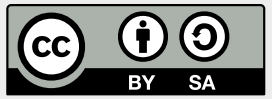
Deep Generative Models

Milan Straka

May 4, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



Generative Models

Generative models are given a set $\mathcal{X}$ of realizations of a random variable $\mathbf{x}$ and their goal is to estimate $P(\boldsymbol{x})$.

Usually the goal is to be able to sample from $P(\mathbf{x})$, but sometimes an explicit calculation of $P(\boldsymbol{x})$ is also possible.

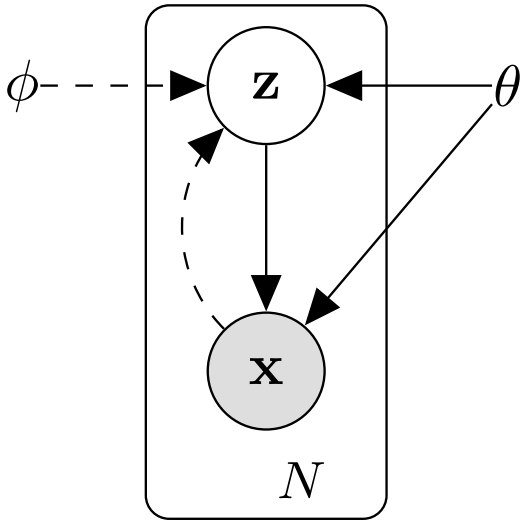
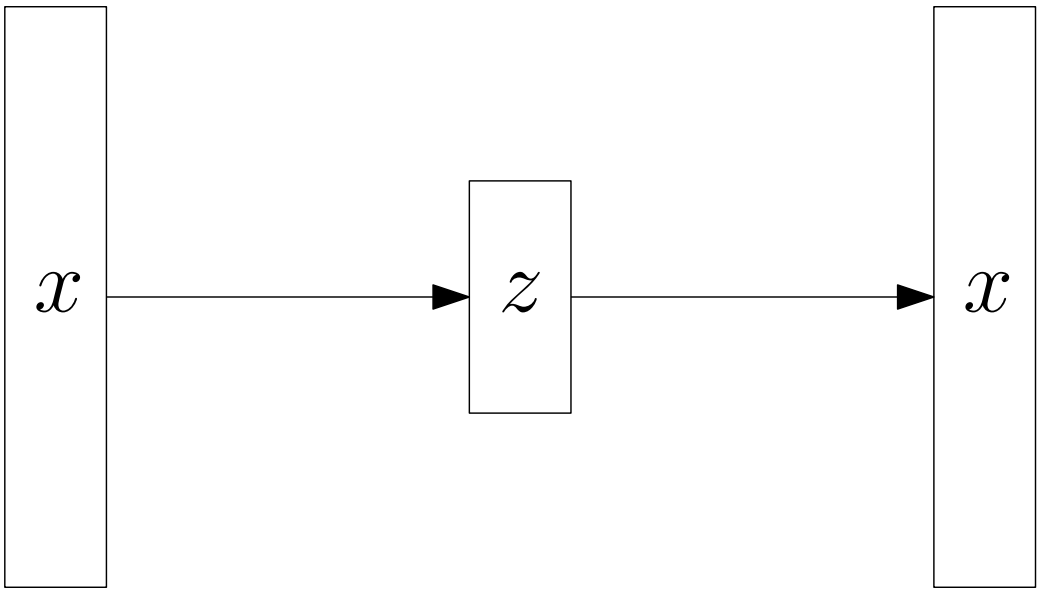


Figure 1 of paper "Auto-Encoding Variational Bayes", https://arxiv.org/abs/1312.6114.

One possible approach to estimate $P(\boldsymbol{x})$ is to assume that the random variable $\mathbf{x}$ depends on a *latent variable* $\mathbf{z}$:

$$P(\boldsymbol{x}) = \sum_{\boldsymbol{z}} P(\boldsymbol{z})P(\boldsymbol{x}|\boldsymbol{z}) = \mathbb{E}_{\boldsymbol{z}\sim P(\mathbf{z})} P(\boldsymbol{x}|\boldsymbol{z}).$$

We use neural networks to estimate the conditional probability with $P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$.



- Autoencoders are useful for unsupervised feature extraction, especially when performing input compression (i.e., when the dimensionality of the latent space $\boldsymbol{z}$ is smaller than the dimensionality of the input).
- When $\boldsymbol{x} + \boldsymbol{\varepsilon}$ is used as input, autoencoders can perform denoising.
- However, the latent space $\boldsymbol{z}$ does not need to be fully covered, so a randomly chosen $\boldsymbol{z}$ does not need to produce a valid $\boldsymbol{x}$.

Variational AutoEncoders

We assume $P(\mathbf{z})$ is fixed and independent on $\mathbf{x}$.

We approximate $P(\boldsymbol{x}|\boldsymbol{z})$ using $P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. However, in order to train an autoencoder, we need to know the posterior $P_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, which is usually intractable.

We therefore approximate $P_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ by a trainable $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$.

Variational AutoEncoders

Let us define *variational lower bound* or *evidence lower bound* (ELBO), denoted $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x})$, as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x}) = \log P_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{\mathrm{KL}}\big(Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) || P_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big).$$

Because KL-divergence is non-negative, $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x}) \leq \log P_{\boldsymbol{\theta}}(\boldsymbol{x})$.

By using simple properties of conditional and joint probability, we get that

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x}) &= \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}) + \log P_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) - \log Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})] \\
&= \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})] \\
&= \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log P(\boldsymbol{z}) - \log Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})] \\
&= \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - D_{\mathrm{KL}}(Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) || P(\boldsymbol{z})).
\end{aligned}$$

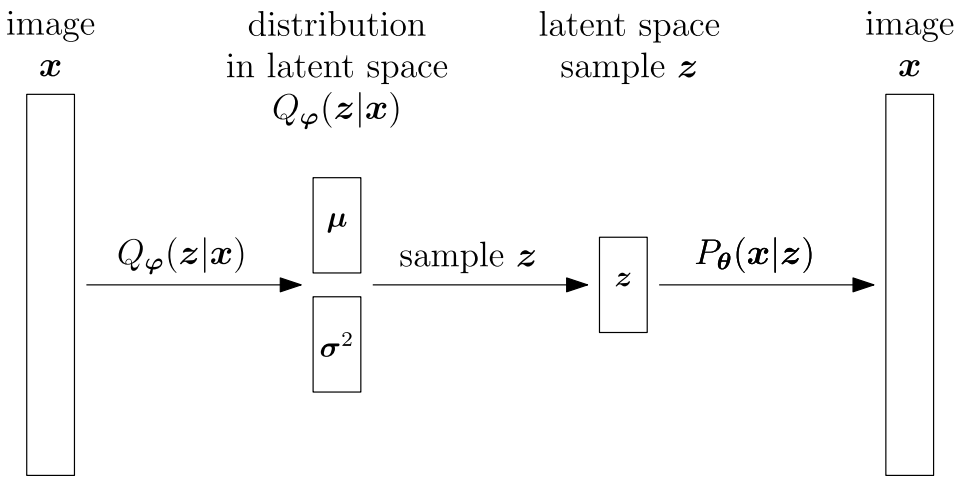
Variational AutoEncoders

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x}) = \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - D_{\mathrm{KL}}(Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})||P(\boldsymbol{z}))$$

- We train a VAE by maximizing $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x})$.
- The distribution $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ is parametrized as a normal distribution $\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, with the model predicting $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}^2$ given $\boldsymbol{x}$.
  - The normal distribution is used, because we can sample from it efficiently, we can backpropagate through it and we can compute D_{KL} analytically; furthermore, if we decide to parametrize $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ using mean and variance, the maximum entropy principle suggests we should use the normal distribution.
- The $\mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}$ is estimated using a single sample.
- We use a prior $P(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{1})$.

Variational AutoEncoders

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x}) = \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - D_{\mathrm{KL}}(Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})||P(\boldsymbol{z}))$$



Note that the loss has 2 intuitive components:

- **reconstruction loss** – starting with $\boldsymbol{x}$, passing though $Q_{\boldsymbol{\varphi}}$, sampling $\boldsymbol{z}$ and then passing through $P_{\boldsymbol{\theta}}$ should arrive back at $\boldsymbol{x}$;
- **latent loss** – over all $\boldsymbol{x}$, the distribution of $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ should be as close as possible to the prior $P(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{1})$, which is independent on $\boldsymbol{x}$.

VAE – Reparametrization Trick

In order to backpropagate through $\boldsymbol{z} \sim Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$, note that if

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2),$$

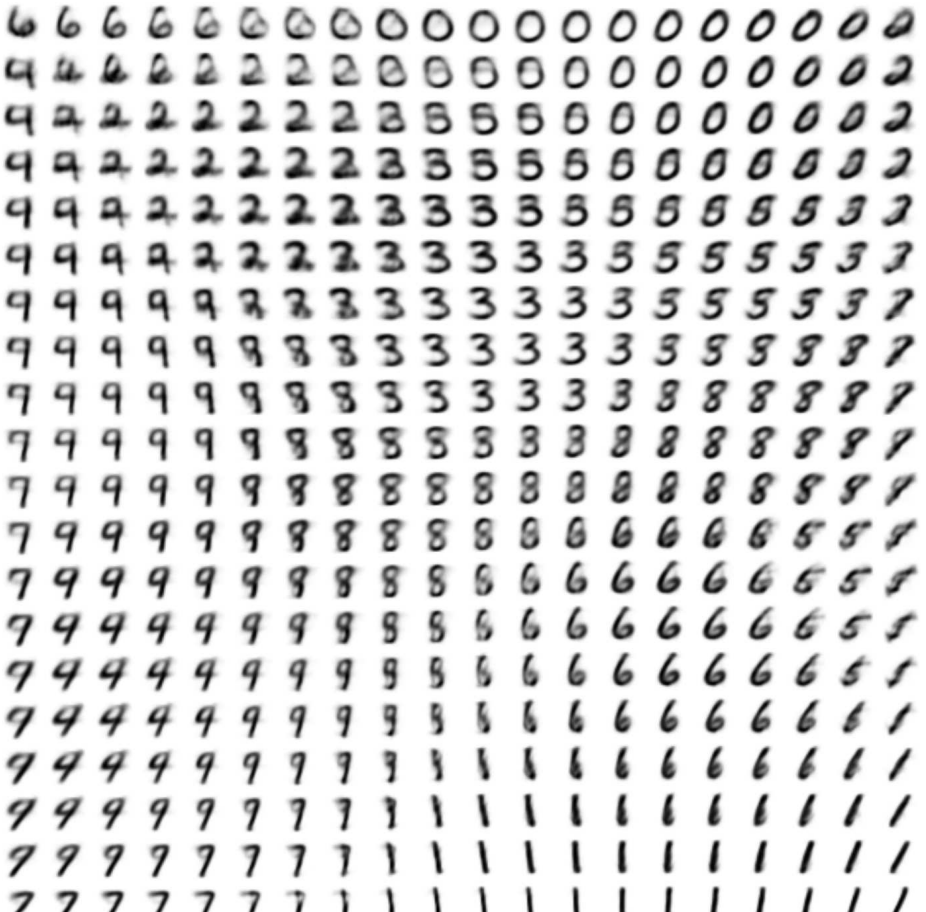
we can write $\boldsymbol{z}$ as

$$\boldsymbol{z} \sim \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \mathcal{N}(\mathbf{0}, \mathbf{1}).$$

Such formulation then allows differentiating $\boldsymbol{z}$ with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and is called a *reparametrization trick* (Kingma and Welling, 2013).

