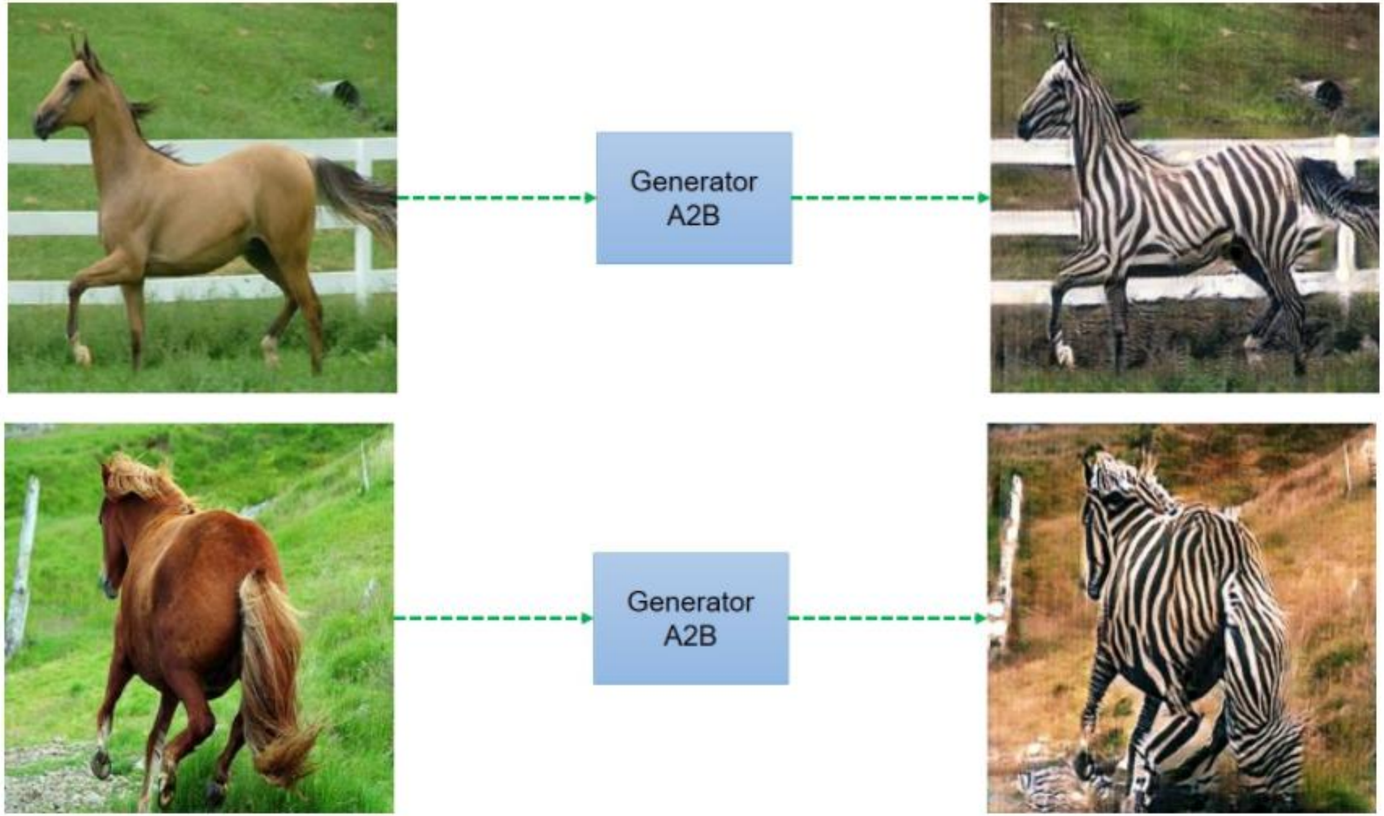
BigGAN: HD Image Generation

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

StyleGAN: Face Image Generation

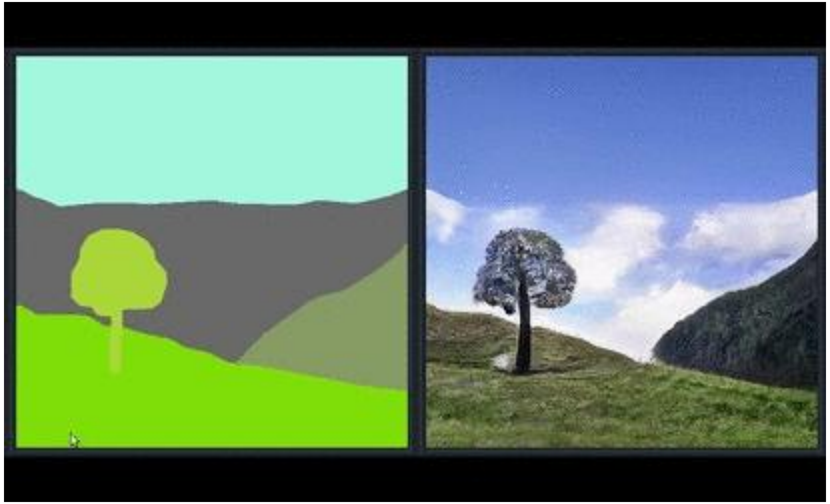
[Karras et al., '18] StyleGAN : A Style-Based Generator Architecture for Generative Adversarial Networks
[Karras et al., '19] StyleGAN2 : Analyzing and Improving the Image Quality of StyleGAN

