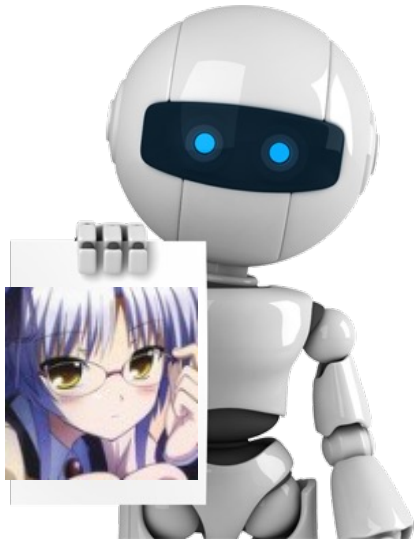
Generative models

Outline

1. **Preview: Auto-Encoders, VAE**
2. Generative models with GAN
3. GAN architectures



Drawing? => learning from examples

Review: Auto-encoder

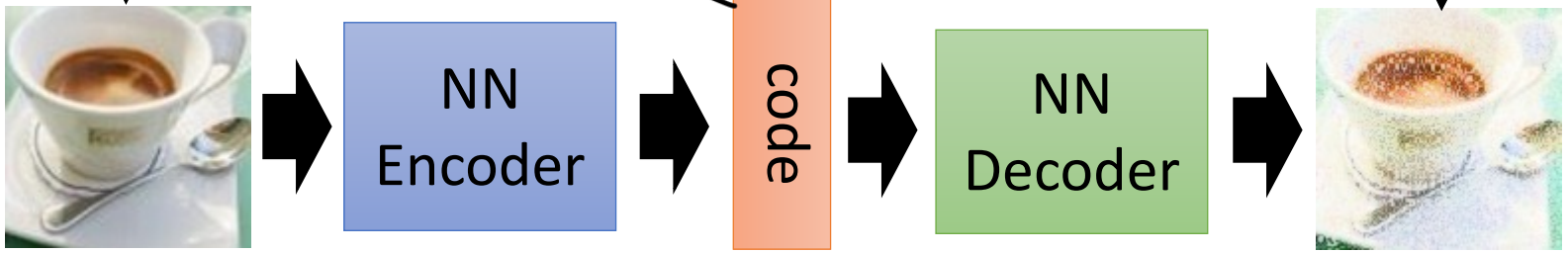
(no label, just the input x)
unsupervised learning

Minimize reconstruction error

very compact latent representation

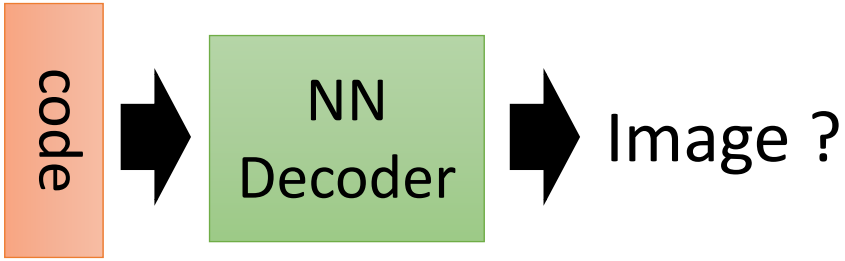
As close as possible

train:

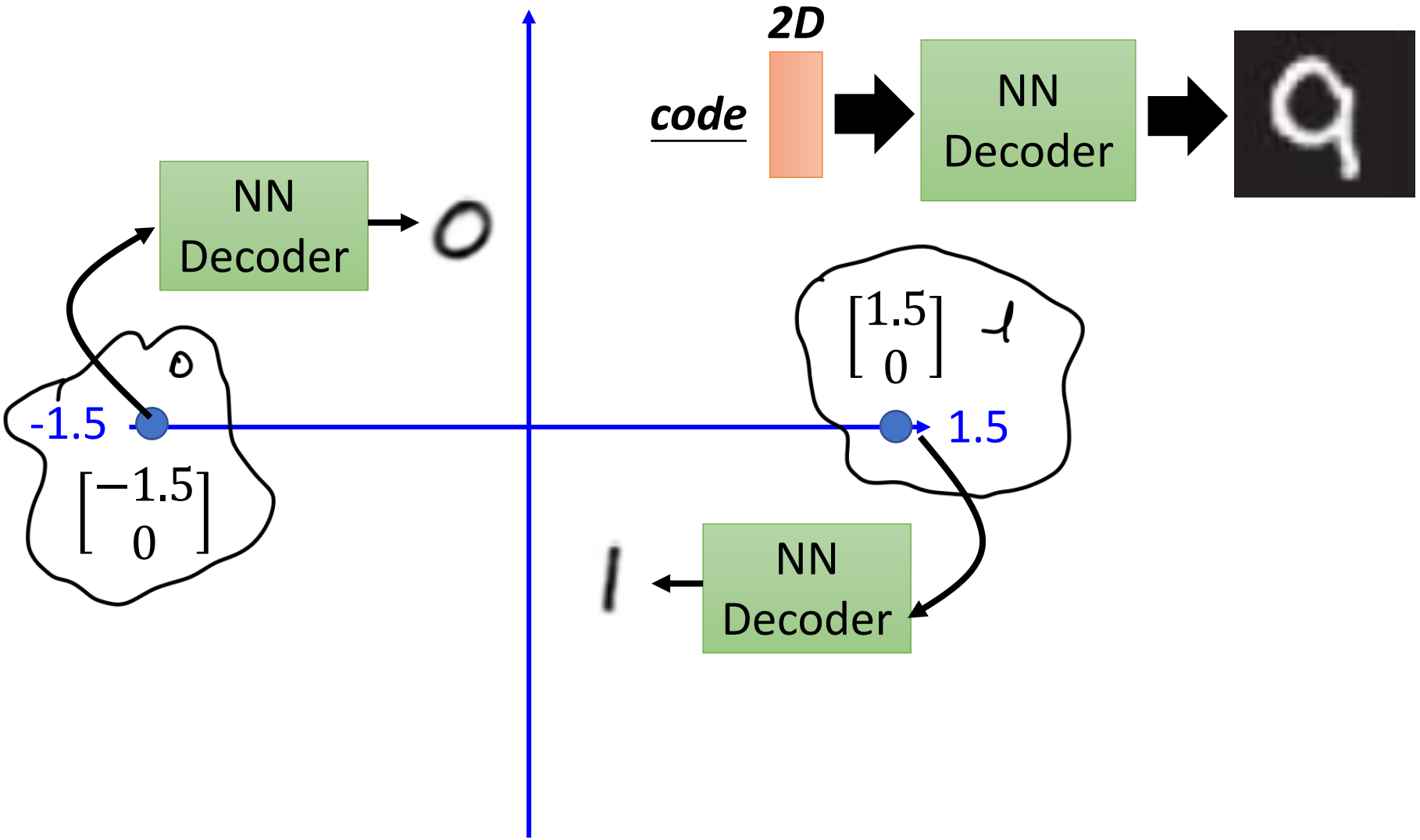


inference:

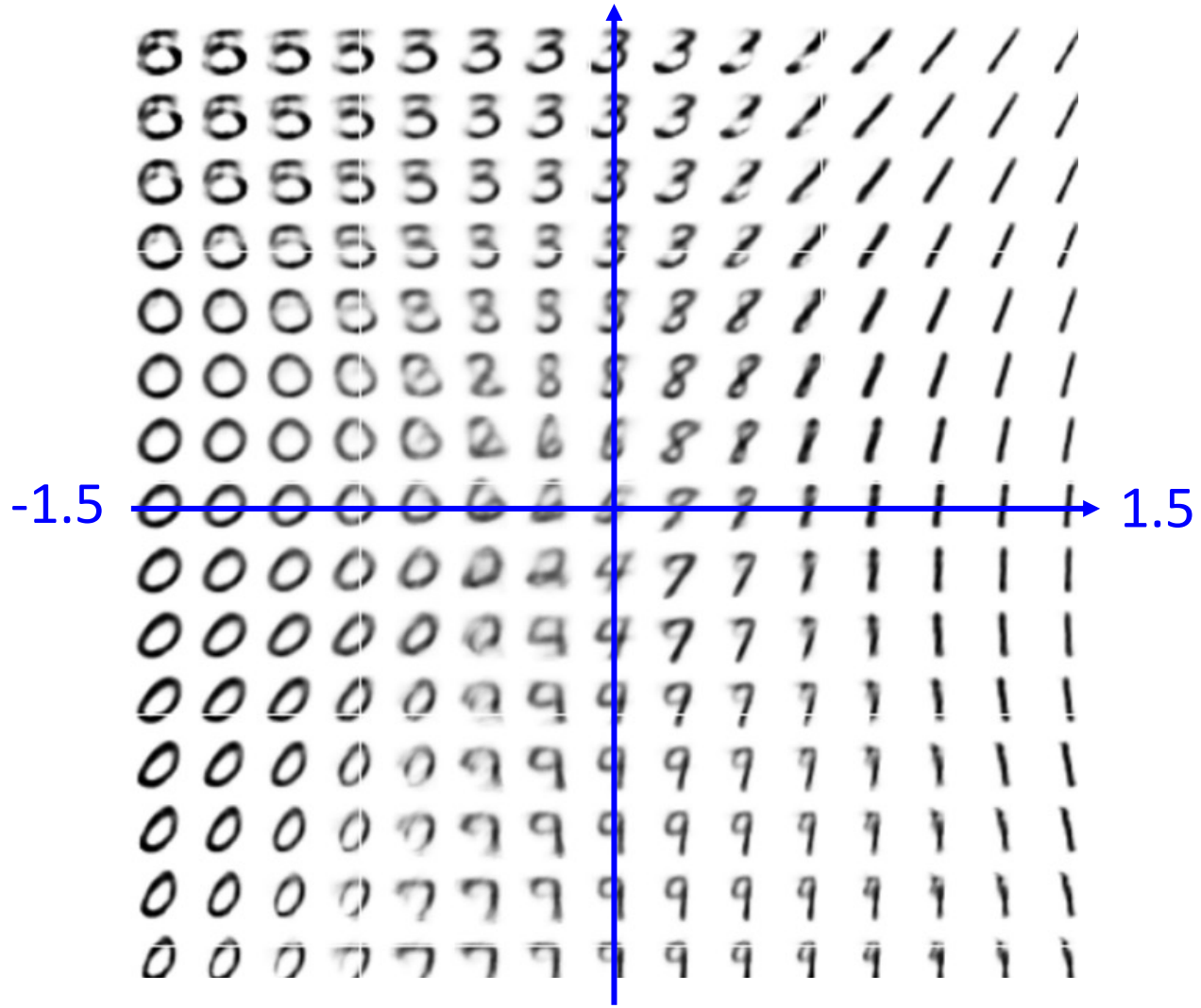
Randomly generate a vector as code



Review: Auto-encoder

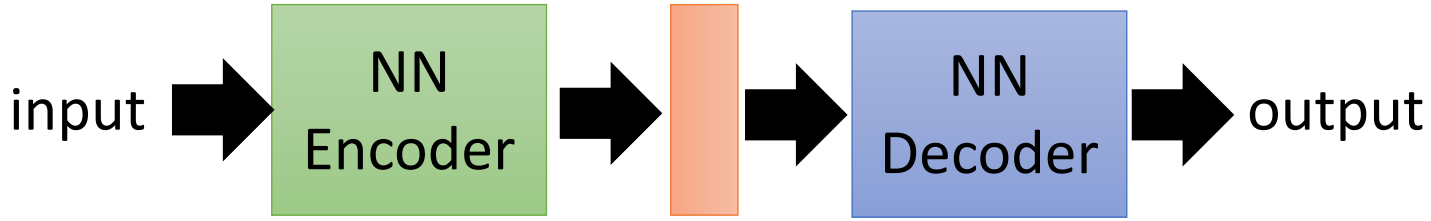


Review: Auto-encoder



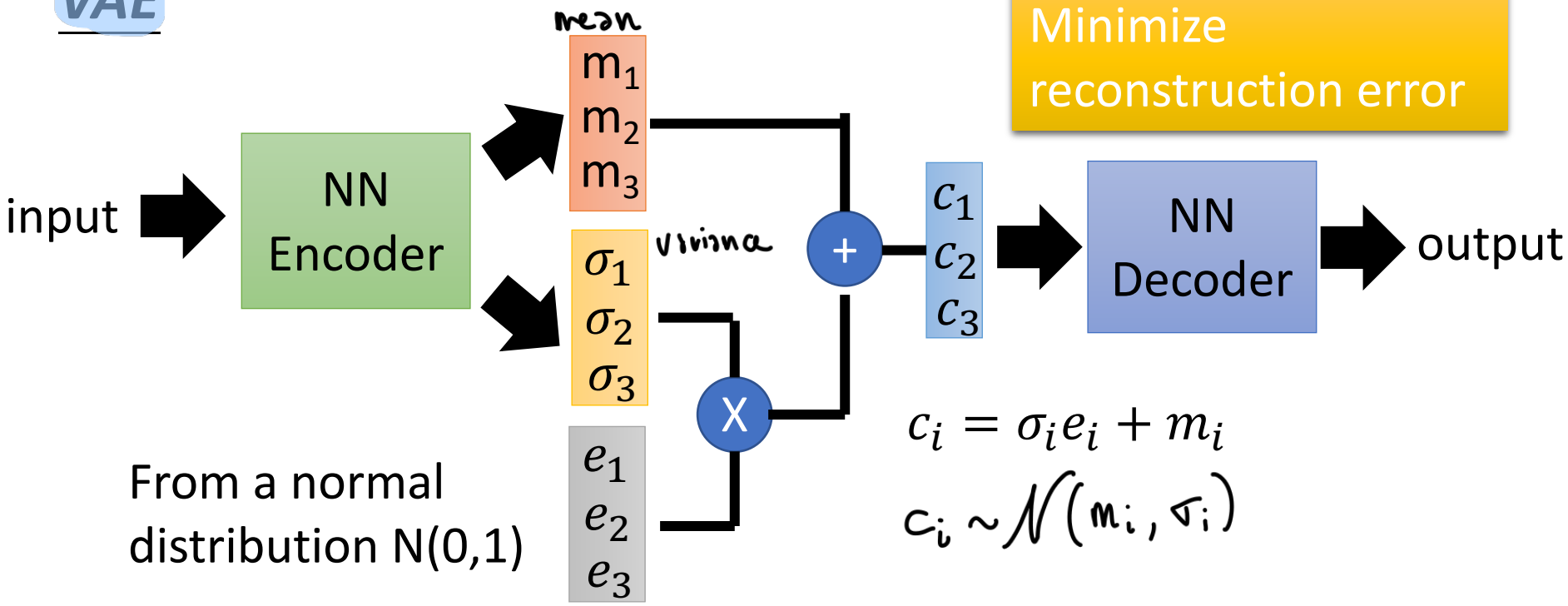
Auto-encoder

VAE: generate not only one point in the latent space, but a prob. distribution $\sim \mathcal{N}(m, \sigma)$



