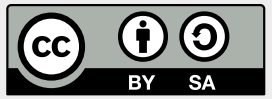
Deep Generative Models

Milan Straka

May 6, 2019

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.

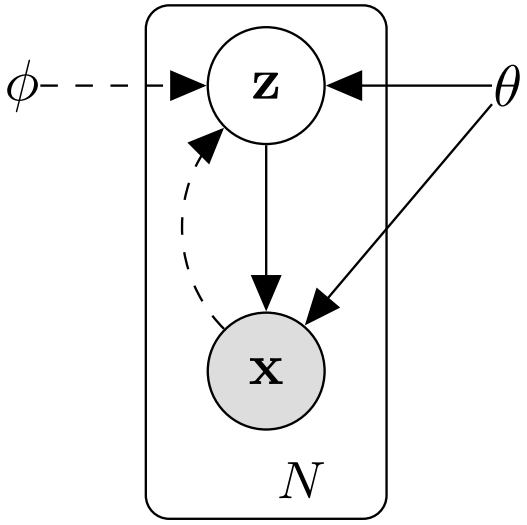


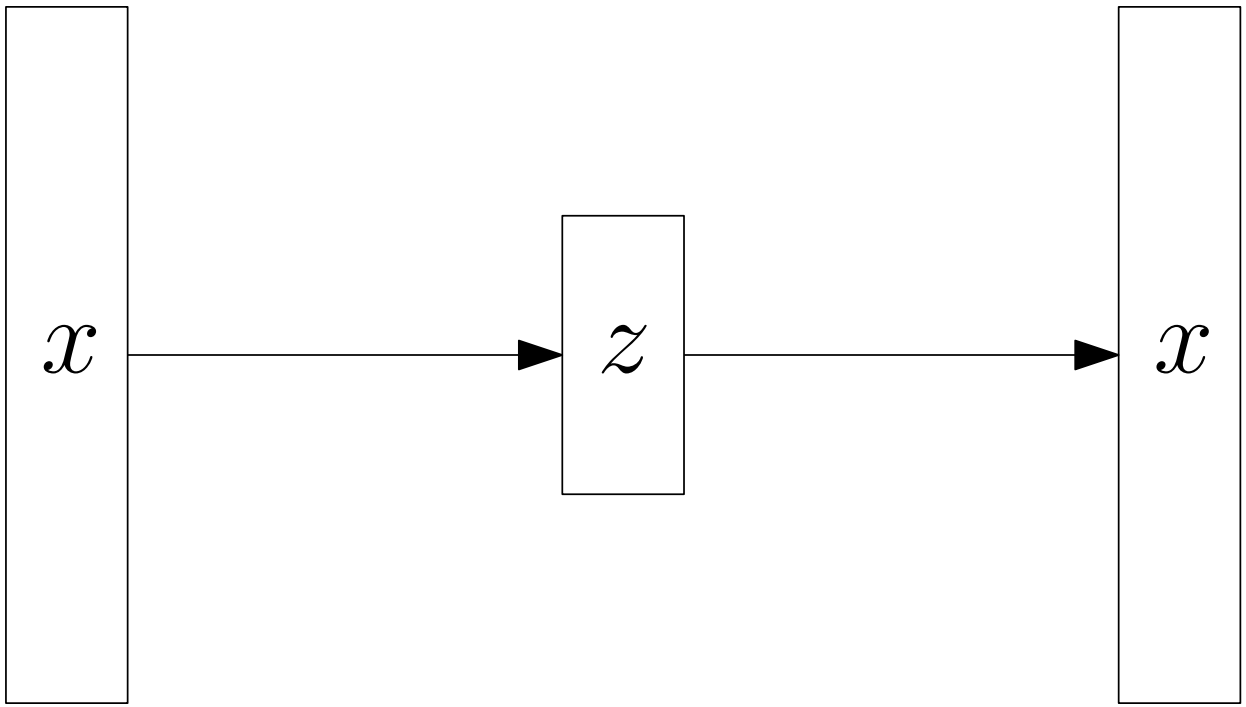
Figure 1 of paper "Auto-Encoding Variational Bayes", https://arxiv.org/abs/1312.6114.

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

$$P(\boldsymbol{x}) = P(\boldsymbol{z})P(\boldsymbol{x}|\boldsymbol{z}).$$

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

AutoEncoders



- unsupervised feature extraction
- input compression for $z < x$
- when $\boldsymbol{x} + \boldsymbol{\varepsilon}$ is used as input, autoencoders can perform denoising

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

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

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 $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x})$, taking a single point estimate of the expectation and using a prior $P(\boldsymbol{z}) = \mathcal{N}(0, 1)$.

Note that the loss has 2 intuitive components:

- reconstruction loss: Starting with $\boldsymbol{x}$, and passing though Q and then again through P should arrive back at $\boldsymbol{x}$.
- latent loss: The distribution of $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ should be as close to the prior $P(\boldsymbol{z}) = \mathcal{N}(0, 1)$, which is independent on $\boldsymbol{x}$.

