NPFL114, Lecture 10

Deep Generative Models

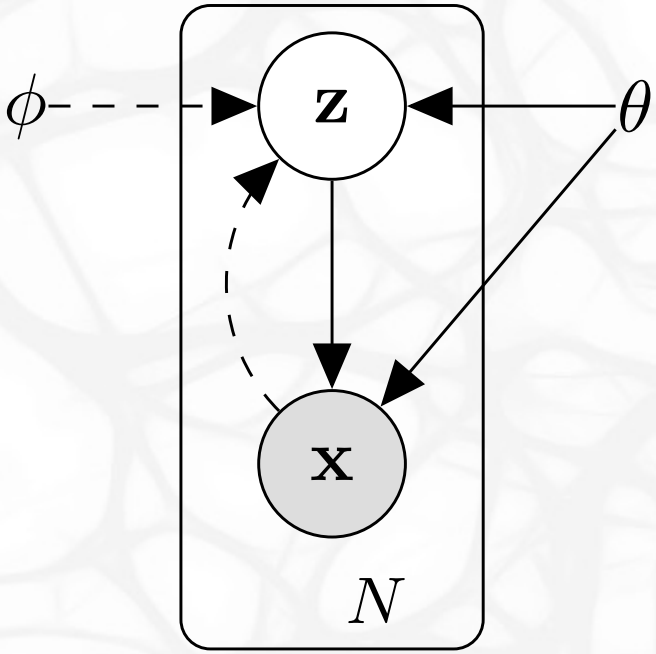
Milan Straka

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.

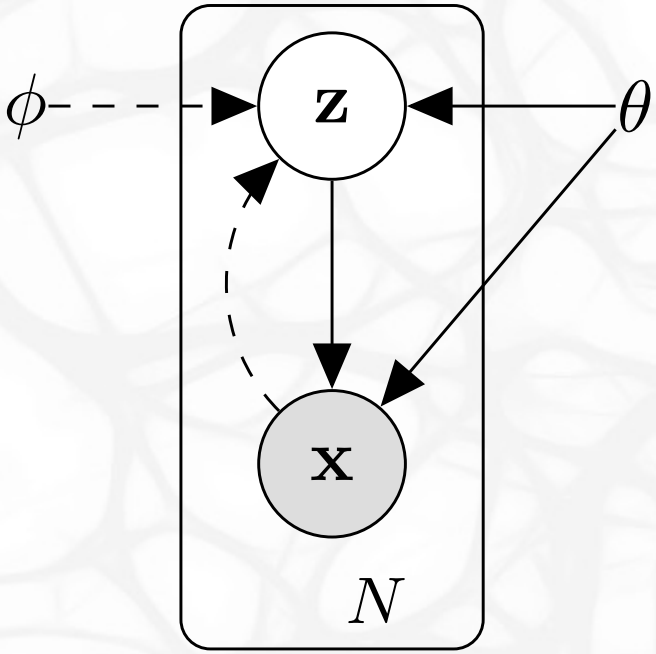
Deep Generative Models



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

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

Deep Generative Models

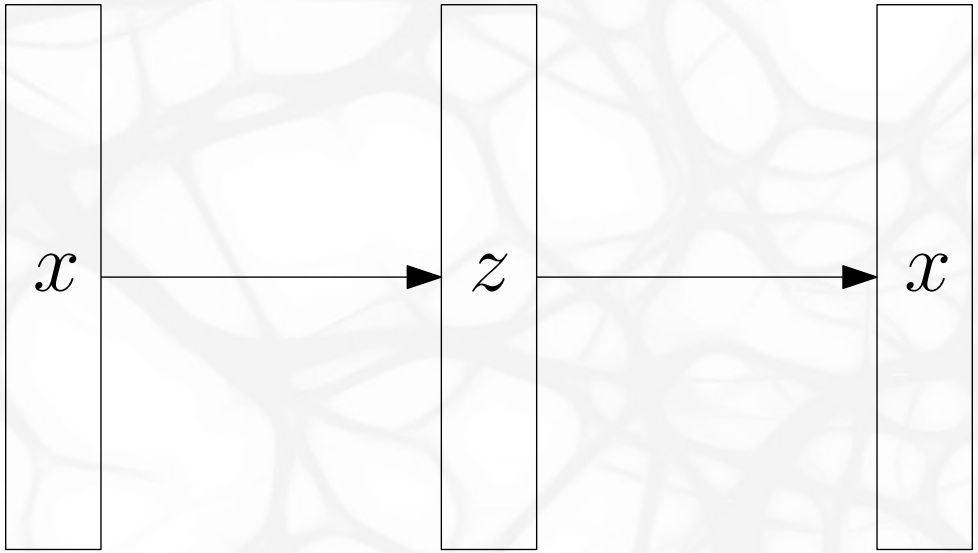


One possible approach to estimate $P(\boldsymbol{x})$ is to assume that the random variable $\mathbf{x}$ depends on a *latent variable* $\mathbf{z}$:

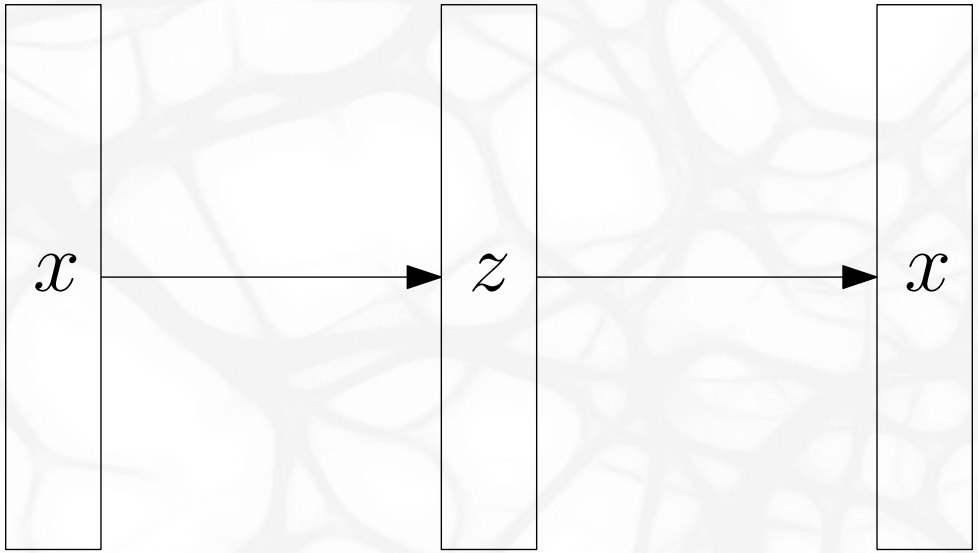
$$P(\boldsymbol{x}) = P(\boldsymbol{z})P(\boldsymbol{x}|\boldsymbol{z}).$$

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

AutoEncoders



AutoEncoders



- unsupervised feature extraction
- input compression
- when $x + \varepsilon$ is used as input, autoencoders can perform denoising

Variational AutoEncoders

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

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

Variational AutoEncoders

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

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

We will 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

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

Variational AutoEncoders

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

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

It can then be checked that

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

where the last inequality follows from KL-divergence being non-negative.