code

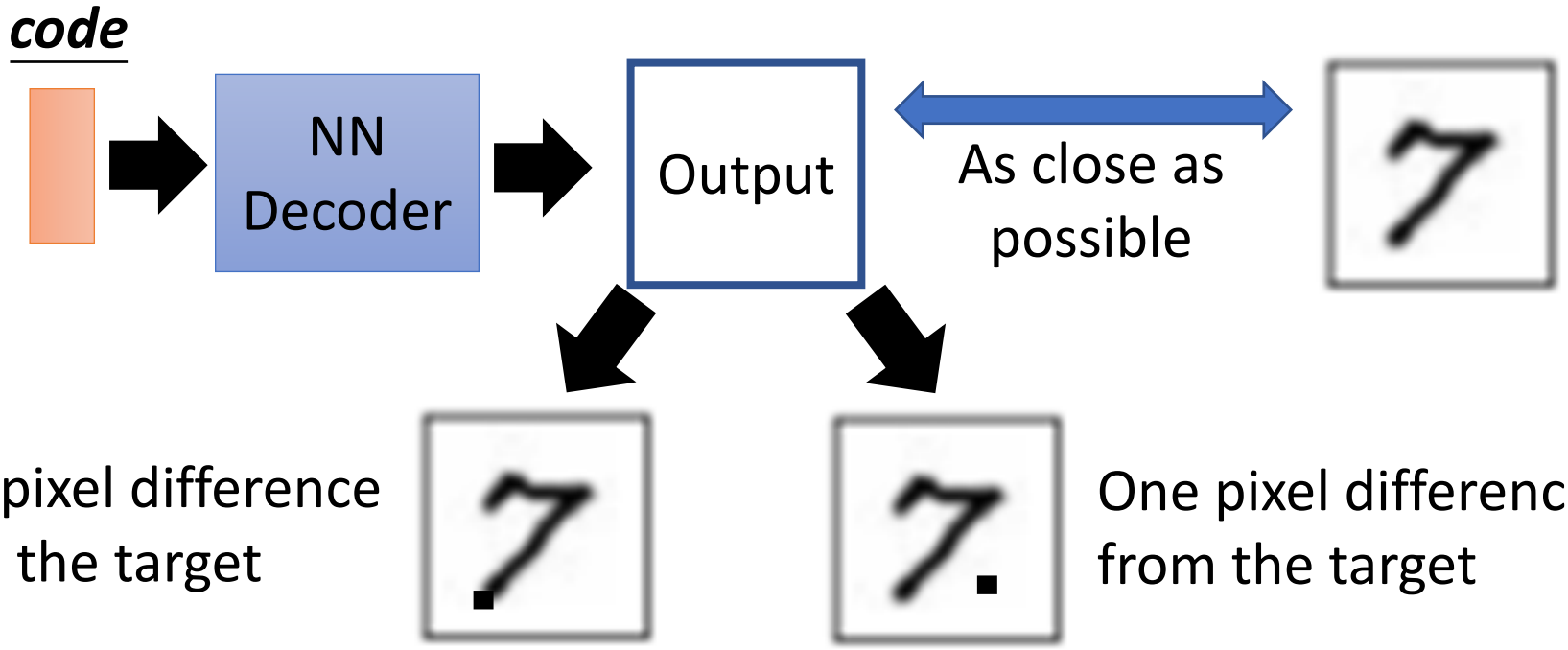
VAE



Auto-Encoding Variational Bayes, https://arxiv.org/abs/1312.6114

Problems of AE/VAE

- It does not really try to simulate real images

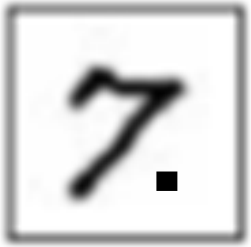


They have same reconstruction error!

Problems of AE/VAE

GAN to tackle this pb:

Realistic



Non Realistic

GAN: generative adversarial networks

Game scenario:

min D's correct classification

Player1, **Generator,** produces samples

max correct classification

Player2, – Its adversary **Discriminator**, attempts to distinguish real samples from fake generated ones (produced by Player1) !

Player1 aims at producing Realistic images to fool the Player2

Generative models

Outline

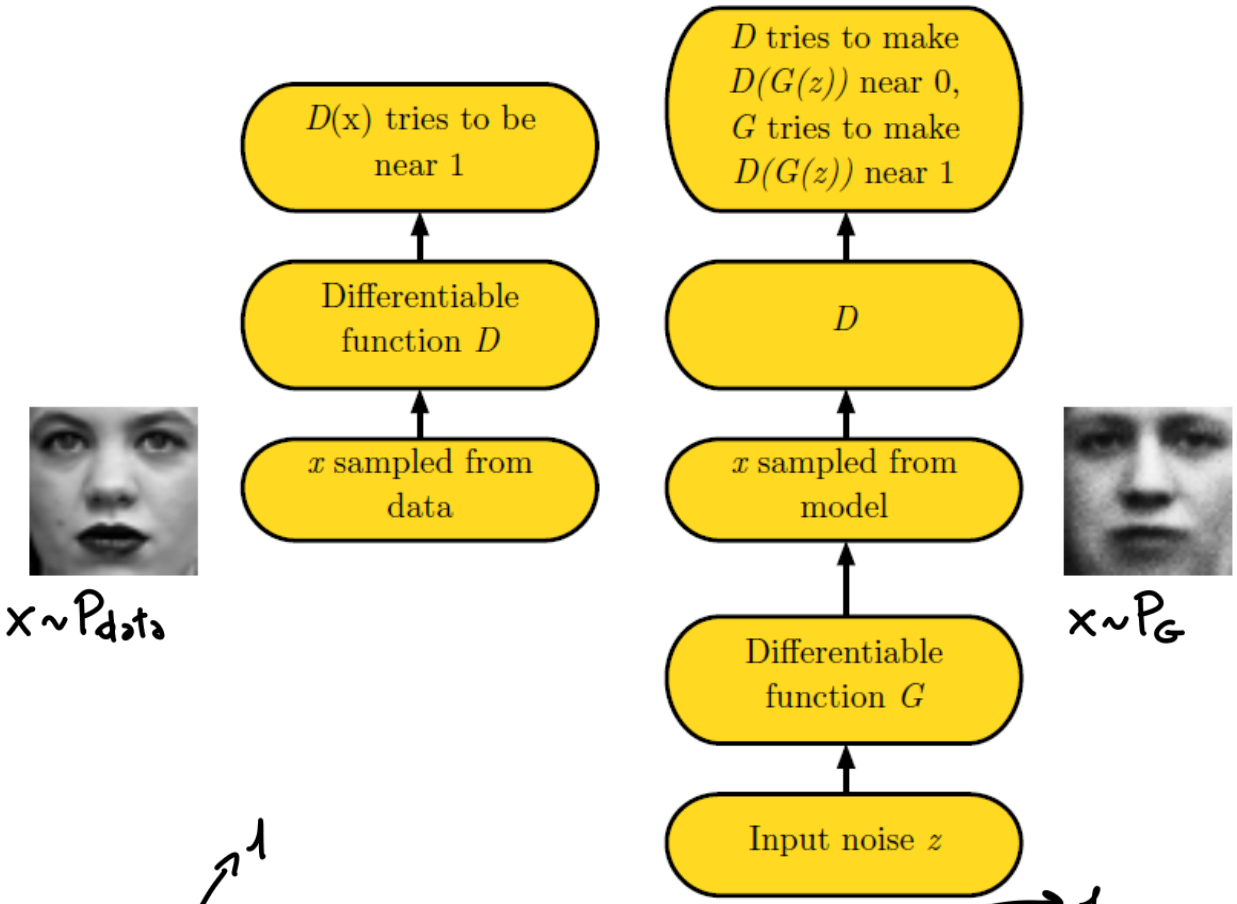
1. Preview: Auto-Encoders, VAE
2. **Generative models with GAN**
   - GAN Algorithm

Adversarial Nets Framework

$D(x) = \begin{cases} 1, & x \sim P_{data} \\ 0, & x \sim P_G \end{cases}$

Game scenario:

Player1, Generator G
Player2, Discriminator D

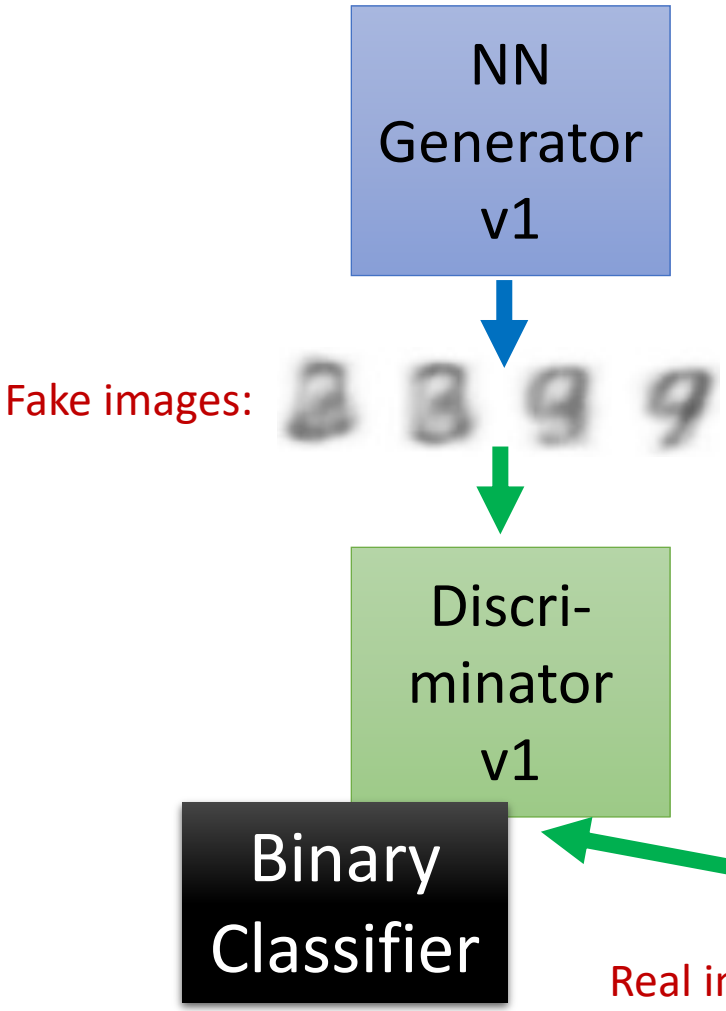


$$V(G,D) = \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{x\sim P_G}[log(1-D(x))]$$

$$G^* = arg \min_G \max_D V(G,D)$$

G updated to minimize this part, to get D(x) → 1

GAN Learning – D and G updates

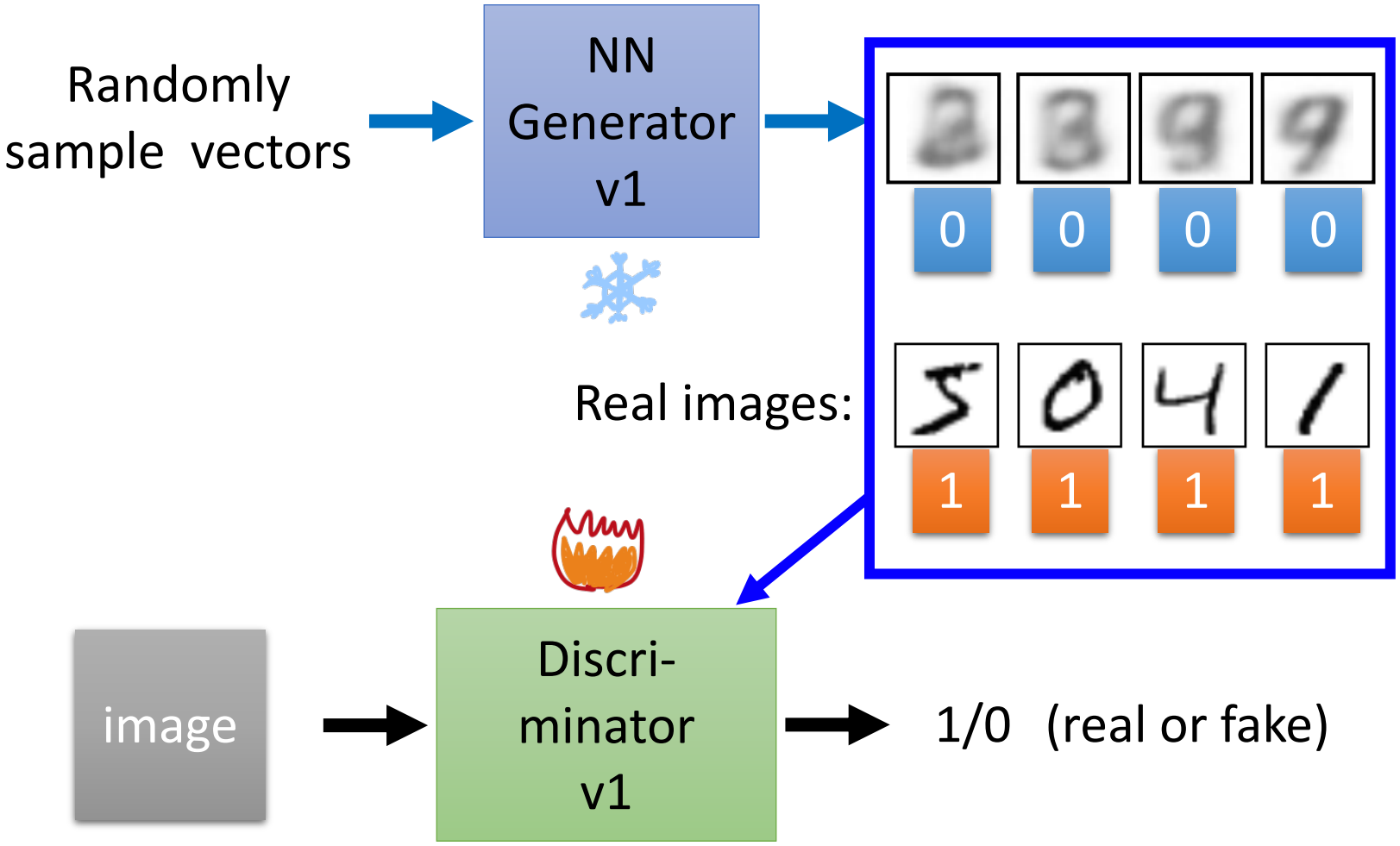


Game scenario:

Player1, Generator G, produces samples
Player2, – Its adversary **Discriminator D**, attempts to distinguish real samples from fake generated ones (produced by Player1) !