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

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

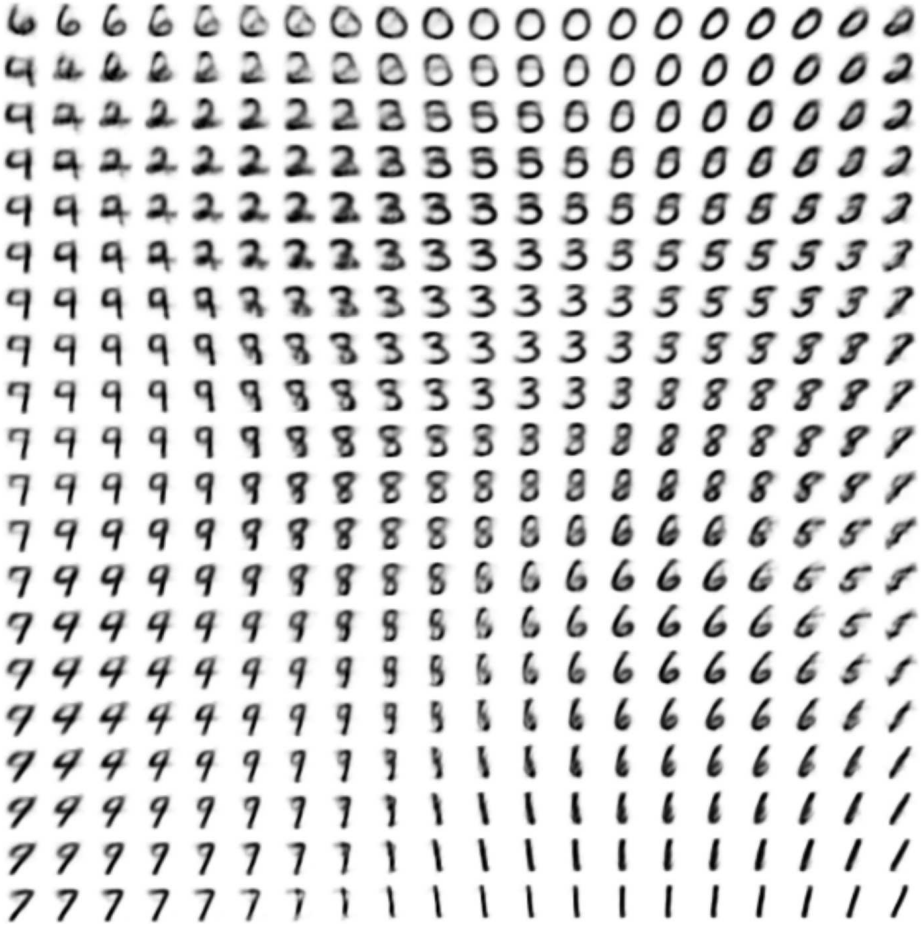
we can write $\boldsymbol{z}$ as

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

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

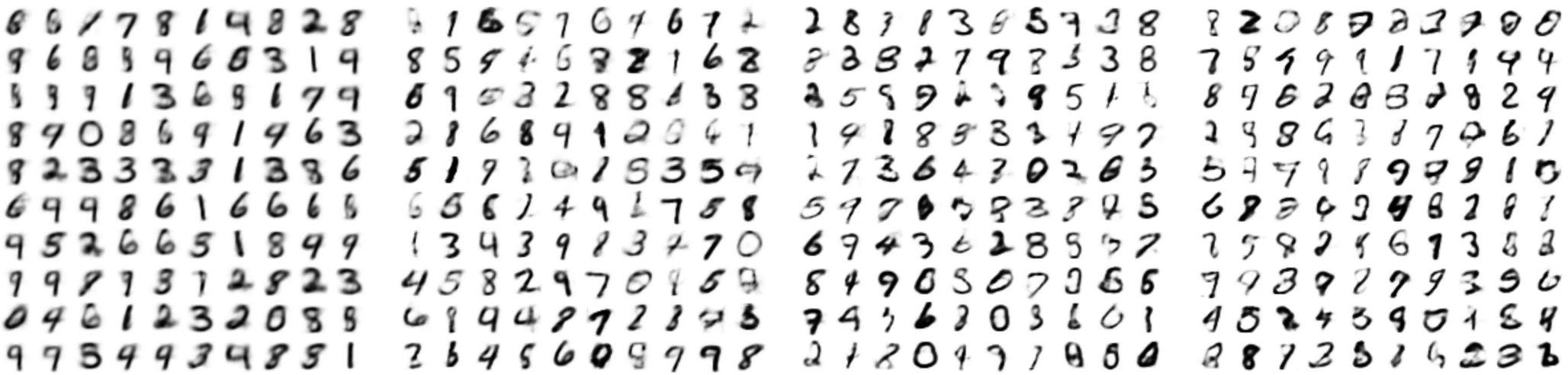
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