Variational AutoEncoders

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

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

It can then be checked that

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

where the last inequality follows from KL-divergence being non-negative.

The variational lower bound can also be written as

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

Variational AutoEncoders

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

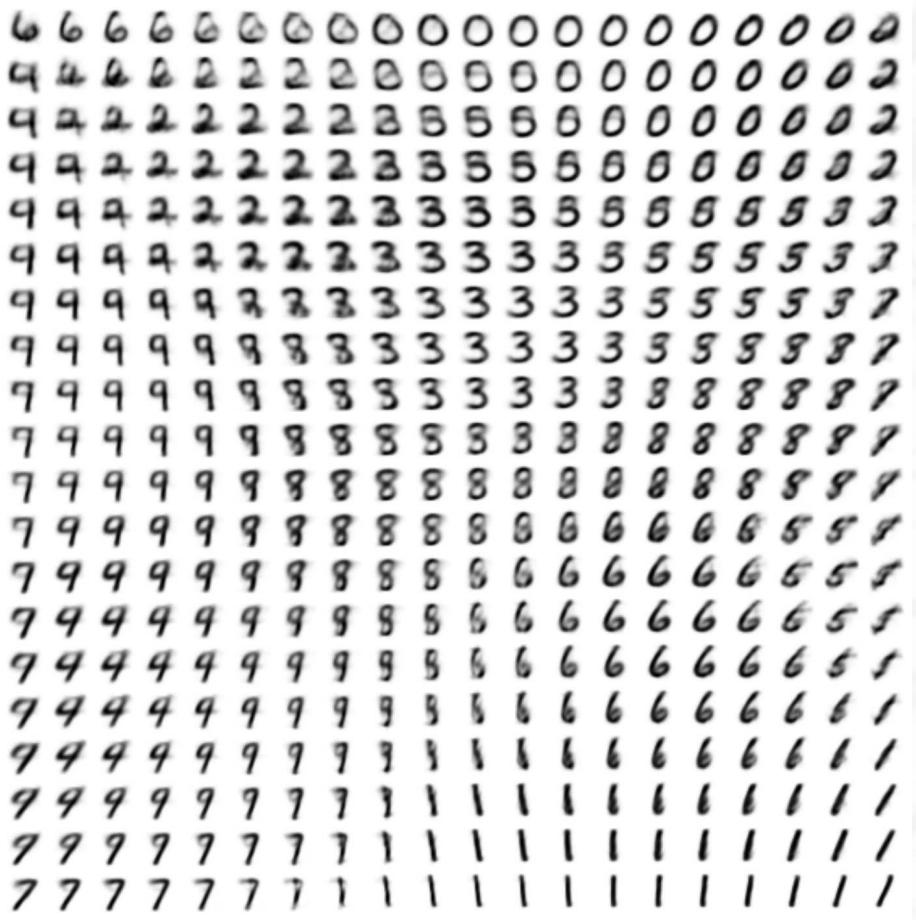
$$\boldsymbol{z} \sim \mathcal{N}(\mu, \sigma^2),$$