Player1 aims at producing Realistic images to fool the Player2

GAN - Discriminator



Discriminator Optimization on a batch of images:

Use gradient descent to update the parameters in the discriminator, with a **freezed** generator

GAN - Generator

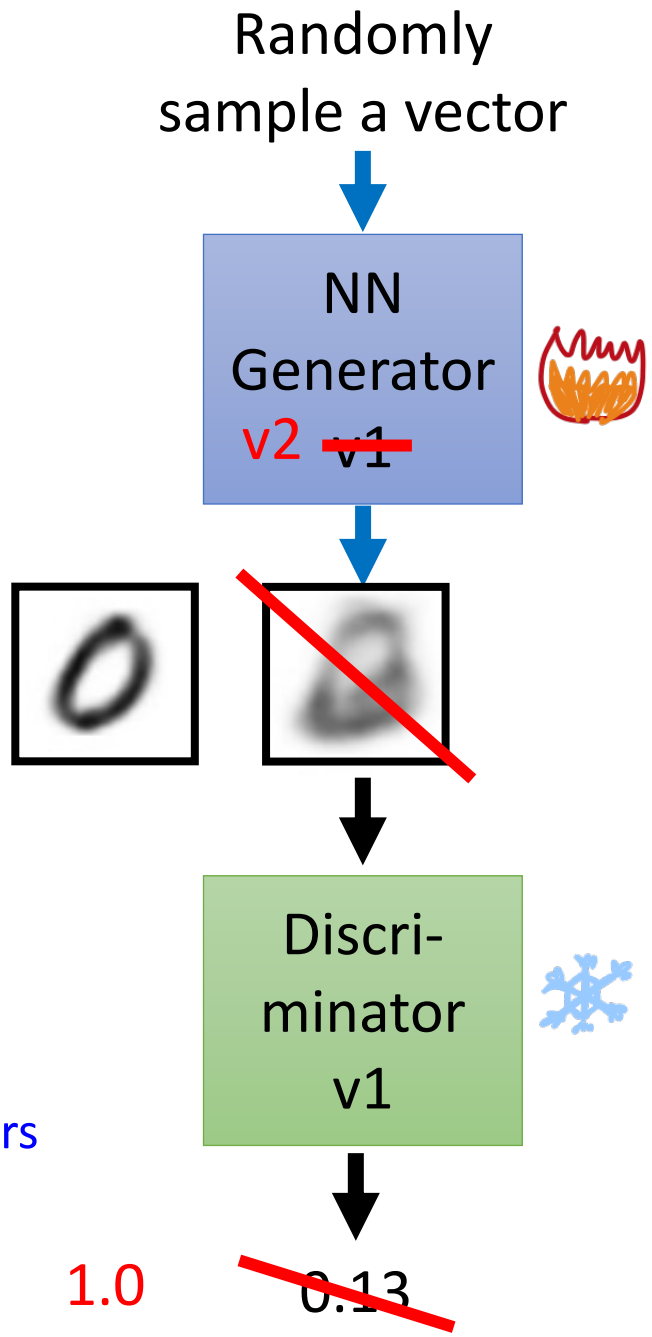
Updating the parameters of generator

The output be classified as "real" (as close to 1 as possible)

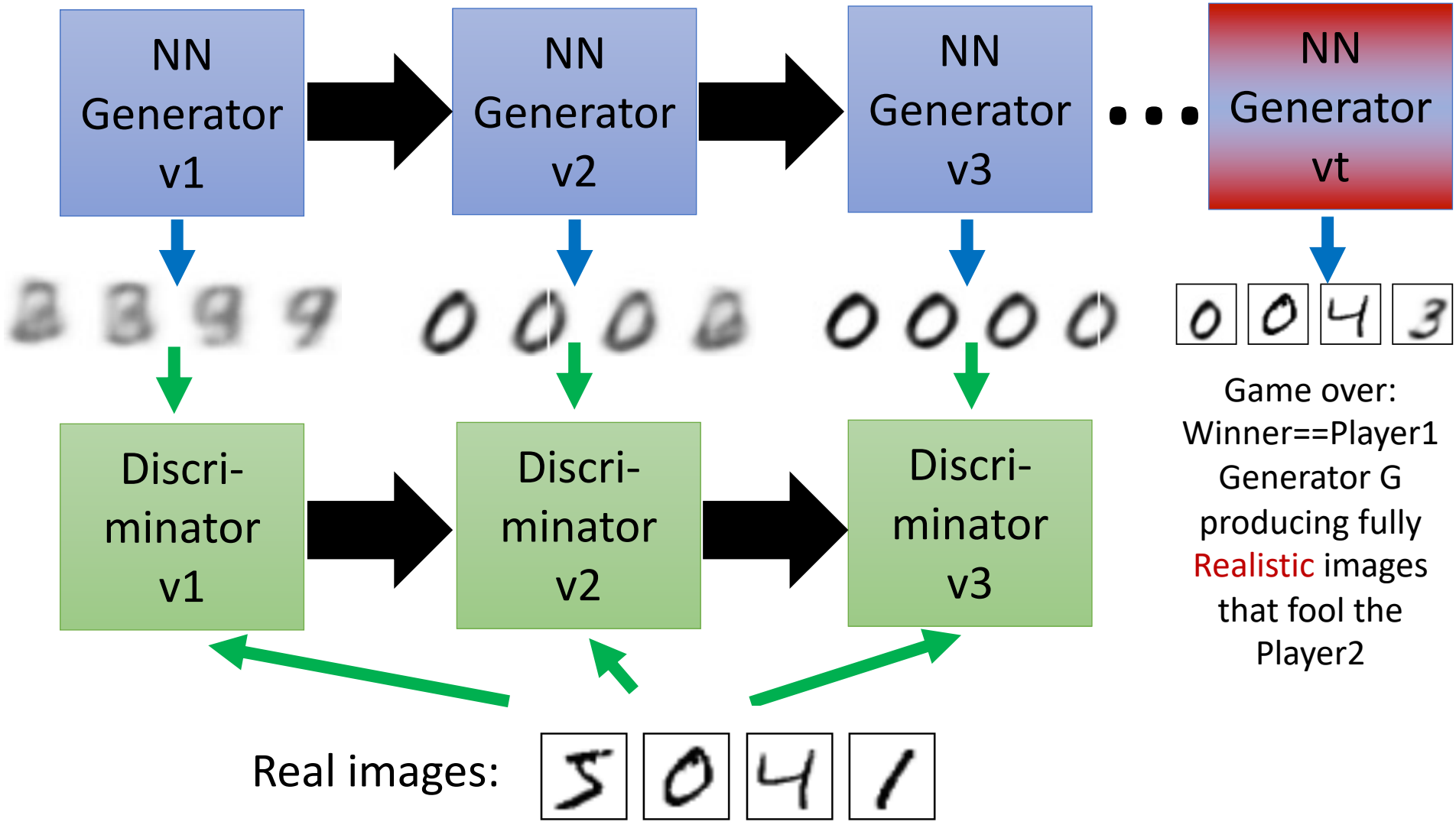
Generator + Discriminator = a network

Optimization:

Use gradient descent to update the parameters in the generator, with a **freezed** discriminator



GAN Learning – D and G updates



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

    • Update the discriminator by ascending its stochastic gradient: only D

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

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

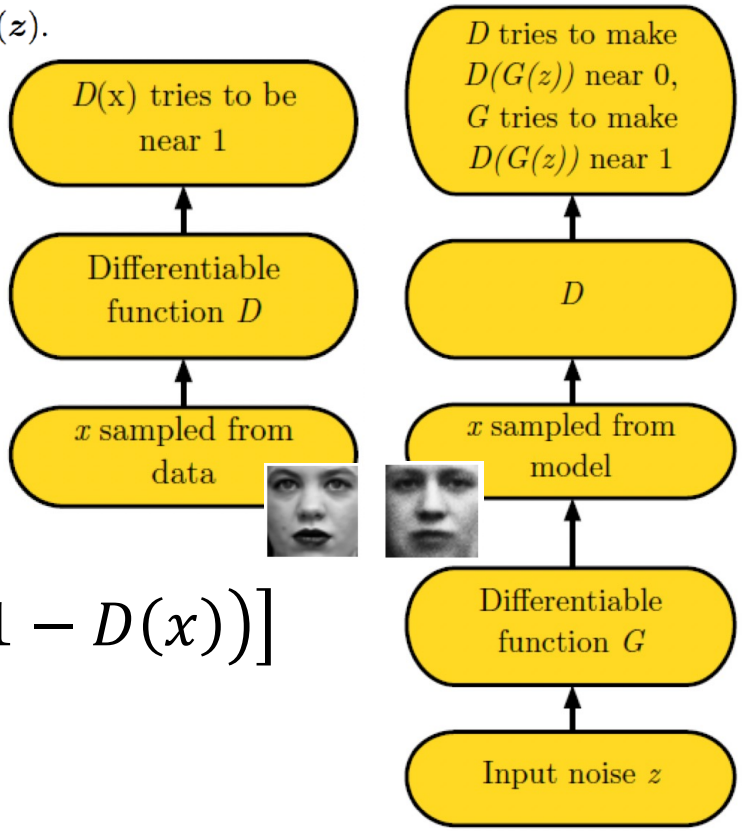
  • Update the generator by descending its stochastic gradient: only G

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

end for

1 step for G every K steps for D (hyperparameter to select)

GAN algorithm



$$V = \mathbb{E}_{x \sim P_{data}}[logD(x)] + \mathbb{E}_{x \sim P_G}[log(1 - D(x))]$$

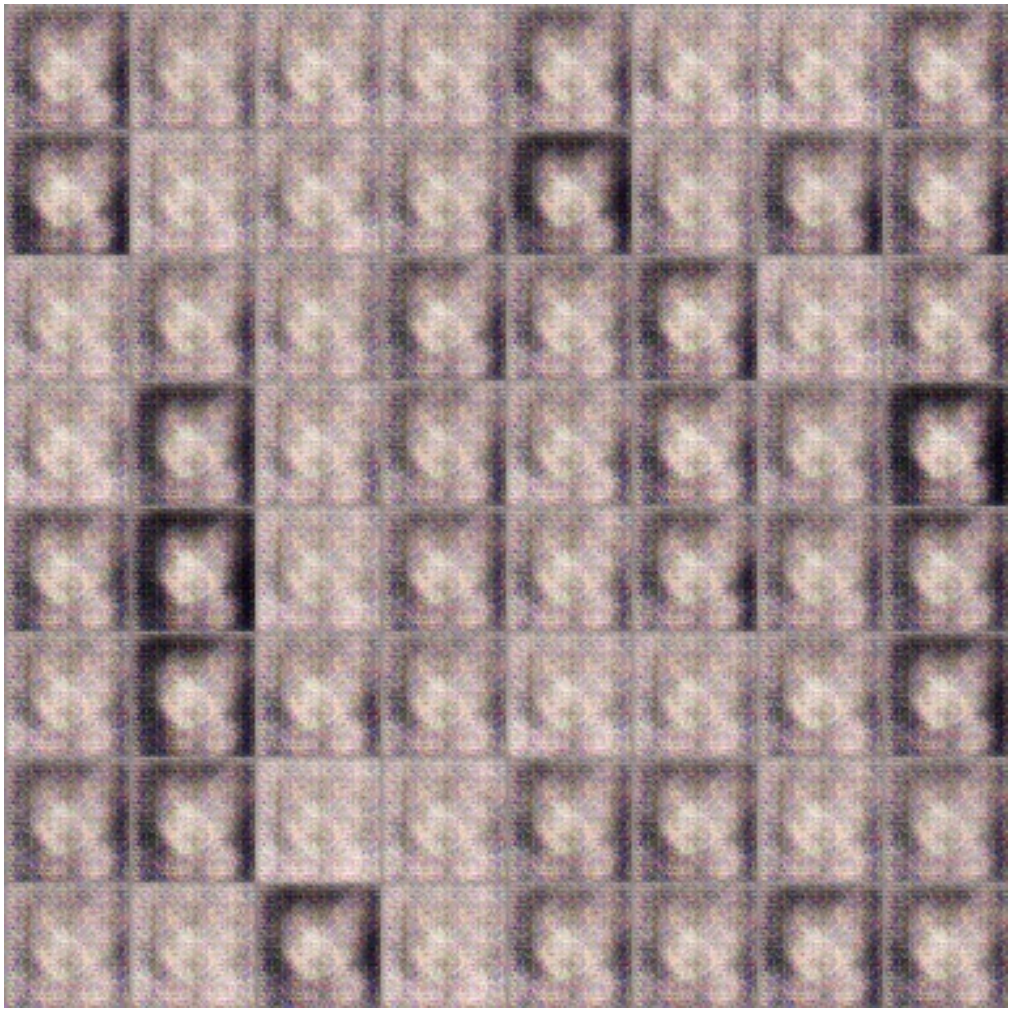
$$G^* = arg \min_G \max_D V(G, D)$$

One example GAN

Source of images: https://zhuanlan.zhihu.com/p/24767059

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

GAN



100 rounds

GAN

1000 rounds

GAN

50,000 rounds

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
   - GAN Algorithm
   - **KL vs. Jensen Shannon Divergence**

$$V(G, D) = \mathbb{E}_{x \sim P_{data}}[logD(x)] + \mathbb{E}_{x \sim P_G}\left[log\left(1 - D(x)\right)\right]$$

$$G^* = arg \min_G \max_D V(G, D)$$

Which measure to evaluate how $P_G(x;\theta)$ is close to $P_{data}(x)$ in Maximum Likelihood optimization?