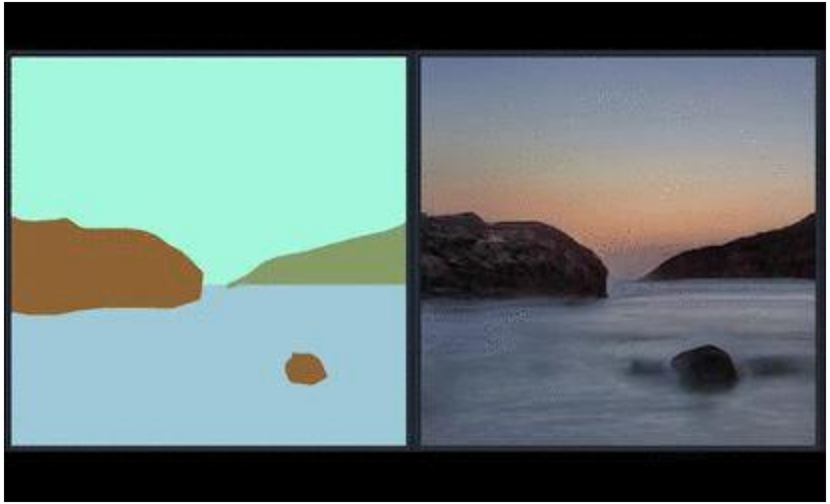
Cycle GAN: Unpaired Image-to-Image Translation