we can write $\boldsymbol{z}$ as

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

Variational AutoEncoders

(a) Learned Frey Face manifold

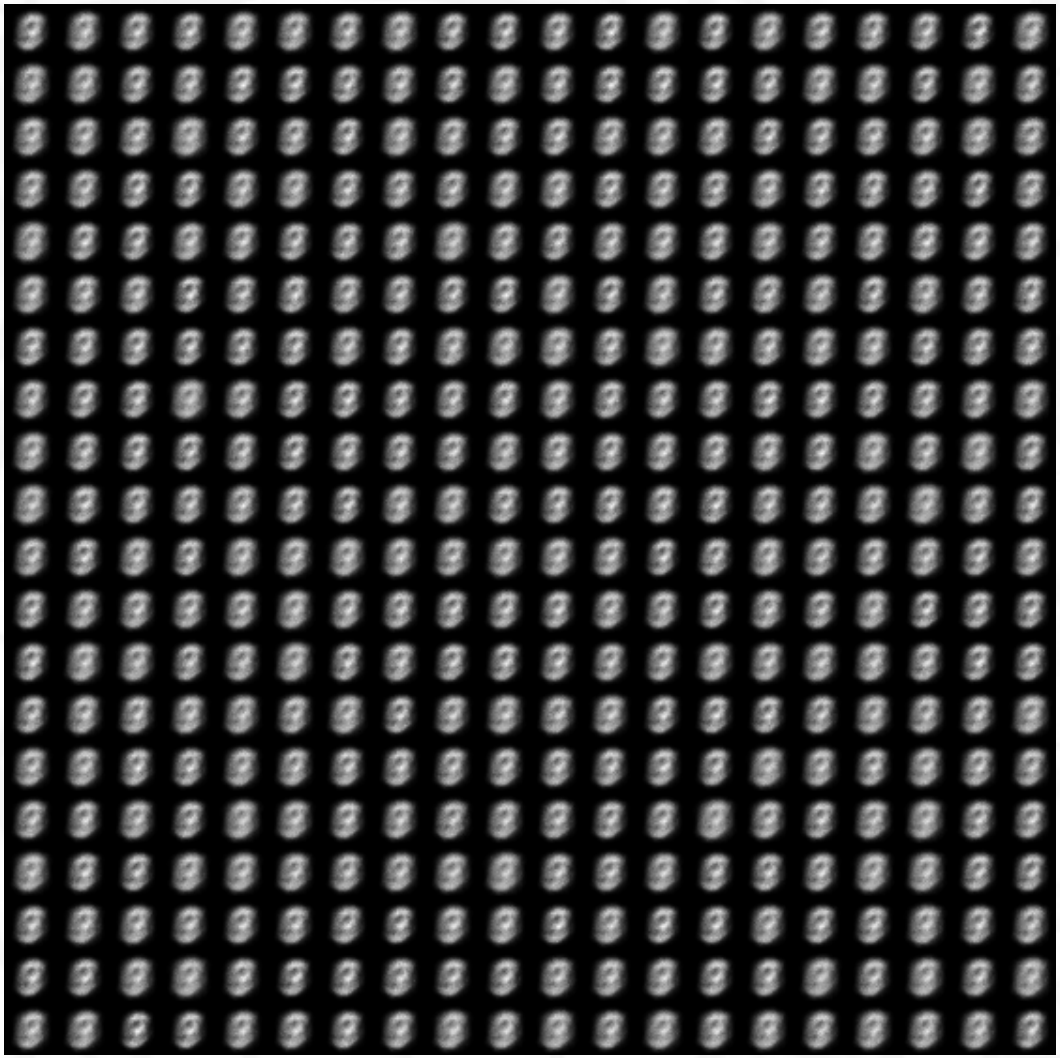


(b) Learned MNIST manifold

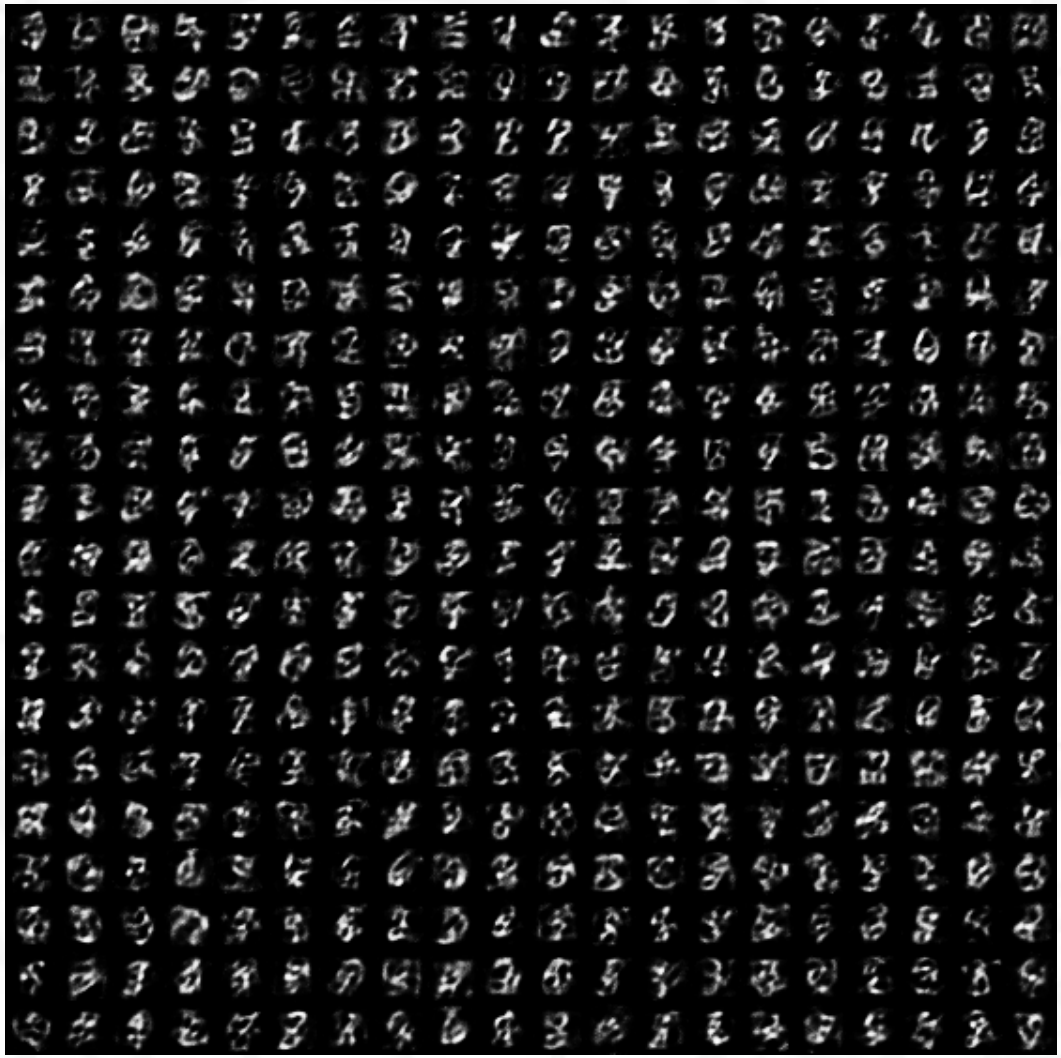
Variational AutoEncoders

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

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 $G(\boldsymbol{z}; \boldsymbol{\theta}_g)$.

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

Generative Adversarial Networks

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

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

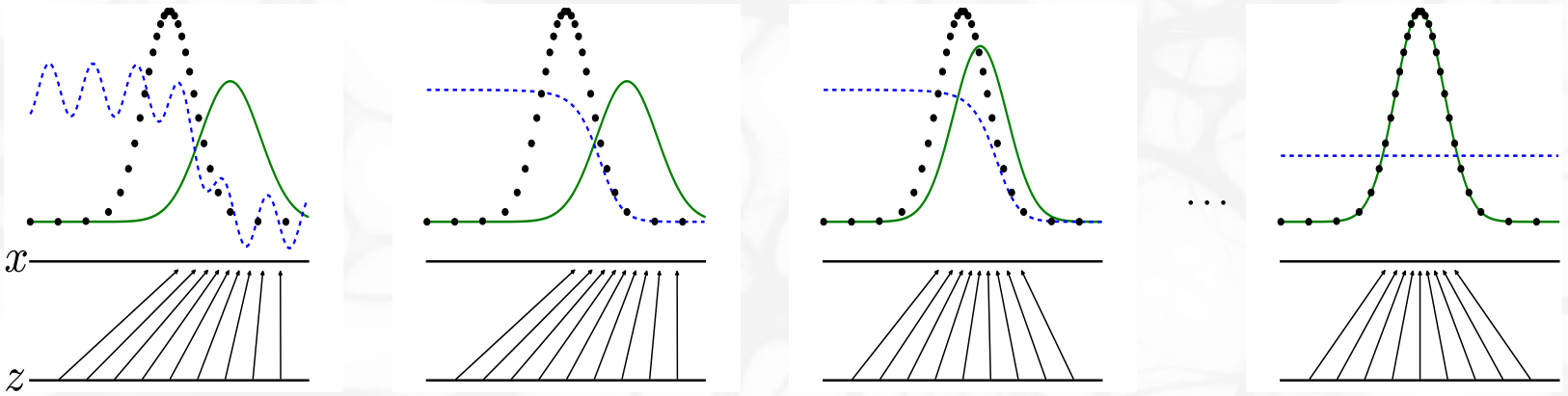
Then we have a *discriminator*, which given data $\boldsymbol{x}$ generates a probability whether $\boldsymbol{x}$ comes from real data or is generated by a generator.