- Given a data distribution $P_{data}(x)$
- We have a distribution $P_G(x;\theta)$ parameterized by θ
  - E.g. $P_G(x;\theta)$ is a Gaussian Mixture Model, θ are means and variances of the Gaussians
  - We want to find θ such that $P_G(x;\theta)$ close to $P_{data}(x)$

Sample $\{x^1, x^2, \dots, x^m\}$ from $P_{data}(x)$

We can compute $P_G(x^i;\theta)$

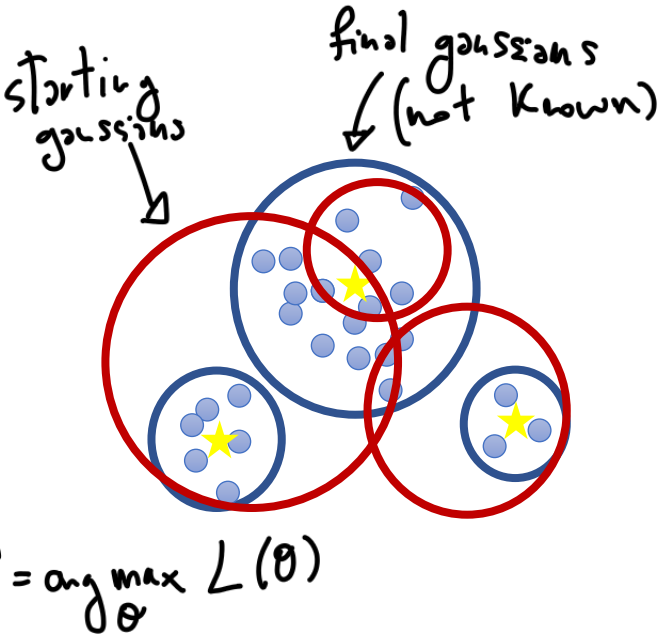
Likelihood of generating the samples

considering independent samples

$$L = \prod_{i=1}^{m} P_G(x^i;\theta)$$

Find θ^* maximizing the likelihood

$$\theta^* = \arg\max_{\theta} L(\theta)$$



Which measure to evaluate how $P_G(x;\theta)$ is close to $P_{data}(x)$ in Maximum Likelihood optimization?

log doesn't change θ^*

$$\theta^* = arg\max_\theta \prod_{i=1}^{m} P_G(x^i;\theta) = arg\max_\theta log \prod_{i=1}^{m} P_G(x^i;\theta)$$

$$= arg\max_\theta \sum_{i=1}^{m} logP_G(x^i;\theta)$$

$\{x^1, x^2, \ldots, x^m\}$ from $P_{data}(x)$

$$\approx arg\max_\theta \mathbb{E}_{x\sim P_{data}}[logP_G(x;\theta)]$$

added constant w.r.t. θ

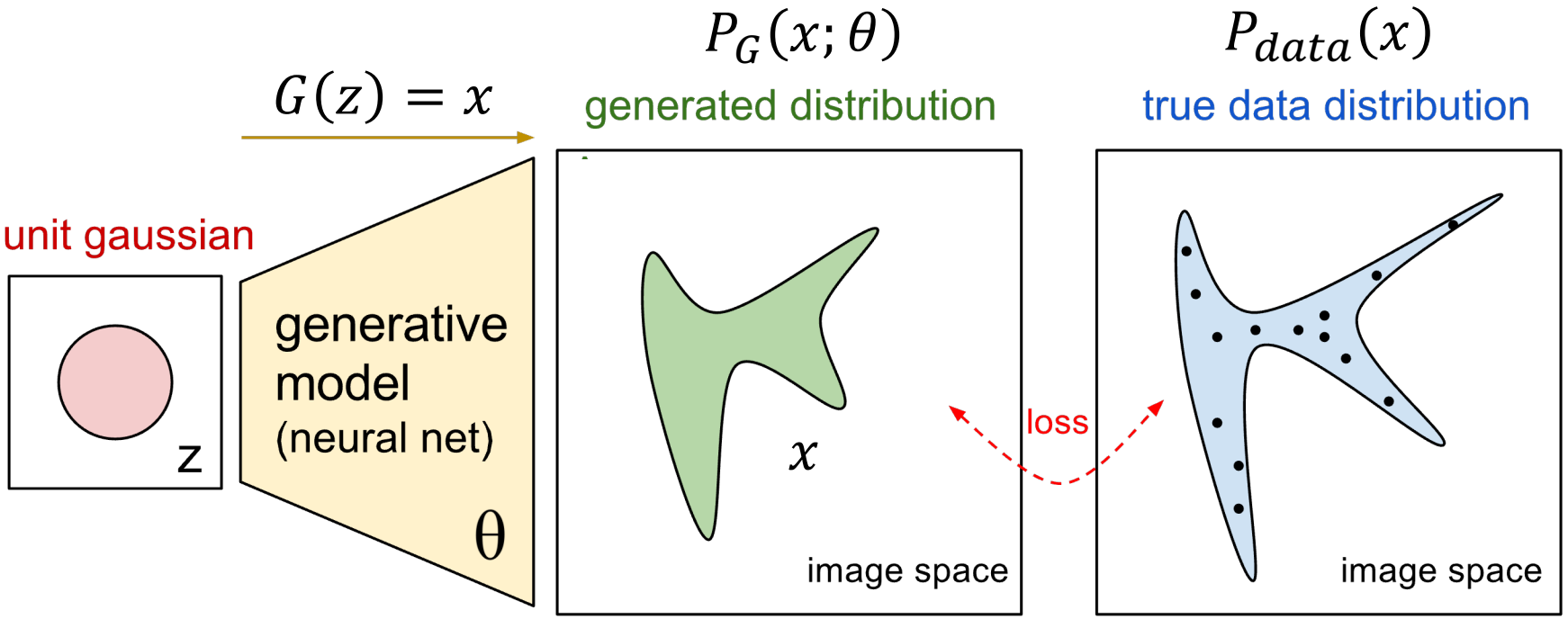
$$= arg\max_\theta \int_x P_{data}(x)logP_G(x;\theta)dx - \int_x P_{data}(x)logP_{data}(x)dx$$

$$= arg\min_\theta KL\left(P_{data}(x)||P_G(x;\theta)\right)$$

$$KL(P||Q) = \int_x P(x)log\frac{P(x)}{Q(x)}dx$$

In Maximum Likelihood it is a KLD Kullback Leibler Divergence

If $P_G(x;\theta)$ is a coming with a NN



$$P_G(x;\theta) = \int_z P_{prior}(z) I_{[G(z)=x]} dz$$

It is difficult to compute the likelihood.

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

Basic Idea of GAN: the 2 players G-D game

- Generator G

  Hard to learn by maximum likelihood

  - G is a function, input z, output x
  - Given a prior distribution $P_{prior}(z)$, a probability distribution $P_G(x)$ is defined by function G (and P_{prior})
- Discriminator D
  - D is a function, input x, output scalar
  - Evaluate the "difference" between $P_G(x)$ and $P_{data}(x)$
- Global objective function V(G,D)

$$\theta^* = G^* = arg \min_G \max_D V(G, D)$$

Basic Idea

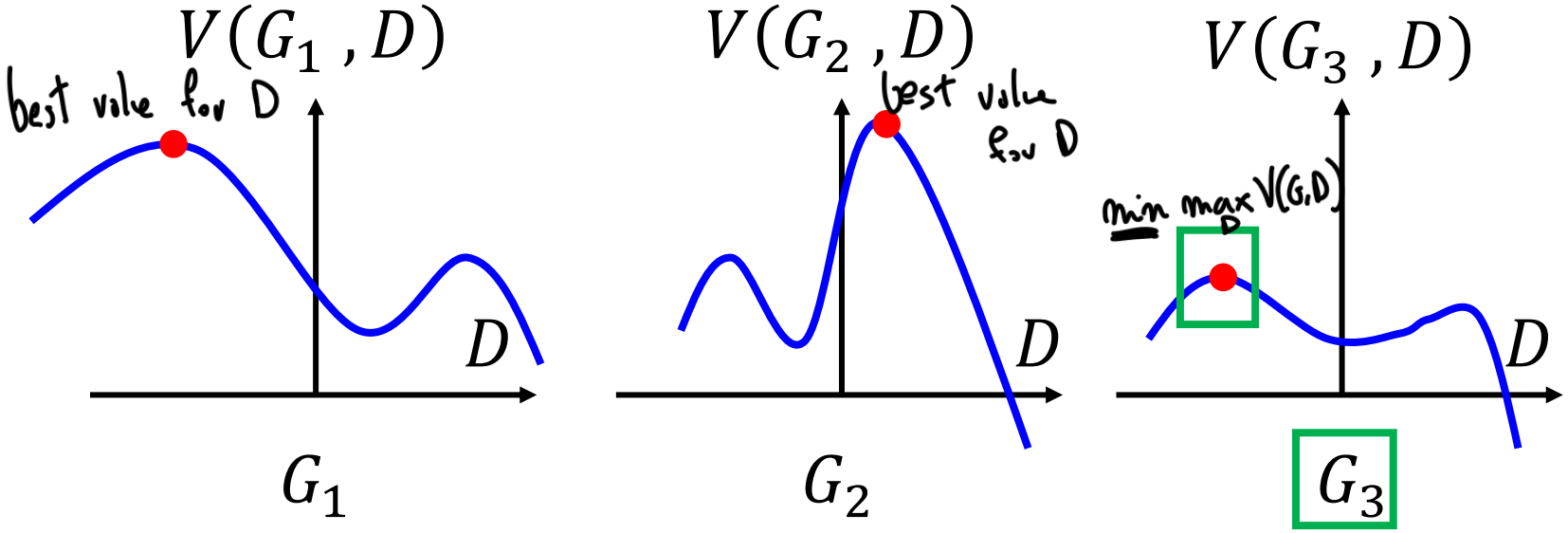
G^* picks the minimum among the maximum of $V(G,D)$

$$G^* = arg \min_G \max_D V(G,D)$$

$$V = \mathbb{E}_{x \sim P_{data}}[logD(x)] + \mathbb{E}_{x \sim P_G}[log(1 - D(x))]$$

Given a generator G, $\max_D V(G,D)$ evaluate the "difference" between P_G and P_{data}

Pick the G defining P_G most similar to P_{data}



$$\max_D V(G,D) \qquad G^* = arg \min_G \max_D V(G,D)$$

- Given G, what is the optimal D* maximizing

$$V = \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{x\sim P_G}\left[log\left(1-D(x)\right)\right]$$

$$= \int_x P_{data}(x) logD(x)\, dx + \int_x P_G(x) log\left(1-D(x)\right) dx$$

$$= \int_x \left[P_{data}(x) logD(x) + P_G(x) log\left(1-D(x)\right)\right] dx$$

Assume that D(x) can have any value here

- Given x, the optimal D* maximizing

$$P_{data}(x) logD(x) + P_G(x) log\left(1-D(x)\right)$$

$$\max_D V(G,D) \qquad G^* = arg\min_G \max_D V(G,D)$$

- Given x, the optimal D* maximizing

$$P_{data}(x)logD(x) + P_G(x)log\big(1-D(x)\big)$$

a D b D

- Find D* maximizing: $\mathrm{f}(D) = \mathrm{a}log(D) + blog(1-D)$

$$\frac{d\mathrm{f}(D)}{dD} = a\times\frac{1}{D} + b\times\frac{1}{1-D}\times(-1) \quad = 0$$

$$a\times\frac{1}{D^*} = b\times\frac{1}{1-D^*} \qquad a\times(1-D^*) = b\times D^*$$

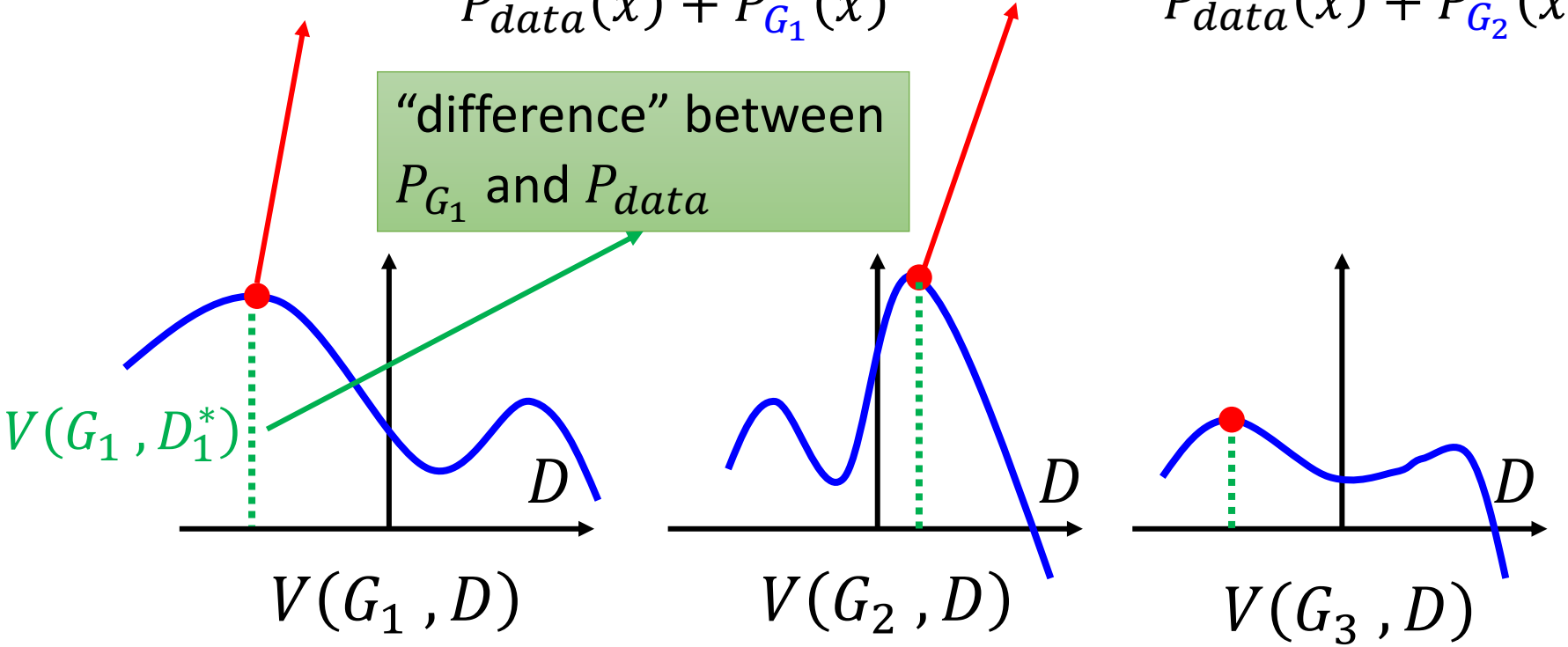
$$a - aD^* = bD^*$$

$$D^* = \frac{a}{a+b} \Rightarrow \qquad D^*(x) = \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}$$