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

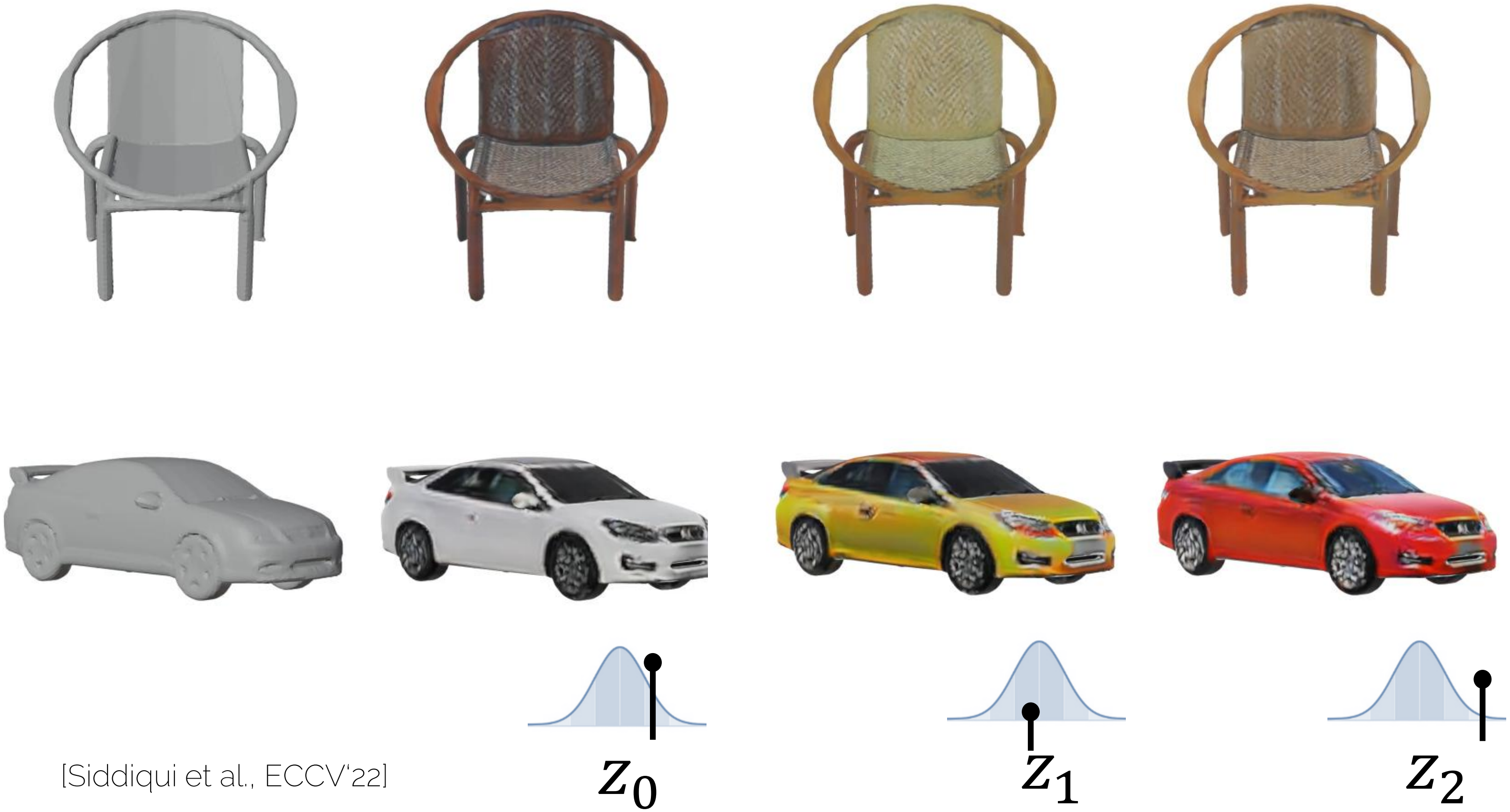
SPADE: GAN-Based Image Editing





[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

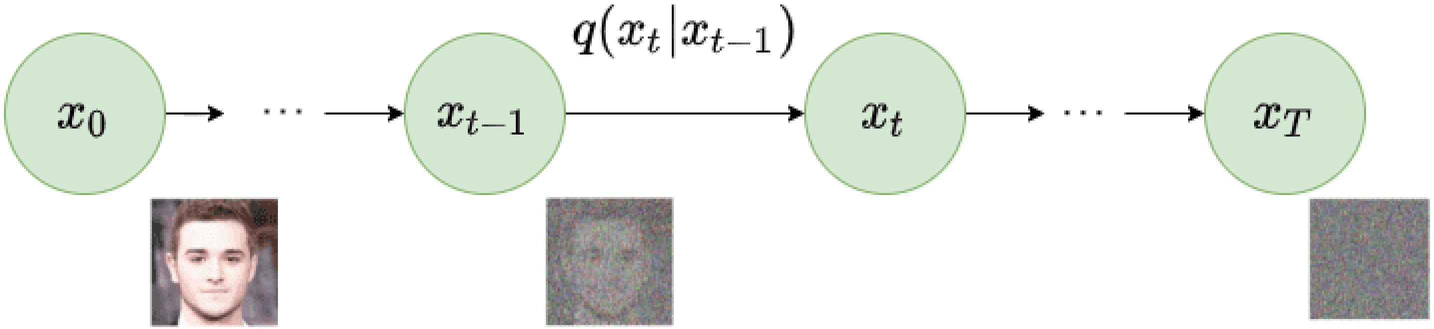
Source: Google Trends

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

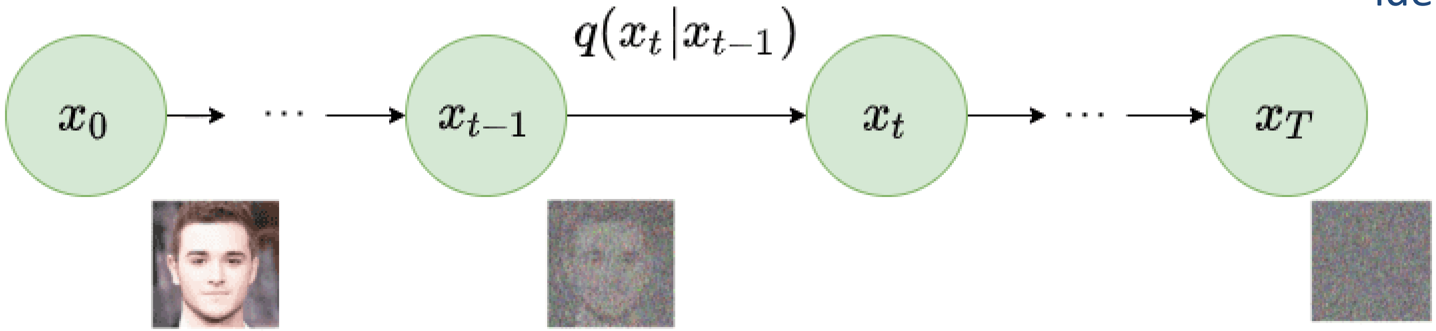
- Markov chain of T steps
  - Each step depends only on previous
- Adds noise to x_0 sampled from real distribution $q(x)$