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

In other words, D and G play the following game:

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

Generative Adversarial Networks



Generative Adversarial Networks

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

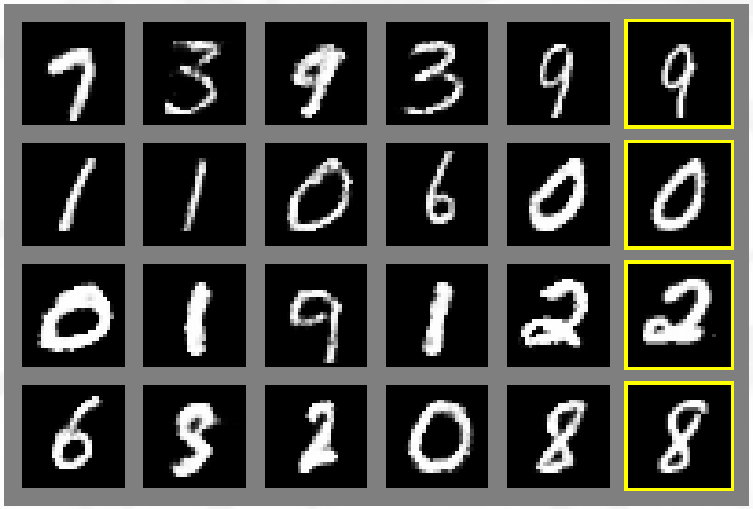
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

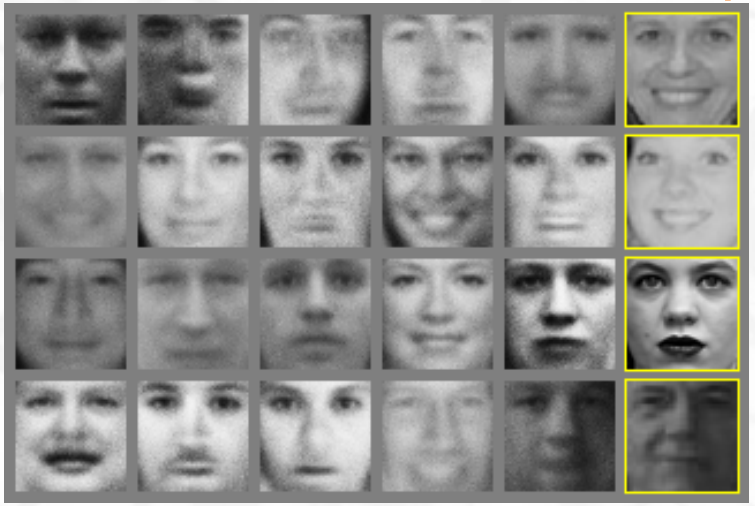
end for

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

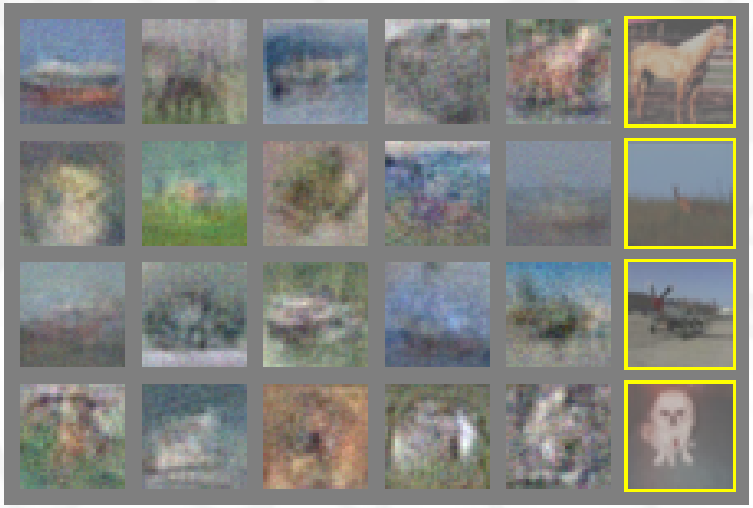
Generative Adversarial Networks



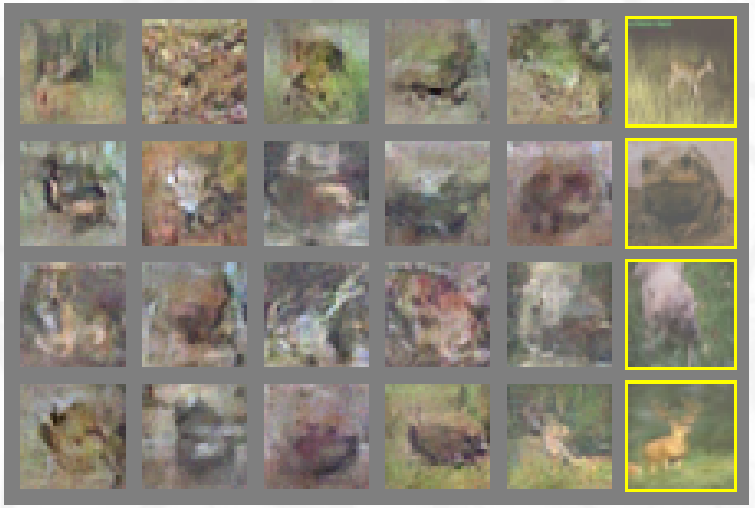
a)



b)