$0 <$ < 1

$$\max_D V(G,D) \qquad G^* = arg \min_G \max_D V(G,D)$$

$$D_1^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_{G_1}(x)} \qquad D_2^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_{G_2}(x)}$$



$$\max_D V(G,D)$$

$$V = \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{x\sim P_G}[log(1-D(x))]$$

$$\max_D V(G,D) = V(G,D^*) \qquad D^*(x) = \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}$$

$$= \mathbb{E}_{x\sim P_{data}}\left[log\frac{P_{data}(x)}{P_{data}(x)+P_G(x)}\right] + \mathbb{E}_{x\sim P_G}\left[log\frac{P_G(x)}{P_{data}(x)+P_G(x)}\right]$$

$$= \int_x P_{data}(x) log \frac{\frac{1}{2}\,P_{data}(x)}{\frac{P_{data}(x)+P_G(x)}{2}} dx + \int_x P_G(x) log \frac{\frac{1}{2}\,P_G(x)}{\frac{P_{data}(x)+P_G(x)}{2}} dx$$

$$=> +2log\frac{1}{2} = -2log2$$

$$\max_D V(G,D)$$

$\mathrm{JSD}(P||Q) = \frac{1}{2}\mathrm{KL}(P||M) + \frac{1}{2}\mathrm{KL}(Q||M)$

$\mathrm{M} = \frac{1}{2}(P+Q)$

$$\max_D V(G,D) = V(G,D^*) \qquad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$= -2log2 + \int_x P_{data}(x) log \frac{P_{data}(x)}{\big(P_{data}(x) + P_G(x)\big)/2} dx$$

$$+ \int_x P_G(x) log \frac{P_G(x)}{\big(P_{data}(x) + P_G(x)\big)/2} dx$$

$$= -2\log2 + \mathrm{KL}\left(\mathrm{P_{data}(x)}||\frac{\mathrm{P_{data}(x)} + \mathrm{P_G(x)}}{2}\right)$$

$$+\mathrm{KL}\left(\mathrm{P_G(x)}||\frac{\mathrm{P_{data}(x)} + \mathrm{P_G(x)}}{2}\right)$$

$$= -2log2 + 2JSD\big(P_{data}(x)||P_G(x)\big)$$ Jensen-Shannon divergence

In the end ……

$$V = \mathbb{E}_{x\sim P_{data}}[logD(x)] + \mathbb{E}_{x\sim P_G}[log(1-D(x))]$$

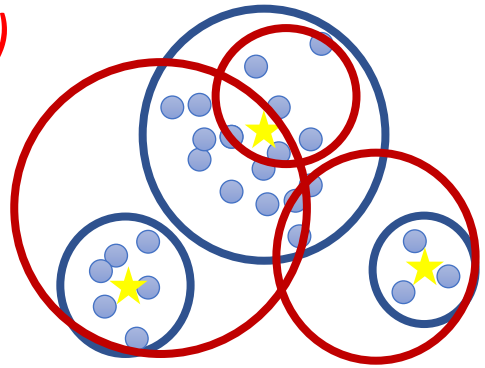
- Generator G, Discriminator D
- Looking for G* such that $G^* = arg\min_G \max_D V(G,D)$
- Given G, $\max_D V(G,D) = -2log2 + 2JSD(P_{data}(x)||P_G(x))$

  0 < … < log 2
- What is the optimal G?

  $P_G(x) = P_{data}(x)$

  with/using the JS(P_G, P_{data}) Divergence

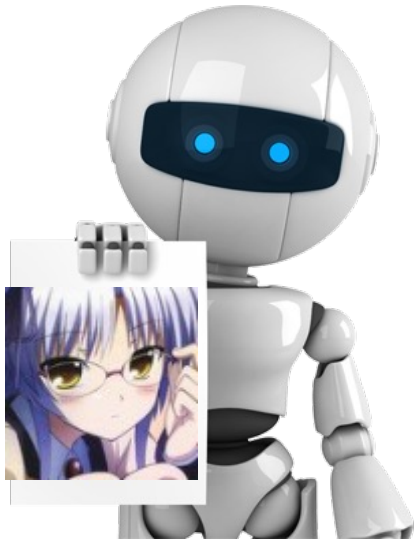
  (In Maximum Likelihood it is a KL Divergence)



Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. **GAN architectures**



Drawing? => learning from examples

Recall Algo GAN

for number of training iterations **do**

for k steps **do**

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

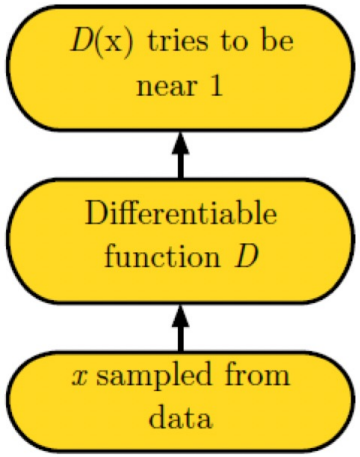
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

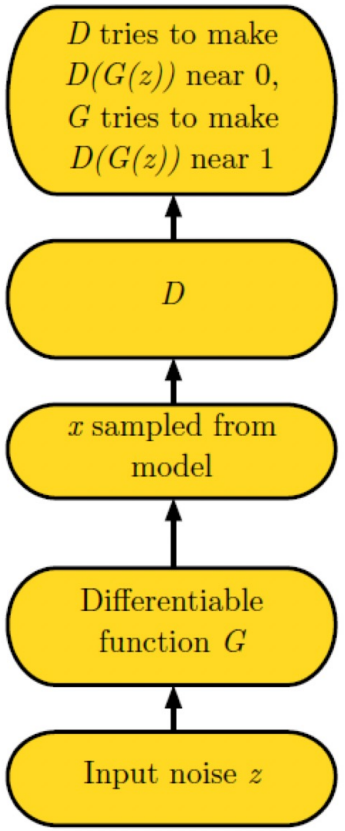
end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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





Functions G and D are NN

Question:

Which architectures for G and D?

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. **GAN architectures**
    1. **Basics**

Basic Archi for G and D and expe

Models

G and D fully connected nets
or convolutional for D, (de)convolutional for G (as seen for segmentation nets)
ReLU and/or sigmoids, dropout

Datasets

MNIST, Toronto Face Database, CIFAR-10

GAN - Evaluation : How to evaluate automatically generative models?

- Approximate p_g by fitting a Gaussian Parzen window on the generated images.
- Cross-validate σ to maximize likelihood of validation set
- Compute the likelihood of the test set

compare prob distr. of sampled images w. th a real dataset

FID

Evaluation not trivial, can be done using generated images as inputs for deep nets => inception scores

Frechet inception distance using 2 Gaussian (data,gen) over inception features: $\mathrm{FID} = ||\mu - \mu_w||_2^2 + \mathrm{tr}(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{1/2})$

GAN - Qualitative results 1/2

Figure: Right col nearest from dataset. a) MNIST, b) TFD, c) CIFAR-10 (fully connected), d) CIFAR-10 (convolutional D, deconvolutional G)

GAN - Qualitative results 2/2

Figure: Linear interpolation between 2 points in z space

- **Advantages:**
  - Computational advantages (no complex likelihood inference)
  - Can represent sharper distributions
- **Disadvantages:**
  - G and D must be well synchronized for the algorithm to converge correctly

GAN architectures

- How to improve result quality?
  - Spatial resolution

    $\Rightarrow$Cascade of GAN
  - Object quality

    => Progressive growing of spatial resolution in G

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. **LaPGAN**

Laplacian Pyramid GANs (LAPGANs)

- GANs do not work well for complex / high level / natural images.
- Idea: decompose the generation in successive tasks using Laplacian Pyramid (of GANs)

Let $d(I)$ and $u(I)$ be down-sampling and up-sampling operations.
Gaussian pyramid:

$$\mathcal{G}(I) = [I_0, I_1, ..., I_K], I_k = d^{(k)}(I)$$

Laplacian pyramid:

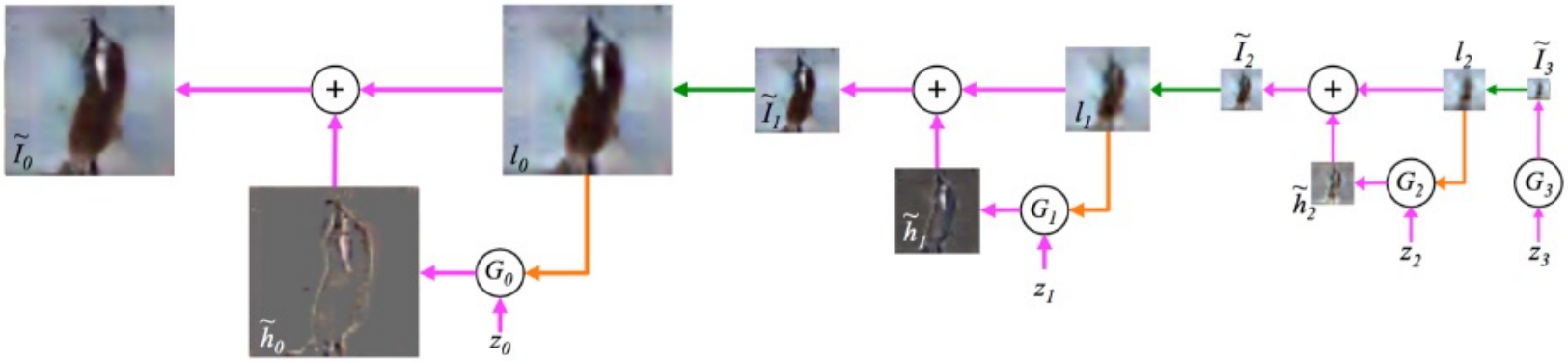
$$h_k = \mathcal{L}_k(I) = I_k - u(I_{k+1})$$

Reconstruction:

$$I_k = u(I_{k+1}) + h_k$$

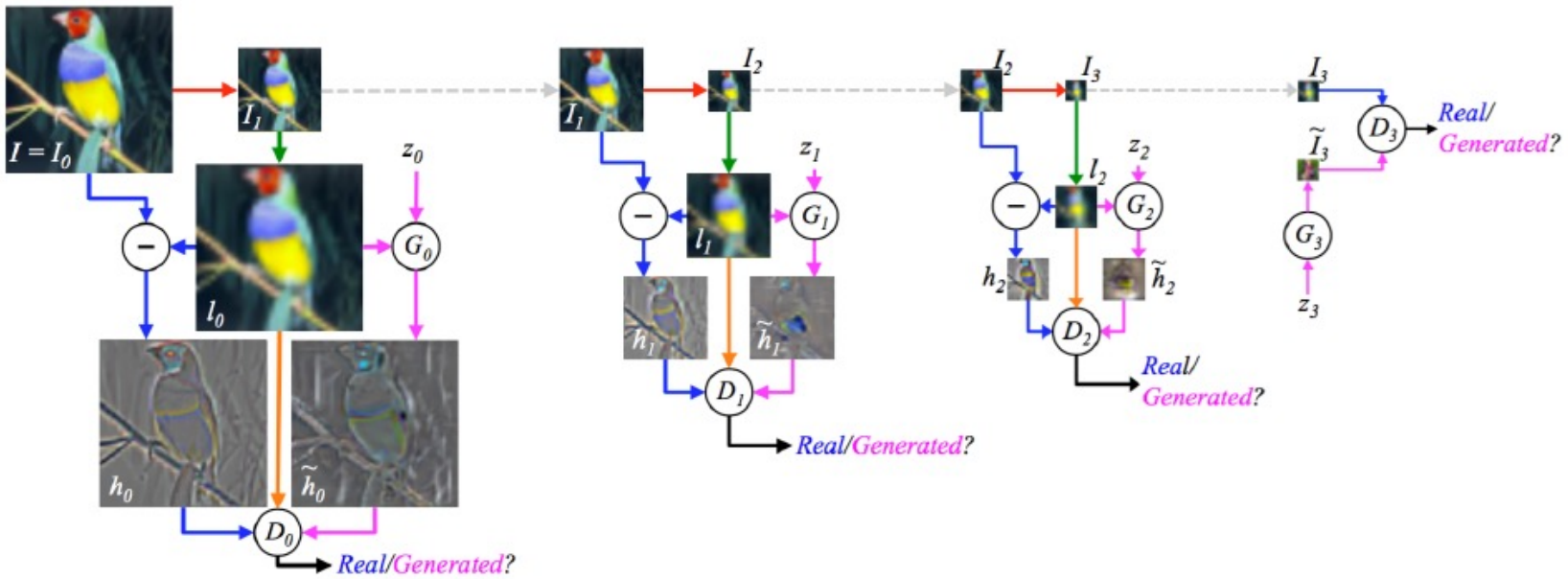
LAPGAN model - sampling

- Set of generative convnets: $G_0, \ldots, G_K$
- Generated details: $\tilde{h}_k = G_k(z_k, u(\tilde{I}_{k+1}))$
- Reconstructed image: $\tilde{I}_k = u(\tilde{I}_{k+1}) + \tilde{h}_k \ (\tilde{I}_{K+1} = 0)$