Variational AutoEncoders

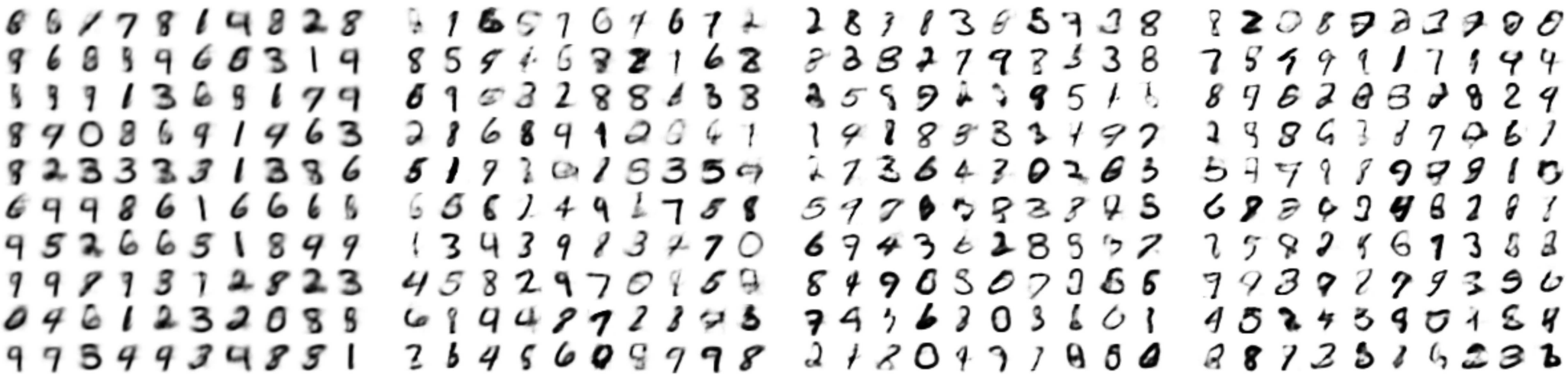
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

Figure 4 of paper "Auto-Encoding Variational Bayes", https://arxiv.org/abs/1312.6114.

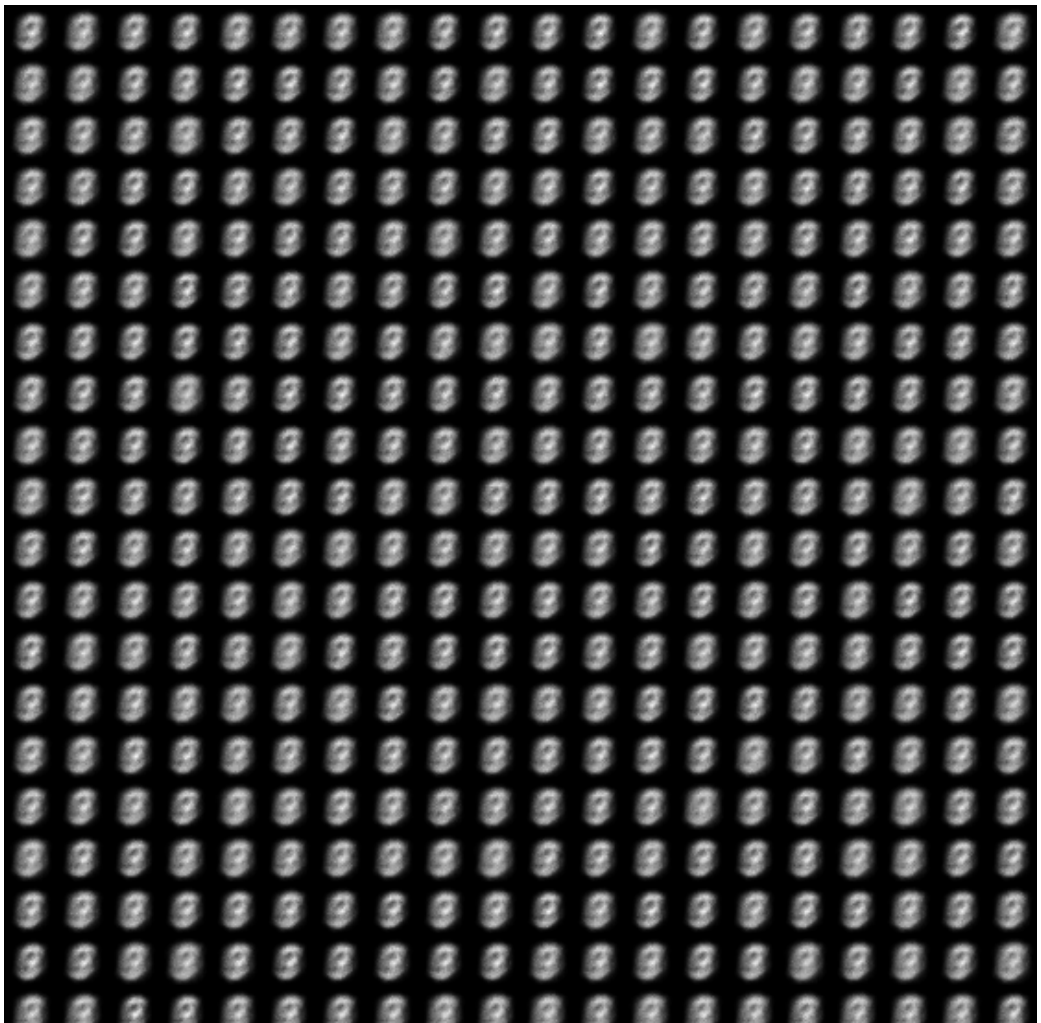
Variational AutoEncoders



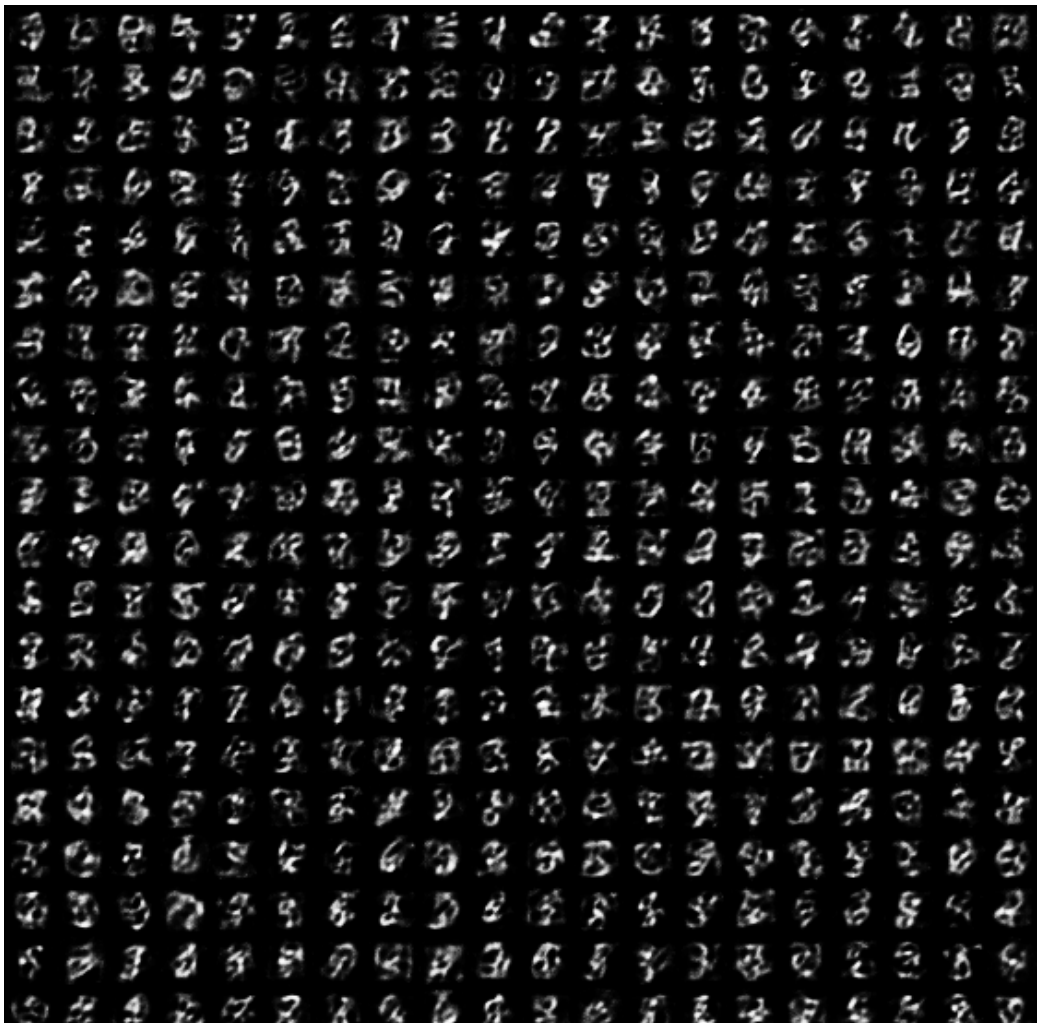
(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 5 of paper "Auto-Encoding Variational Bayes", https://arxiv.org/abs/1312.6114.

VAE – Too High Latent Loss



VAE – Too High Reconstruction Loss



Generative Adversarial Networks

We have a *generator*, which given $\boldsymbol{z} \sim P(\mathbf{z})$ generates data $\boldsymbol{x}$.

We denote the generator as $G(\boldsymbol{z}; \boldsymbol{\theta}_g)$.

Then we have a *discriminator*, which given data $\boldsymbol{x}$ generates a probability whether $\boldsymbol{x}$ comes from real data or is generated by a generator.

We denote the discriminator as $D(\boldsymbol{x}; \boldsymbol{\theta}_d)$.

The discriminator and generator play the following game:

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x} \sim P_{\text{data}}} [\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim P(\mathbf{z})} [\log(1 - D(G(\boldsymbol{z})))].$$

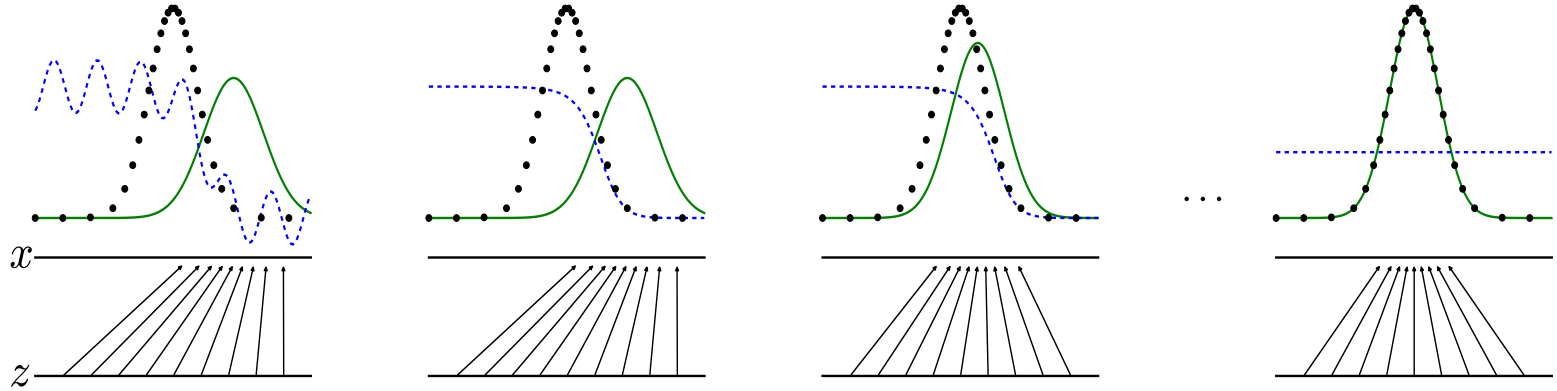


Figure 1 of paper "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661.

The generator and discriminator are alternately trained, the discriminator by

$$\arg\max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P(\mathbf{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

and the generator by

$$\arg\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{z}\sim P(\mathbf{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

In a sense, the discriminator acts as a trainable loss for the generator.

Because $\log(1 - D(G(\boldsymbol{z})))$ can saturate in the beginning of the training, where the discriminator can easily distinguish real and generated samples, the generator can be trained by

$$\arg\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{z} \sim P(\mathbf{z})}[-\log D(G(\boldsymbol{z}))]$$

instead, which results in the same fixed-point dynamics, but much stronger gradients early in learning.

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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

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

Algorithm 1 of paper "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661.

Generative Adversarial Networks

Figure 2 of paper "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661.

Conditional GAN

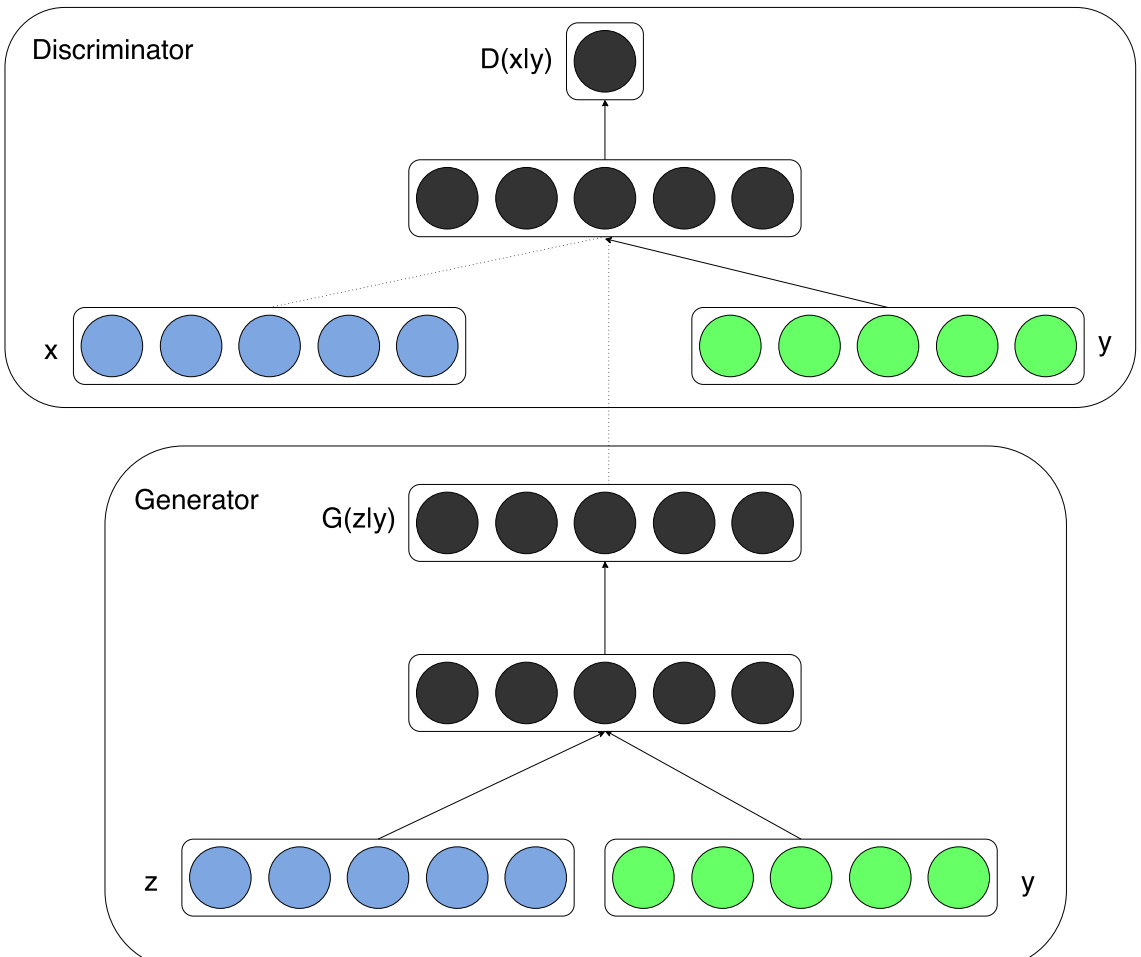
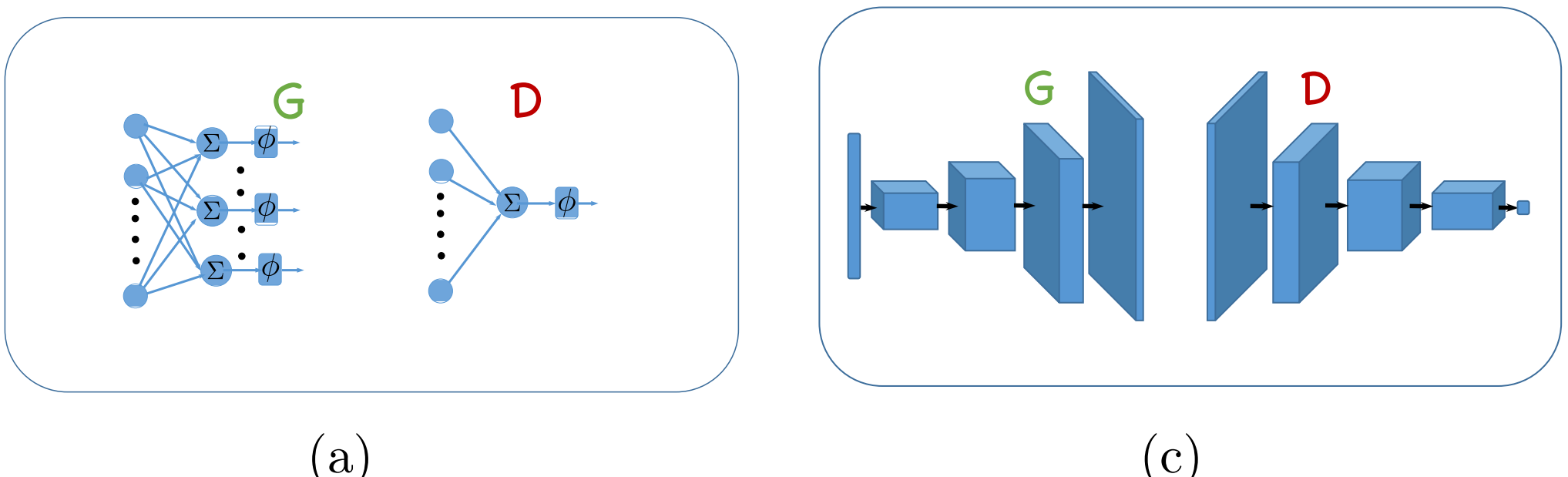


Figure 1 of paper "Conditional Generative Adversarial Nets", https://arxiv.org/abs/1411.1784.

Deep Convolutional GAN

In Deep Convolutional GAN, the discriminator is a convolutional network (with batch normalization) and the generator is also a convolutional network, utilizing transposed convolutions.



(a) (c)

Figure 1 of paper "An Online Learning Approach to Generative Adversarial Networks", https://arxiv.org/abs/1706.03269.

Deep Convolutional GAN

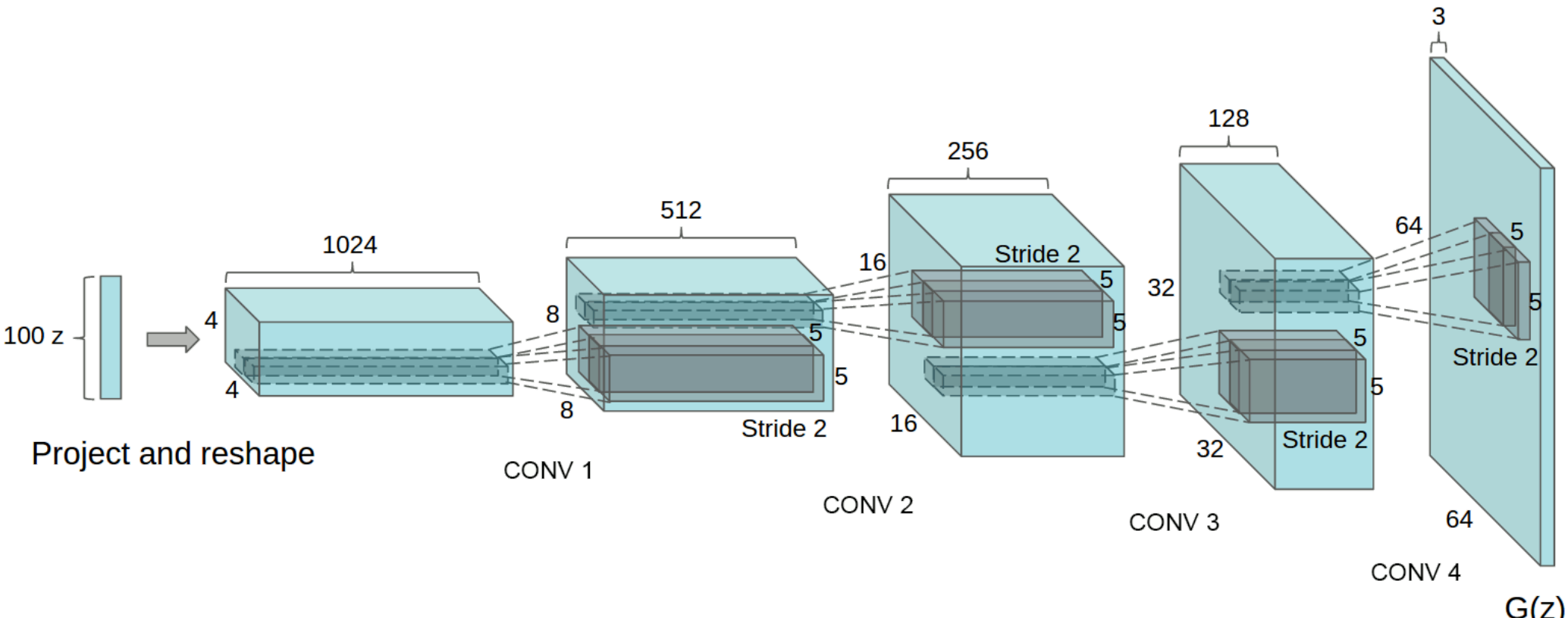


Figure 1 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

Figure 2 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

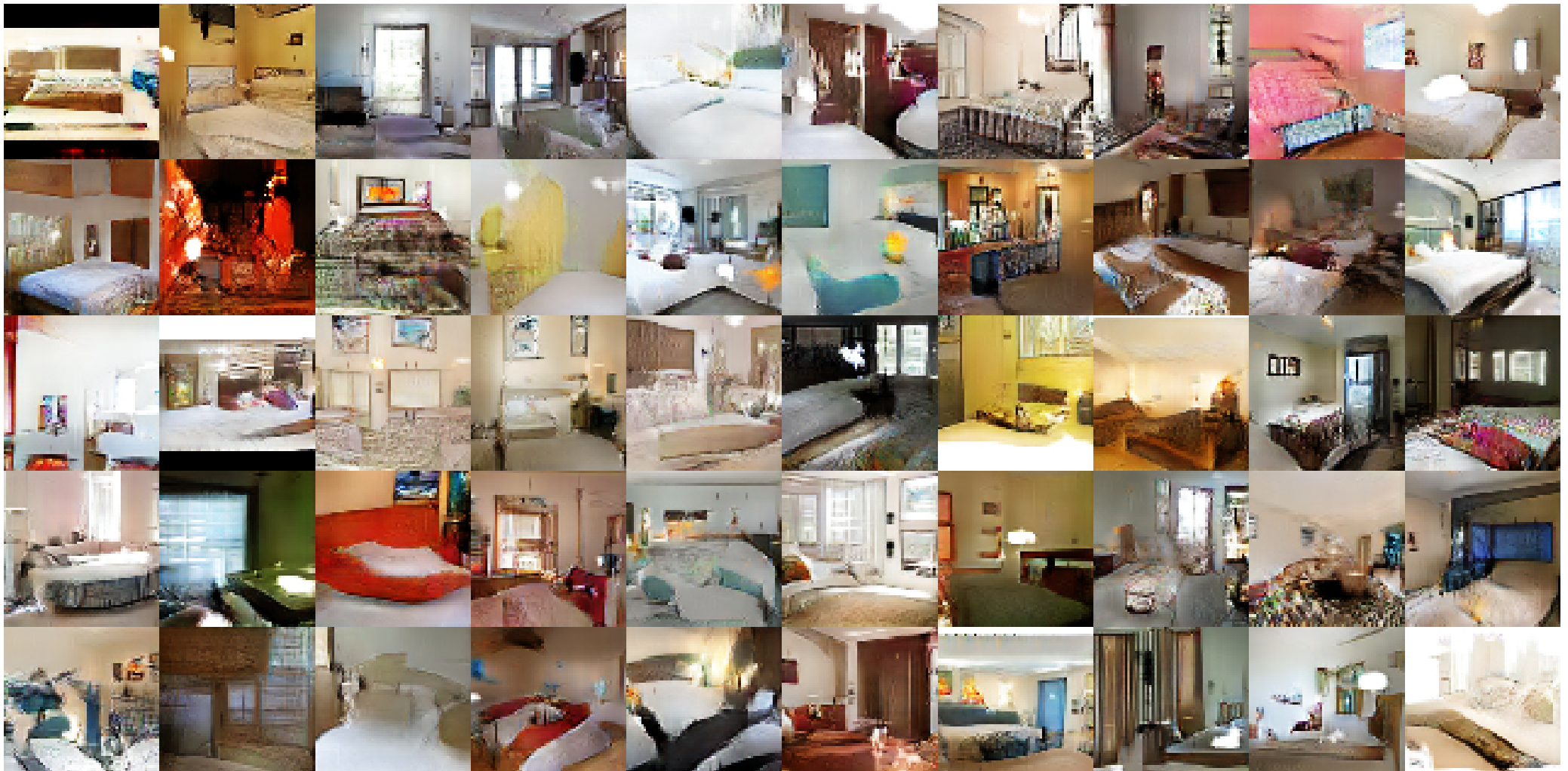


Figure 3 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

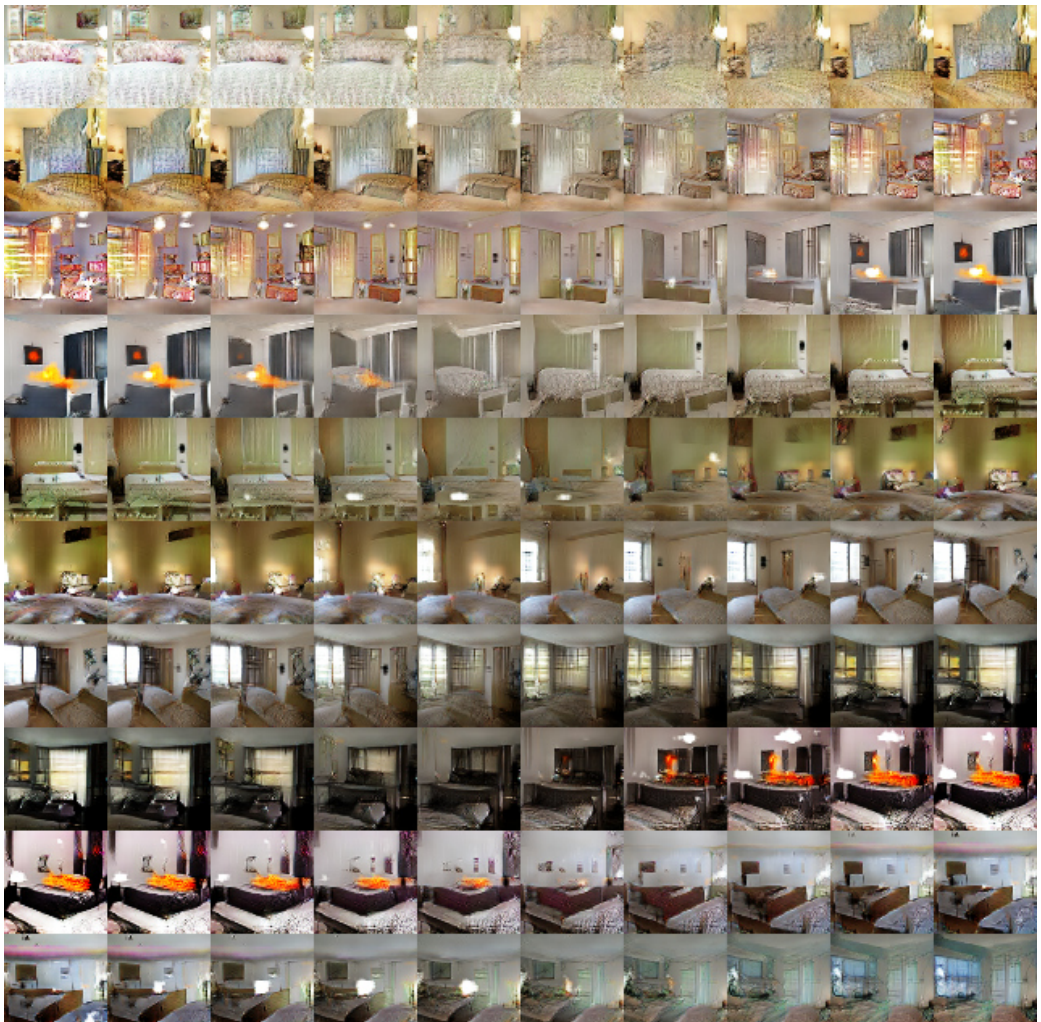


Figure 4 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

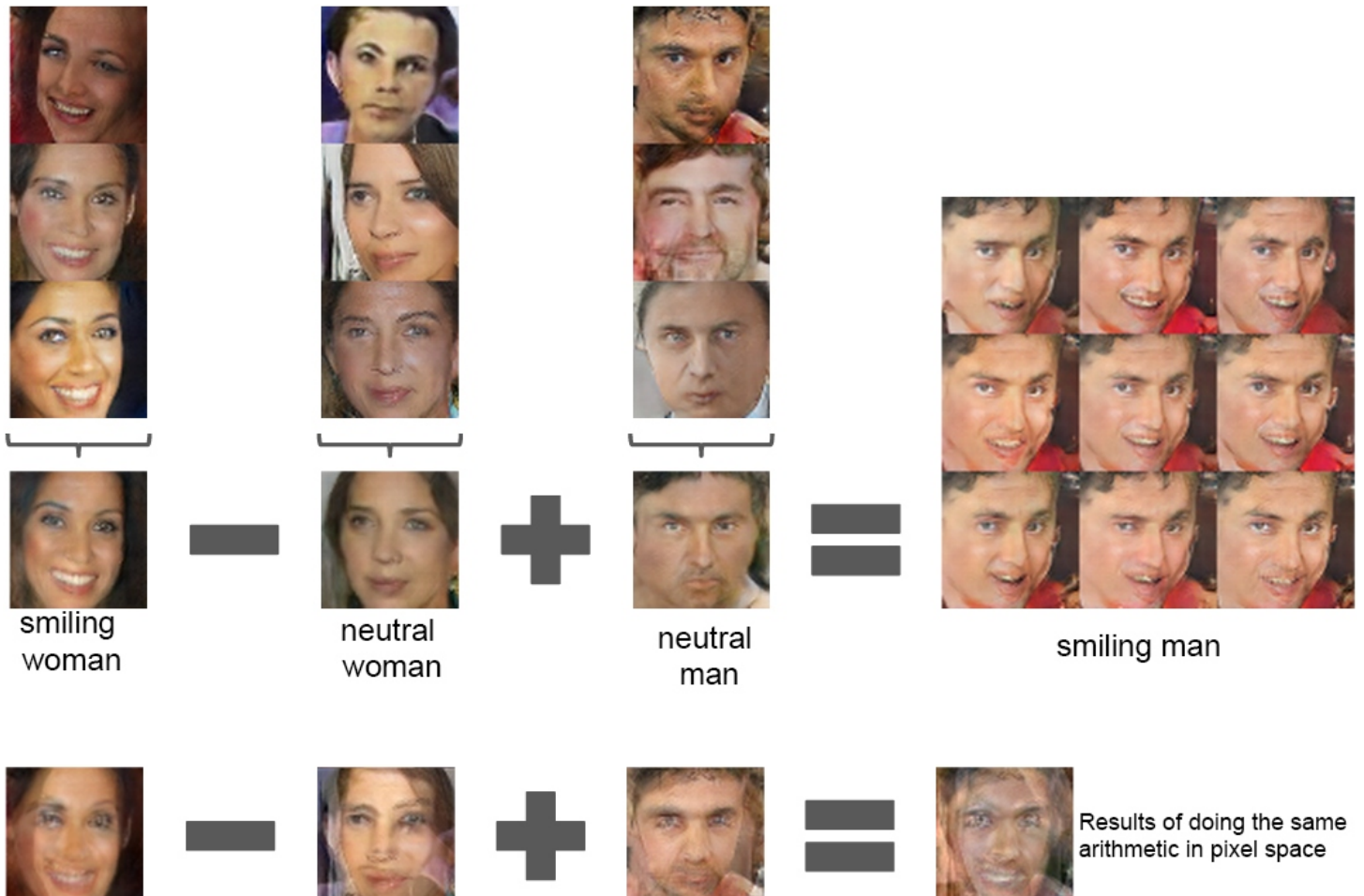


Figure 7 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

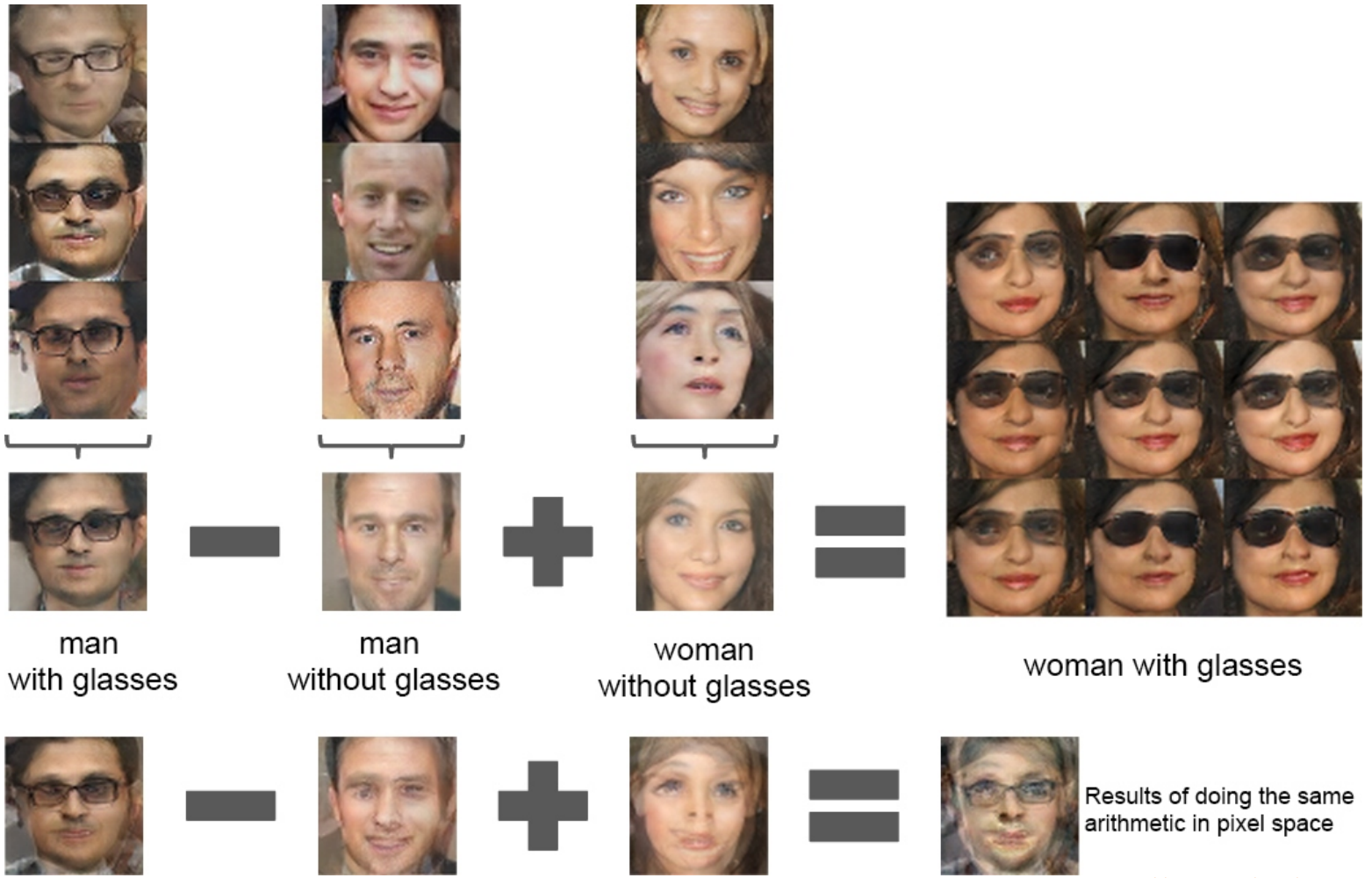


Figure 7 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

Figure 8 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

GANs are Problematic to Train

Unfortunately, alternating SGD steps are not guaranteed to reach even a local optimum of a minimax problem – consider the following one:

$$\min_x \max_y x \cdot y.$$

The update rules of x and y for learning rate α are

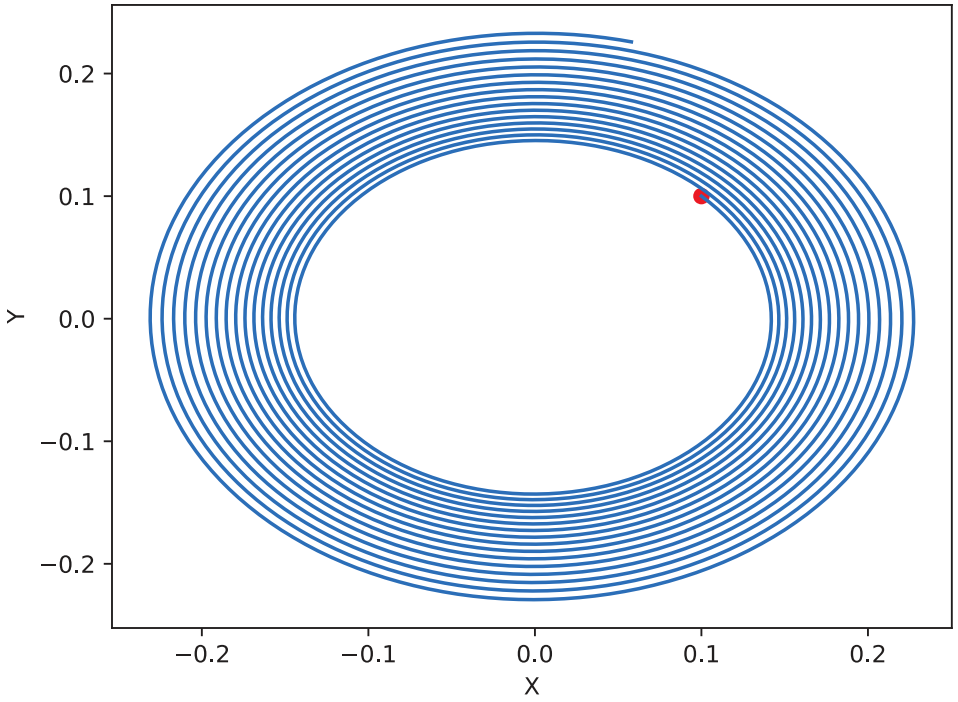
$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -\alpha \\ \alpha & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}.$$

The update matrix is a rotation matrix multiplied by a constant $\sqrt{1+\alpha^2} > 1$

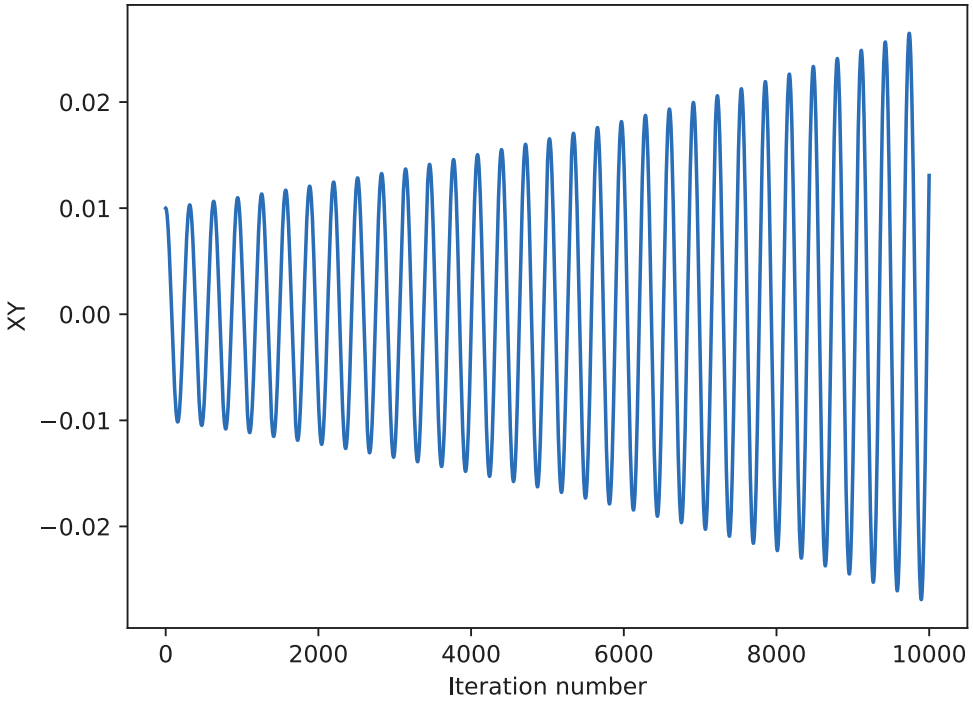
$$\begin{bmatrix} 1 & -\alpha \\ \alpha & 1 \end{bmatrix} = \sqrt{1+\alpha^2} \cdot \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix},$$

so the SGD will not converge with arbitrarily small step size.

GANs are Problematic to Train



(a)



(b)

Fig. 1: Performance of gradient method with fixed step size for Example 2. (a) illustrates the choices of x and y as iteration processes, the red point $(0.1, 0.1)$ is the initial value. (b) illustrates the value of xy as a function of iteration numbers.

Figure 1 of paper "Fictitious GAN: Training GANs with Historical Models", https://arxiv.org/abs/1803.08647.

GANs are Problematic to Train

- Mode collapse

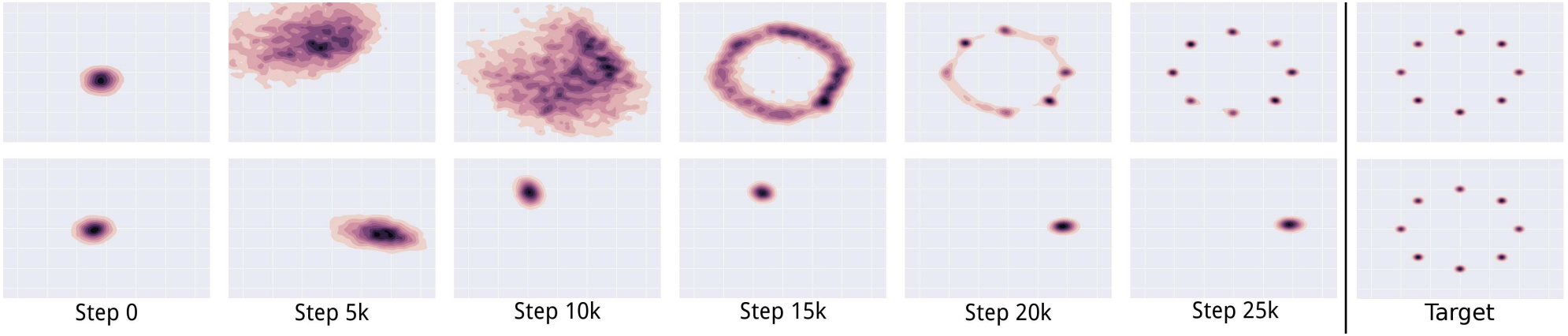
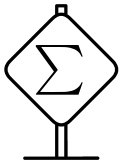


Figure 2 of paper "Unrolled Generative Adversarial Networks", https://arxiv.org/abs/1611.02163.

  - If the generator could see the whole batch, similar samples in it would be candidates for fake images.
    - Batch normalization helps a lot with this.
  - Historical averaging

- Label smoothing of only positive samples helps with the gradient flow.



Standard GANs optimize the Jensen-Shannon divergence

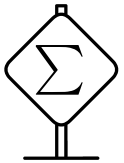
$$JS(p,q) = \frac{1}{2} D_{\mathrm{KL}}\left(p||(p+q)/2\right) + \frac{1}{2} D_{\mathrm{KL}}\left(q||(p+q)/2\right),$$

because for a fixed generator G, the optimum discriminator $D^*_G(\boldsymbol{x}) = \frac{P_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + P_{\text{generator}}(\boldsymbol{x})}$.

Therefore,

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P(\mathbf{z})}[\log(1 - D^*_G(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim P_{\text{generator}}}[\log(1 - D^*_G(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}\left[\log \frac{P_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + P_{\text{generator}}(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim P_{\text{generator}}}\left[\log \frac{P_{\text{generator}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + P_{\text{generator}}(\boldsymbol{x})}\right] \\
&= D_{\mathrm{KL}}\left(P_{\text{data}}\middle\|\frac{P_{\text{data}} + P_{\text{generator}}}{2}\right) + D_{\mathrm{KL}}\left(P_{\text{generator}}\middle\|\frac{P_{\text{data}} + P_{\text{generator}}}{2}\right) + c.
\end{aligned}$$

Wasserstein GAN



Instead of minimizing JS divergence

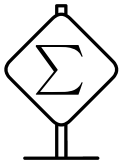
$$JS(p,q) = \frac{1}{2} D_{\mathrm{KL}}\big(p||(p+q)/2\big) + \frac{1}{2} D_{\mathrm{KL}}\big(q||(p+q)/2\big),$$

Wasserstein GAN minimizes Earth-Mover distance

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}\big[||x-y||\big].$$

The joint distribution $\gamma \in \Pi(p,q)$ indicates how much "mass" must be transported from x to y, and EM is the "cost" of the optimal transport plan.

Wasserstein GAN



The EM distance behaves much better than JS.

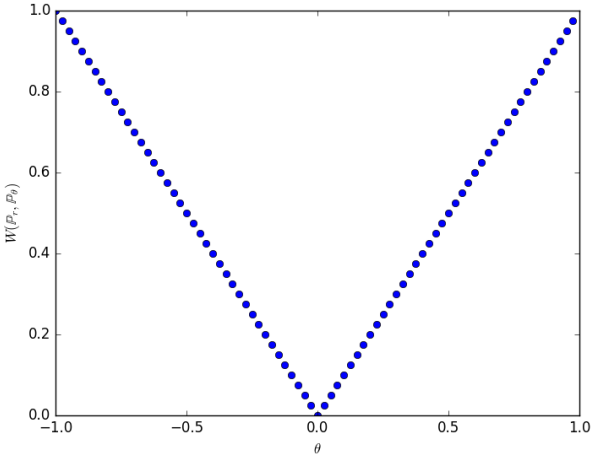
For example, imagine that P_0 is a distribution on $\mathbb{R}^2$, which is uniform on $(0, y)$ for $0 \leq y \leq 1$ and that P_θ is a distribution on $\mathbb{R}^2$ uniform on (θ, y) for $0 \leq y \leq 1$.

Then

$$JS(P_0, P_\theta) = \begin{cases} 0 & \text{if } \theta = 0 \\ \log_2 & \text{if } \theta \neq 0 \end{cases},$$

while

$$W(P_0, P_\theta) = |\theta|.$$



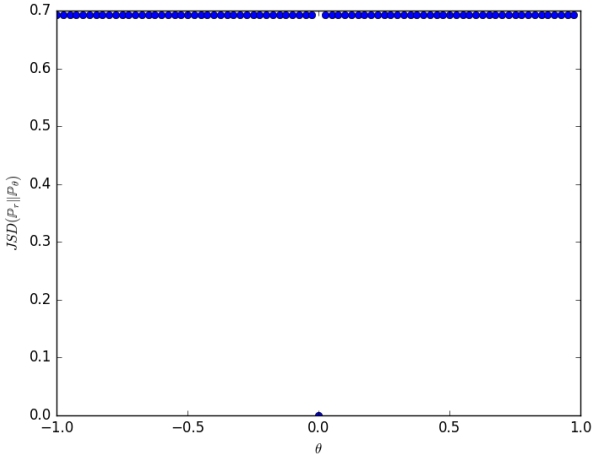
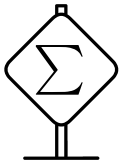


Figure 1 of paper "Wasserstein GAN", https://arxiv.org/abs/1701.07875.

Wasserstein GAN

Using a dual version of the Earth-Mover definition, we arrive at



$$W(p,q) = \sup_{f, ||f||_L \leq 1} \mathbb{E}_{x \sim p}\big[f(x)\big] - \mathbb{E}_{y \sim q}\big[f(x)\big],$$

so the discriminator returns a single output without activation and it needs to be 1-Lipschitz.

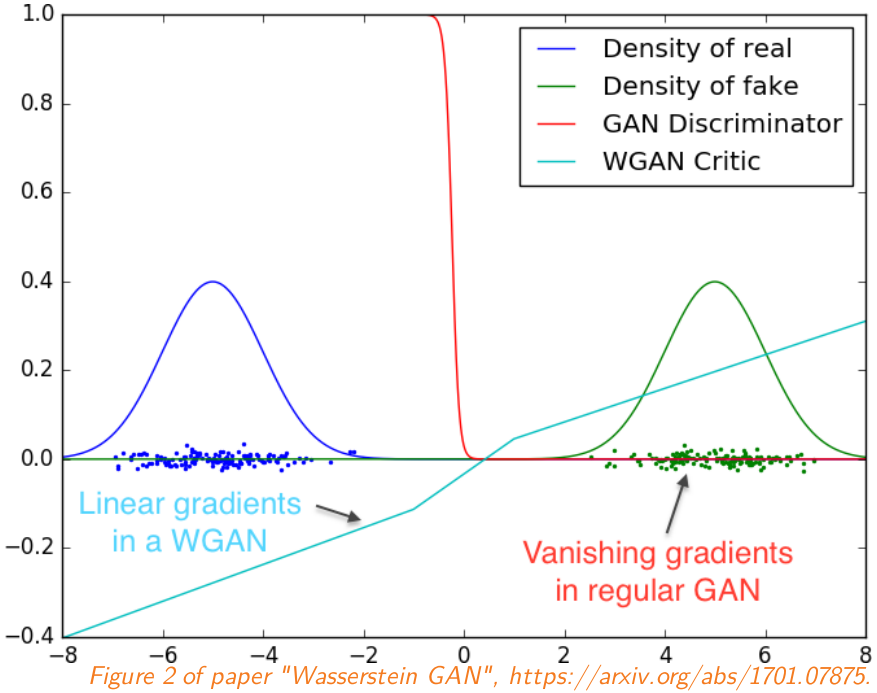


Figure 2 of paper "Wasserstein GAN", https://arxiv.org/abs/1701.07875.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Algorithm 1 of paper "Wasserstein GAN", https://arxiv.org/abs/1701.07875.

Wasserstein GAN

Figure 5: Algorithms trained with a DCGAN generator. Left: WGAN algorithm. Right: standard GAN formulation. Both algorithms produce high quality samples.

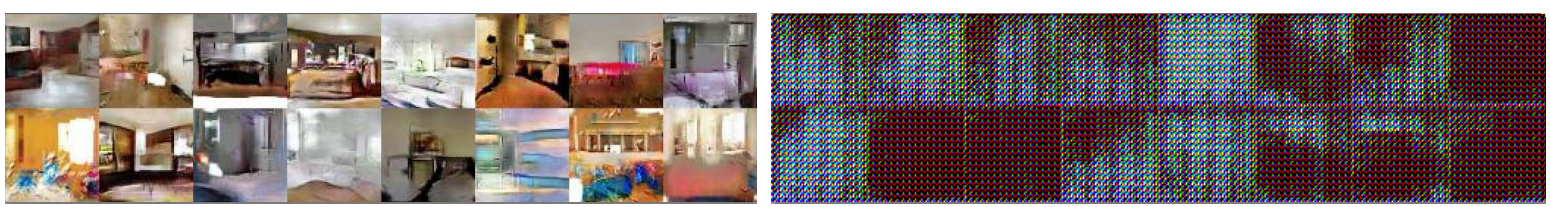


Figure 6: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in [18]). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude. Left: WGAN algorithm. Right: standard GAN formulation. As we can see the standard GAN failed to learn while the WGAN still was able to produce samples.

Figures 5 and 6 of paper "Wasserstein GAN", https://arxiv.org/abs/1701.07875.

Wasserstein GAN

Figure 7: Algorithms trained with an MLP generator with 4 layers and 512 units with ReLU nonlinearities. The number of parameters is similar to that of a DCGAN, but it lacks a strong inductive bias for image generation. Left: WGAN algorithm. Right: standard GAN formulation. The WGAN method still was able to produce samples, lower quality than the DCGAN, and of higher quality than the MLP of the standard GAN. Note the significant degree of mode collapse in the GAN MLP.

Figure 7 of paper "Wasserstein GAN", https://arxiv.org/abs/1701.07875.

Development of GANs

Generative Adversarial Networks are still in active development:

- Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen: **Progressive Growing of GANs for Improved Quality, Stability, and Variation** https://arxiv.org/abs/1710.10196
- Takeru Miyato, Toshiki Kataoka, Masanori Koyama, Yuichi Yoshida: **Spectral Normalization for Generative Adversarial Networks** https://arxiv.org/abs/1802.05957
- Zhiming Zhou, Yuxuan Song, Lantao Yu, Hongwei Wang, Jiadong Liang, Weinan Zhang, Zhihua Zhang, Yong Yu: **Understanding the Effectiveness of Lipschitz-Continuity in Generative Adversarial Nets** https://arxiv.org/abs/1807.00751
- Andrew Brock, Jeff Donahue, Karen Simonyan: **Large Scale GAN Training for High Fidelity Natural Image Synthesis** https://arxiv.org/abs/1809.11096
- Tero Karras, Samuli Laine, Timo Aila: **A Style-Based Generator Architecture for Generative Adversarial Networks** https://arxiv.org/abs/1812.04948

Alternative approaches are also explored: Diederik P. Kingma, Prafulla Dhariwal: **Glow: Generative Flow with Invertible 1x1 Convolutions** https://arxiv.org/abs/1807.03039

BigGAN

Figure 1 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.

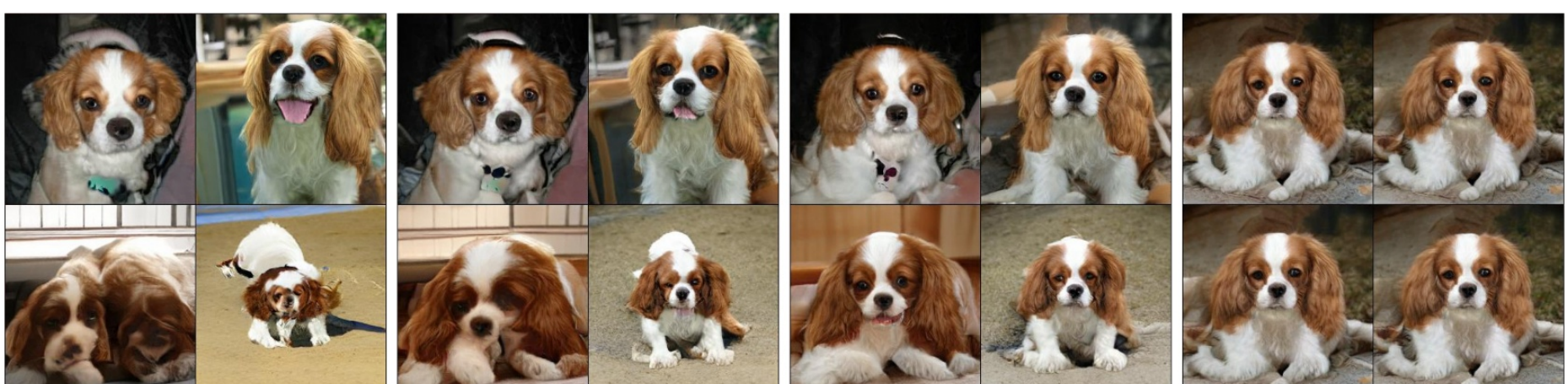


Figure 2 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.

BigGAN

Figure 7 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.

BigGAN Ingredients – Hinge Loss

The Wasserstein GAN formulation can be considered a linear classifier, which tries to maximize the mean distance of real and generated images using their features.

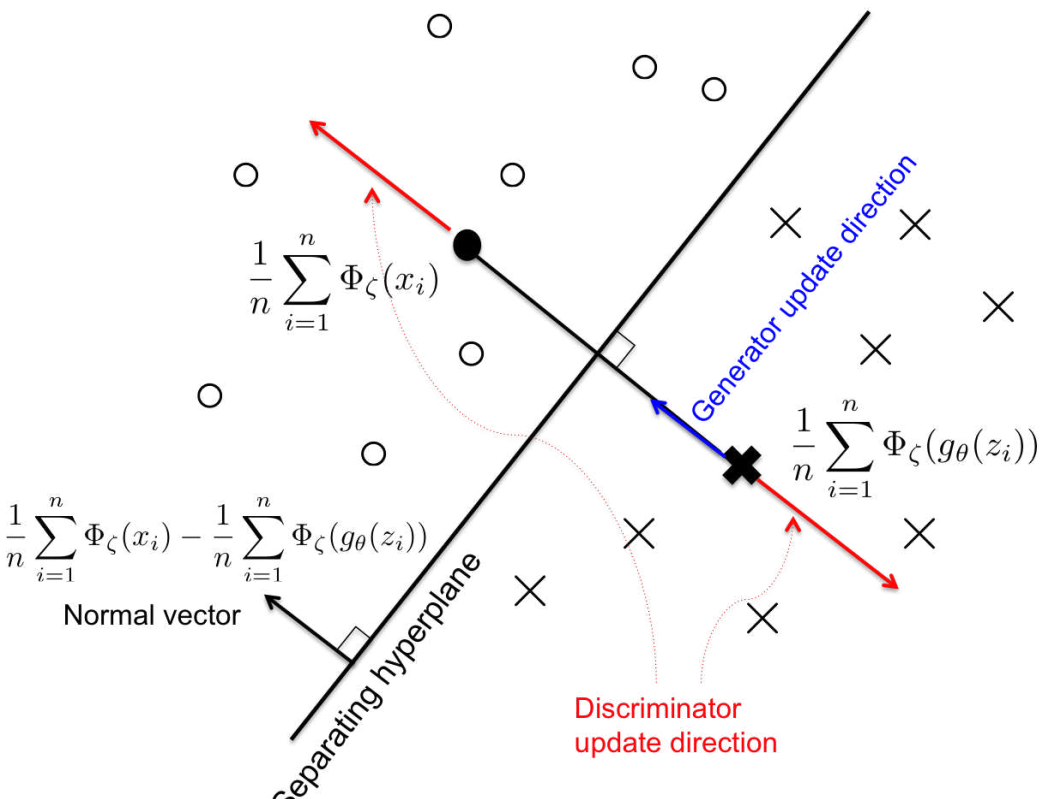
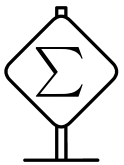


Figure 2 of paper "Geometric GAN", https://arxiv.org/abs/1705.02894.

BigGAN Ingredients – Hinge Loss



We could aim for maximum margin classifier by using Hinge loss, updating the discriminator by

$$\arg\max_{\boldsymbol{\theta}_d} \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}[\min(0, -1 + D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P(\mathbf{z})}[\min(0, -1 - D(G(\boldsymbol{z}))]$$

and the generator by

$$\arg\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{z}\sim P(\mathbf{z})}[-D(G(\boldsymbol{z}))].$$

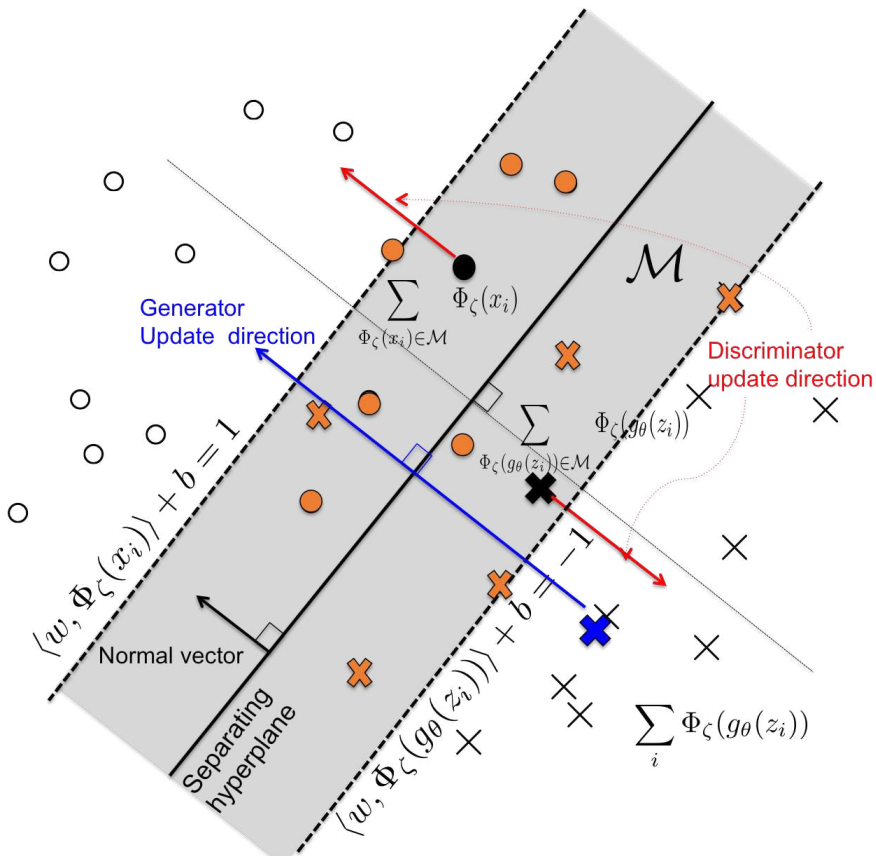
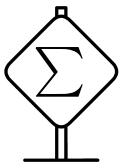


Figure 2 of paper "Geometric GAN", https://arxiv.org/abs/1705.02894.

BigGAN Ingredients – Spectral Normalization



Satisfying the Lipschitz constraint by truncation is not very effective. Better approaches were proposed, by using for example gradient penalties (WGAN-GP) or *spectral normalization*.

In spectral normalization, the idea is to keep the *spectral norm* (the largest singular value) of all convolutional and dense layers equal or close to 1, which in turn guarantees the Lipschitz constraint of the model.

Spectral normalization can be implemented efficiently by performing one step of power iteration each time the kernel in question is used in training.

Algorithm 1 SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer l:
  1. Apply power iteration method to a unnormalized weight W^l:

$$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2 \tag{20}$$

$$\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2 \tag{21}$$

  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l \tag{22}$$

  3. Update W^l with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate α:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M) \tag{23}$$

Algoritm 1 of paper "Spectral Normalization for Generative Adversarial Networks", https://arxiv.org/abs/1802.05957.

BigGAN Ingredients – Self Attention

Because convolutions process local information only, non-local *self attention* module has been proposed.

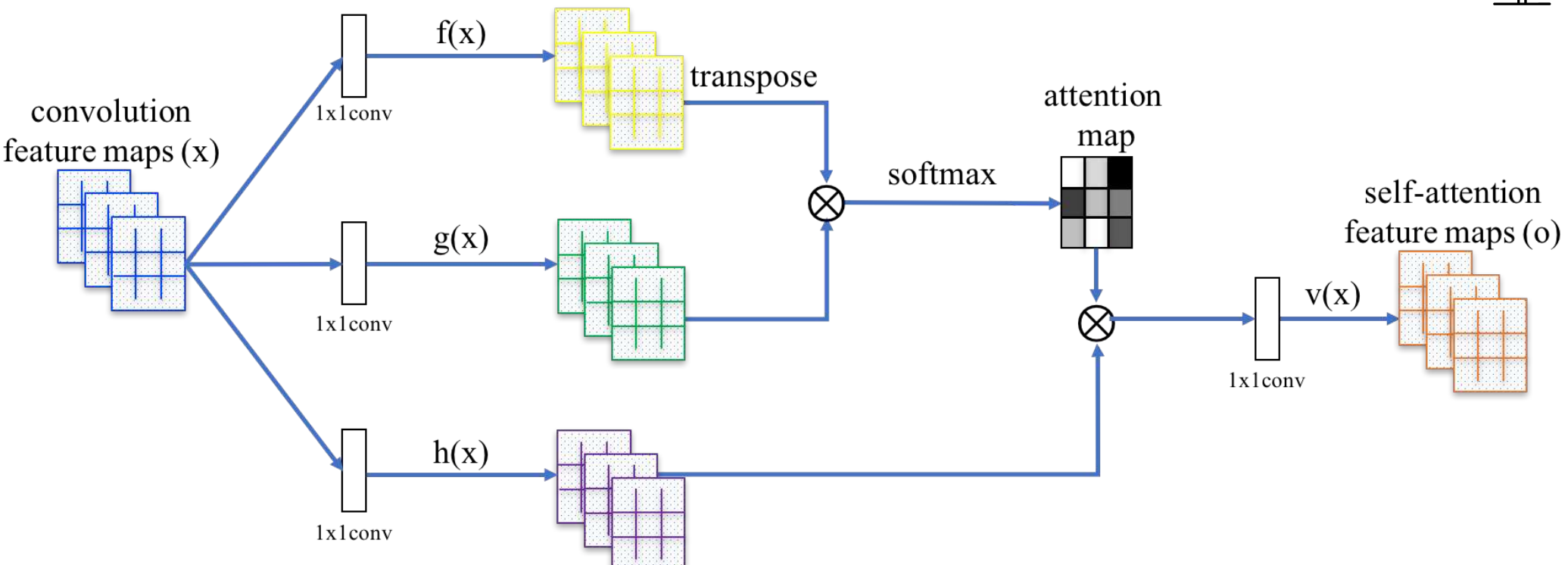


Figure 2 of paper "Self-Attention Generative Adversarial Networks", https://arxiv.org/abs/1805.08318.

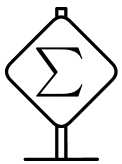
BigGAN Ingredients – Self Attention

```
def attention(self, x, ch):
  f = conv(x, ch // 8, kernel=1, stride=1) # [bs, h, w, c']
  g = conv(x, ch // 8, kernel=1, stride=1) # [bs, h, w, c']
  h = conv(x, ch, kernel=1, stride=1) # [bs, h, w, c]

  # N = h * w
  s = tf.matmul(
        hw_flatten(g), hw_flatten(f), transpose_b=True) # [bs, N, N]
  beta = tf.nn.softmax(s)  # attention map

  o = tf.matmul(beta, hw_flatten(h)) # [bs, N, C]
  gamma = tf.get_variable("gamma", initializer=[0.0])

  o = tf.reshape(o, shape=x.shape) # [bs, h, w, C]
  x = gamma * o + x
  return x
```



BigGAN Ingredients – Architecture

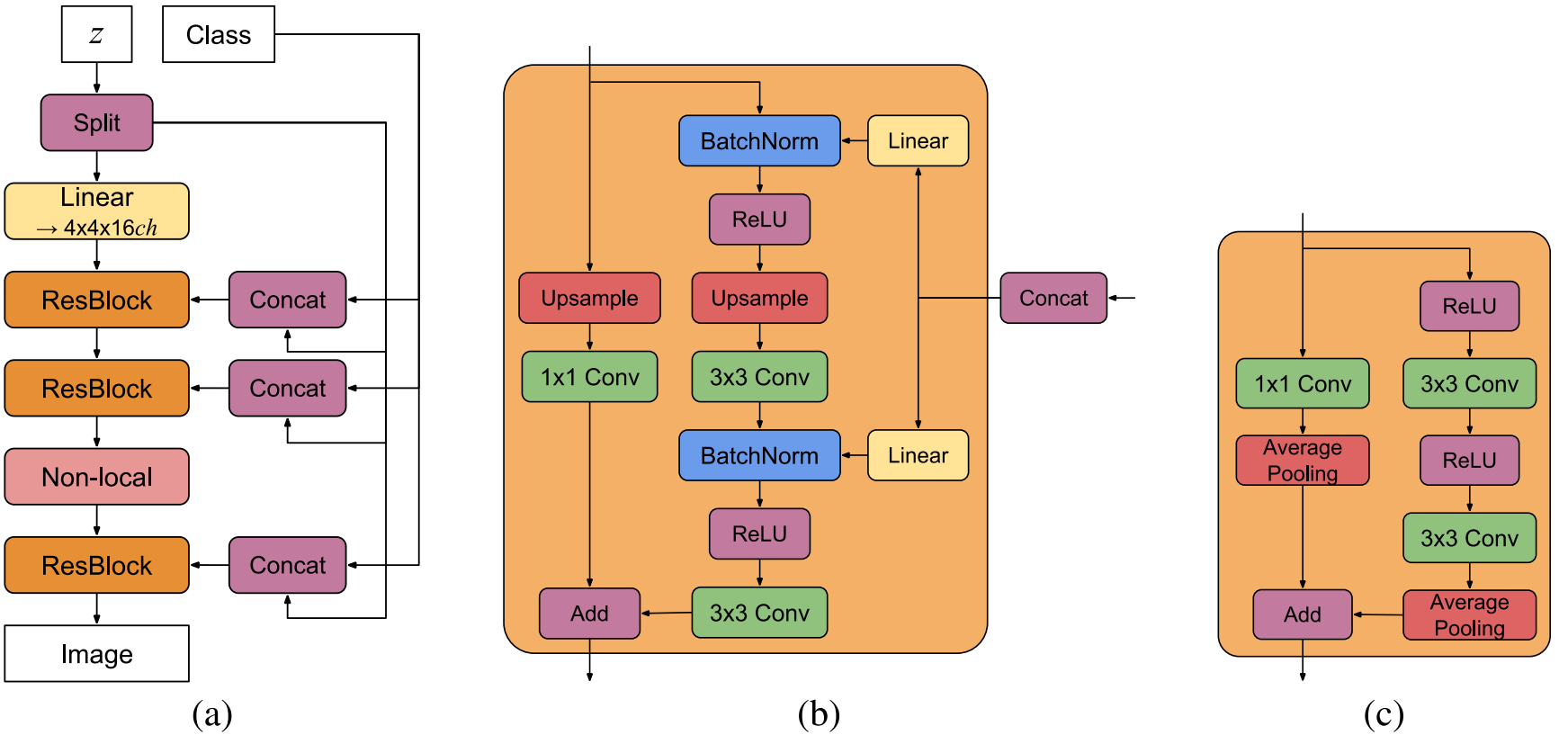
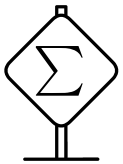


Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

Figure 15 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.

BigGAN Ingredients – Architecture

Table 4: BigGAN architecture for 128×128 images. ch represents the channel width multiplier in each network from Table 1.



$z \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \to 4 \times 4 \times 16ch$
ResBlock up $16ch \to 16ch$
ResBlock up $16ch \to 8ch$
ResBlock up $8ch \to 4ch$
ResBlock up $4ch \to 2ch$
Non-Local Block (64×64)
ResBlock up $2ch \to ch$
BN, ReLU, 3×3 Conv $ch \to 3$
Tanh

(a) Generator

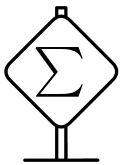
RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down $ch \to 2ch$
Non-Local Block (64×64)
ResBlock down $2ch \to 4ch$
ResBlock down $4ch \to 8ch$
ResBlock down $8ch \to 16ch$
ResBlock down $16ch \to 16ch$
ResBlock $16ch \to 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\to 1$)

(b) Discriminator

Table 4 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.

BigGAN Ingredients – Architecture

Table 4: BigGAN architecture for 128×128 images. ch represents the channel width multiplier in each network from Table 1.



$z \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \to 4 \times 4 \times 16ch$
ResBlock up $16ch \to 16ch$
ResBlock up $16ch \to 8ch$
ResBlock up $8ch \to 4ch$
ResBlock up $4ch \to 2ch$
Non-Local Block (64×64)
ResBlock up $2ch \to ch$
BN, ReLU, 3×3 Conv $ch \to 3$
Tanh

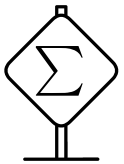
(a) Generator

RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down $ch \to 2ch$
Non-Local Block (64×64)
ResBlock down $2ch \to 4ch$
ResBlock down $4ch \to 8ch$
ResBlock down $8ch \to 16ch$
ResBlock down $16ch \to 16ch$
ResBlock $16ch \to 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\to 1$)

(b) Discriminator

Table 4 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.

BigGAN Ingredients – Truncation Trick



The so-called **tuncation trick** is used to trade between fidelity and variety – during training, $\boldsymbol{z}$ is sampled from $\mathcal{N}(\mathbf{0}, \mathbf{1})$, while it is sampled from *truncated normal* during generation.

In the following examle, samples were generated using threshold $2, 1, 0.5, 0.04$.

Figure 2 from paper "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096.