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

mean

variance

identity matrix



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

Forward Diffusion

- Go from 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

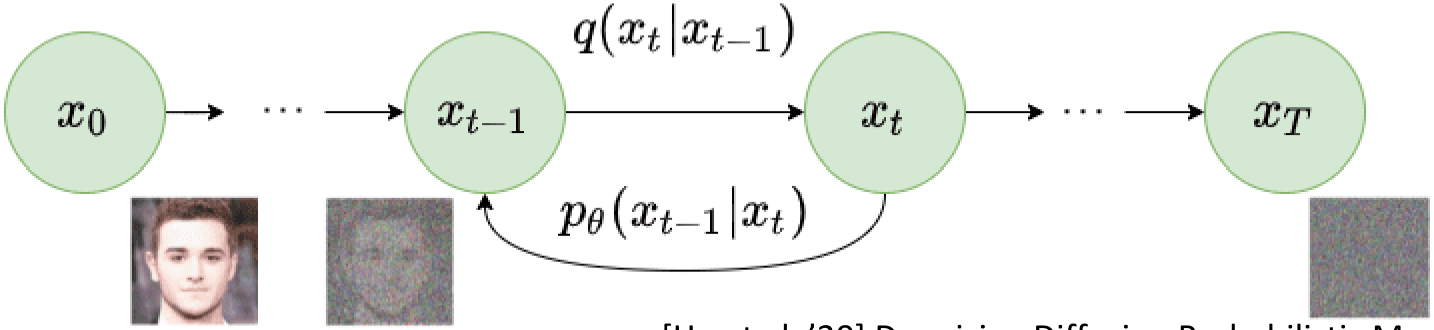
- How to model reverse process?

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

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

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

Training a Diffusion Model

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

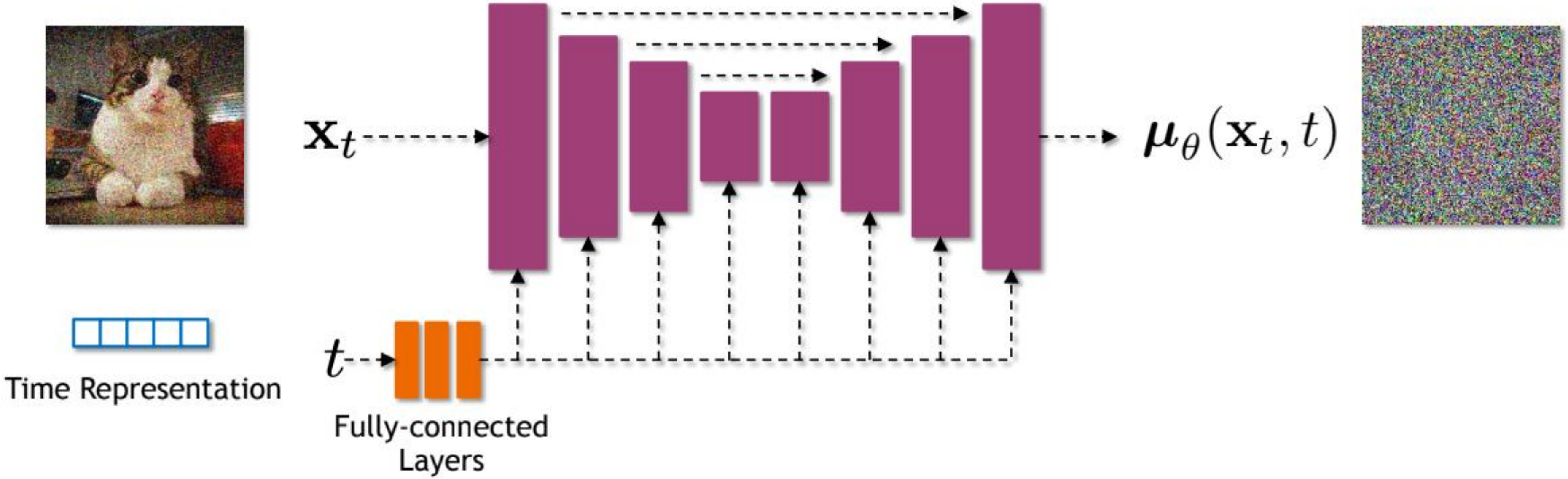
- Predicts diffusion posterior mean

Diffusion Model Architecture

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

Diffusion Model Architecture

Example U-Net as diffusion model

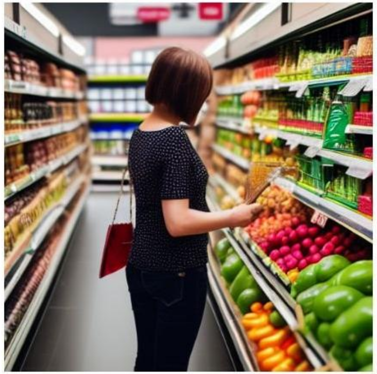


Applications for Diffusion Models

- Text-to-image

Oil Painting

Digital Illustration

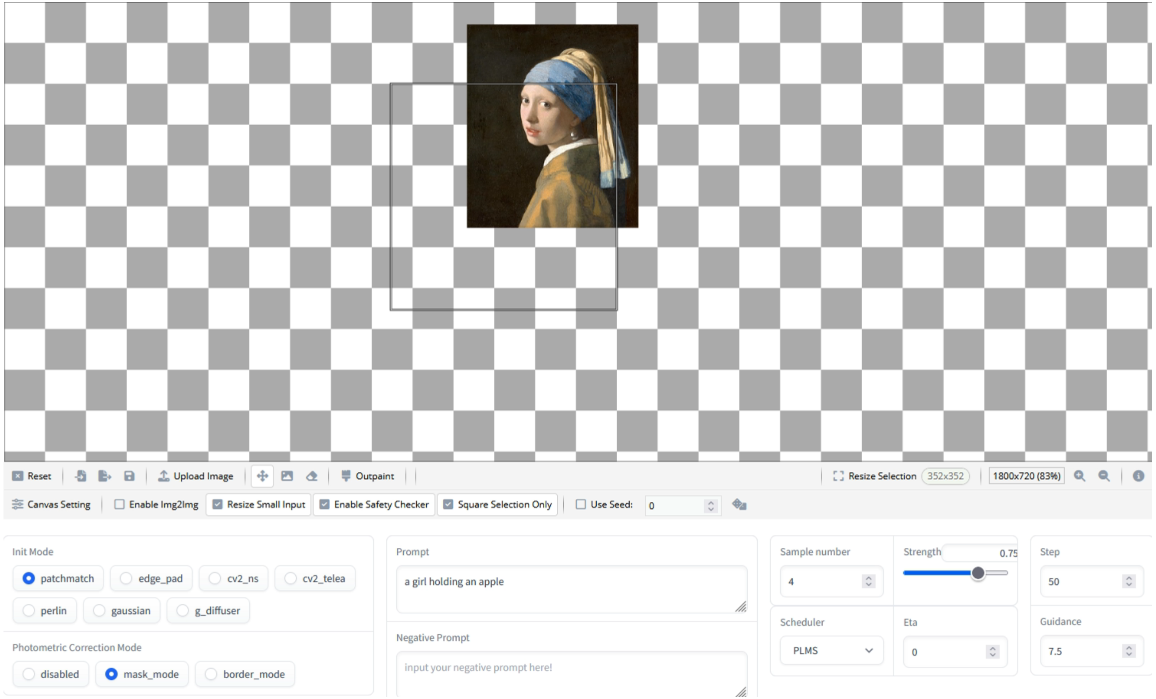


Hyperrealistic

Cartoon

Applications for Diffusion Models

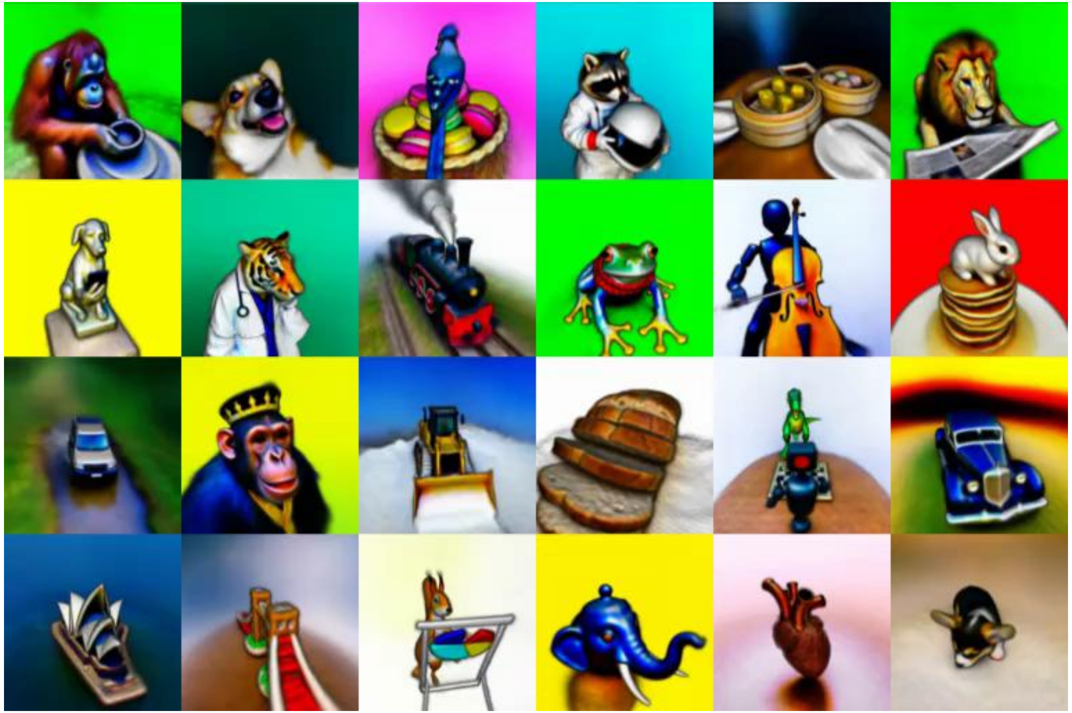
- Image inpainting & outpainting



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

Applications for Diffusion Models

- Text-to-3D Neural Radiance Fields



https://dreamfusion3d.github.io/

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

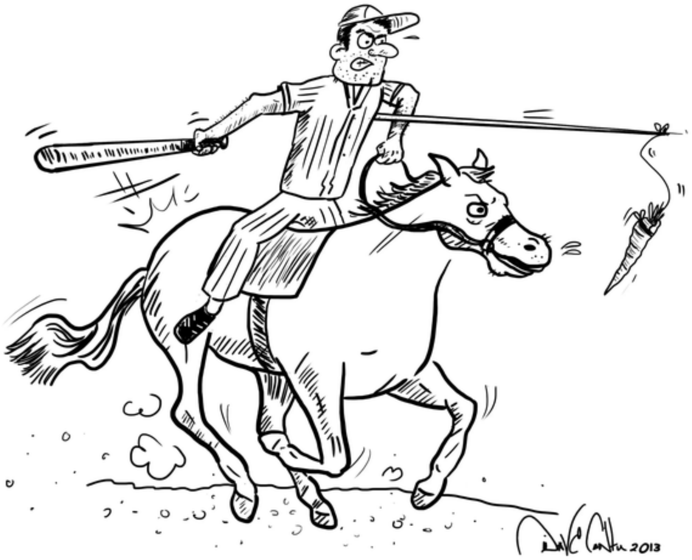
Reinforcement Learning

Sequential data

Learning by interaction with the environment

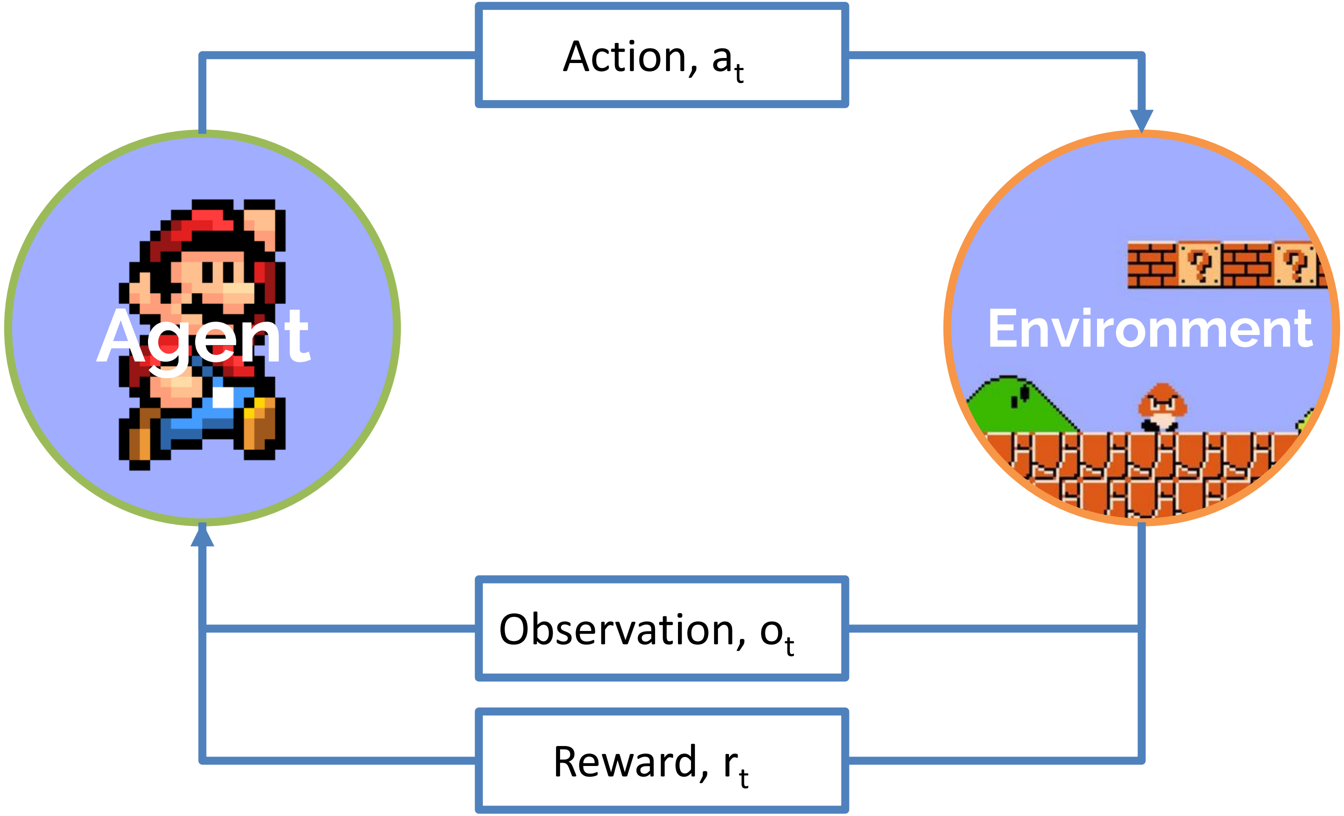
In a Nutshell

- 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

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

- 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:

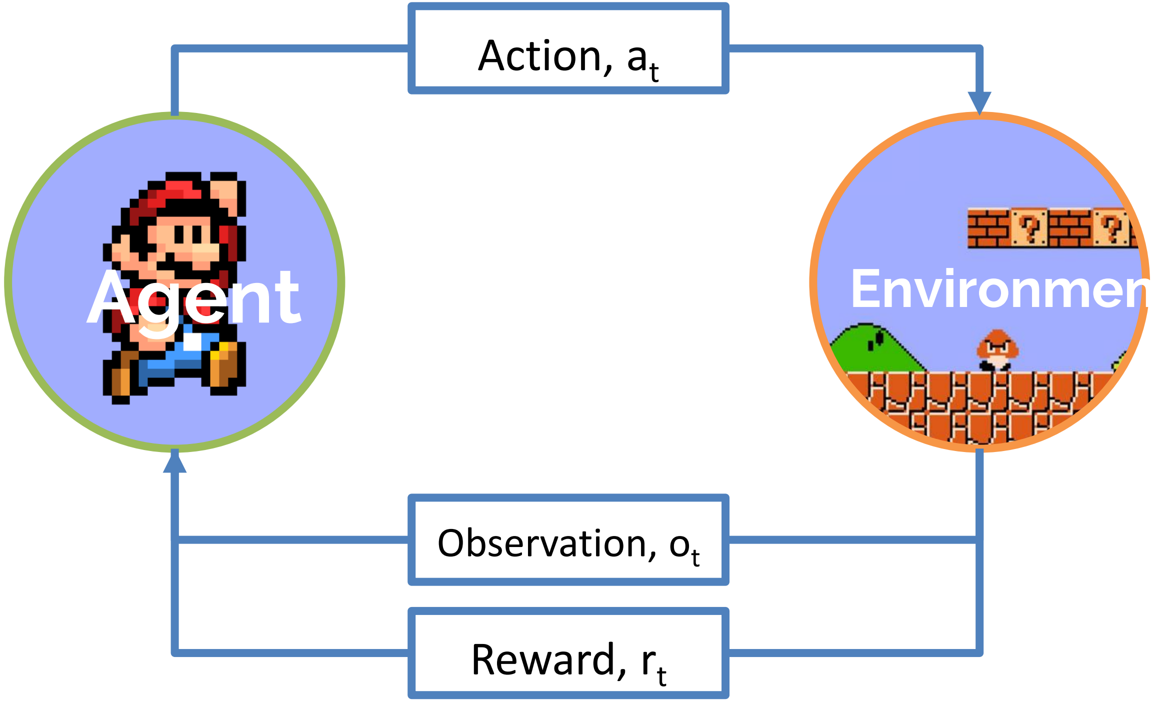
$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \ldots s_t]$$

- "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

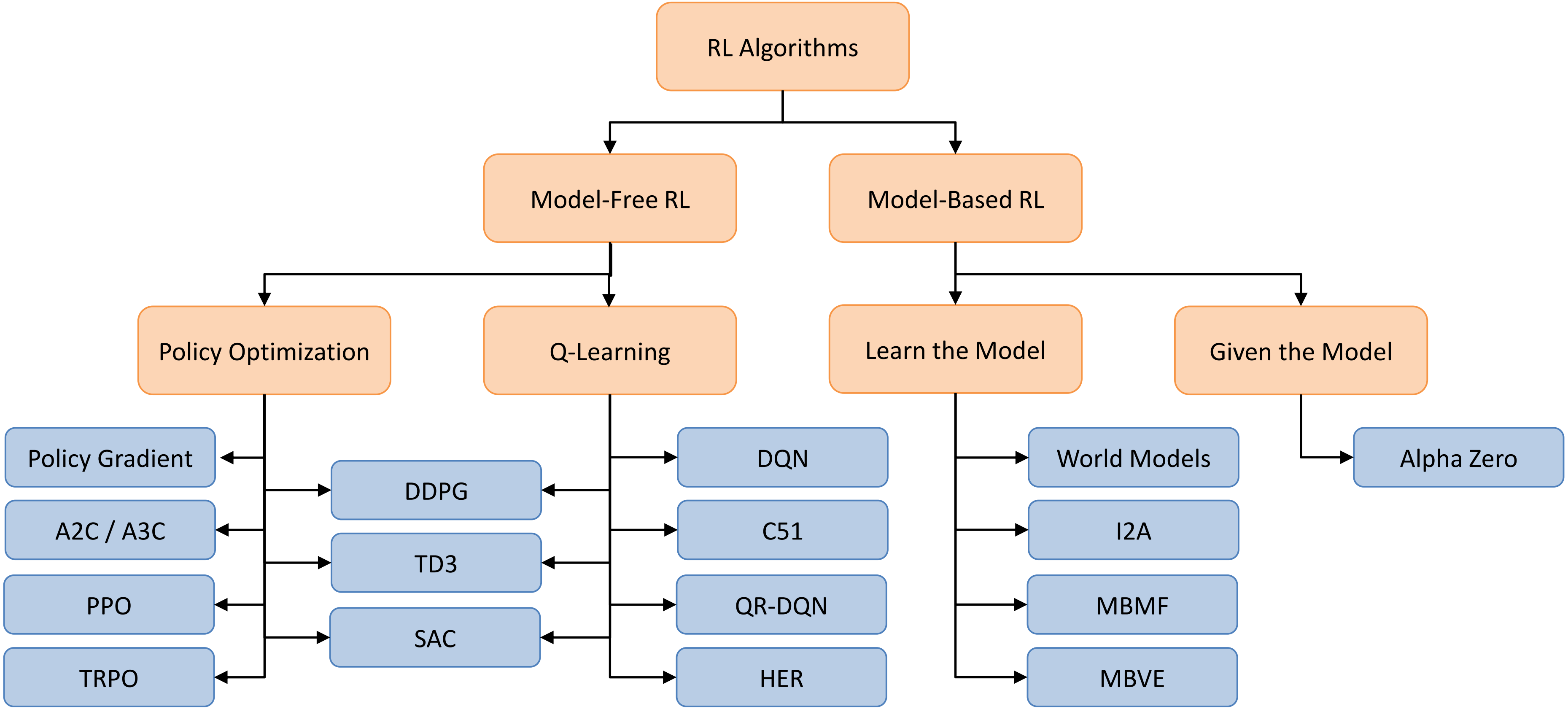
Components of an RL Agent

- **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

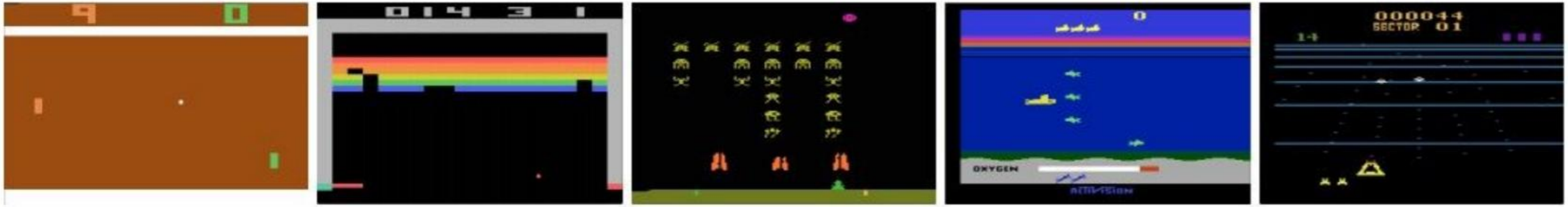
Source: UCL Reinforcement Learning

Taxonomy of RL Algorithms



Source: spinningup.openai.com

RL Milestones: Playing Atari

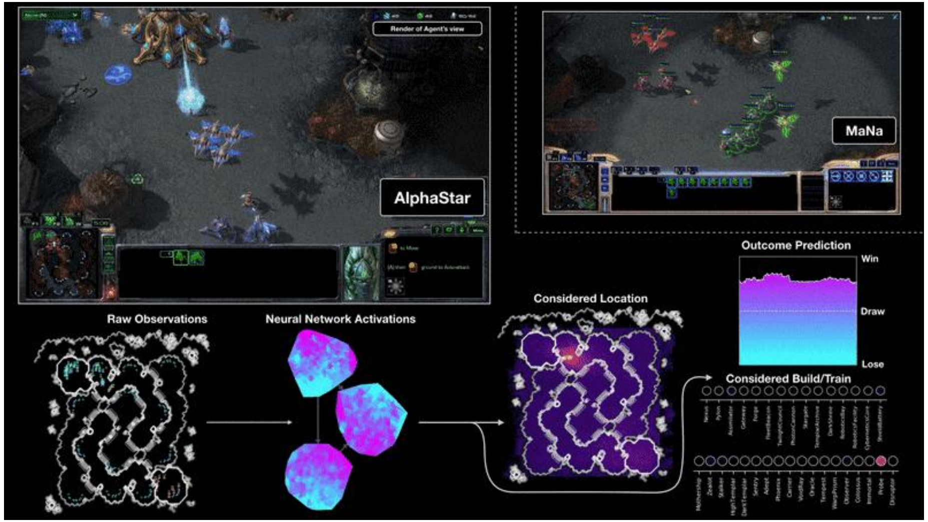


- Mnih et al. 2013, first appearance of DQN
- Successfully learned to play different Atari games like Pong, Breakout, Space Invaders, Seaquest and Beam Rider

[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)



See you next time!