LAPGAN model - training

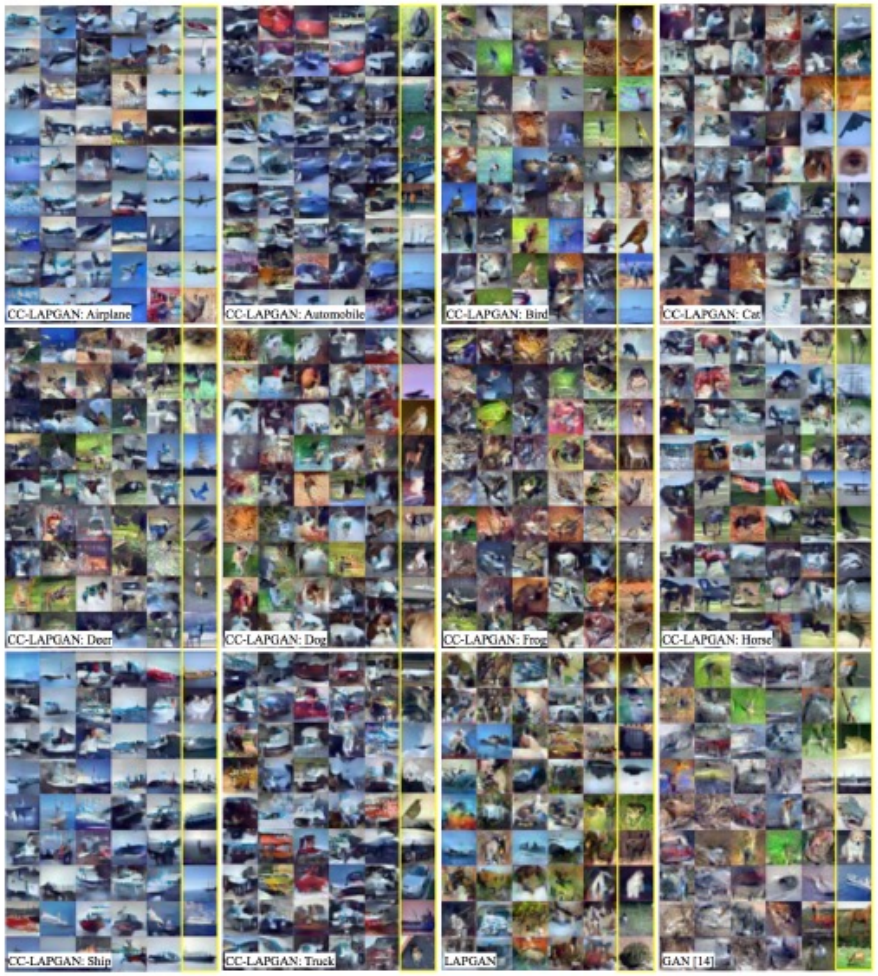
- Low-pass version of l_0: $l_0 = u(d(l_0))$
- Either:
  - High-pass version of l_0: $h_0 = l_0 - l_0$
  - Generate $\tilde{h}_0 = G_0(z_0, l_0)$
- Forward $D_0(l_0 + h_0$ or $\tilde{h}_0)$
- Backward D_0 and G_0
- G_K and D_K are trained as a simple GAN



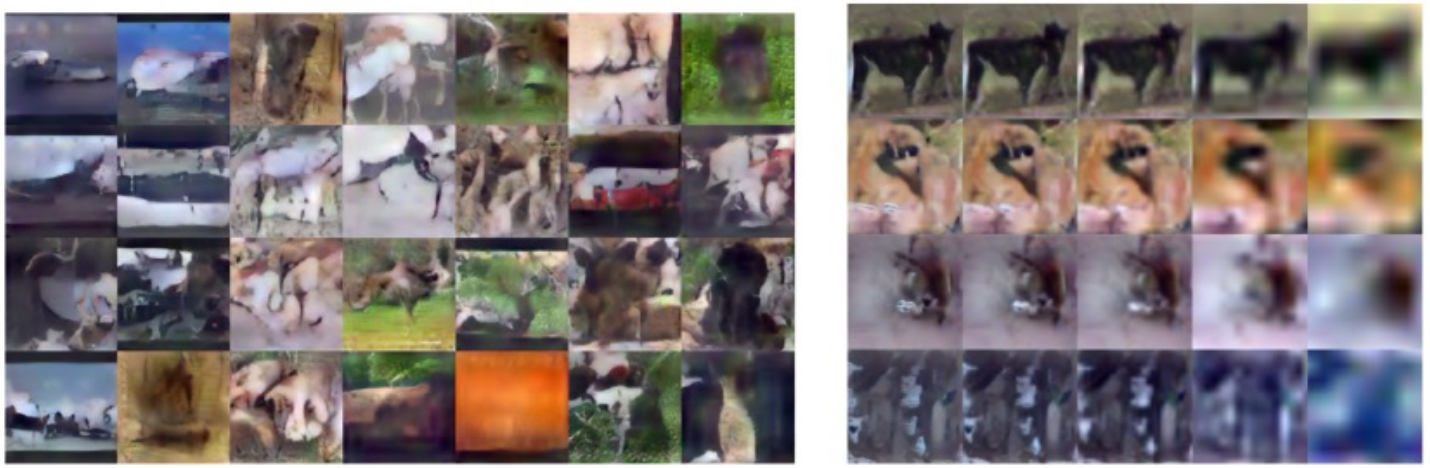
LAPGAN model - Experiments

- **Datasets:** CIFAR-10, STL
- **Initial scale:**
  - G_K and D_K have 2 hidden layers and ReLU
  - $z_K \sim U_{[-1,1]^{100}}$
  - Trained as GAN
- **Subsequent scales:**
  - G_k and D_k convnets with 3 and 2 layers
  - $z_k \sim U_{[-1,1]^{|I_k|}}$ ("color" layer)
  - Trained as CGAN

LAPGAN model - Results - CIFAR



LAPGAN model - Results - STL



(a) (b)

Figure 4: STL samples: **(a)** Random 96x96 samples from our LAPGAN model. **(b)** Coarse-to-fine generation chain.

LAPGAN model - Results - LSUN

LAPGAN

- Good idea (cascade) but Generator structure too weak

=> Other approach: progressive growing of spatial resolution

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. **DCGAN**

Progressive growing of spatial resolution in G

Deep Convolutional GANs (DCGANs)

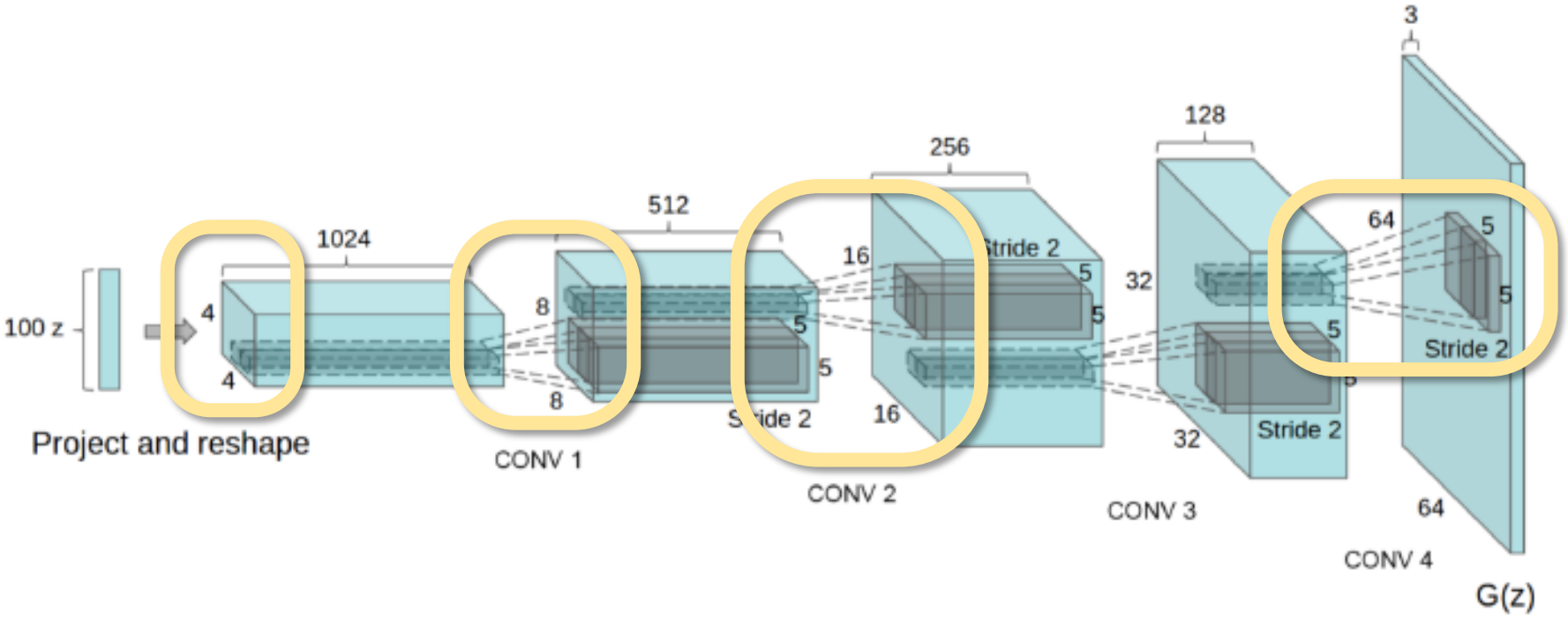
GANs are hard to scale => Identify a family of architectures that gives stable training

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator) =transposed convolutions =deconvolutions
- Use batchnorm in both the generator and the discriminator
- Remove fully connected hidden layers for deeper architectures
- Use ReLU activation in generator for all layers except for the output, which uses Tanh
- Use LeakyReLU activation in the discriminator for all layers

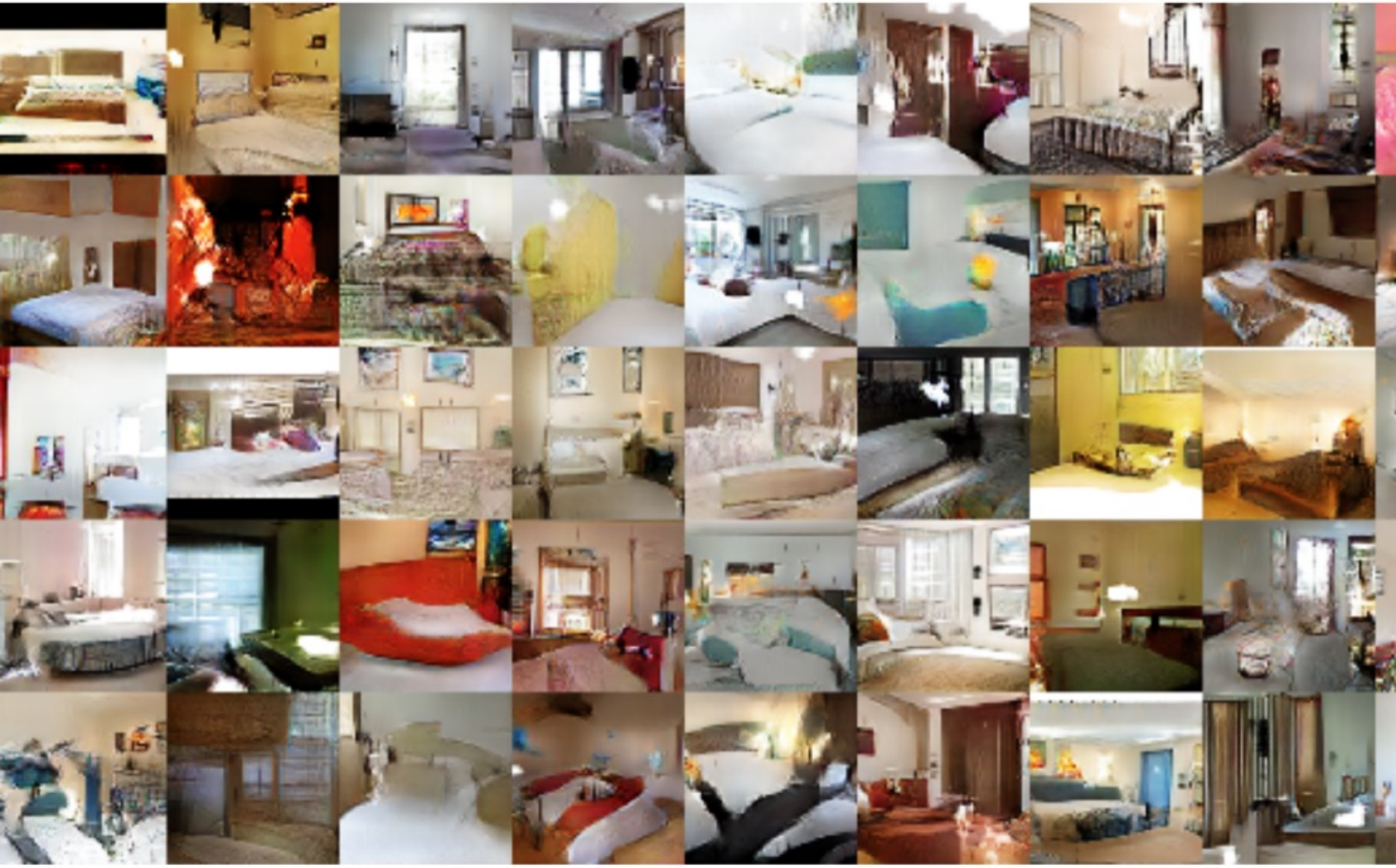
Progressive growing of spatial resolution in G: DCGAN

Upsampling step by step

Combine with convolutional layers



DCGAN - Results - generated bedrooms



DCGAN results - Faces

Generative models

Outline

1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. DCGAN
    4. **ProGAN**

Progressive growing of GANs

[Progressive Growing of GANs for Improved Quality, Stability and Variation, Tero Karras et al. (NVIDIA); ICLR 2018]

Combine idea of LAPGAN (several output reso) and DCGAN (archi prog growing)

1. First, start with training 4x4 output images.
2. When this training has converged, add a new block to generate 8x8 output images.
3. Etc.

The transition to adding a new block is gradual, we first start with more weight on the (upsampled) output of the previous block, and then add more and more weights to the output of the current block.

All weights remain trainable during the whole process.