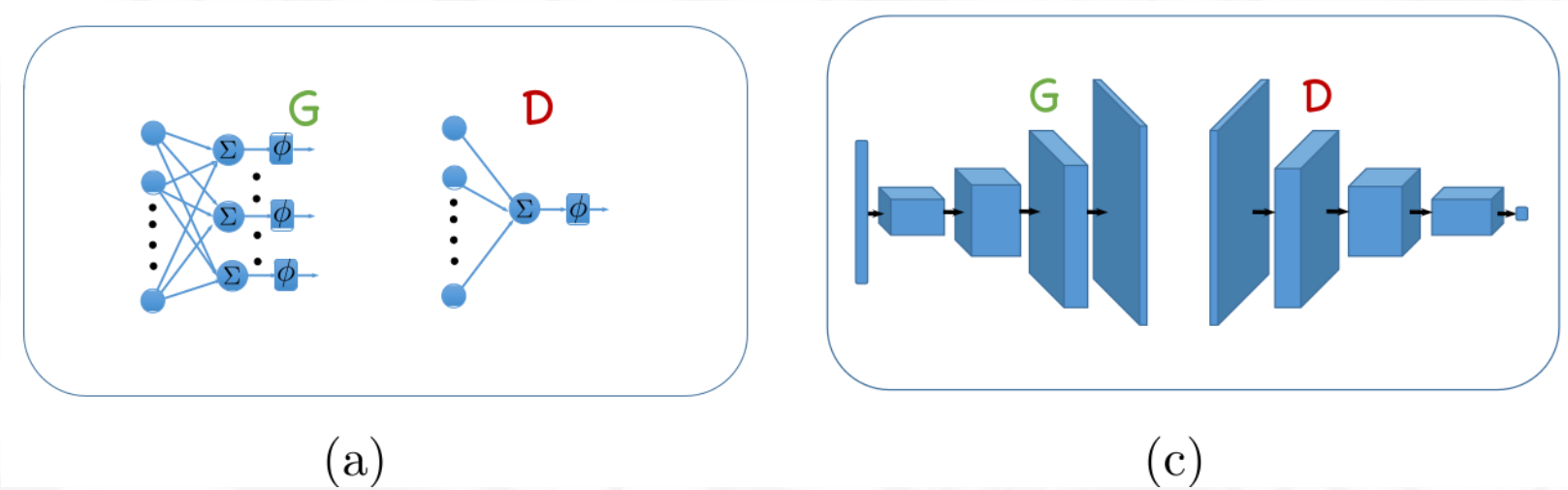


c)



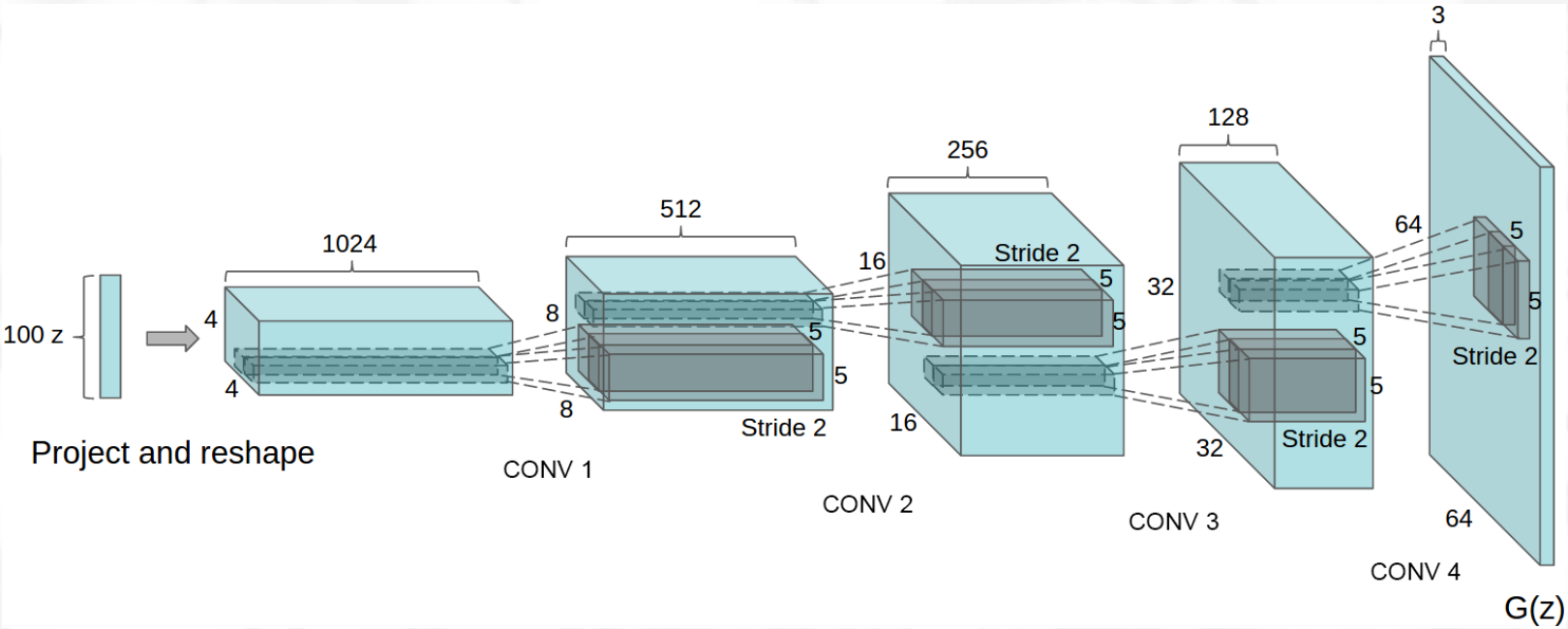
d)

Deep Convolutional GAN



(a) (c)

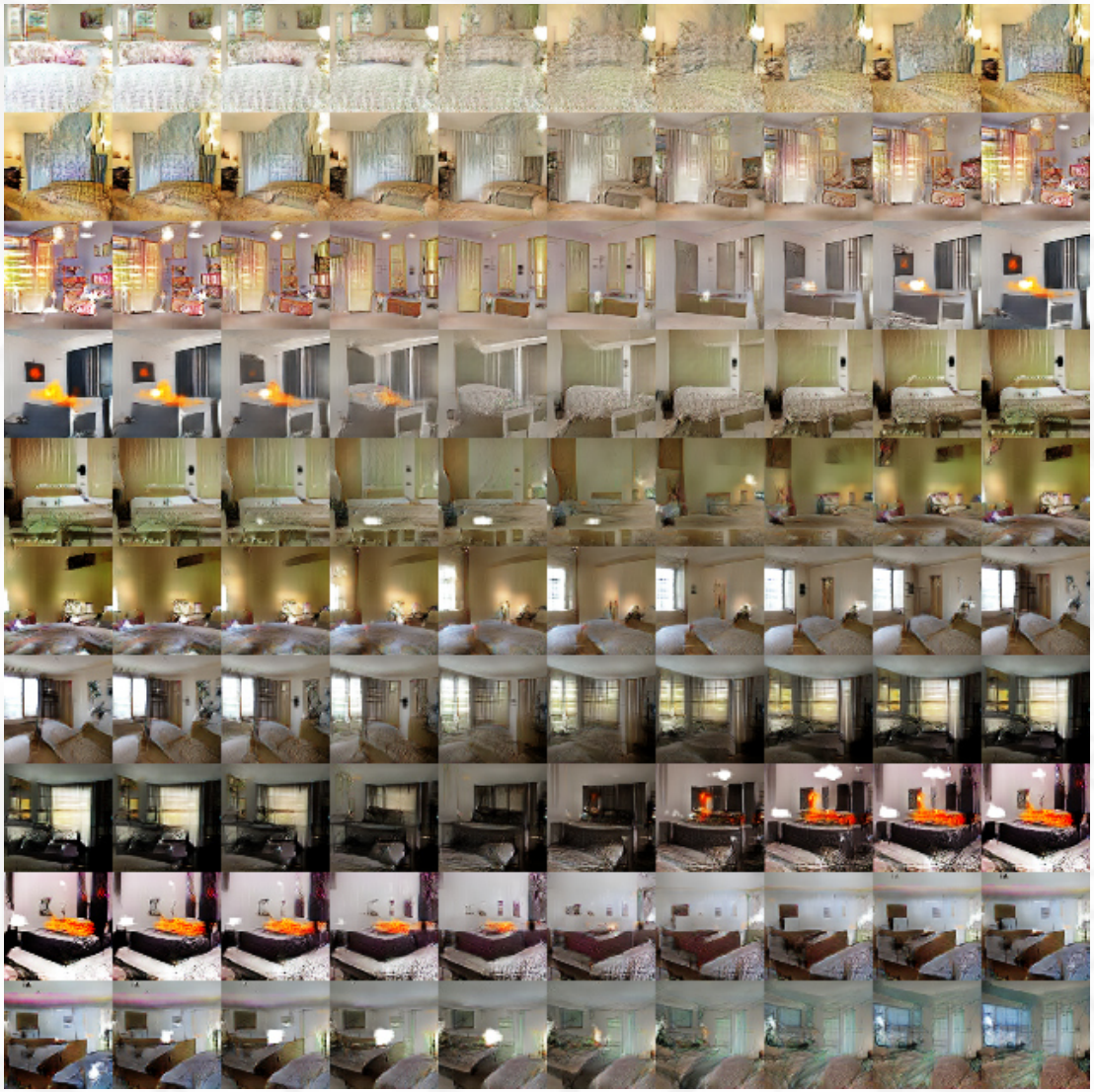
Deep Convolutional GAN



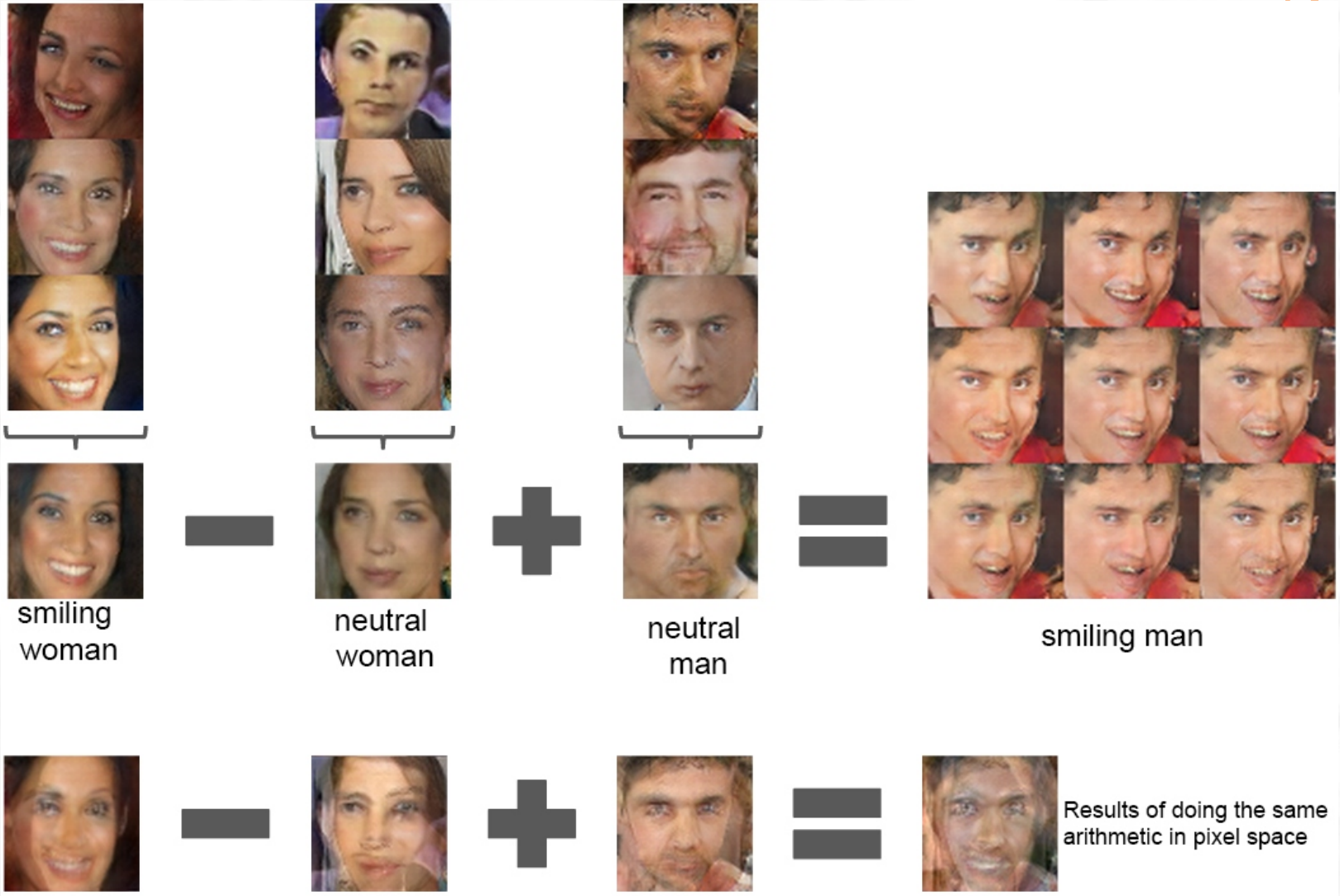
Deep Convolutional GAN

Deep Convolutional GAN

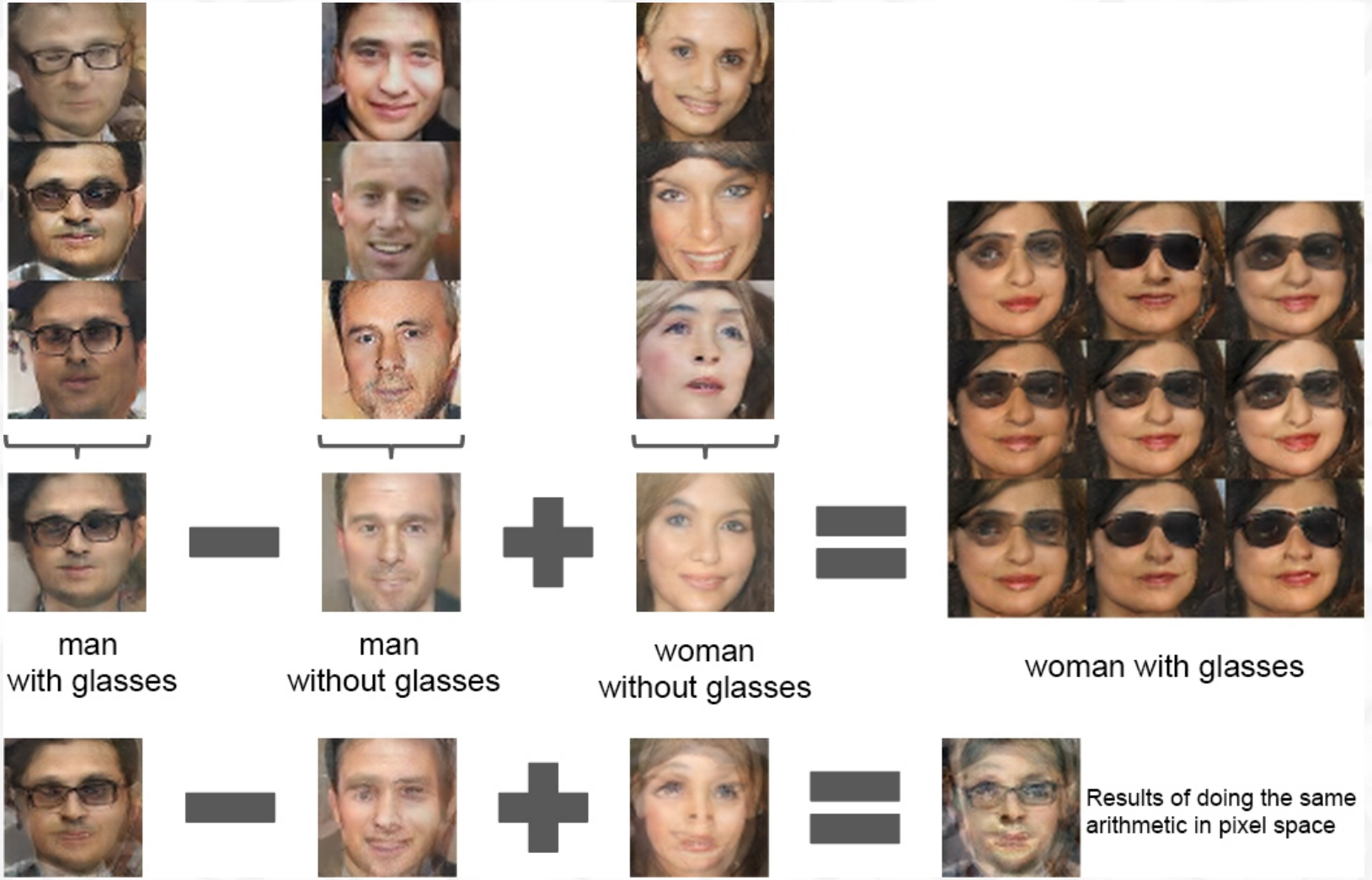
Deep Convolutional GAN



Deep Convolutional GAN



Deep Convolutional GAN

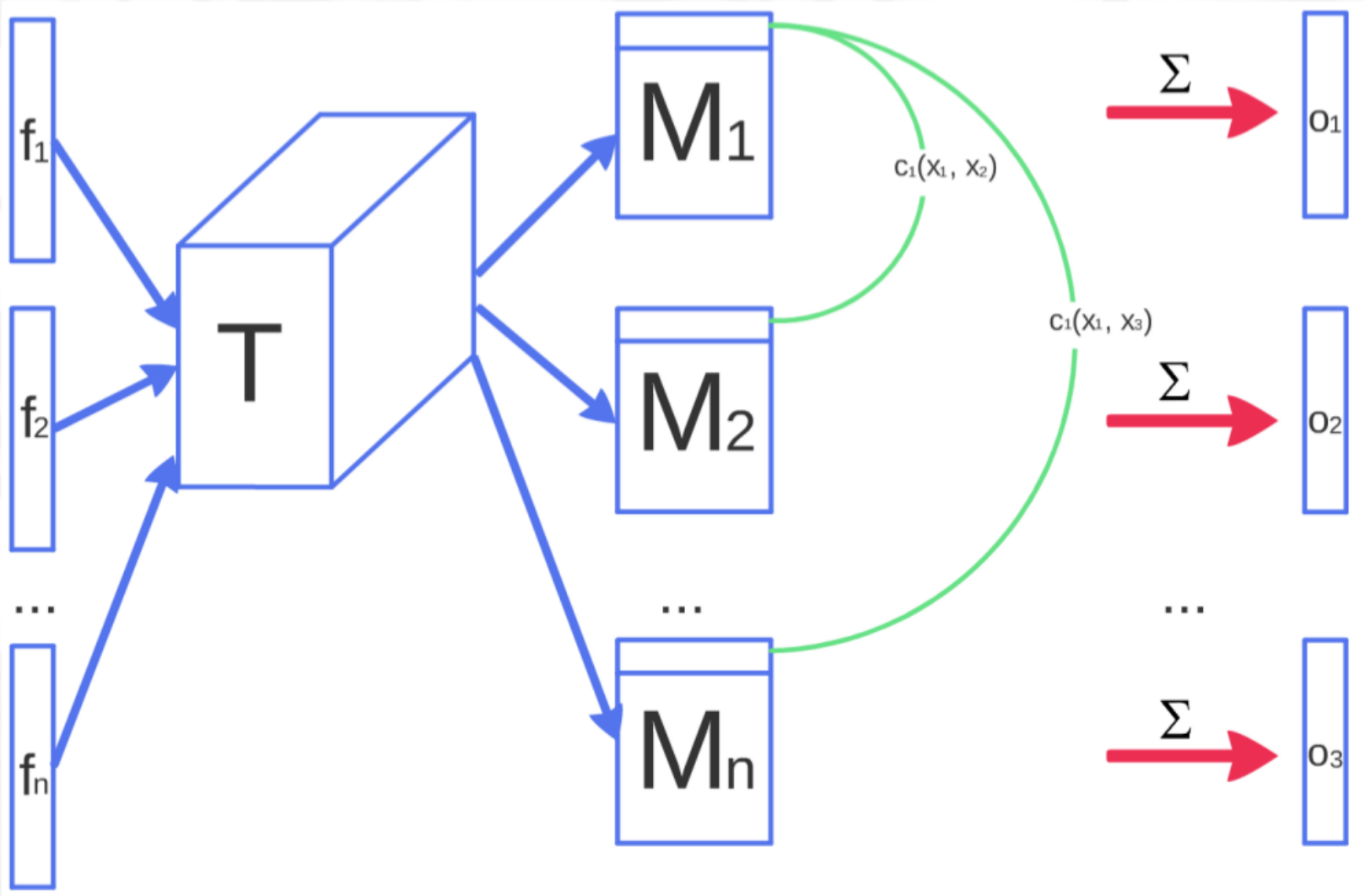


Deep Convolutional GAN

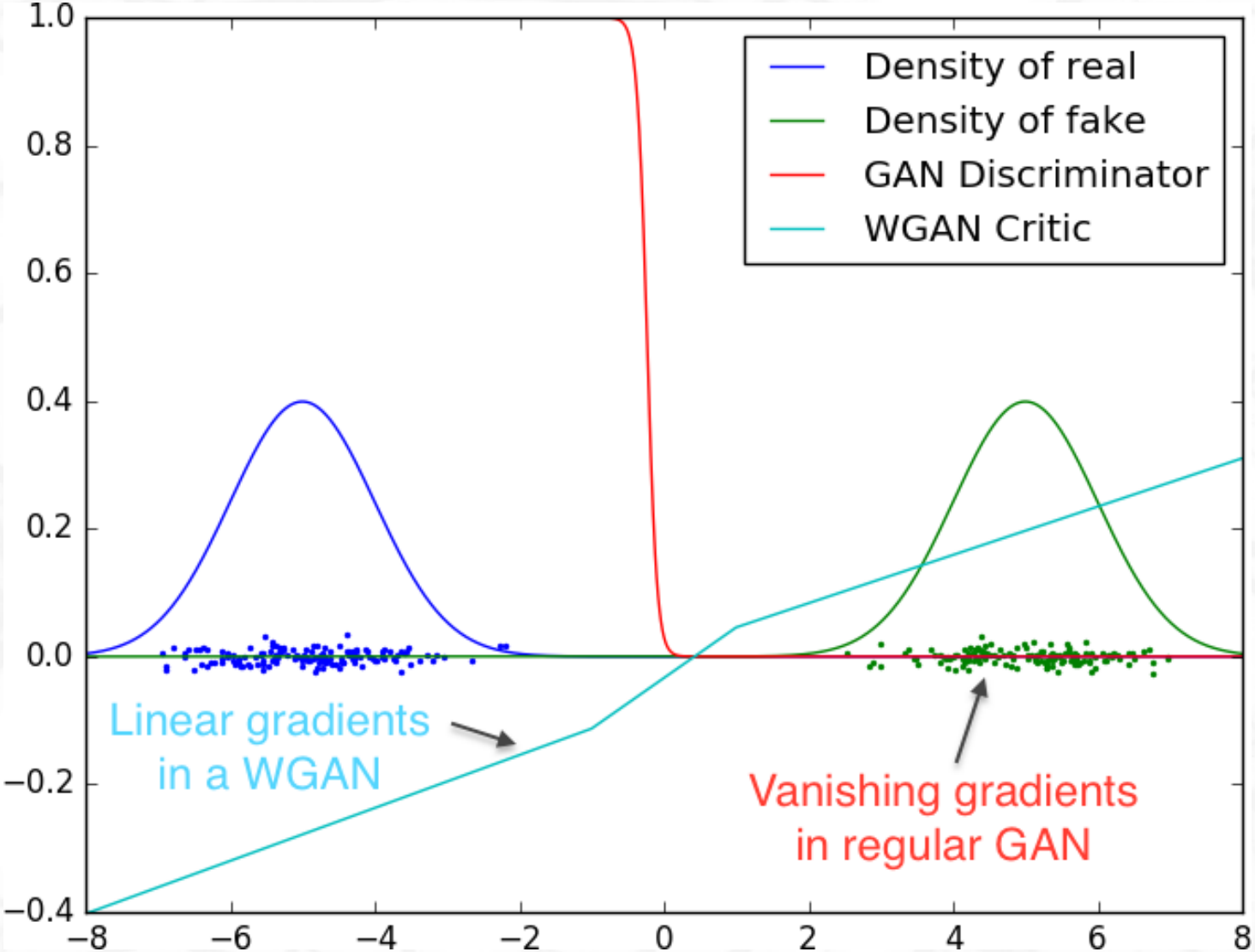
GANs are Problematic to Train

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

Minibatch Discrimination



Wasserstein GAN



Wasserstein GAN

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

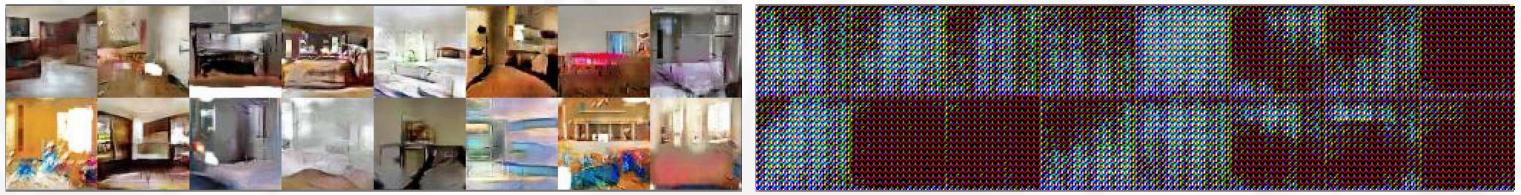


Figure 6: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in [18]). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude.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

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.