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

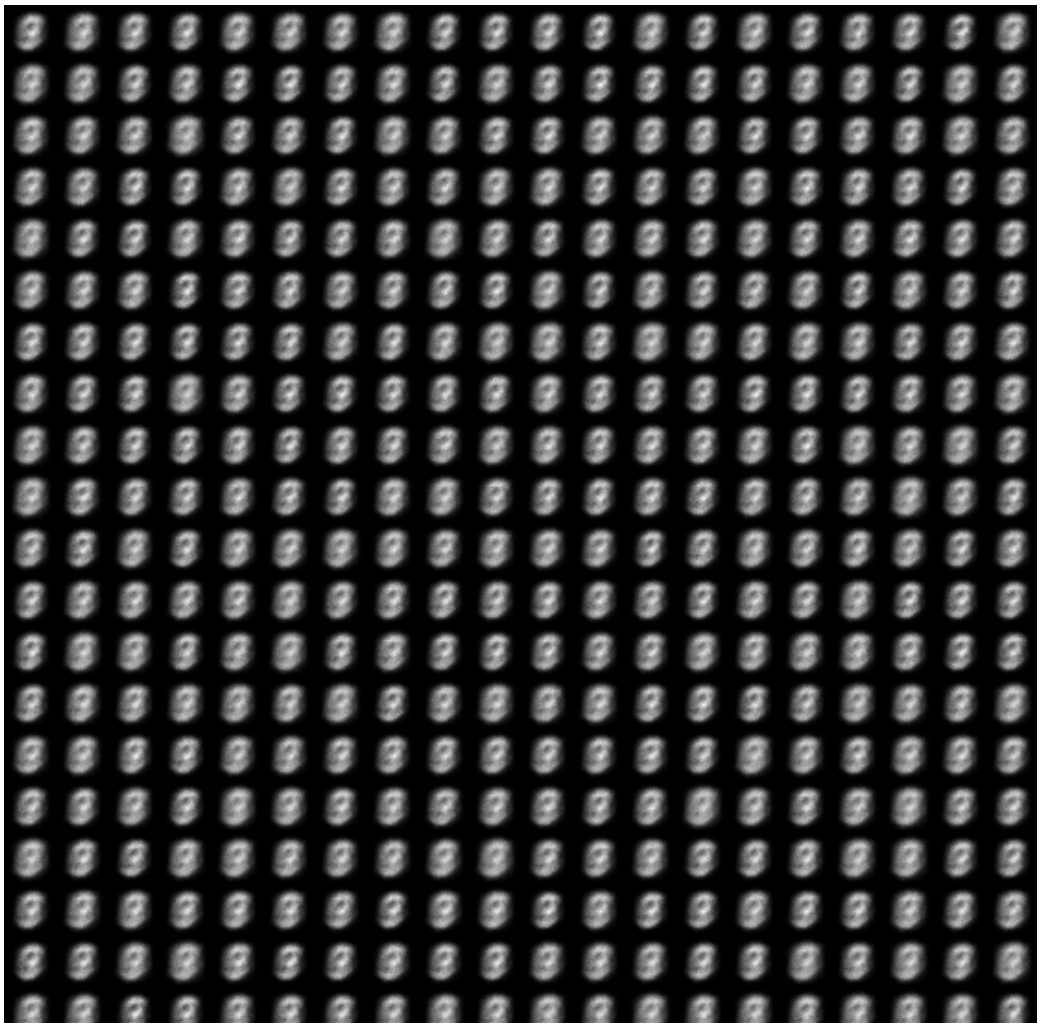
Variational AutoEncoders



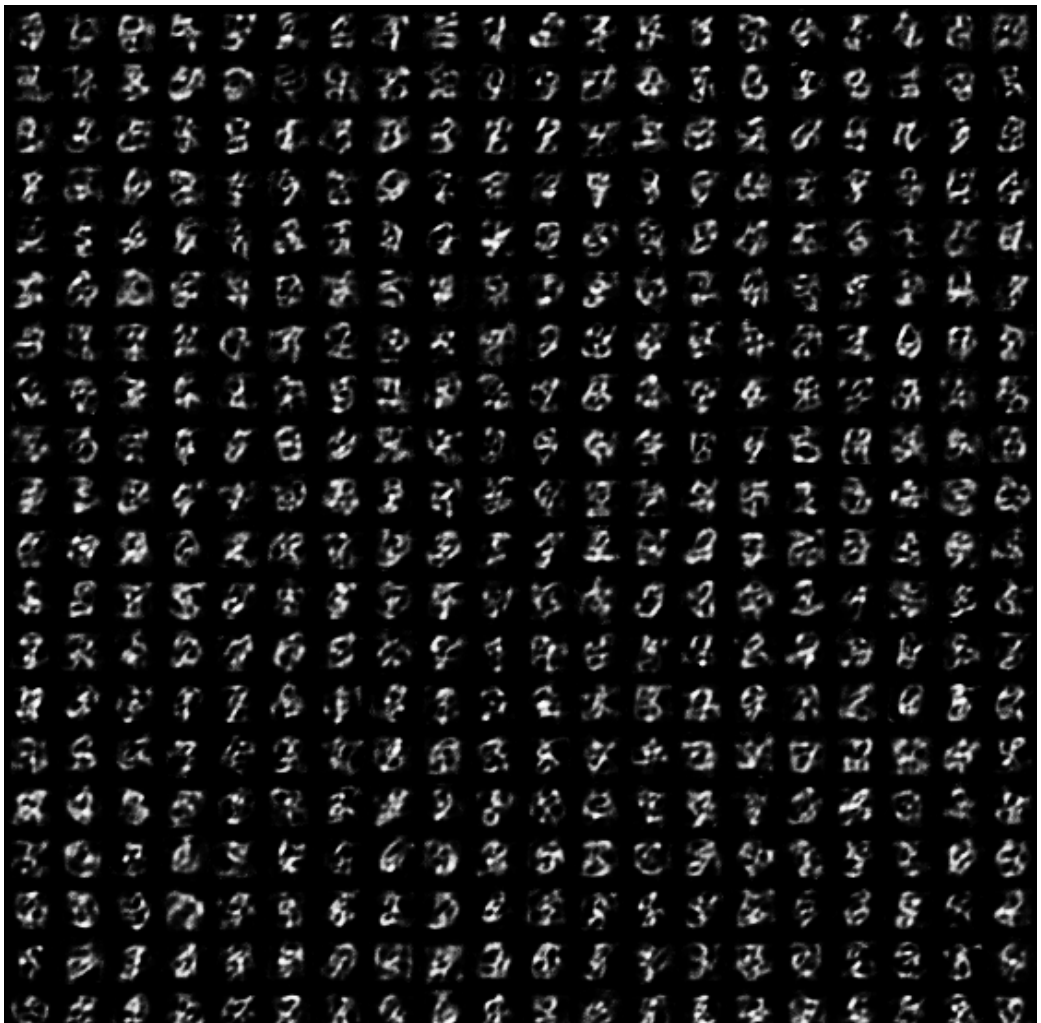
(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

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 discrimininator as $D(\boldsymbol{x}; \boldsymbol{\theta}_d)$.

The discriminator and generator play the following game:

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

Generative Adversarial Networks

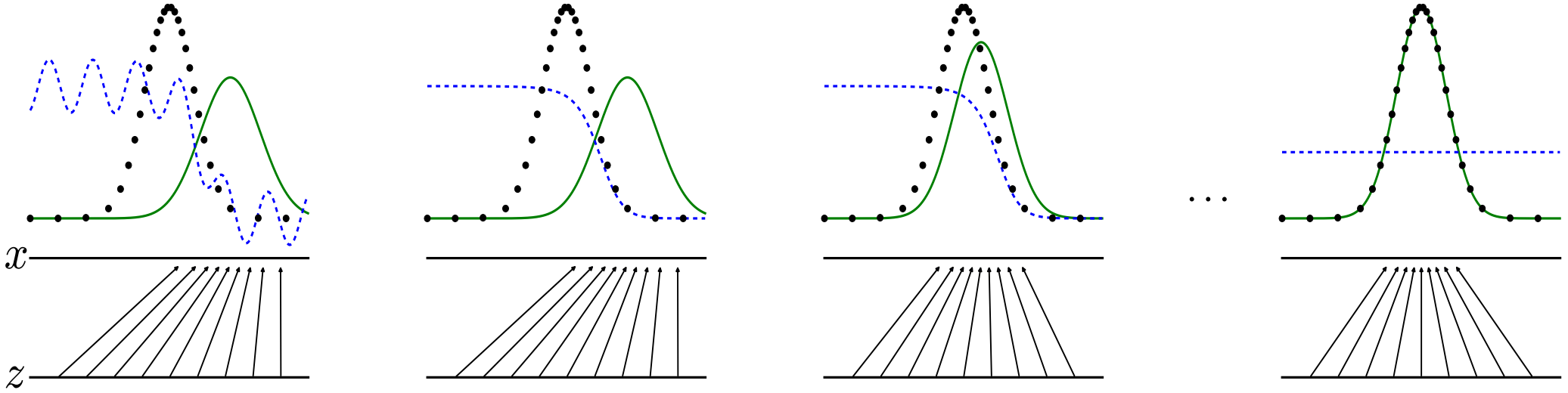


Figure 1 of paper "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661.

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:

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

  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

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

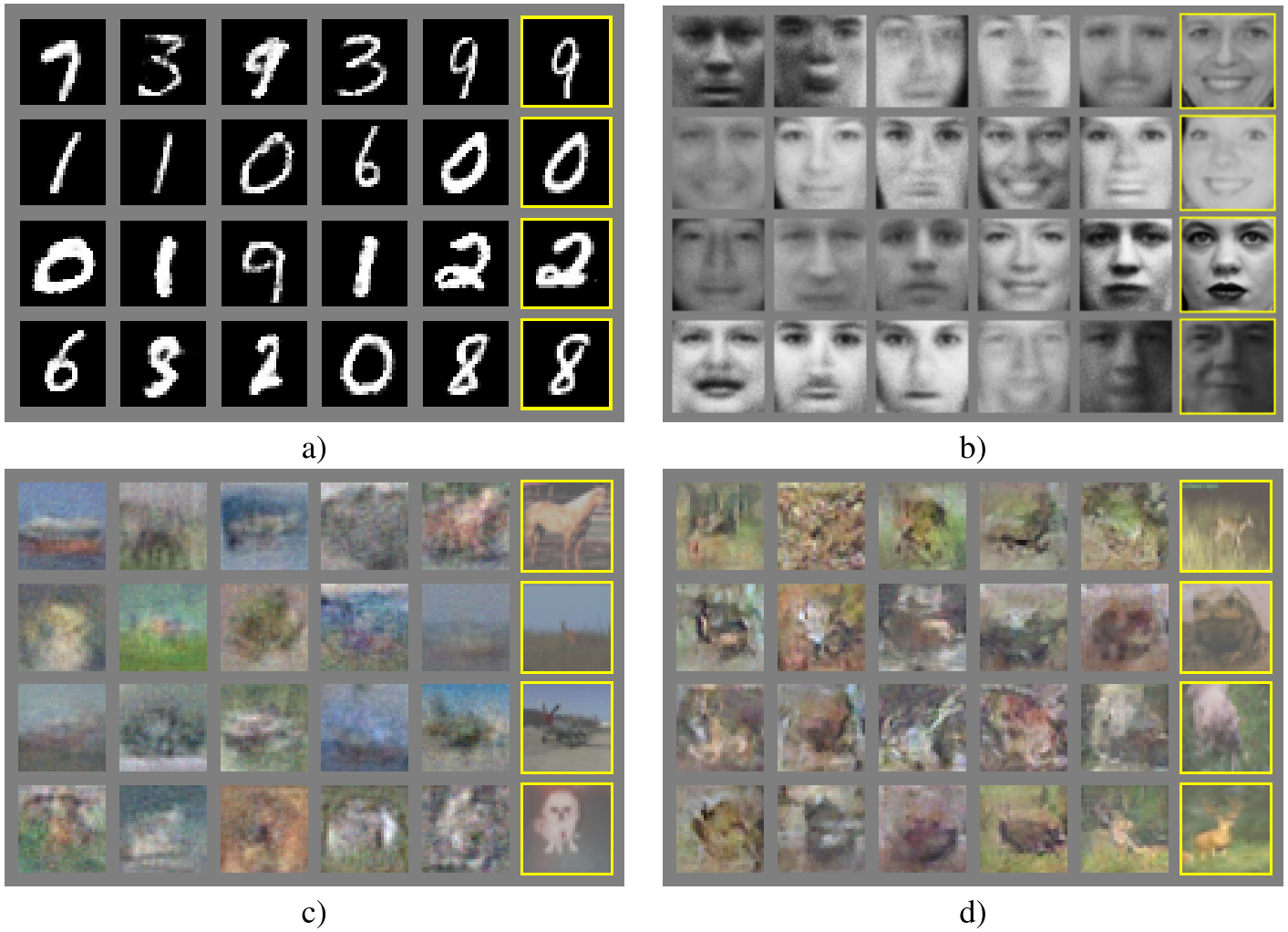


Figure 2 of paper "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661.

Conditional GAN

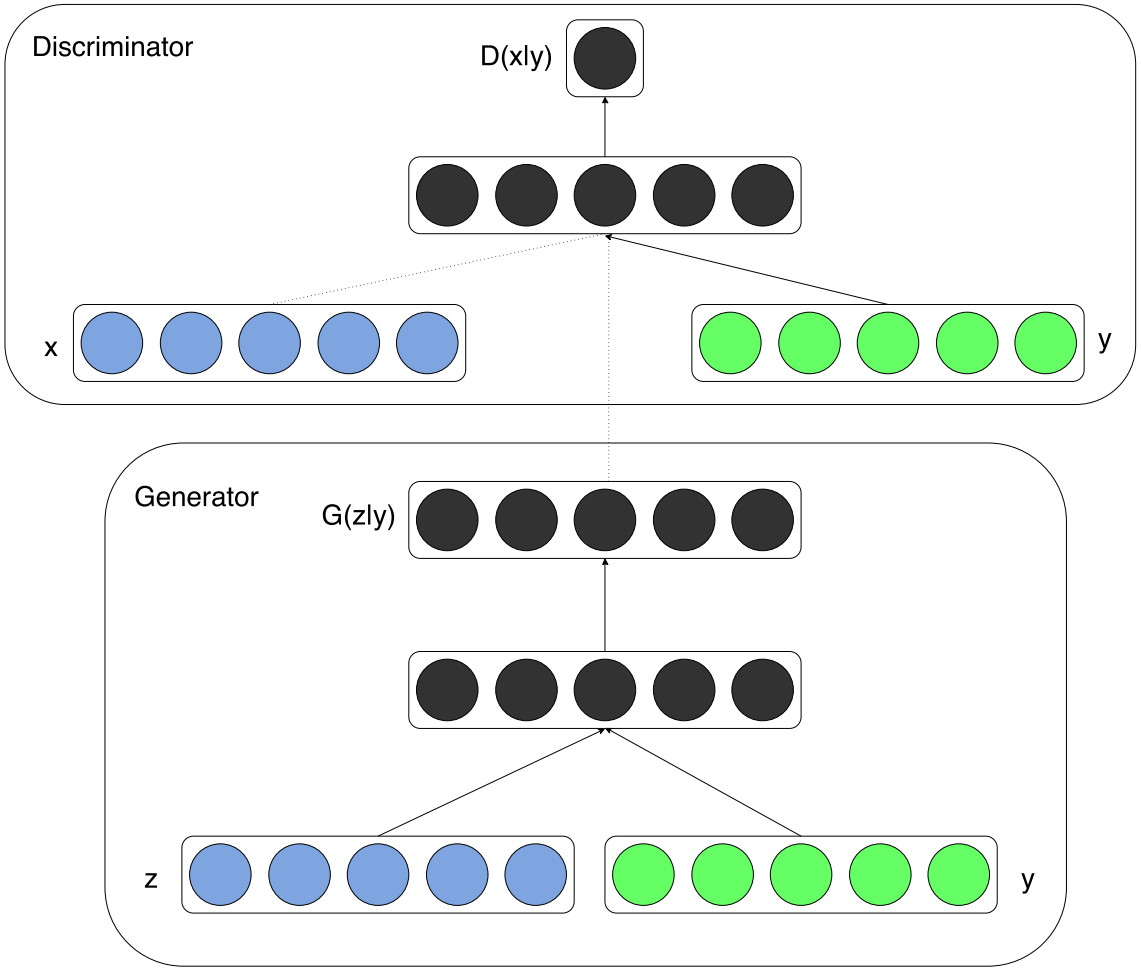
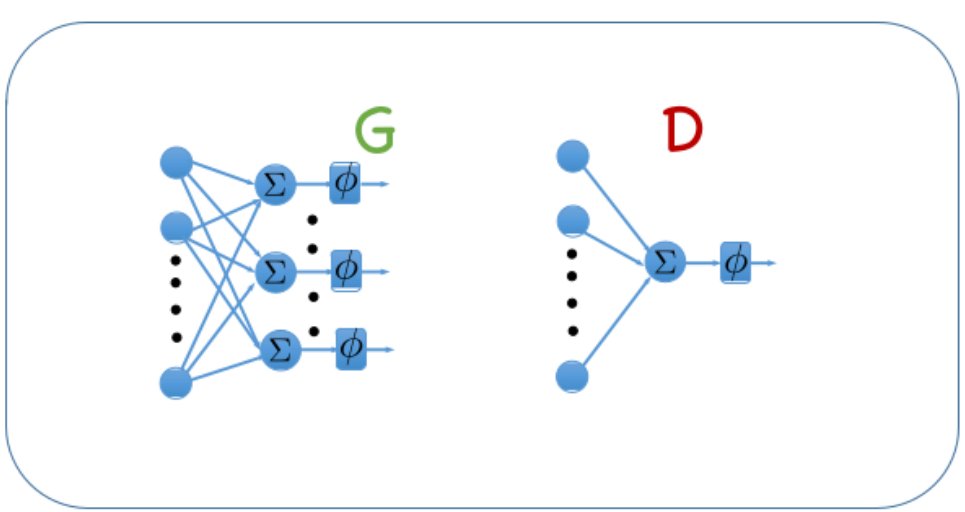
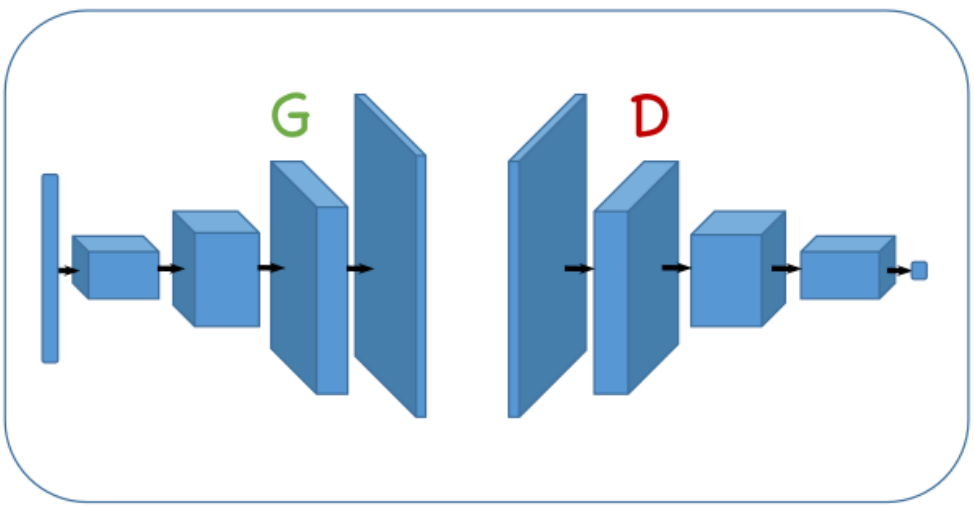


Figure 1 of paper "Conditional Generative Adversarial Nets", https://arxiv.org/abs/1411.1784.

Deep Convolutional GAN



(a)



(c)

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

Deep Convolutional GAN

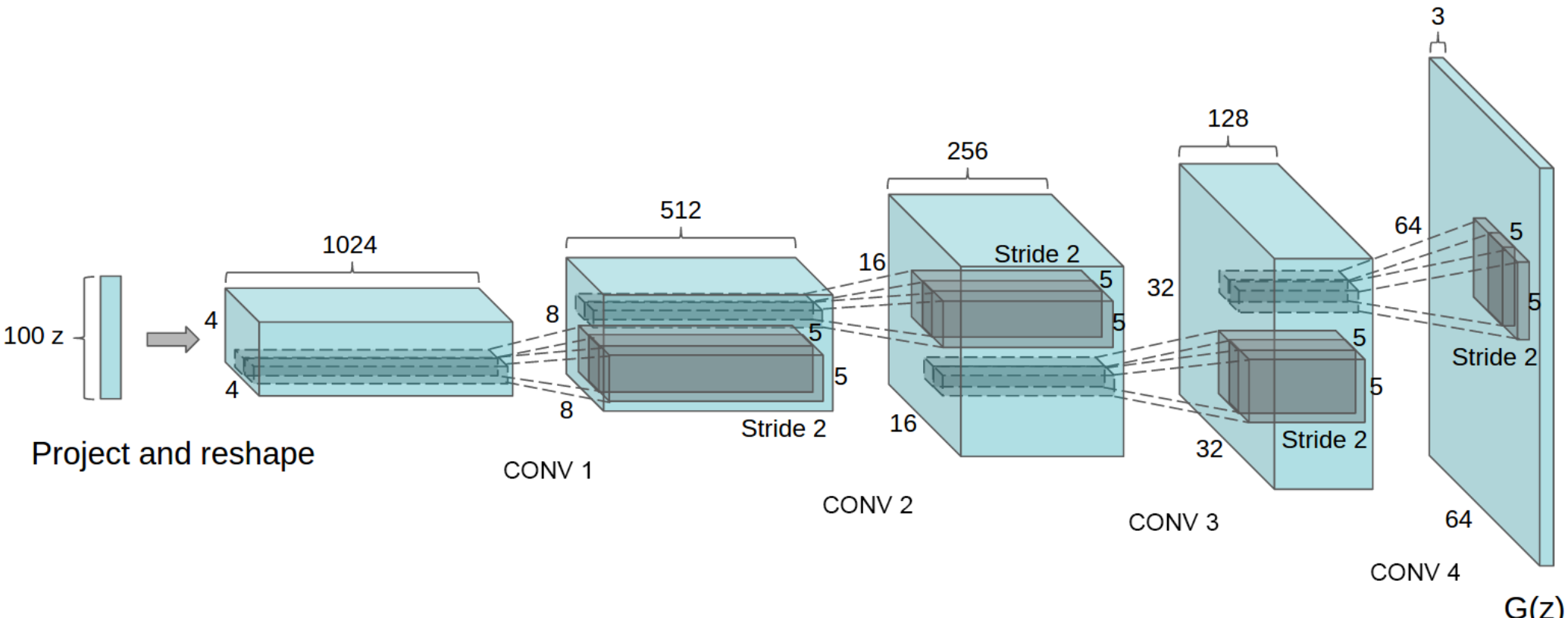


Figure 1 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

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

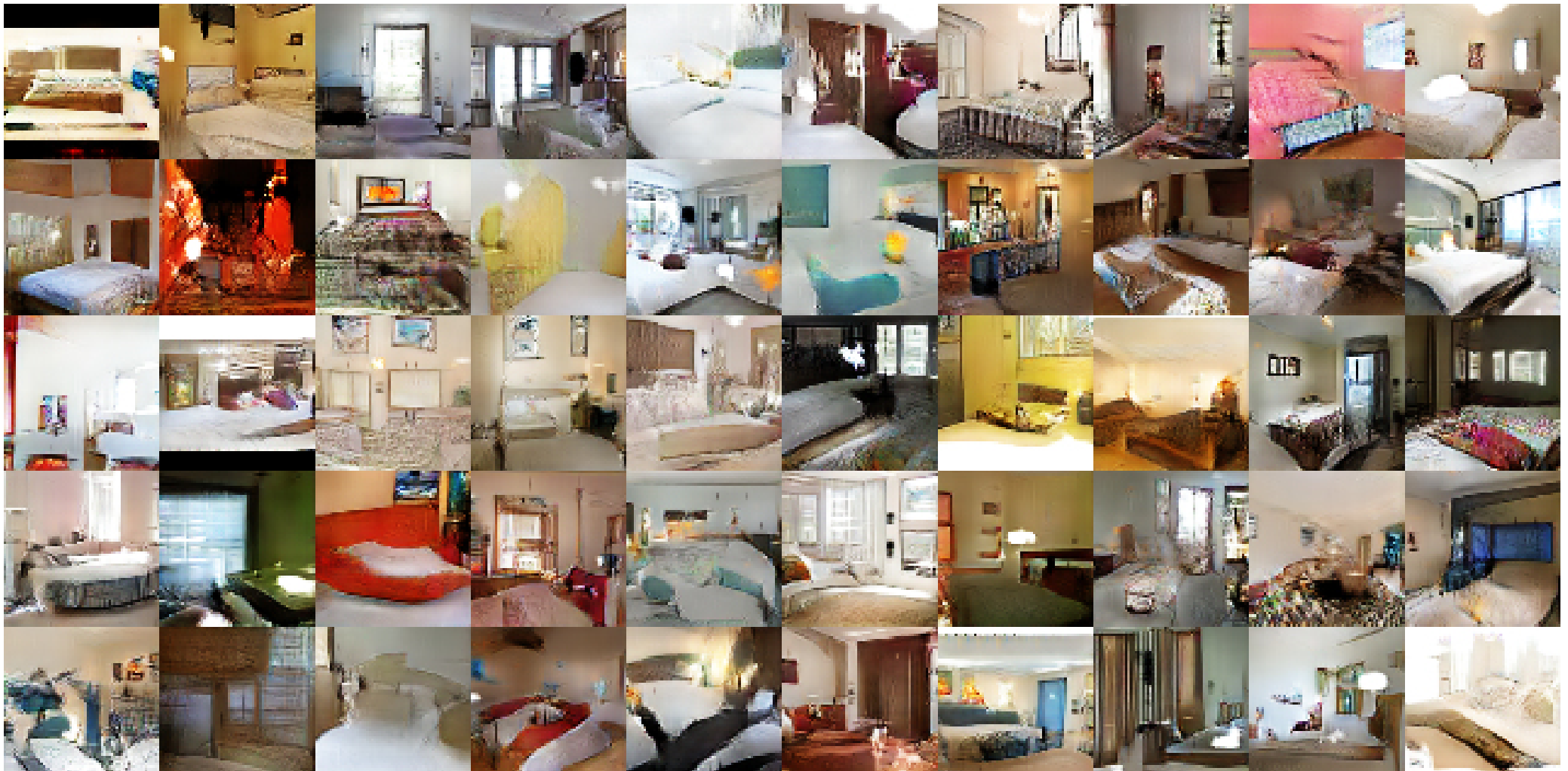


Figure 3 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

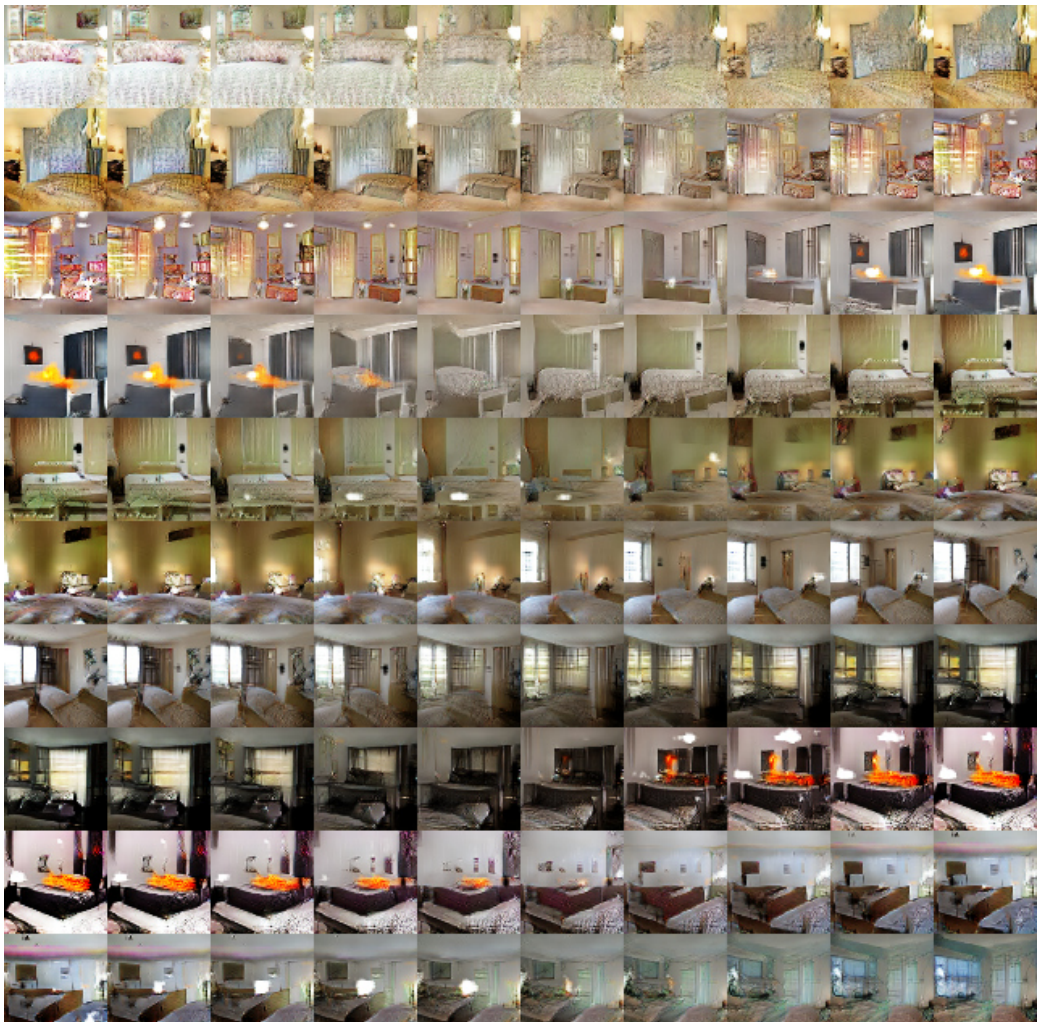


Figure 4 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

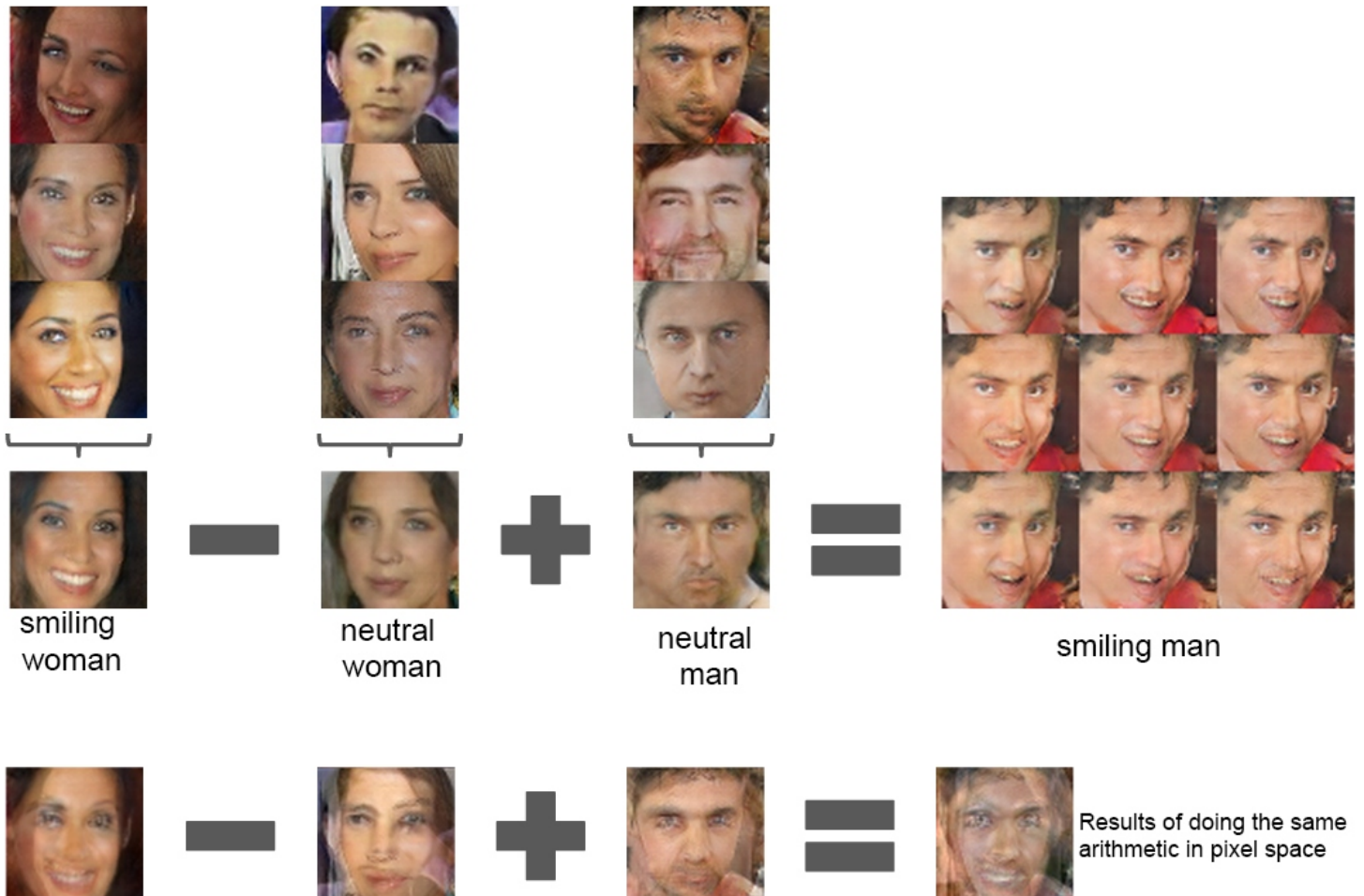


Figure 7 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

Deep Convolutional GAN

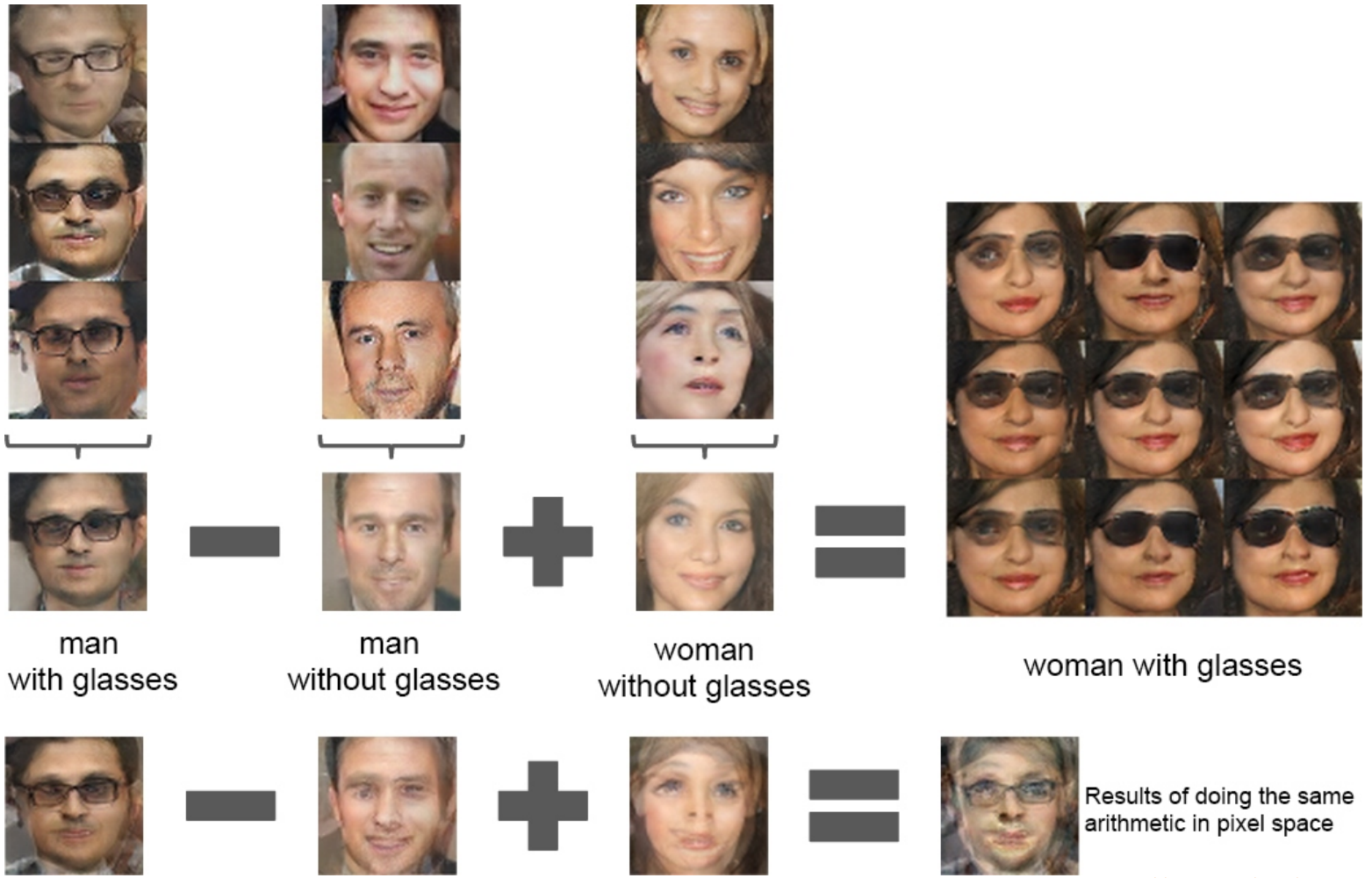


Figure 7 of paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434.

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

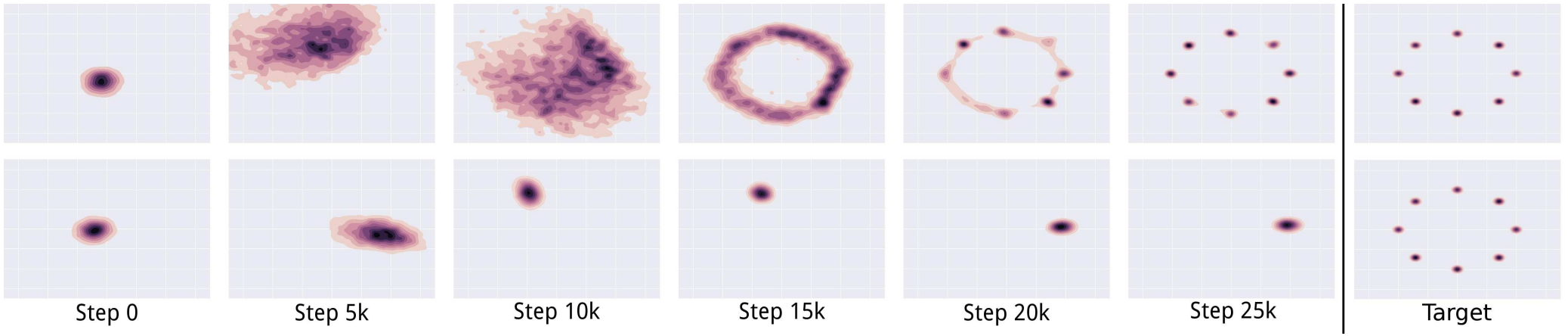


Figure 2 of paper "Unrolled Generative Adversarial Networks", https://arxiv.org/abs/1611.02163.

- Feature matching
- Minibatch discrimination
- Historical averaging
- Label smoothing

Minibatch Discrimination

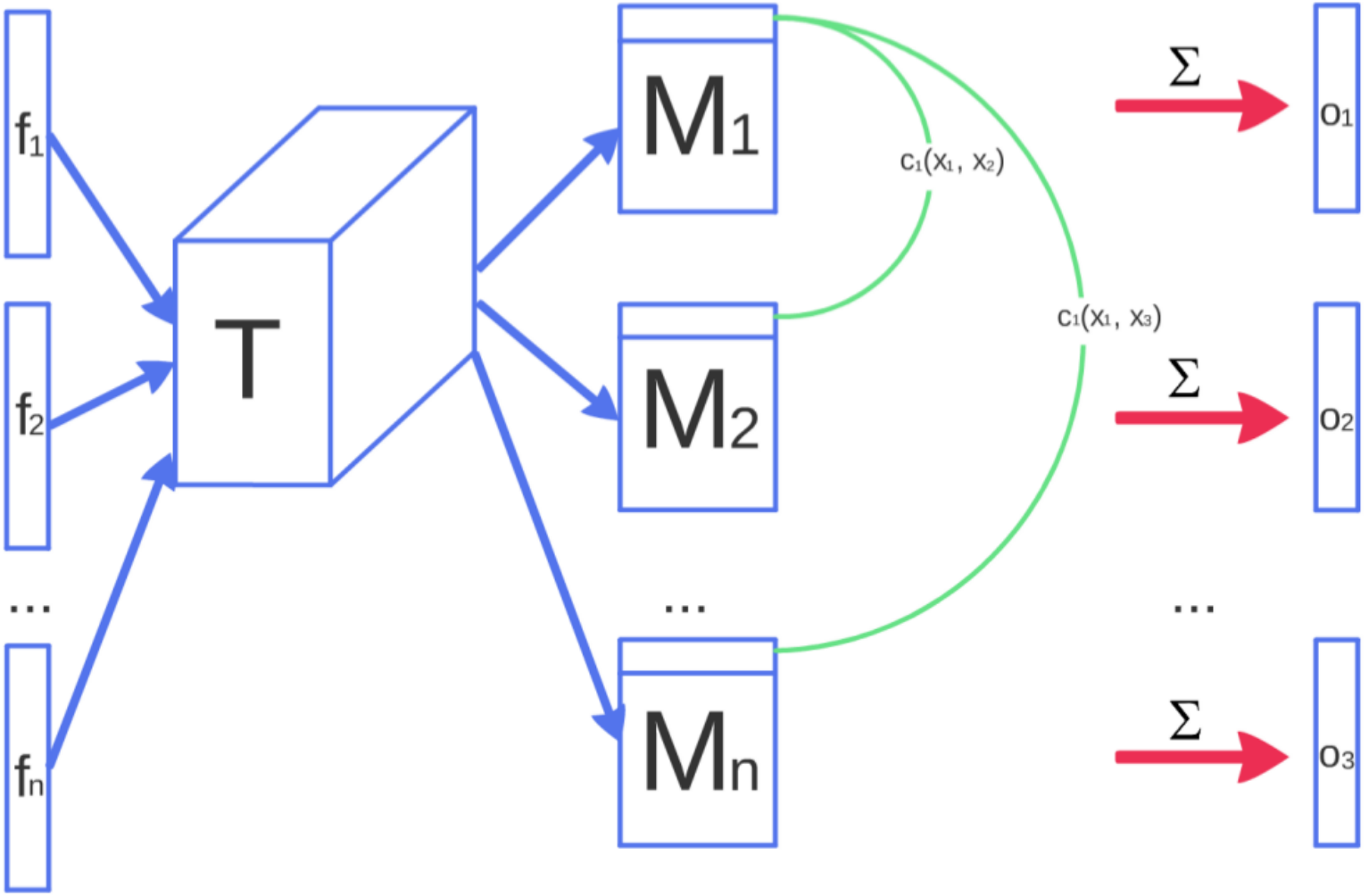


Figure 1 of paper "Improved Techniques for Training GANs", https://arxiv.org/abs/1606.03498.

Wasserstein GAN

Instead of minimizing JS divergence

$$JS(p,q) = KL(p||q) + KL(q||p),$$

Wasserstein GAN minimizes Earth-Mover distance

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

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

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

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

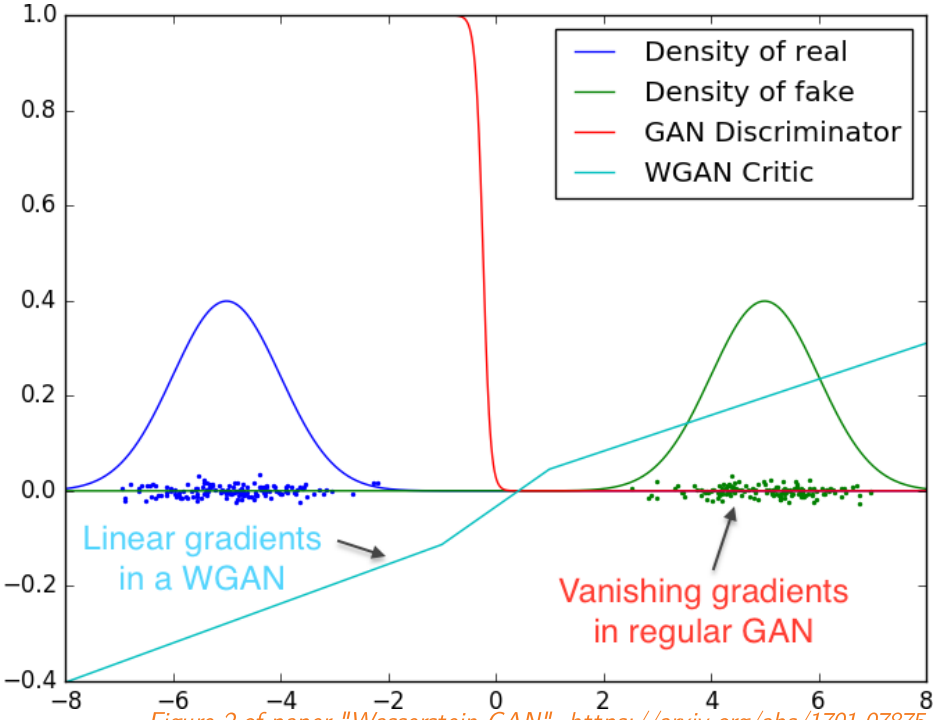


Figure 2 of paper "Wasserstein GAN", https://arxiv.org/abs/1701.07875.

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.

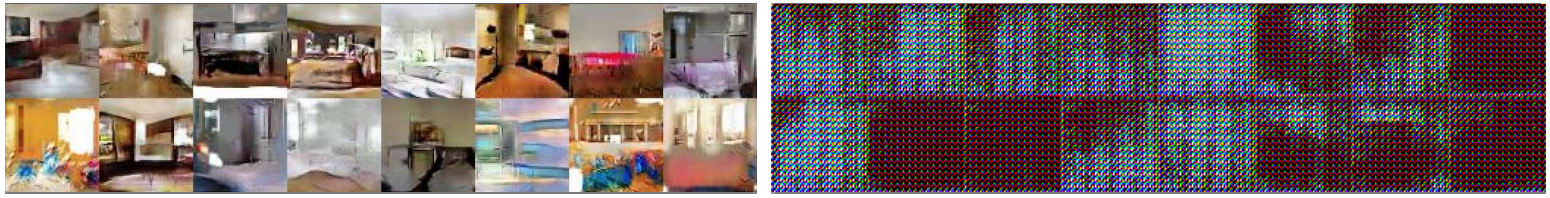


Figure 6: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in [18]). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude. 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