Discriminator = mirror image of generator

Progressive growing of GANs

[Progressive Growing of GANs for Improved Quality, Stability and Variation, Tero Karras et al. (NVIDIA); ICLR 2018]

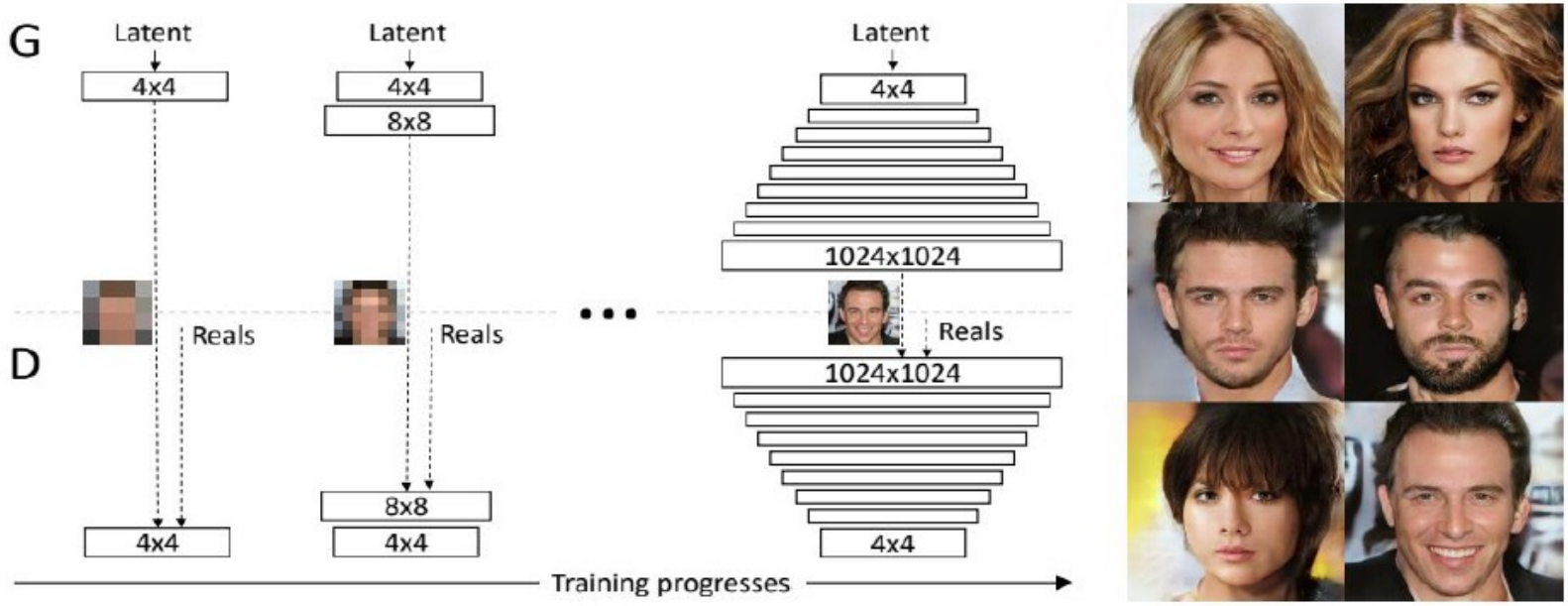


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Progressive growing of GANs

[Progressive Growing of GANs for Improved Quality, Stability and Variation, Tero Karras et al. (NVIDIA); ICLR 2018]

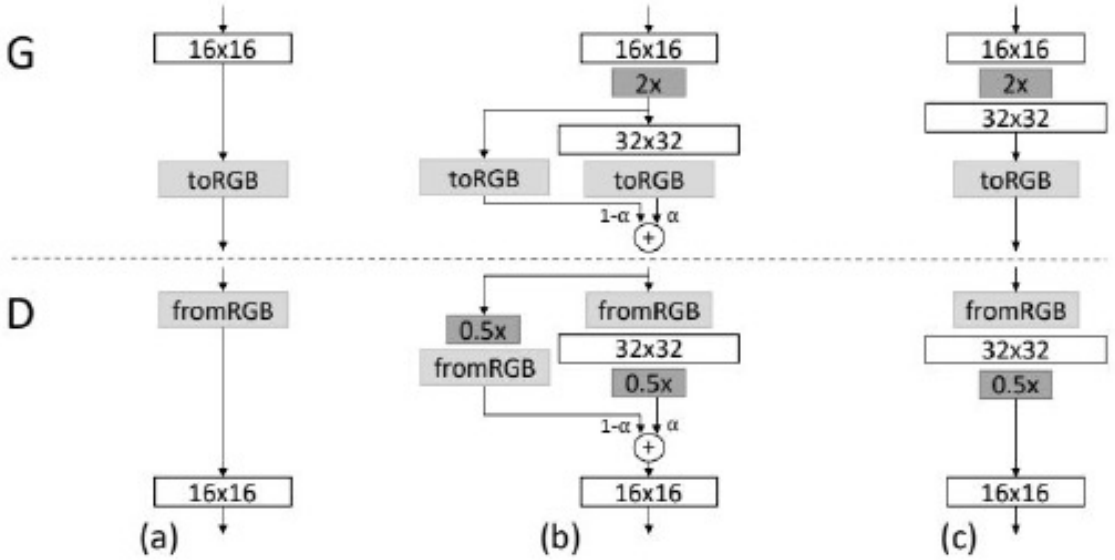


Figure 2: When doubling the resolution of the generator (G) and discriminator (D) we fade in the new layers smoothly. This example illustrates the transition from 16×16 images (a) to 32×32 images (c). During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight α increases linearly from 0 to 1. Here $2\times$ and $0.5\times$ refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The toRGB represents a layer that projects feature vectors to RGB colors and fromRGB does the reverse; both use 1×1 convolutions. When training the discriminator, we feed in real images that are downscaled to match the current resolution of the network. During a resolution transition, we interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.

Generative models

Outline

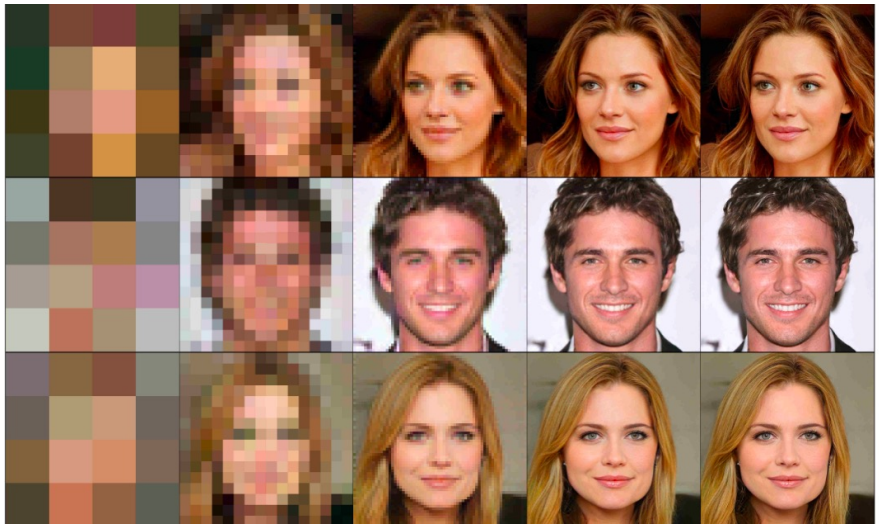
1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. DCGAN
    4. ProGAN
    5. **MSG-GAN**

MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks [CVPR 2020]

Main Idea:

- ProGAN both use progressive growing, but although this gives stability, it introduces many complicated training parameters associated with each new network.
- Training cannot be done “out of the box”, have to adjust parameters for each new dataset.

→ Train all at once without complicated adding on layers



MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks [CVPR 2020]

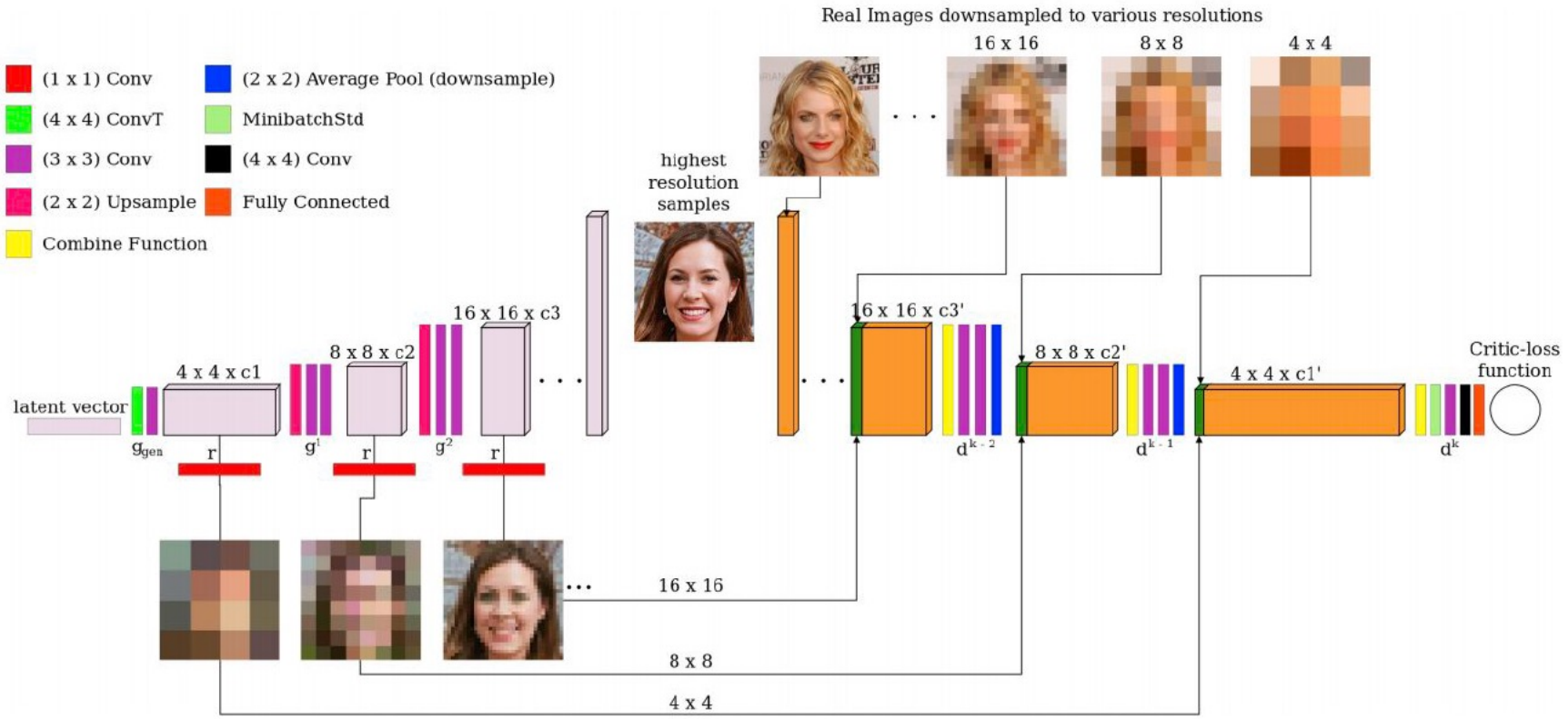


Figure 2: Architecture of MSG-GAN, shown here on the base model proposed in ProGANs [13]. Our architecture includes connections from the intermediate layers of the generator to the intermediate layers of the discriminator. Multi-scale images sent to the discriminator are concatenated with the corresponding activation volumes obtained from the main path of convolutional layers followed by a combine function (shown in yellow).

MSG-GAN: results – Random generated CelebA-HQ Faces at resolution 1024x1024

Generative models

Outline

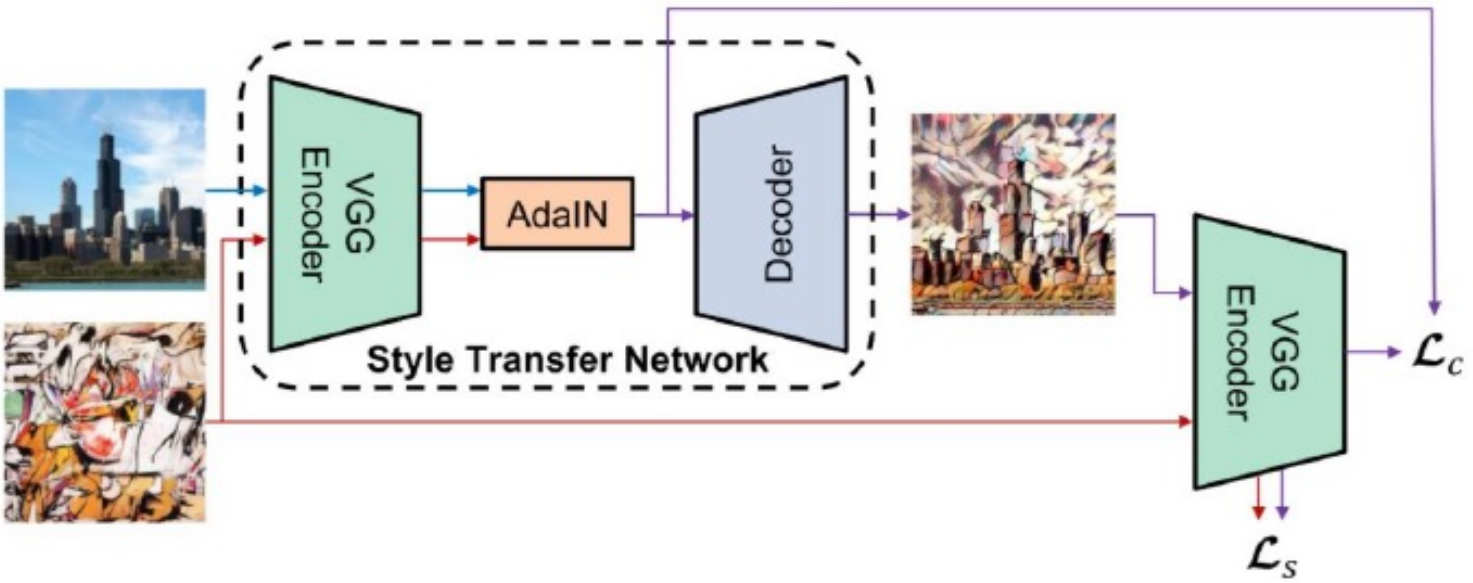
1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
    1. Basics
    2. LaPGAN
    3. DCGAN
    4. ProGAN
    5. MSG-GAN
    6. **StyleGAN**

StyleGAN: A Style-Based Generator Architecture for Generative Adversarial Networks [Karras CVPR 2019]

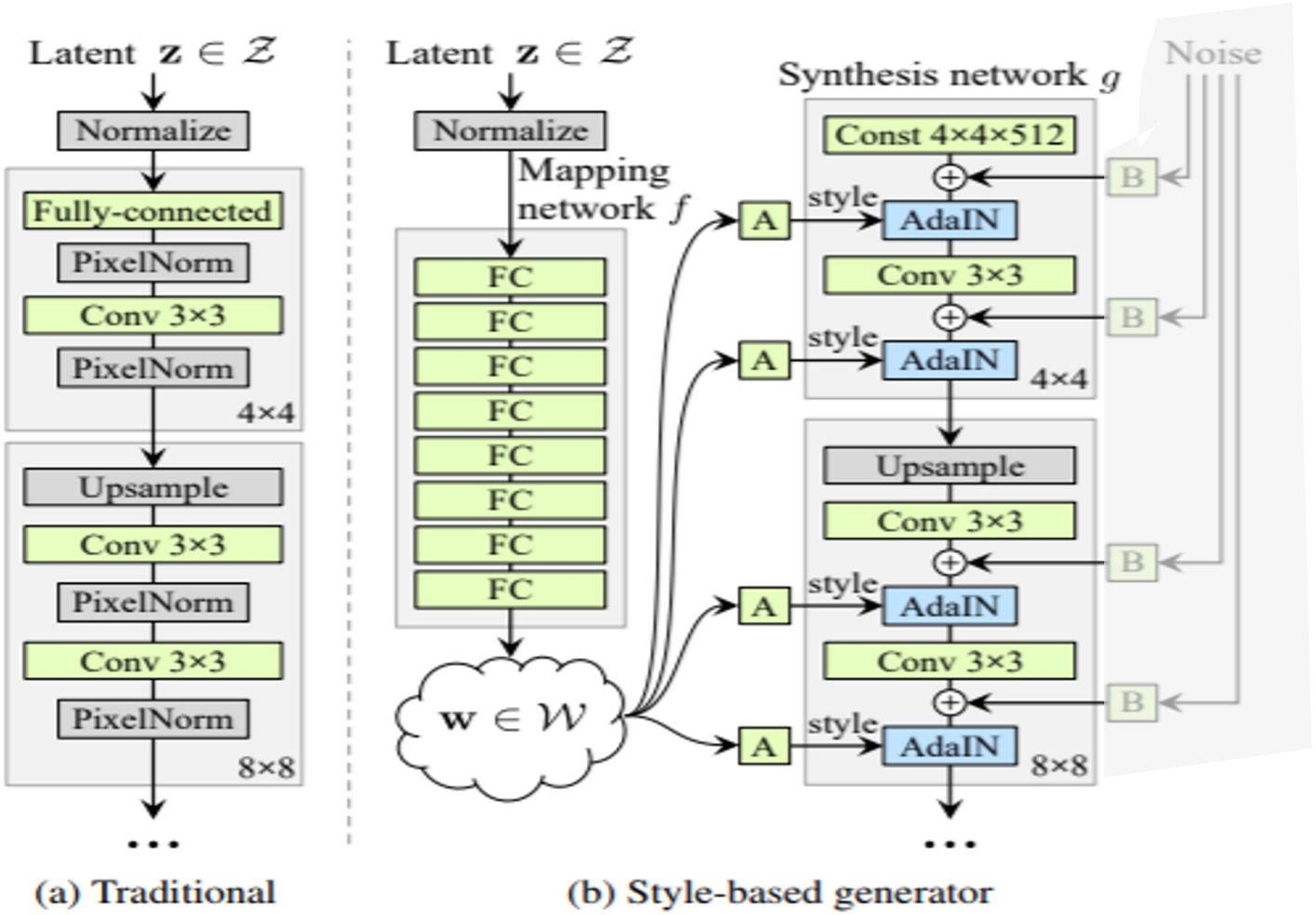
Still progressive growing architecture but with new refinement block based on: Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization (AdaIN)

style

$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$



StyleGAN Network Architecture



Building up the Model

	Method	CelebA-HQ	FFHQ
A	Baseline Progressive GAN [29]	7.79	8.04
B	+ Tuning (incl. bilinear up/down)	6.11	5.25
C	+ Add mapping and styles	5.34	4.85
D	+ Remove traditional input	5.07	4.88
E	+ Add noise inputs	**5.06**	4.42
F	+ Mixing regularization	5.17	**4.40**

Generative models

Outline

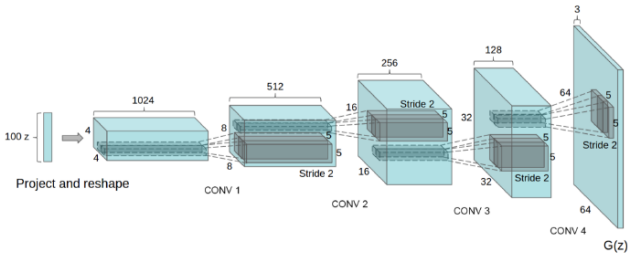
1. Preview: Auto-Encoders, VAE
2. Generative models with GAN
3. GAN architectures
4. **Editing**

GAN editing

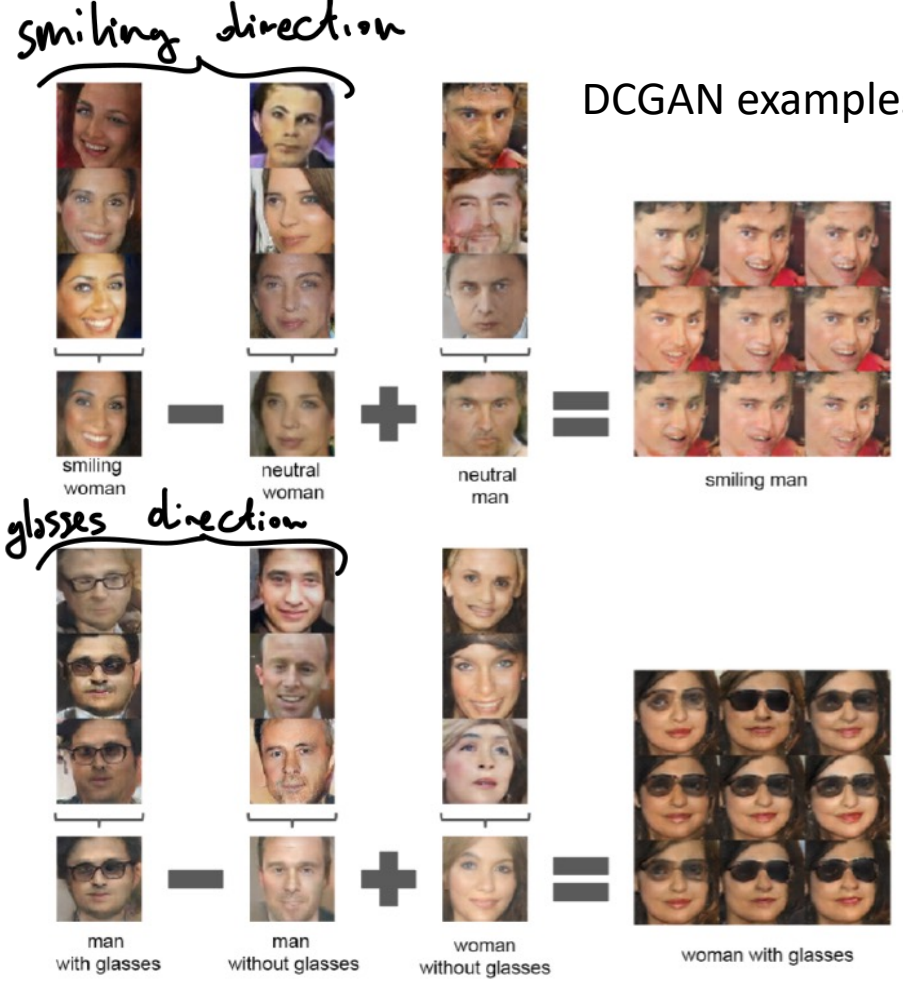
Editing by manipulation in the latent space of z

Image I = G(z)

Editing: changing z to z' and the new image is I' = G(z')



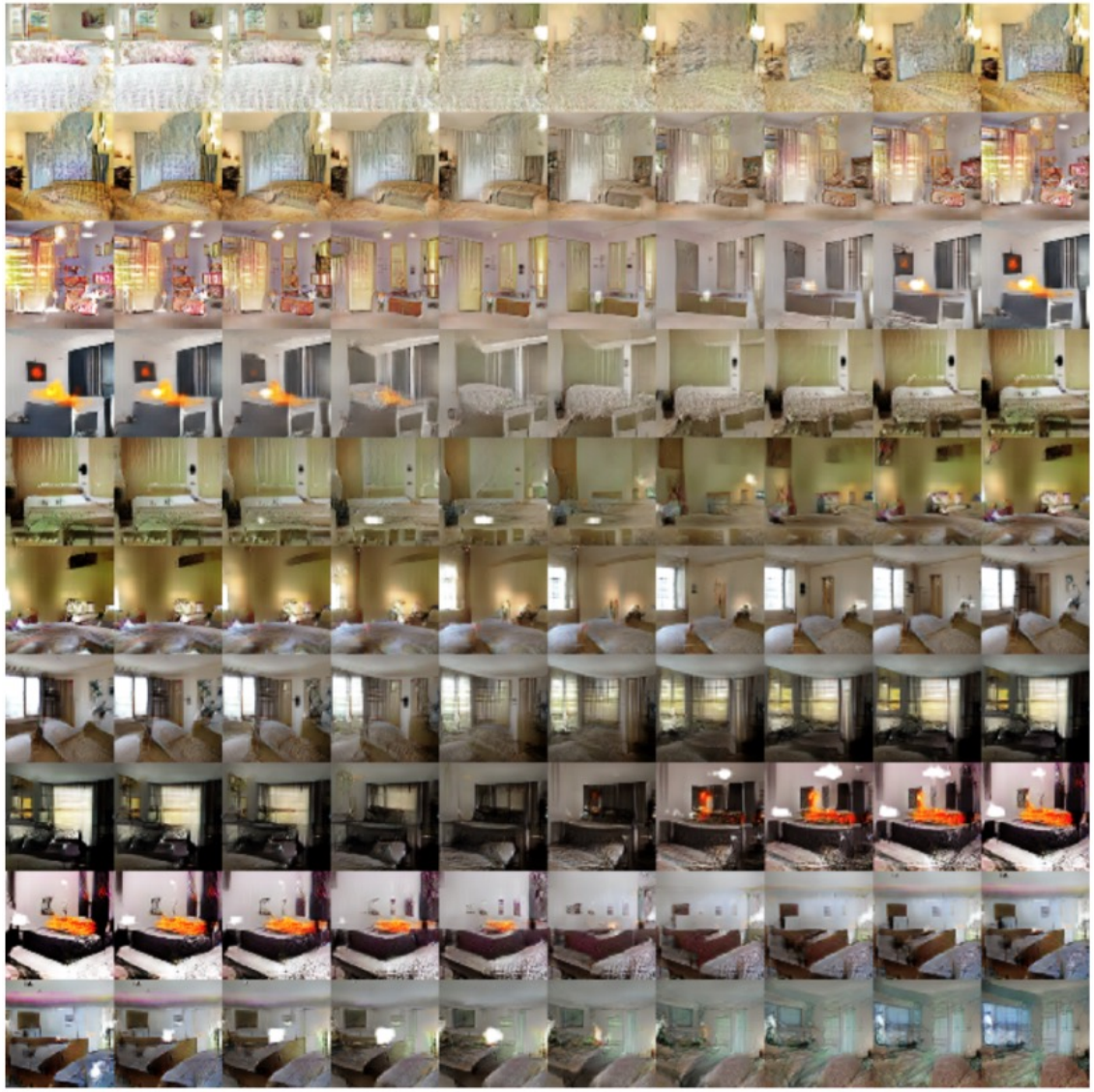
Artithmetics in latent space: vector mean, addition, substraction



DCGAN examples

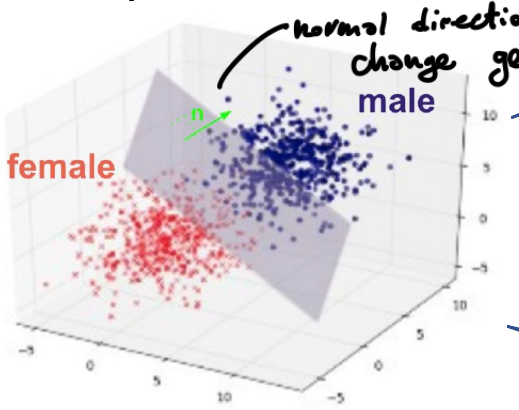
GAN editing

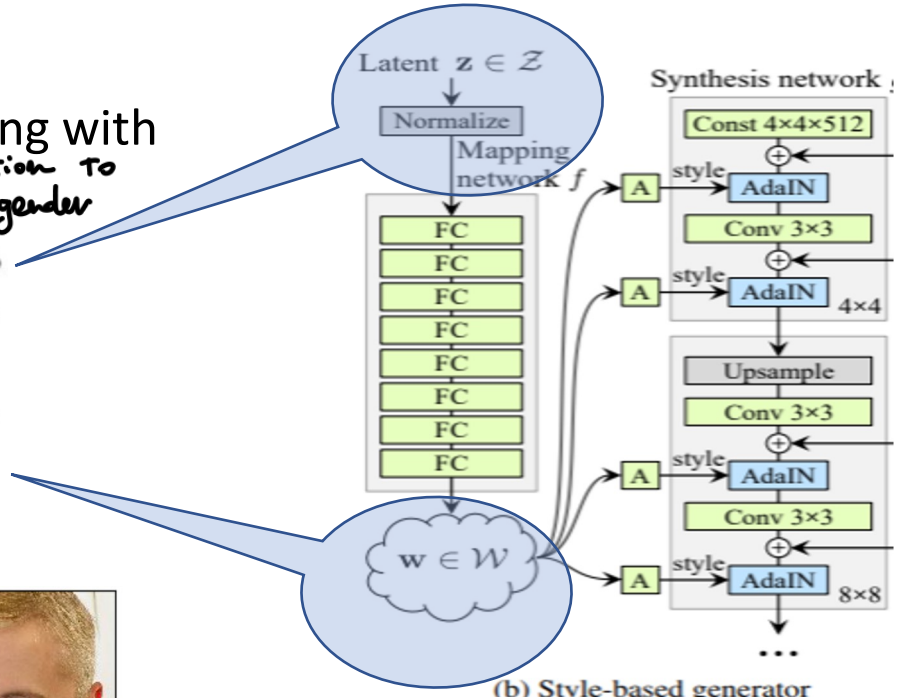
Linear interpolation
in latent space



Gan Editing

Latent space analysis for GAN editing with StyleGAN

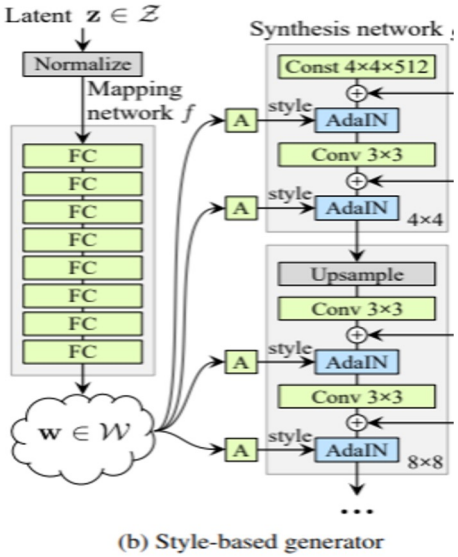




(b) Style-based generator

GAN editing

Latent space analysis for GAN editing with StyleGAN



(b) Style-based generator

Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions (4×4– 8×8) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions (16x16– 32x32) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved.