

Deep Generative Models

Milan Straka

 May 2, 2022



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



unless otherwise stated

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

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

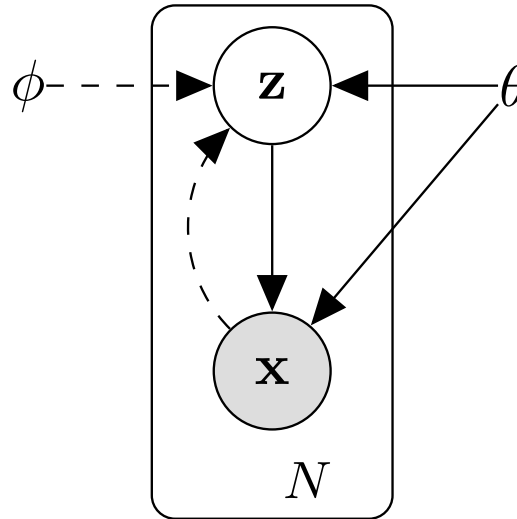
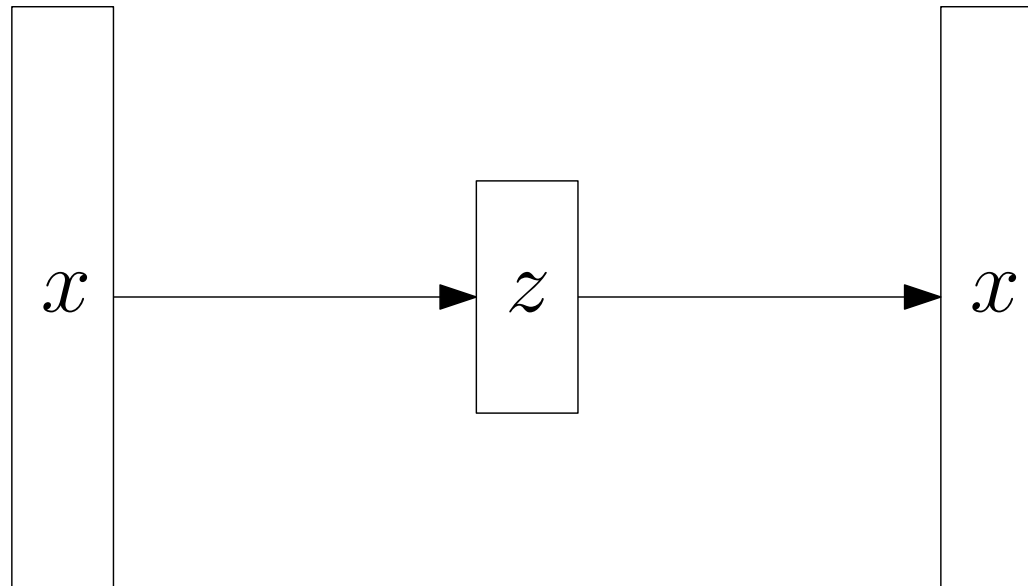


Figure 1 of "Auto-Encoding Variational Bayes", <https://arxiv.org/abs/1312.6114>

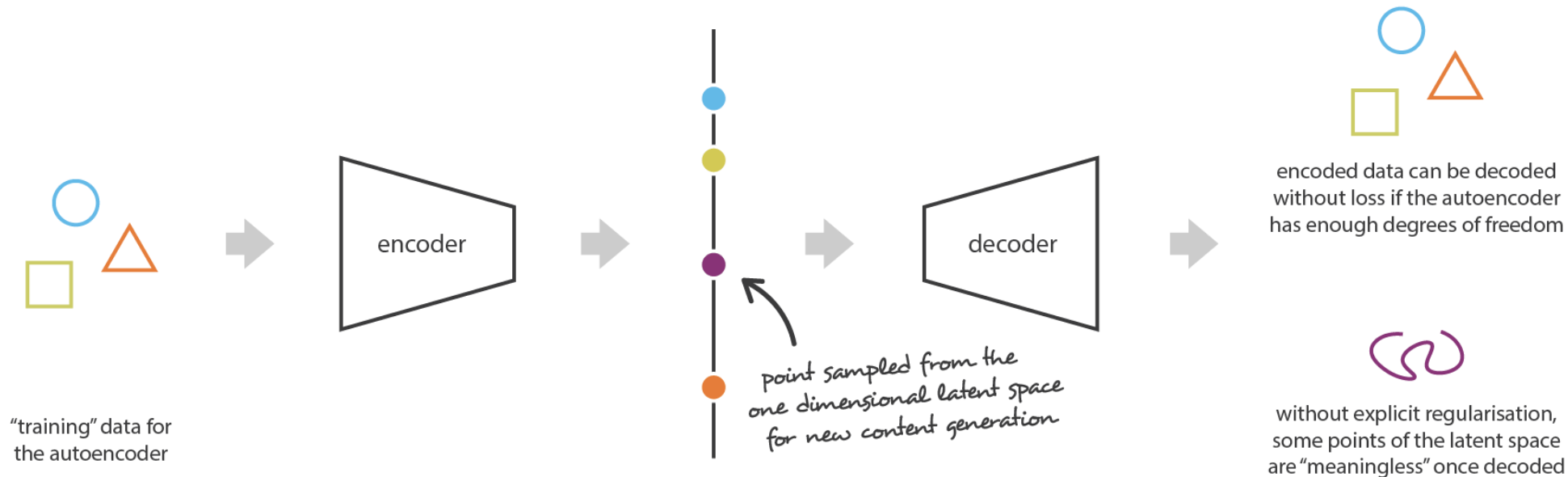
One possible approach to estimate $P(\mathbf{x})$ is to assume that the random variable \mathbf{x} depends on a **latent variable \mathbf{z}** :

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

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



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



https://miro.medium.com/max/3608/1*iSfaVxcGi_ELkKgAG0YRIQ@2x.png

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

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

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

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

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

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

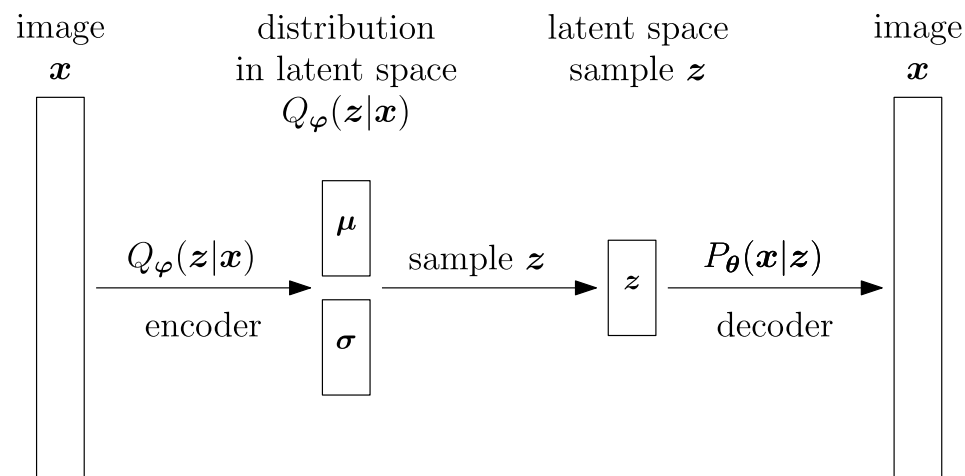
By using simple properties of conditional and joint probability, we get that

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

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

- We train a VAE by minimizing $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}; \mathbf{x})$.
- The distribution $Q_{\boldsymbol{\varphi}}(\mathbf{z}|\mathbf{x})$ is parametrized as a normal distribution $\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, with the model predicting $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ given \mathbf{x} .
 - In order for $\boldsymbol{\sigma}$ to be positive, we can use `exp` activation function (so that the network predicts $\log \boldsymbol{\sigma}$ before the activation), or for example a `softplus` activation function.
 - 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 $Q_{\boldsymbol{\varphi}}(\mathbf{z}|\mathbf{x})$ using mean and variance, the maximum entropy principle suggests we should use the normal distribution.
- The $\mathbb{E}_{Q_{\boldsymbol{\varphi}}(\mathbf{z}|\mathbf{x})}$ is estimated using a single sample.
- We use a prior $P(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

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



Note that the loss has 2 intuitive components:

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

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

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

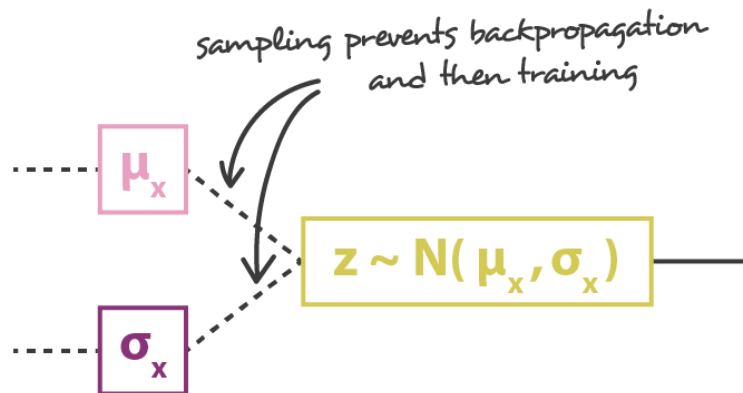
we can write \mathbf{z} as

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

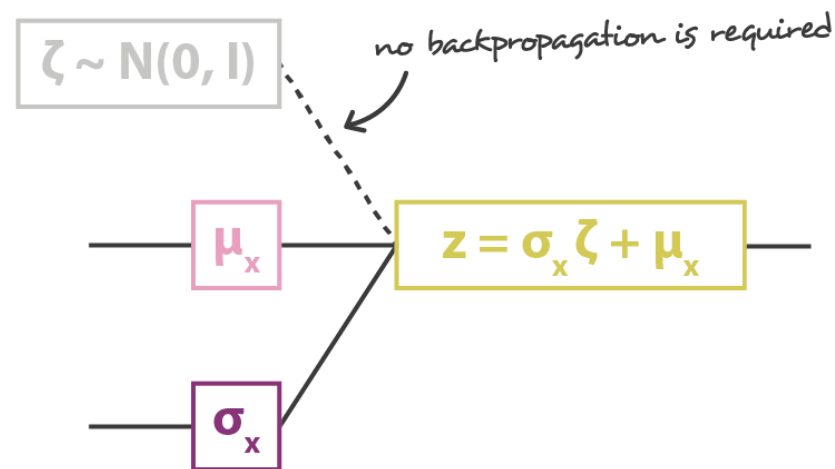
Such formulation then allows differentiating \mathbf{z} with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and is called a **reparametrization trick** (Kingma and Welling, 2013).

—— no problem for backpropagation

..... backpropagation is not possible due to sampling

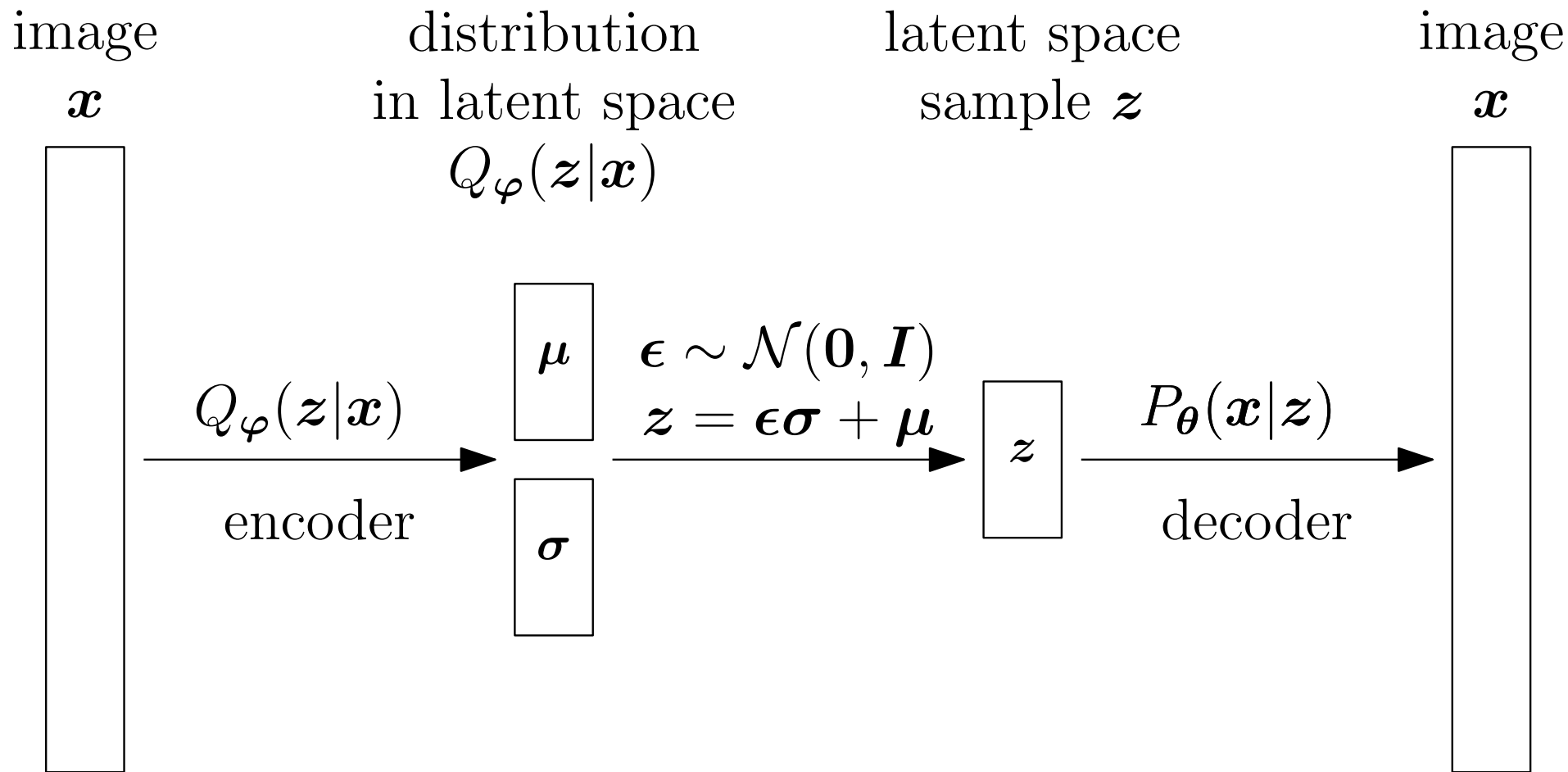


sampling without reparametrization trick



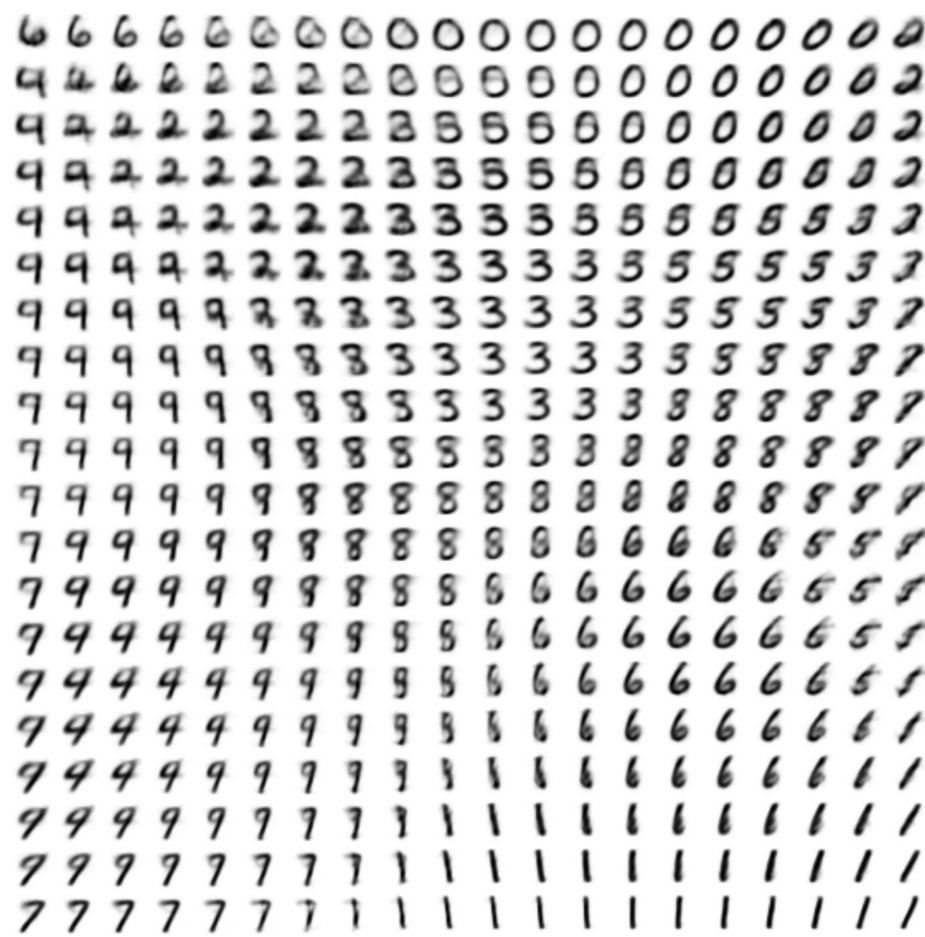
sampling with reparametrization trick

https://miro.medium.com/max/3704/1*S8CoO3TGtFBpzv8GvmgKeg@2x.png





(a) Learned Frey Face manifold



(b) Learned MNIST manifold

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



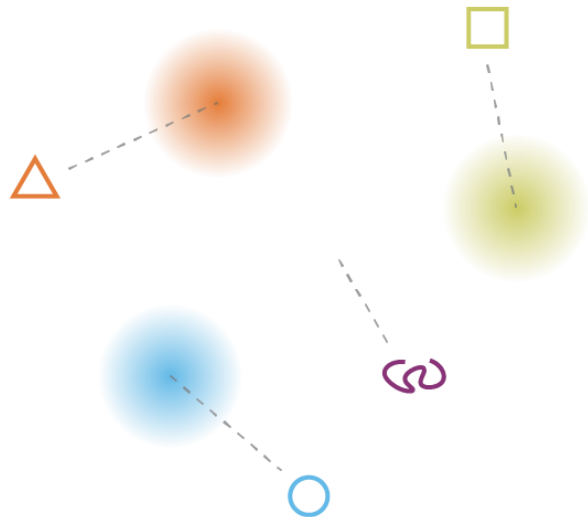
(a) 2-D latent space

(b) 5-D latent space

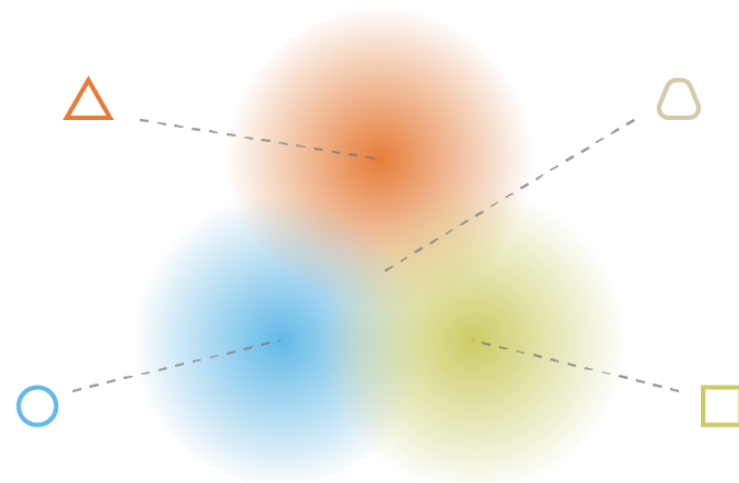
(c) 10-D latent space

(d) 20-D latent space

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

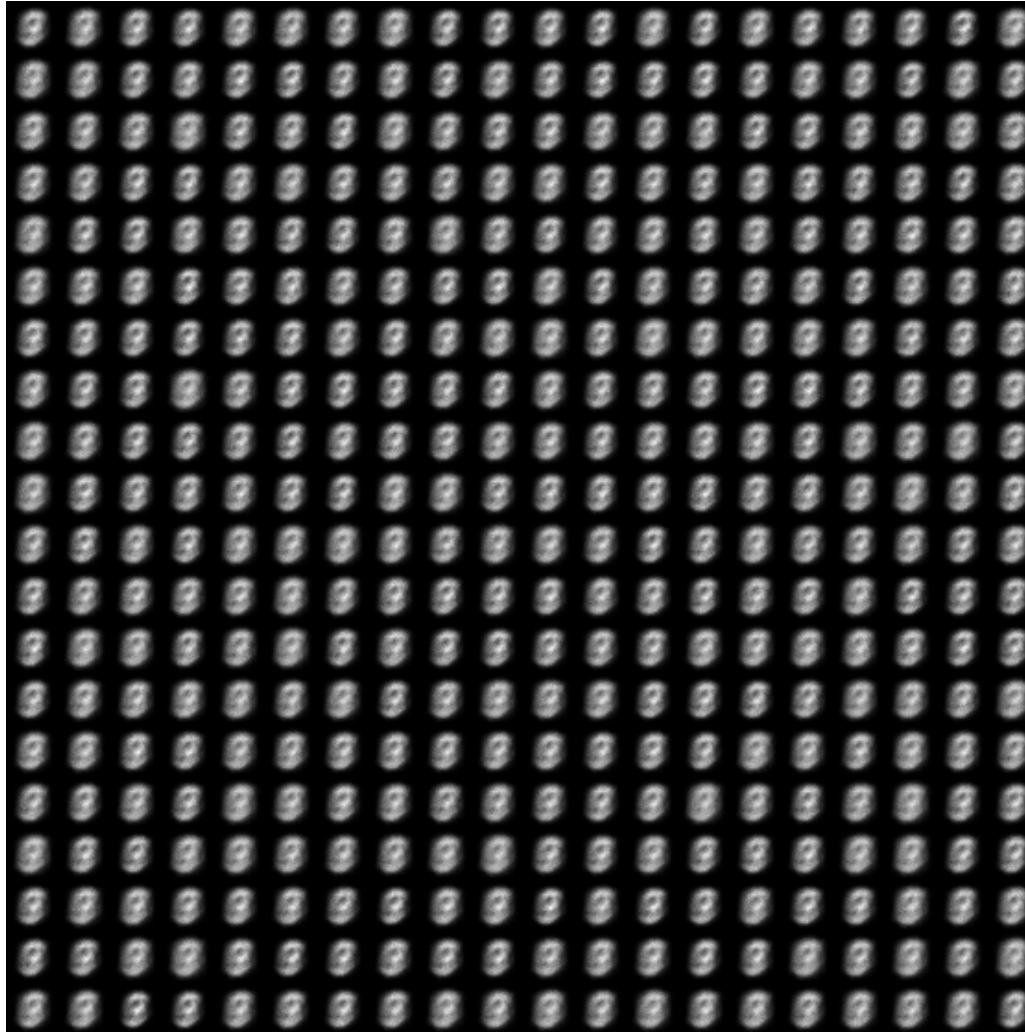


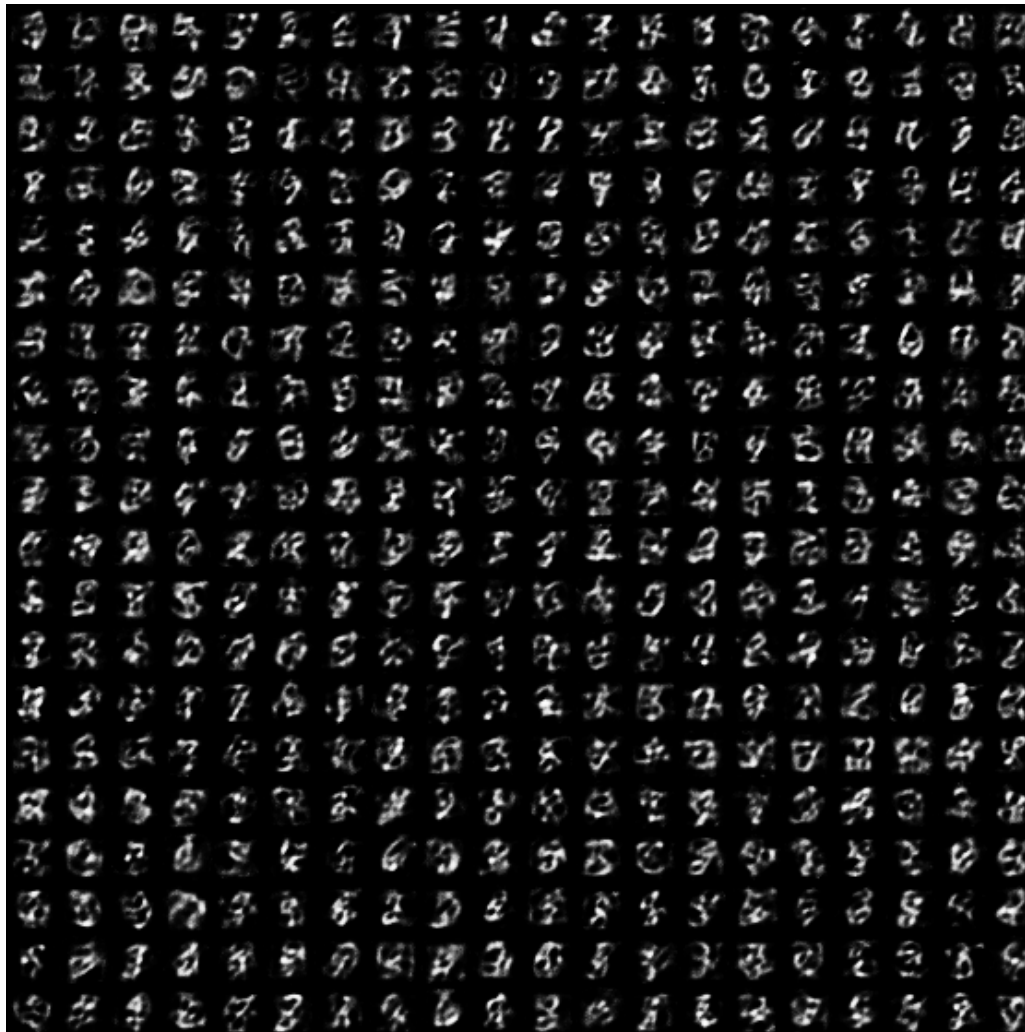
what can happen without regularisation



what we want to obtain with regularisation

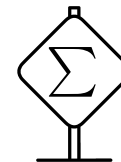
https://miro.medium.com/max/3742/1*9ouOKh2w-b3NNOVx4Mw9bg@2x.png





There is also another way to arrive at the variational lower bound.

First remember that Jensen's inequality states that for a concave function f (like \log),



$$f(\mathbb{E}[\mathbf{x}]) \geq \mathbb{E}[f(\mathbf{x})].$$

Starting with the log probability, we proceed as follows:

$$\begin{aligned} \log P(\mathbf{x}) &= \log \mathbb{E}_{P(\mathbf{z})} [P(\mathbf{x}|\mathbf{z})] \\ &= \log \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})} \left[P(\mathbf{x}|\mathbf{z}) \cdot \frac{P(\mathbf{z})}{Q(\mathbf{z}|\mathbf{x})} \right] \\ &\geq \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})} \left[\log P(\mathbf{x}|\mathbf{z}) + \log \frac{P(\mathbf{z})}{Q(\mathbf{z}|\mathbf{x})} \right] \\ &= \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})} [\log P(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(Q(\mathbf{z}|\mathbf{x}) \| P(\mathbf{z})) \end{aligned}$$

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

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

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

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

The discriminator and generator play the following game:

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

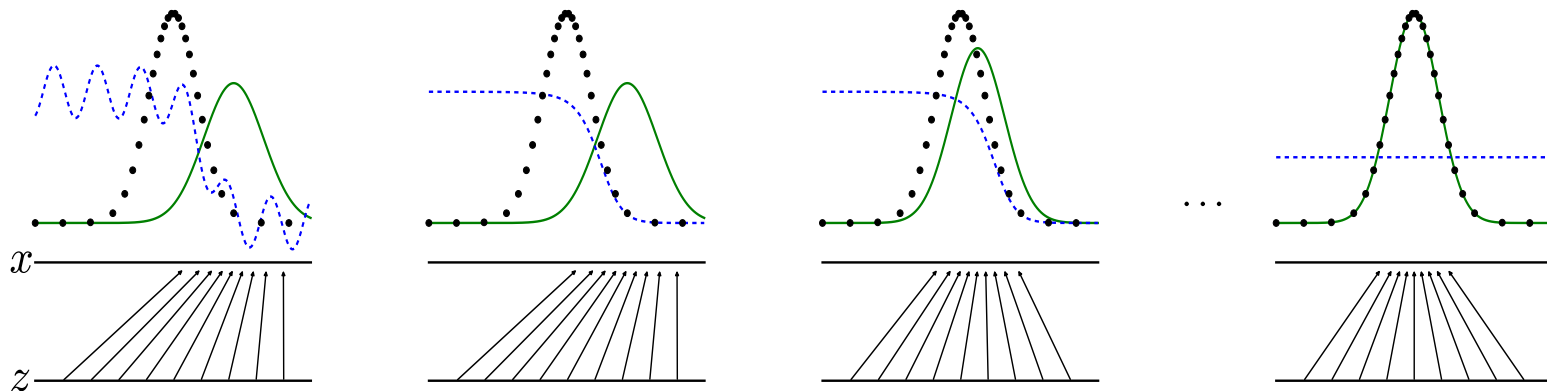


Figure 1 of "Generative Adversarial Nets", <https://arxiv.org/abs/1406.2661>

The generator and discriminator are alternately trained, the discriminator by

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

and the generator by

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

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

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

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

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

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

end for

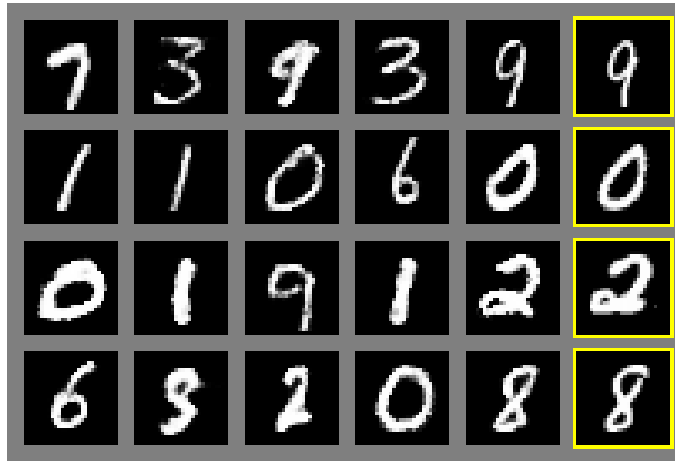
- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending its stochastic gradient:

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

end for

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

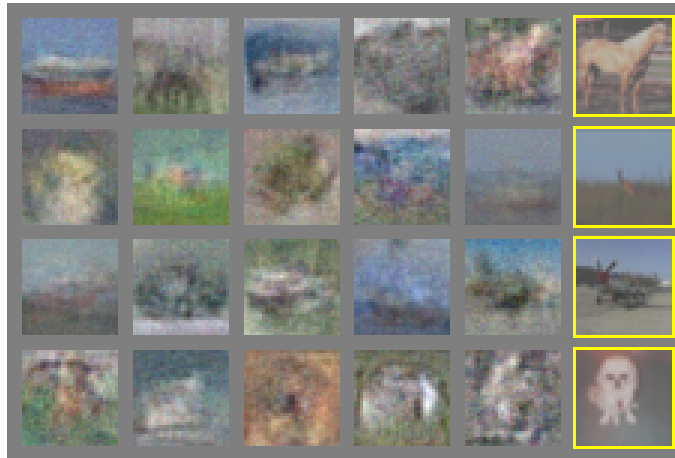
Algorithm 1 of "Generative Adversarial Nets", <https://arxiv.org/abs/1406.2661>



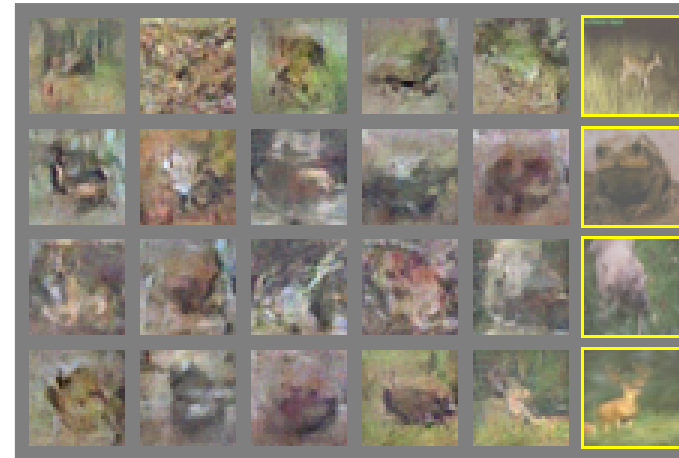
a)



b)



c)



d)

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

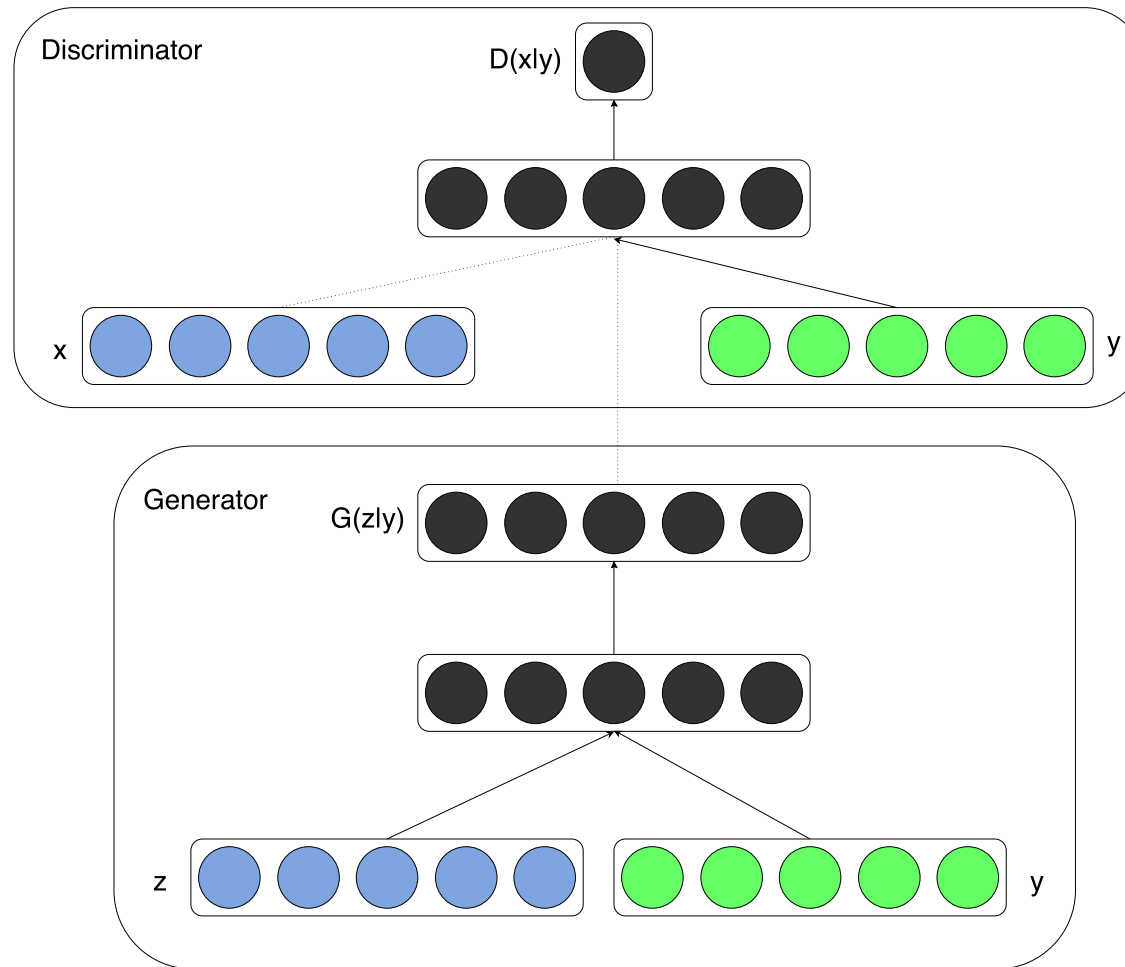
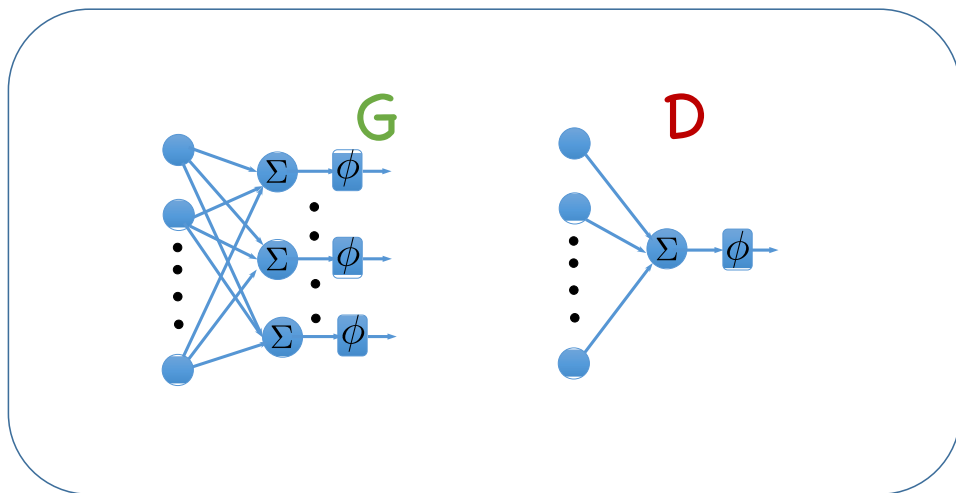


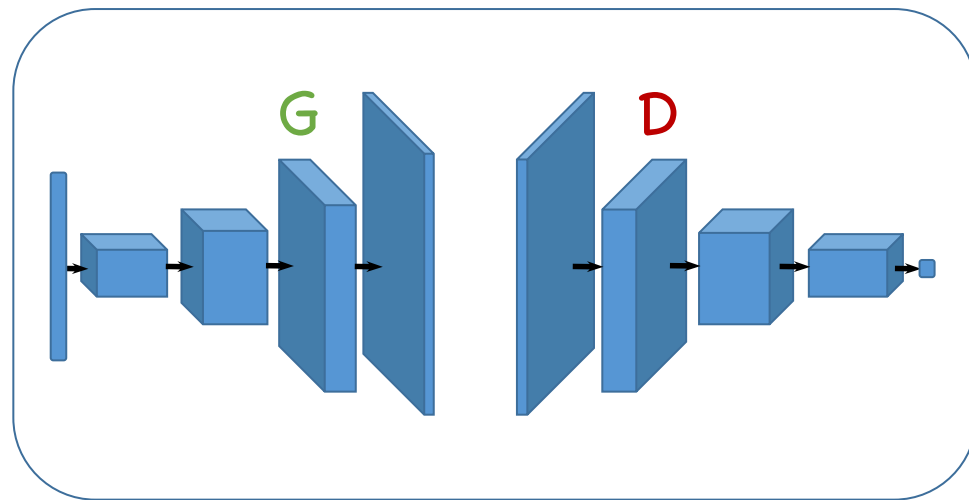
Figure 1 of "Conditional Generative Adversarial Nets", <https://arxiv.org/abs/1411.1784>

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 "An Online Learning Approach to Generative Adversarial Networks", <https://arxiv.org/abs/1706.03269>

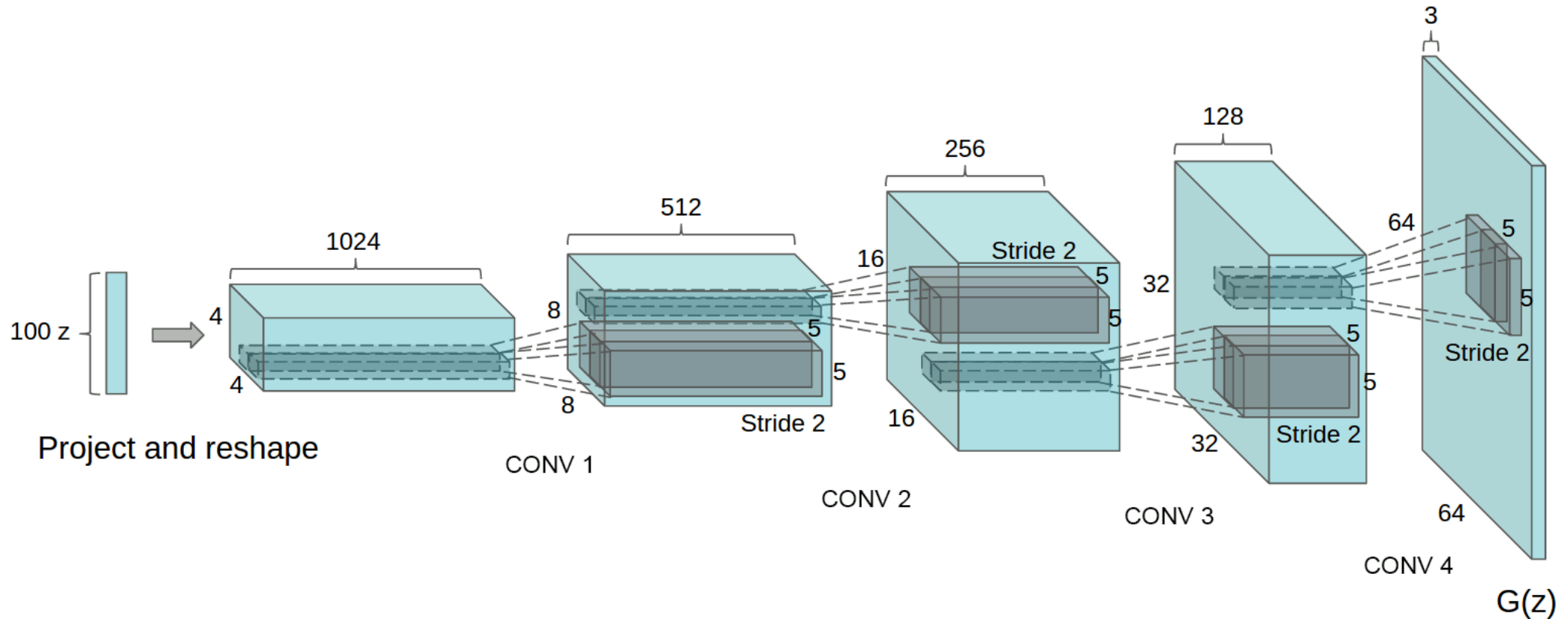


Figure 1 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>



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



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

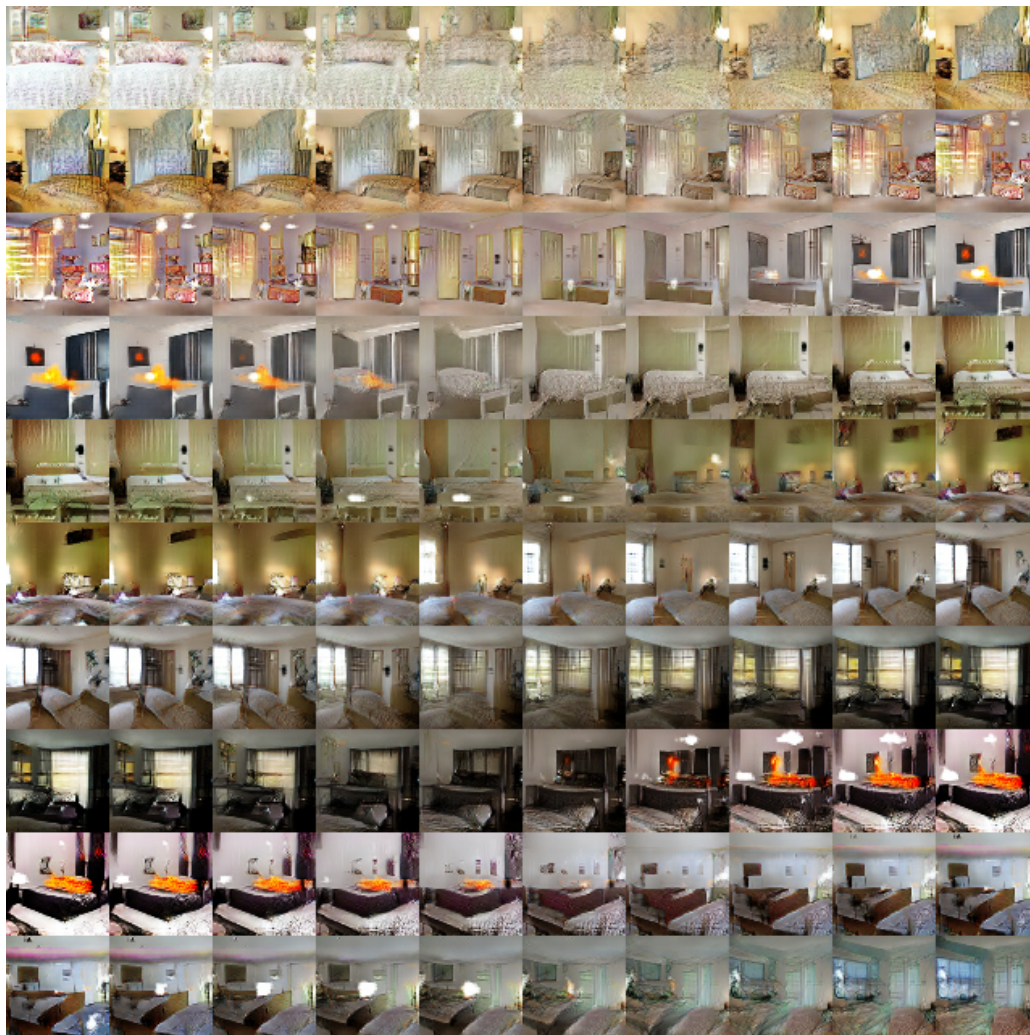


Figure 4 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>

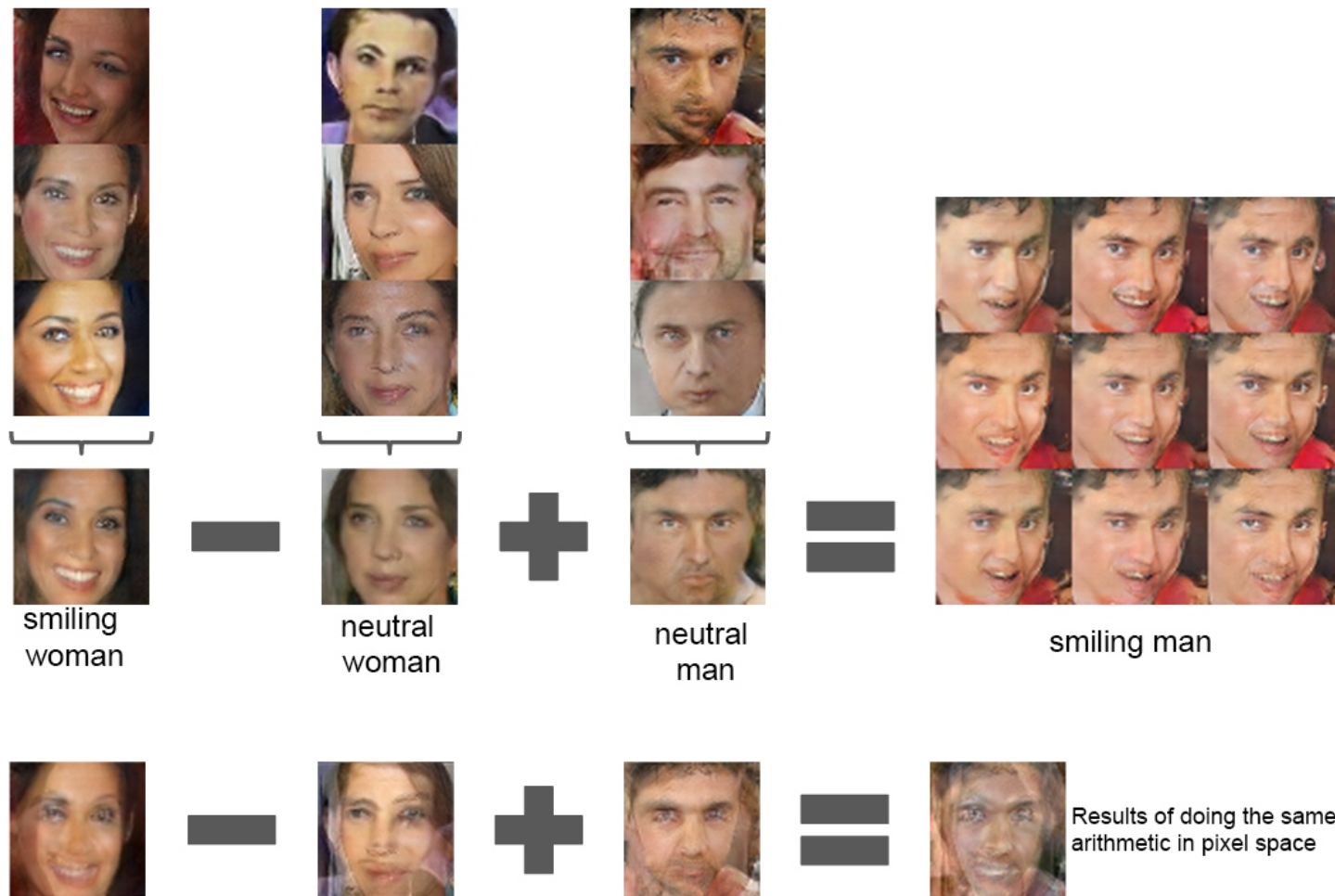


Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>

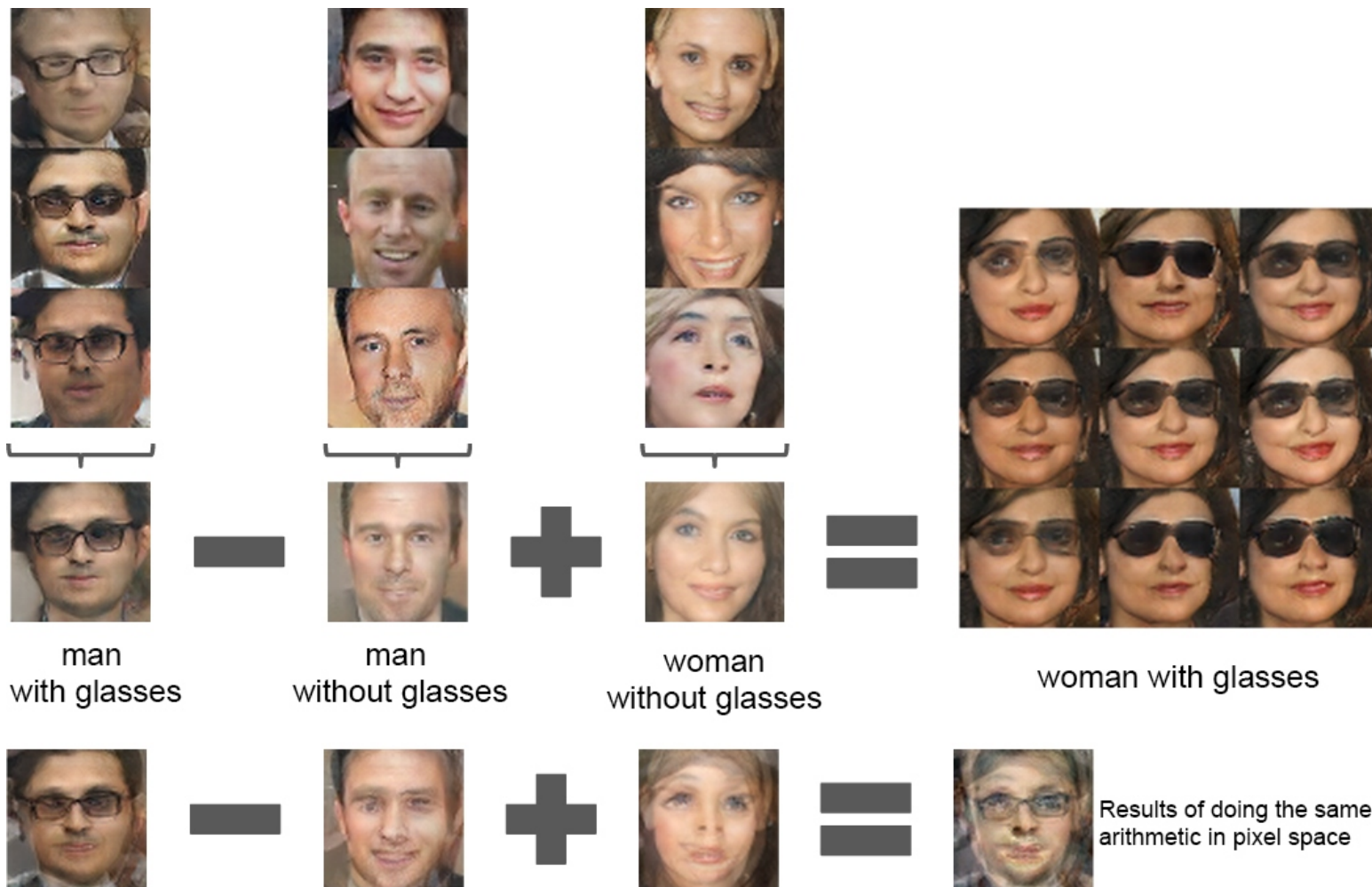


Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>



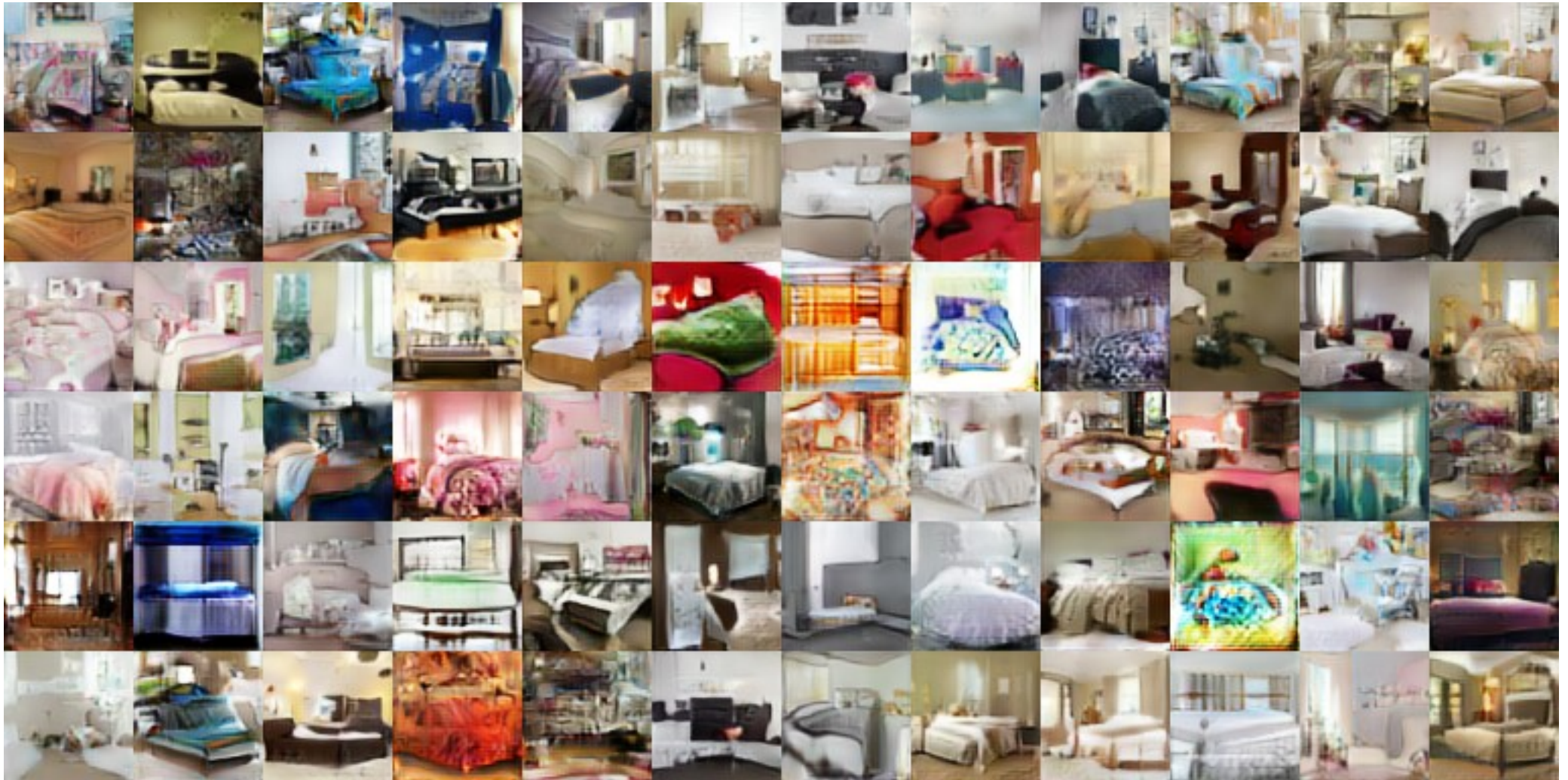
Figure 8 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>

GAN output in paper

Your GAN output



https://miro.medium.com/max/1400/1*r8cuSlAM5oHUERP01TCTxg.jpeg



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:

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

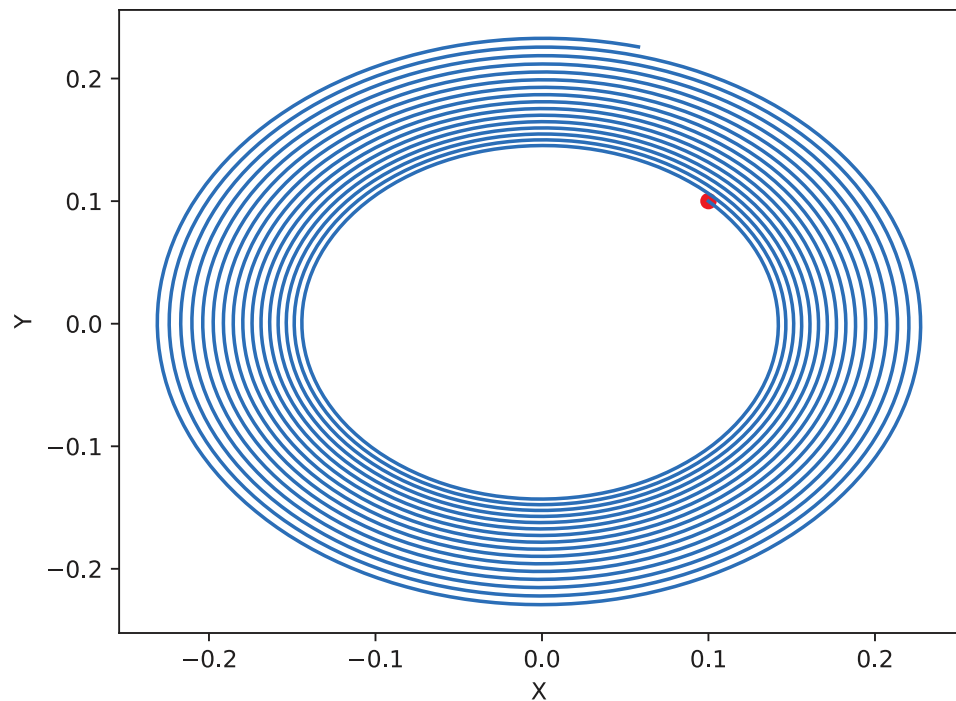
The update rules of x and y for learning rate α are

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

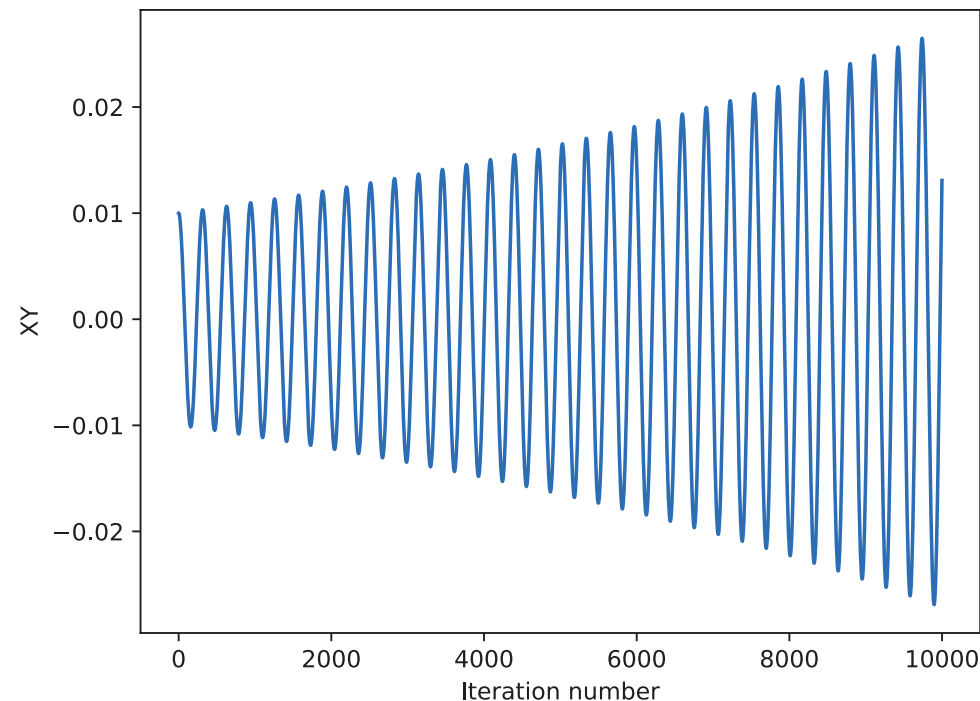
The update matrix is a rotation matrix multiplied by a constant $\sqrt{1 + \alpha^2} > 1$

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

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



(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 "Fictitious GAN: Training GANs with Historical Models", <https://arxiv.org/abs/1803.08647>

GANs are Problematic to Train

- Mode collapse

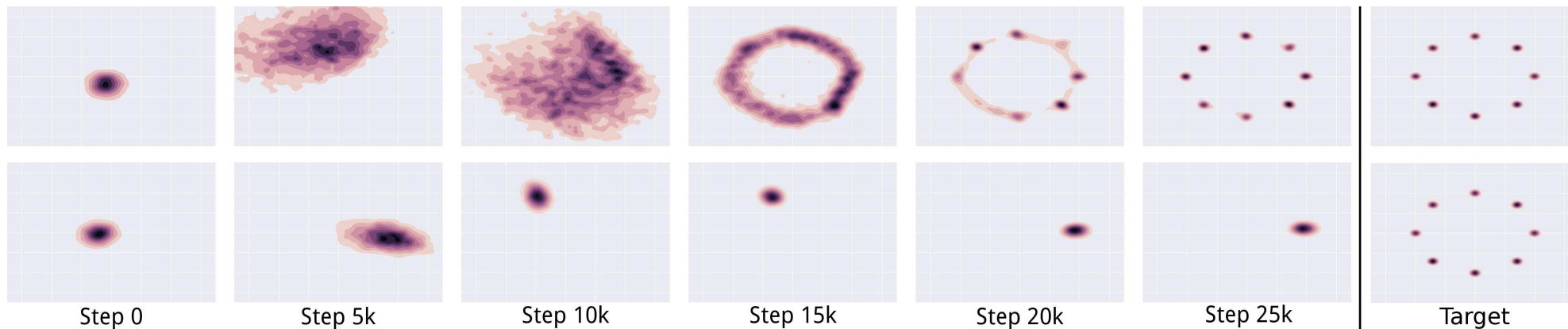


Figure 2 of "Unrolled Generative Adversarial Networks", <https://arxiv.org/abs/1611.02163>

- If the discriminator 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.

The Variational Autoencoders:

- are theoretically-pleasing;
- also provide an encoder, so apart from generation, they can be used as unsupervised feature extraction (the VAE encoder is used in various modeling architectures);
- the generated samples tend to be blurry (because of the sampling used in the reconstruction).

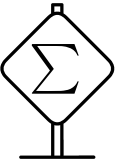
The Generative Adversarial Networks:

- offer very high sample quality;
- are difficult to train and suffer from mode collapse.

In past few years, GANs saw a big development, improving the sample quality substantially. However, since 2019/2020, VAEs have shown remarkable progress (alleviating the blurriness issue via discrete categorical latent variables, usually arranged in a 2D grid), and are being used for generation too. Furthermore, additional approaches (normalizing flows, diffusion models) are being explored, with diffusion models becoming the most promising approach since Q2 of 2021, surpassing both VAEs and GANs.

- Martin Arjovsky, Soumith Chintala, Léon Bottou: **Wasserstein GAN**
<https://arxiv.org/abs/1701.07875>
- Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville: **Improved Training of Wasserstein GANs** <https://arxiv.org/abs/1704.00028>
- 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>

Standard GANs optimize the Jensen-Shannon divergence



$$\text{JSD}(p\|q) = \frac{1}{2}D_{\text{KL}}(p\|(p+q)/2) + \frac{1}{2}D_{\text{KL}}(q\|(p+q)/2),$$

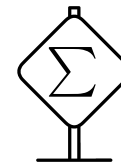
because for a fixed generator G , the optimum discriminator $D_G^*(\mathbf{x}) = \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + P_{\text{generator}}(\mathbf{x})}$.

Therefore,

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

Instead of minimizing JS divergence

$$\text{JSD}(p\|q) = \frac{1}{2}D_{\text{KL}}(p\|(p+q)/2) + \frac{1}{2}D_{\text{KL}}(q\|(p+q)/2),$$



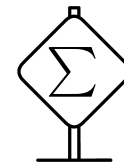
Wasserstein GAN minimizes Earth-Mover distance

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

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.

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

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

while

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

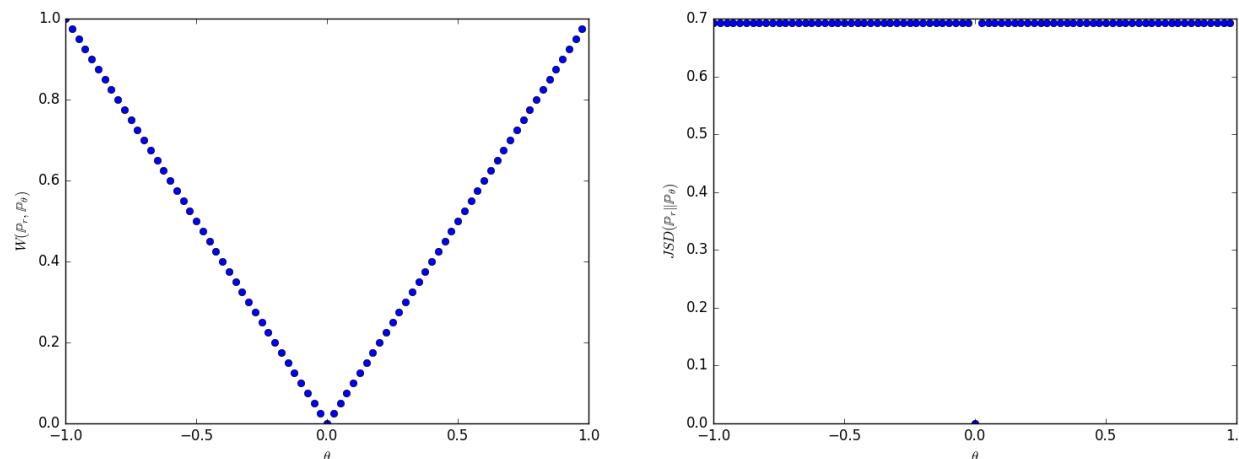
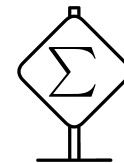


Figure 1 of "Wasserstein GAN", <https://arxiv.org/abs/1701.07875>

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

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



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

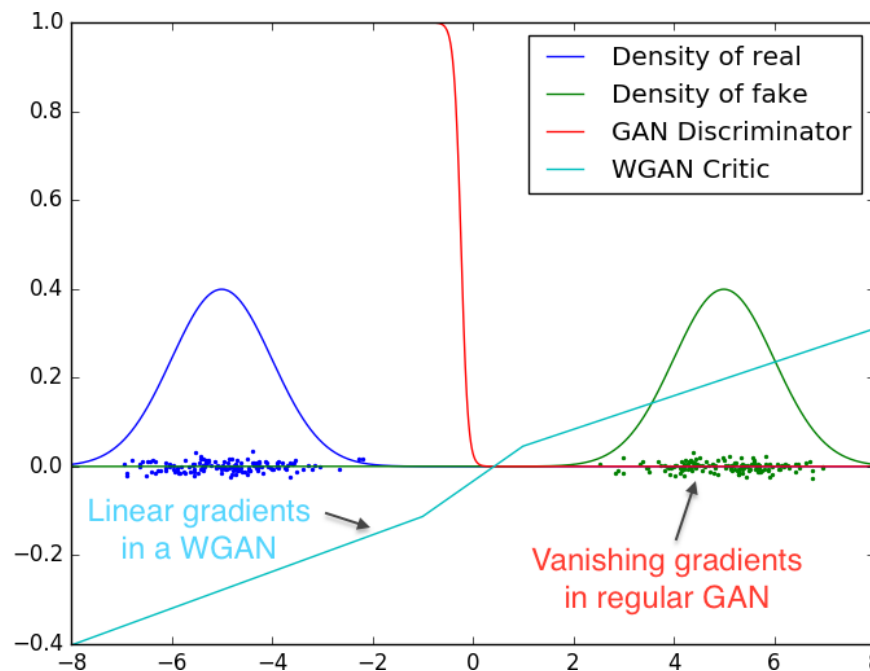


Figure 2 of "Wasserstein GAN", <https://arxiv.org/abs/1701.07875>

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.

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

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



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 "Wasserstein GAN", <https://arxiv.org/abs/1701.07875>



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 "Wasserstein GAN", <https://arxiv.org/abs/1701.07875>



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



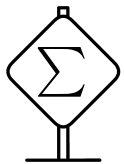
Figure 2 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", <https://arxiv.org/abs/1809.11096>



(a)

(b)

Figure 7 of "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.

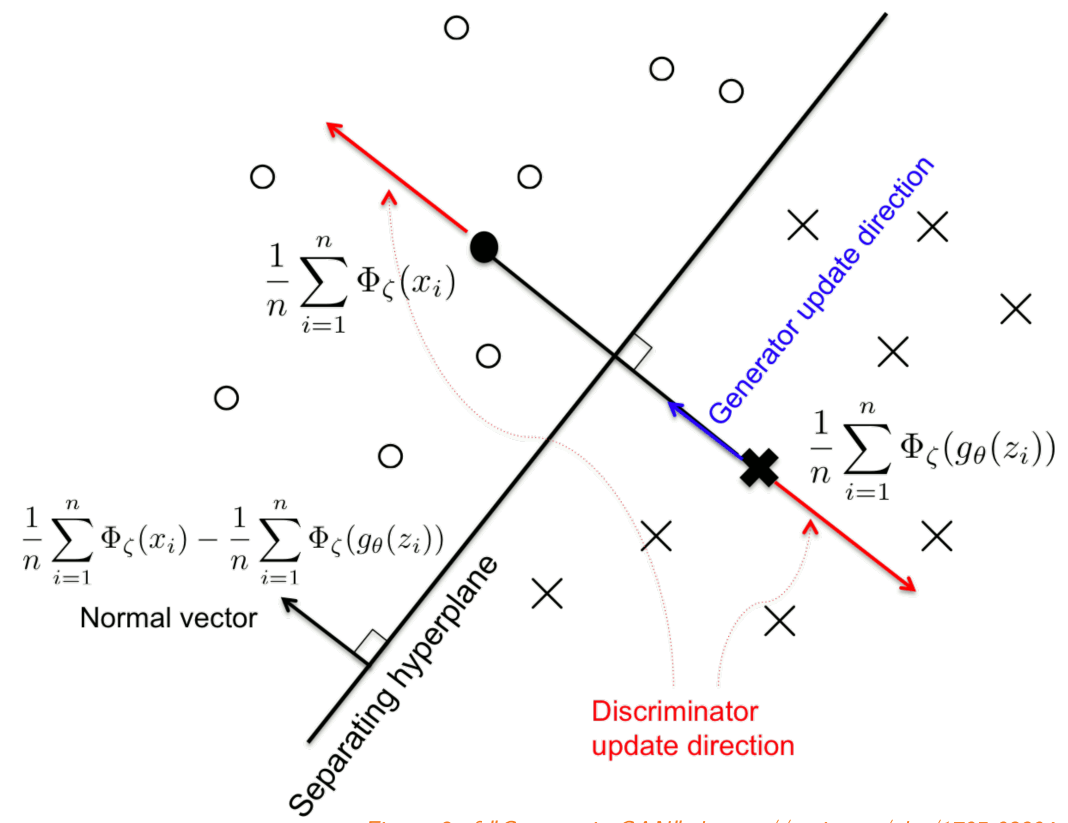


Figure 2 of "Geometric GAN", <https://arxiv.org/abs/1705.02894>

BigGAN Ingredients – Hinge Loss

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

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

and the generator by

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

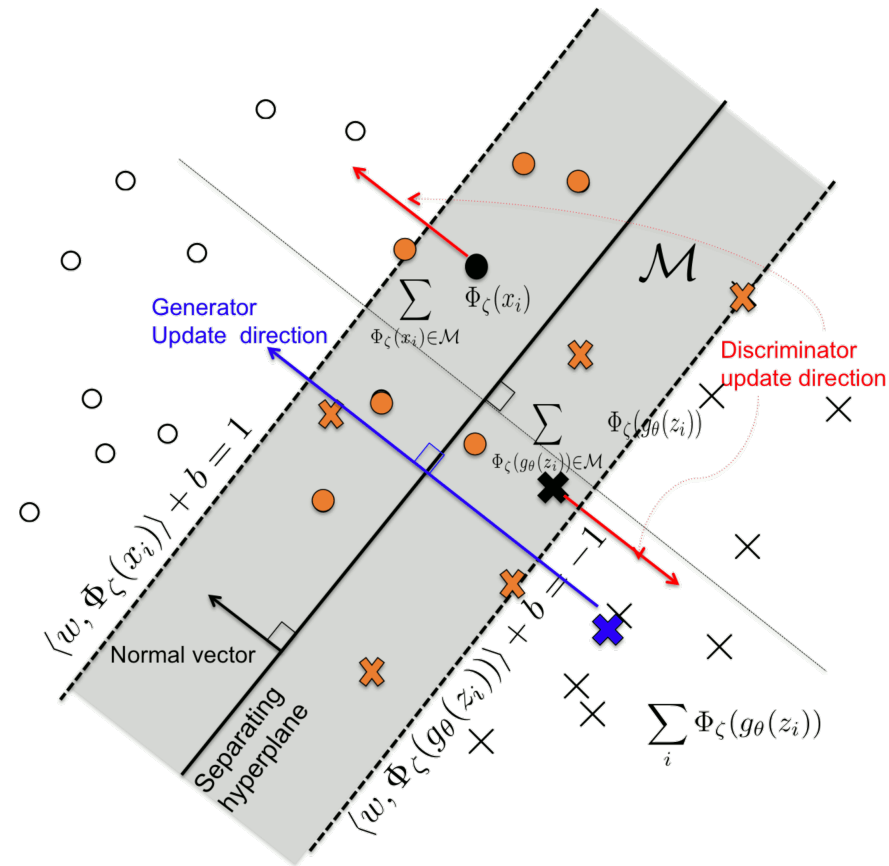
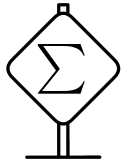
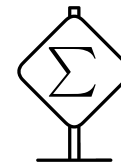


Figure 2 of "Geometric GAN", <https://arxiv.org/abs/1705.02894>

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 $\tilde{\mathbf{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \dots, L$ 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 :

$$\tilde{\mathbf{v}}_l \leftarrow (W^l)^\top \tilde{\mathbf{u}}_l / \|(W^l)^\top \tilde{\mathbf{u}}_l\|_2 \quad (20)$$

$$\tilde{\mathbf{u}}_l \leftarrow W^l \tilde{\mathbf{v}}_l / \|W^l \tilde{\mathbf{v}}_l\|_2 \quad (21)$$

2. Calculate \bar{W}_{SN}^l with the spectral norm:

$$\bar{W}_{\text{SN}}^l(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\mathbf{u}}_l^\top W^l \tilde{\mathbf{v}}_l \quad (22)$$

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

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}_{\text{SN}}^l(W^l), \mathcal{D}_M) \quad (23)$$

Algorithm 1 of "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.

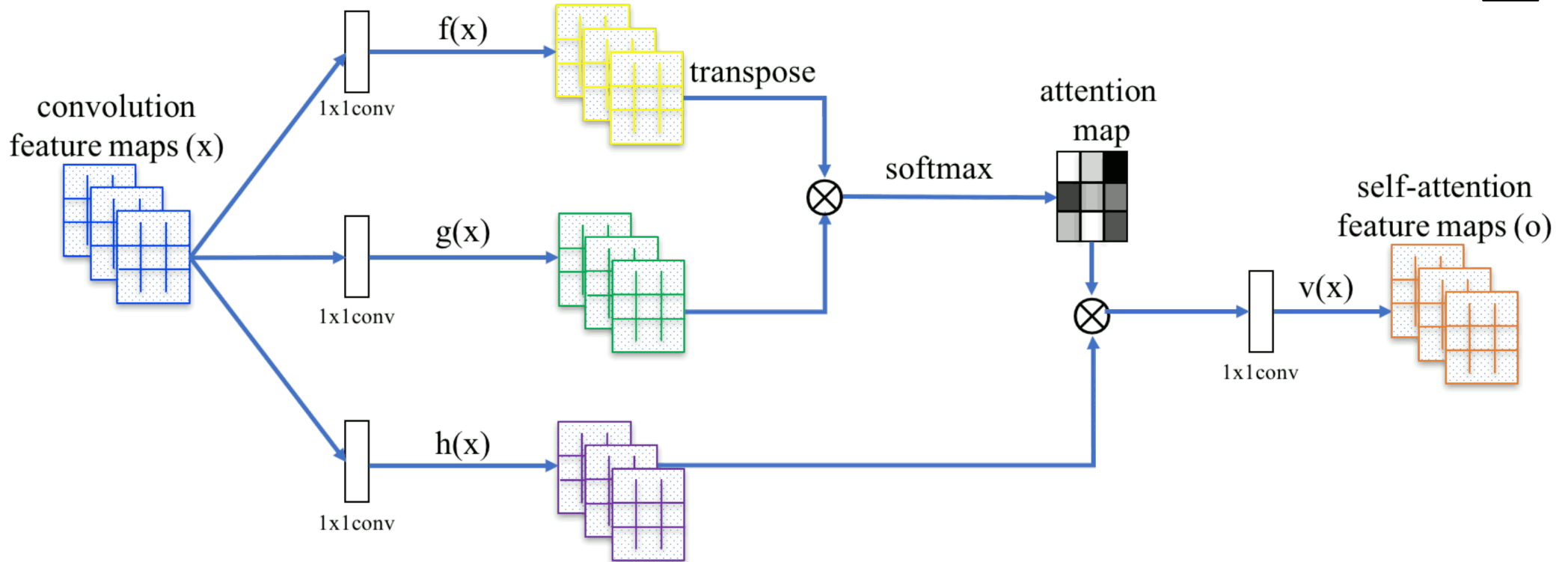
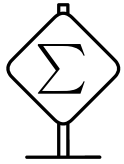


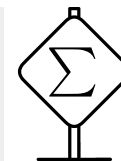
Figure 2 of "Self-Attention Generative Adversarial Networks", <https://arxiv.org/abs/1805.08318>

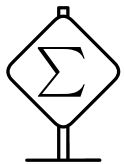

```
def attention(self, x, ch):
    f = conv(x, ch // 8, kernel=1, stride=1) # [bs, h, w, c']
    g = conv(x, ch // 8, kernel=1, stride=1) # [bs, h, w, c']
    h = conv(x, ch, kernel=1, stride=1) # [bs, h, w, c]

    # 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

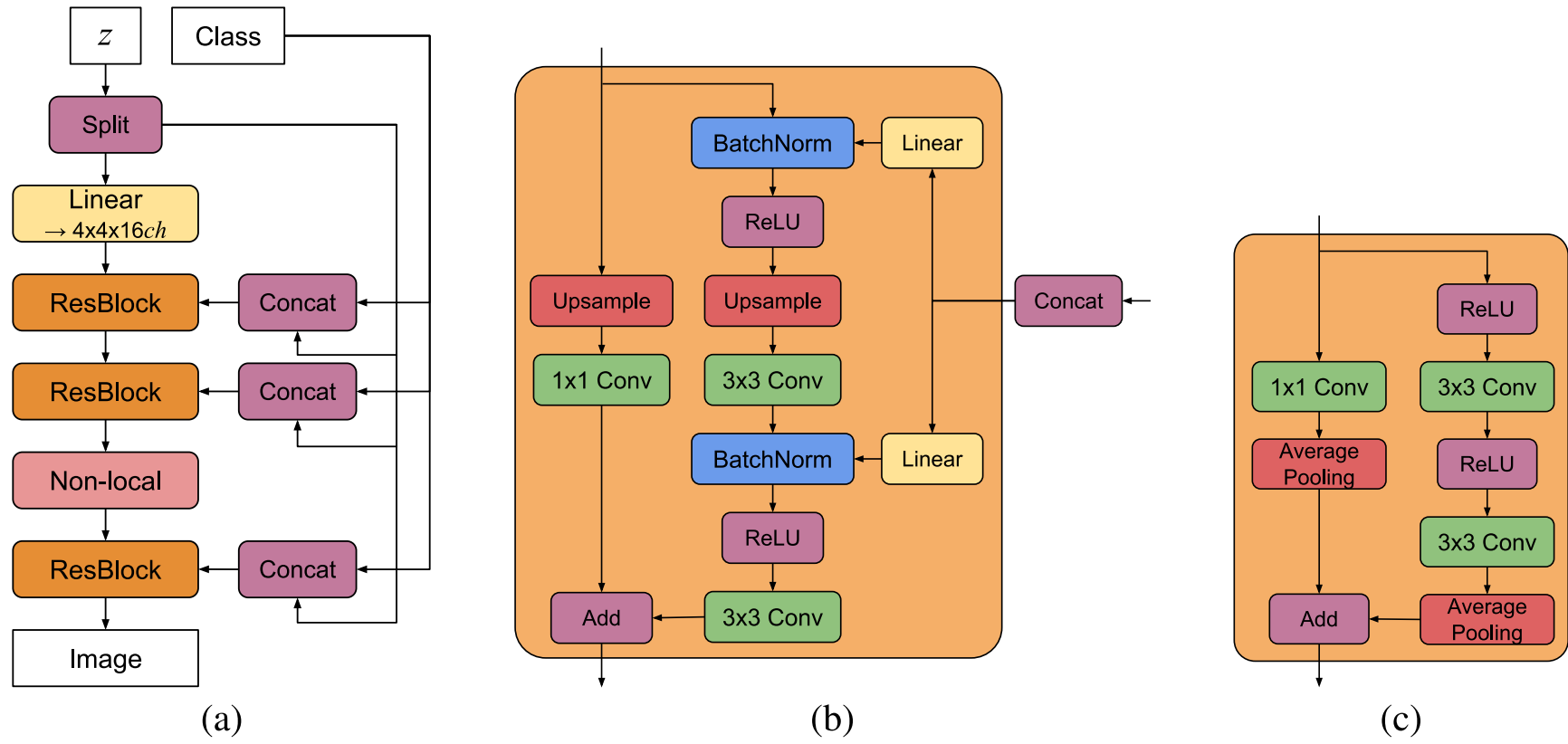
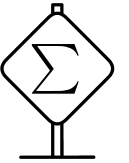


Figure 15: (a) A typical architectural layout for BigGAN's G ; details are in the following tables. (b) A Residual Block ($ResBlock\ up$) in BigGAN's G . (c) A Residual Block ($ResBlock\ down$) in BigGAN's D .

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

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



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

(a) Generator

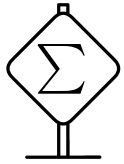
RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down $ch \rightarrow 2ch$
Non-Local Block (64×64)
ResBlock down $2ch \rightarrow 4ch$
ResBlock down $4ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 16ch$
ResBlock down $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ReLU, Global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

(b) Discriminator

Table 4 of "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, \mathbf{z} is sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, 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 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", <https://arxiv.org/abs/1809.11096>

- Aaron van den Oord, Oriol Vinyals, Koray Kavukcuoglu: **Neural Discrete Representation Learning** <https://arxiv.org/abs/1711.00937>
- Ali Razavi, Aaron van den Oord, Oriol Vinyals: **Generating Diverse High-Fidelity Images with VQ-VAE-2** <https://arxiv.org/abs/1906.00446>
- Patrick Esser, Robin Rombach, Björn Ommer: **Taming Transformers for High-Resolution Image Synthesis** <https://arxiv.org/abs/2012.09841>
- Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, Ilya Sutskever: **Zero-Shot Text-to-Image Generation** <https://arxiv.org/abs/2102.12092>

In VQ-VAE, the latent variables are arranged in a **two-dimensional grid** and each latent variable is a **discrete** sample from a categorical distribution.

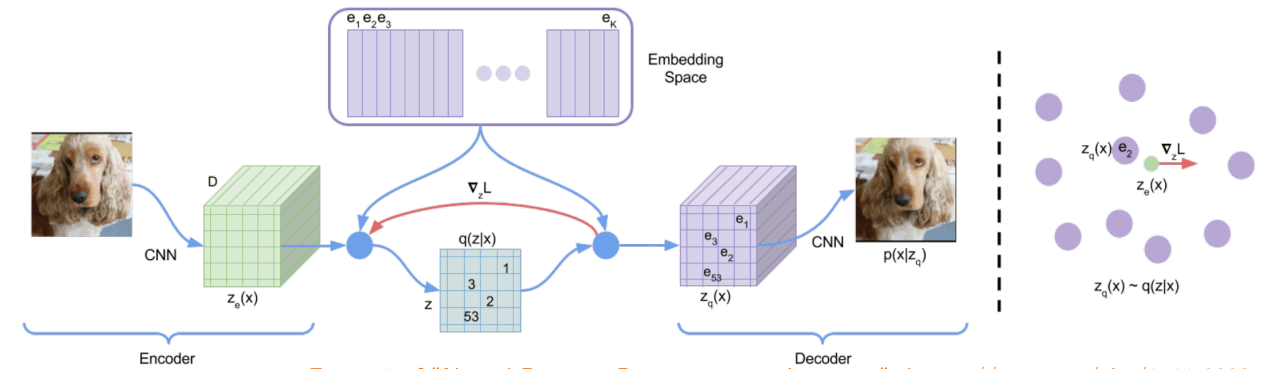


Figure 1 of "Neural Discrete Representation Learning", <https://arxiv.org/abs/1711.00937>



Figure 1 of "Neural Discrete Representation Learning", <https://arxiv.org/abs/1711.00937>

Modeling Latent Structure

With a two-dimensional structure of interdependent latent variables, a question arises of how to generate the latent variables themselves.

- An auto-regressive sampling is used in VQ-VAE (PixelCNN architecture).
- In VQ-VAE-2, the latents are modeled recursively as another VQ-VAE, until finally a PixelCNN is used for the last latents.

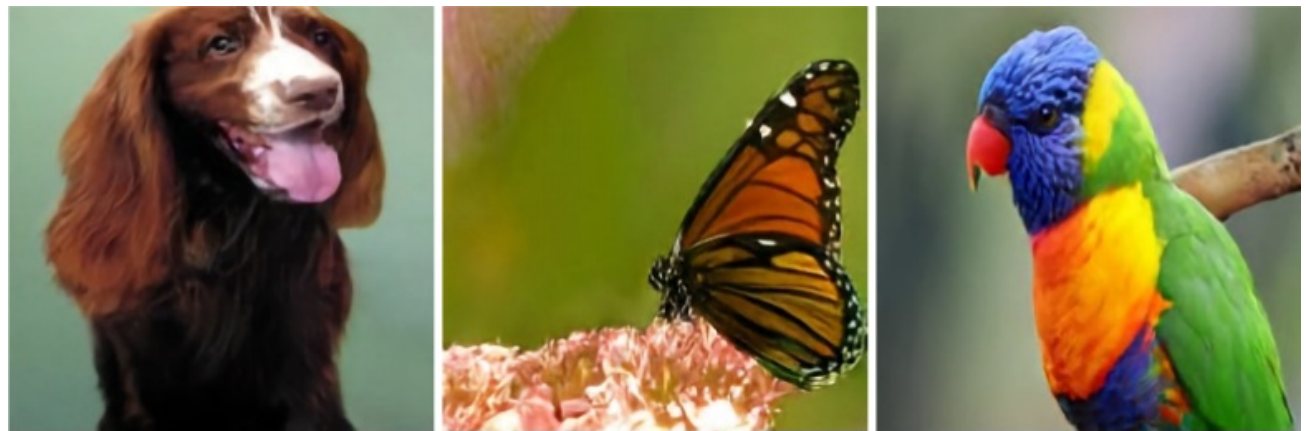
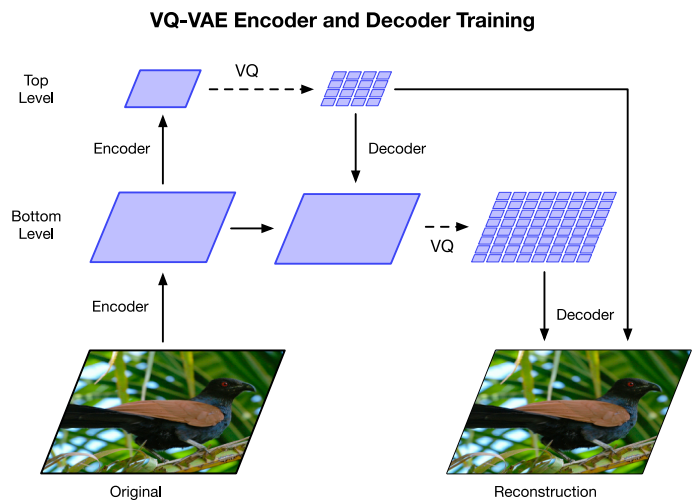


Figure 1 of "Generating Diverse High-Fidelity Images with VQ-VAE-2", <https://arxiv.org/abs/1906.00446>

Figure 2 of "Generating Diverse High-Fidelity Images with VQ-VAE-2", <https://arxiv.org/abs/1906.00446>

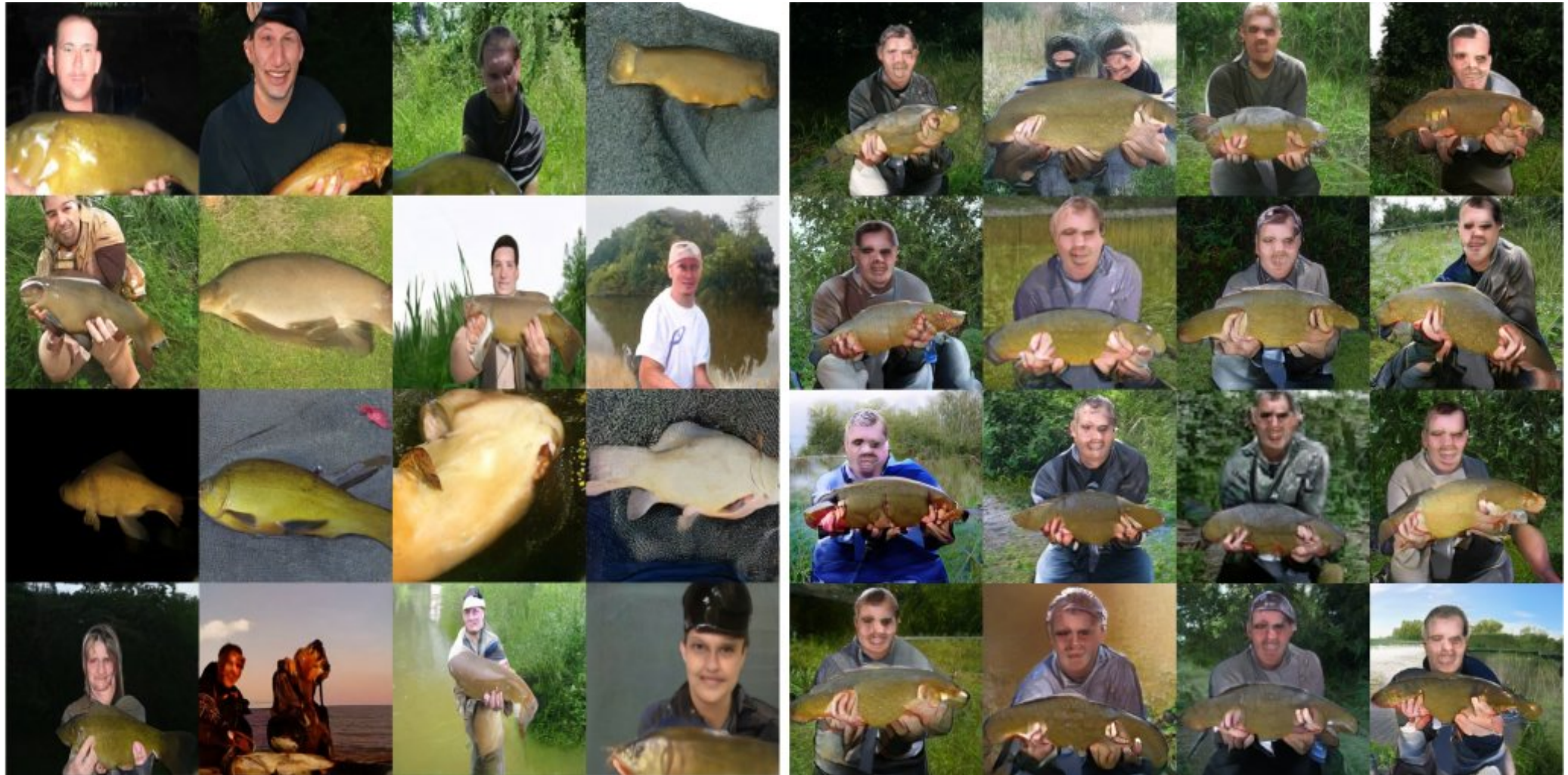


Figure 5 of "Generating Diverse High-Fidelity Images with VQ-VAE-2", <https://arxiv.org/abs/1906.00446>



Figure 5 of "Generating Diverse High-Fidelity Images with VQ-VAE-2", <https://arxiv.org/abs/1906.00446>

Modeling Latent Structure

With a two-dimensional structure of interdependent latent variables, a question arises of how to generate the latent variables themselves.

- In VQGAN, Transformer is used to model the discrete latent variables.

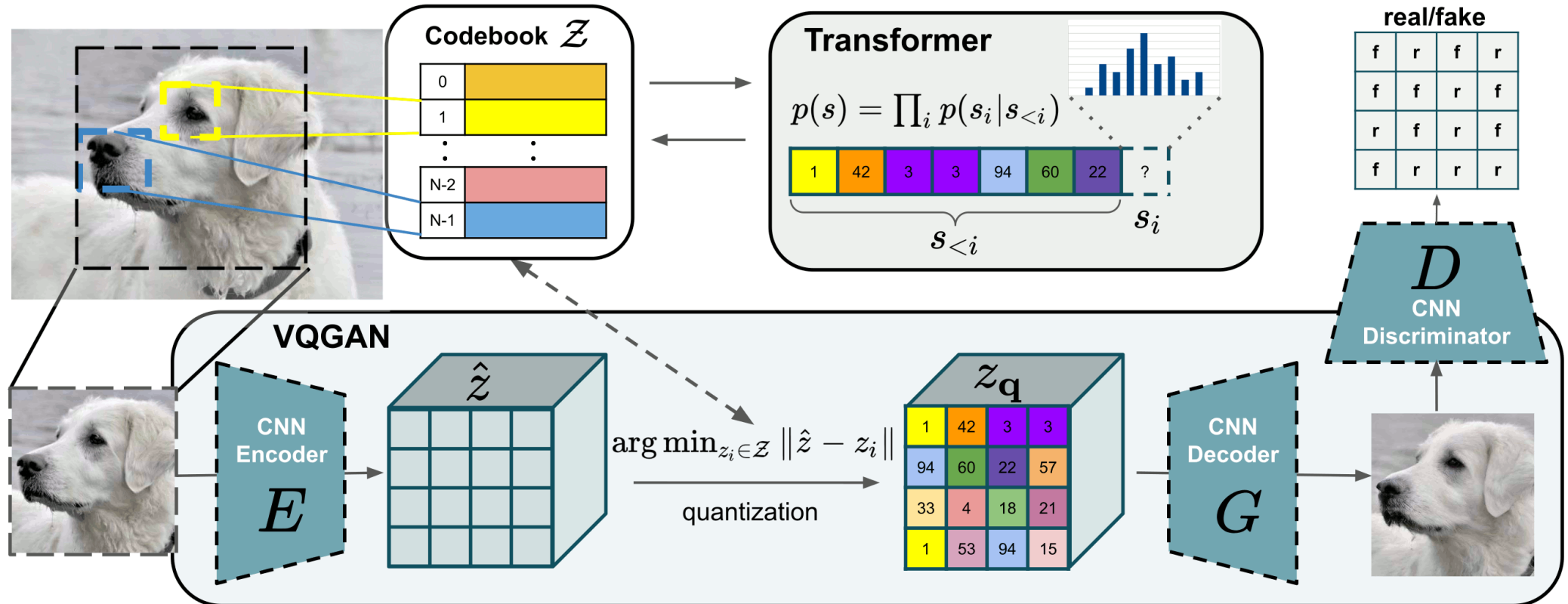
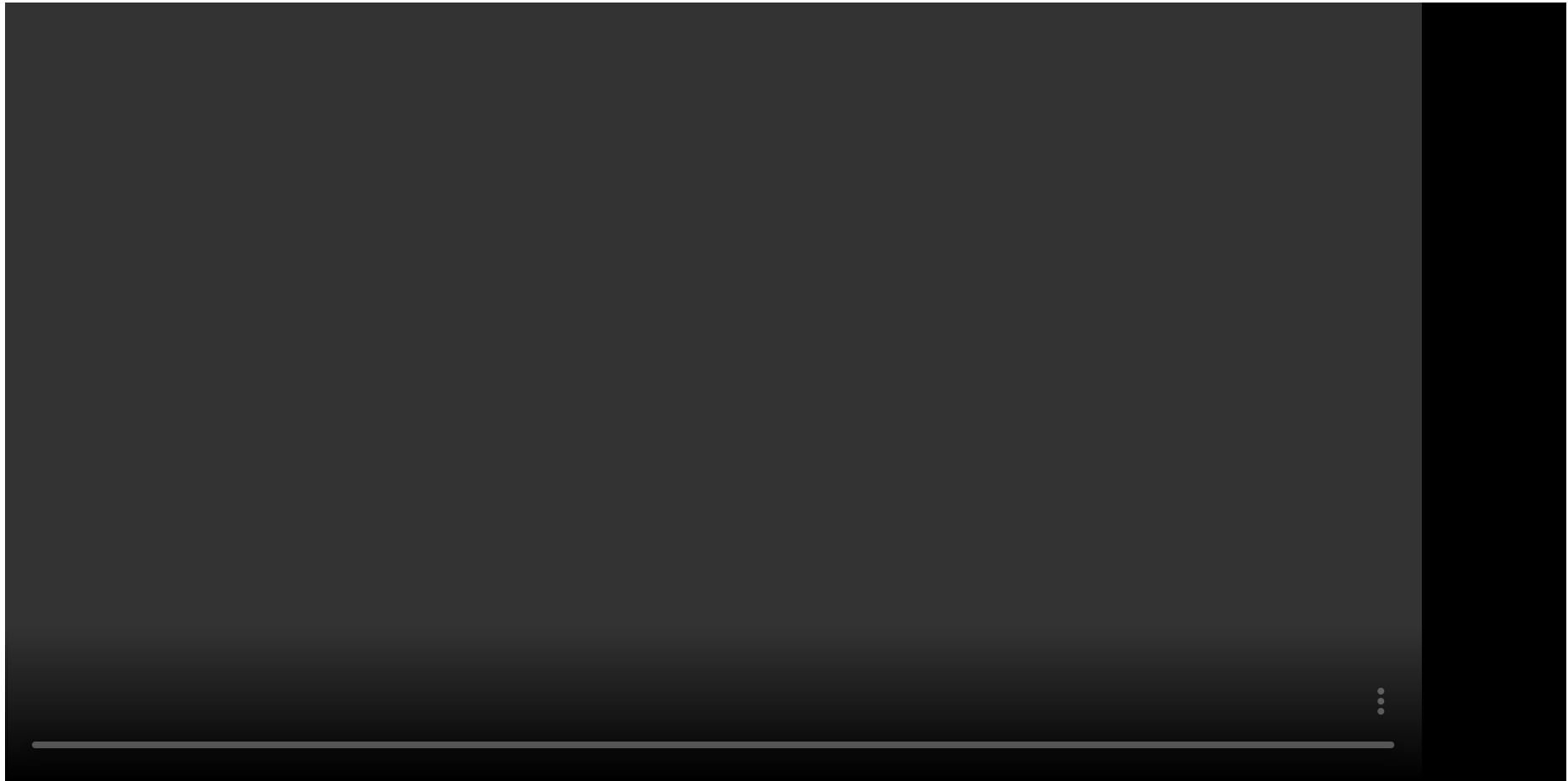


Figure 2 of "Taming Transformers for High-Resolution Image Synthesis", <https://arxiv.org/abs/2012.09841>



Modeling Latent Structure

With a two-dimensional structure of interdependent latent variables, a question arises of how to generate the latent variables themselves.

- In DALL-E, Transformer is used to model a sequence of words followed by a sequence of the discrete image latent variables.



(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

(d) the exact same cat on the top as a sketch on the bottom

Figure 2 of "Zero-Shot Text-to-Image Generation", <https://arxiv.org/abs/2102.12092>

Diffusion Models

- Jonathan Ho, Ajay Jain, Pieter Abbeel: **Denoising Diffusion Probabilistic Models**
<https://arxiv.org/abs/2006.11239>
- Prafulla Dhariwal, Alex Nichol: **Diffusion Models Beat GANs on Image Synthesis**
<https://arxiv.org/abs/2105.05233>

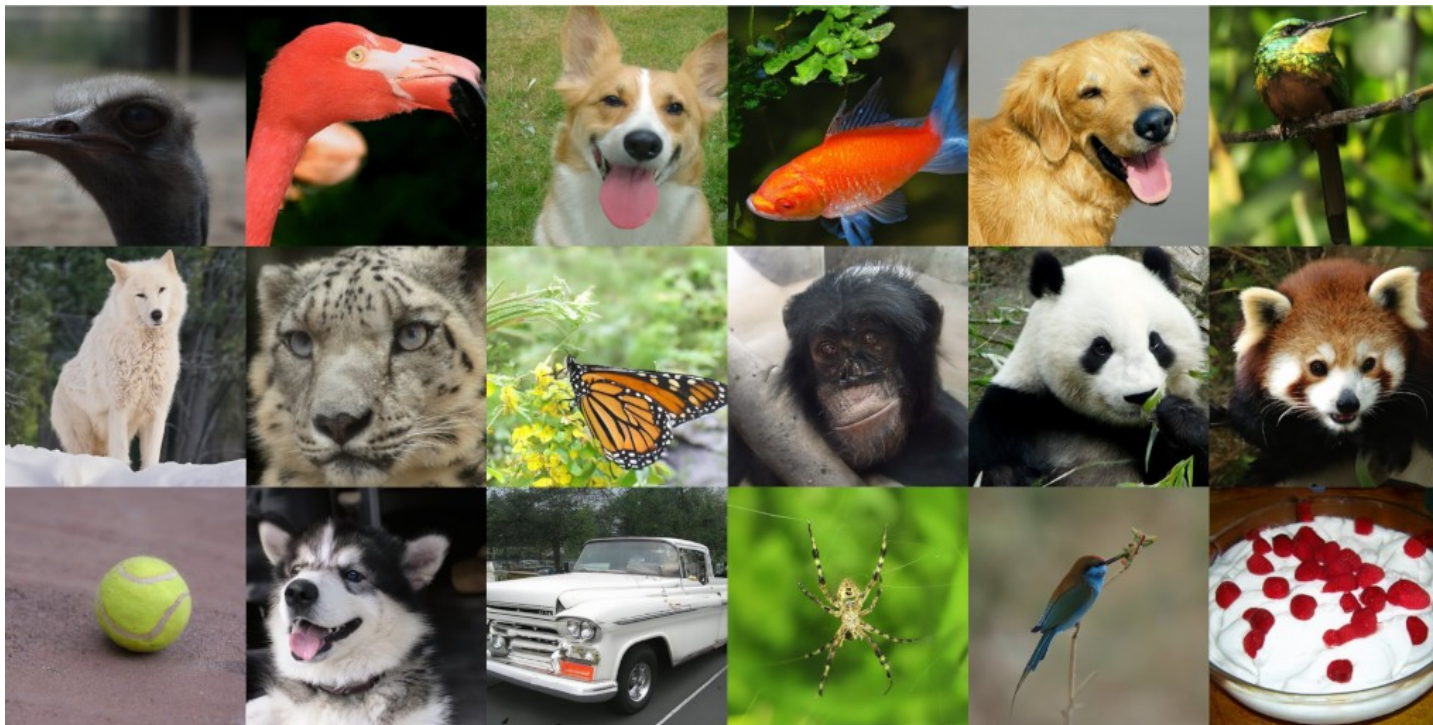
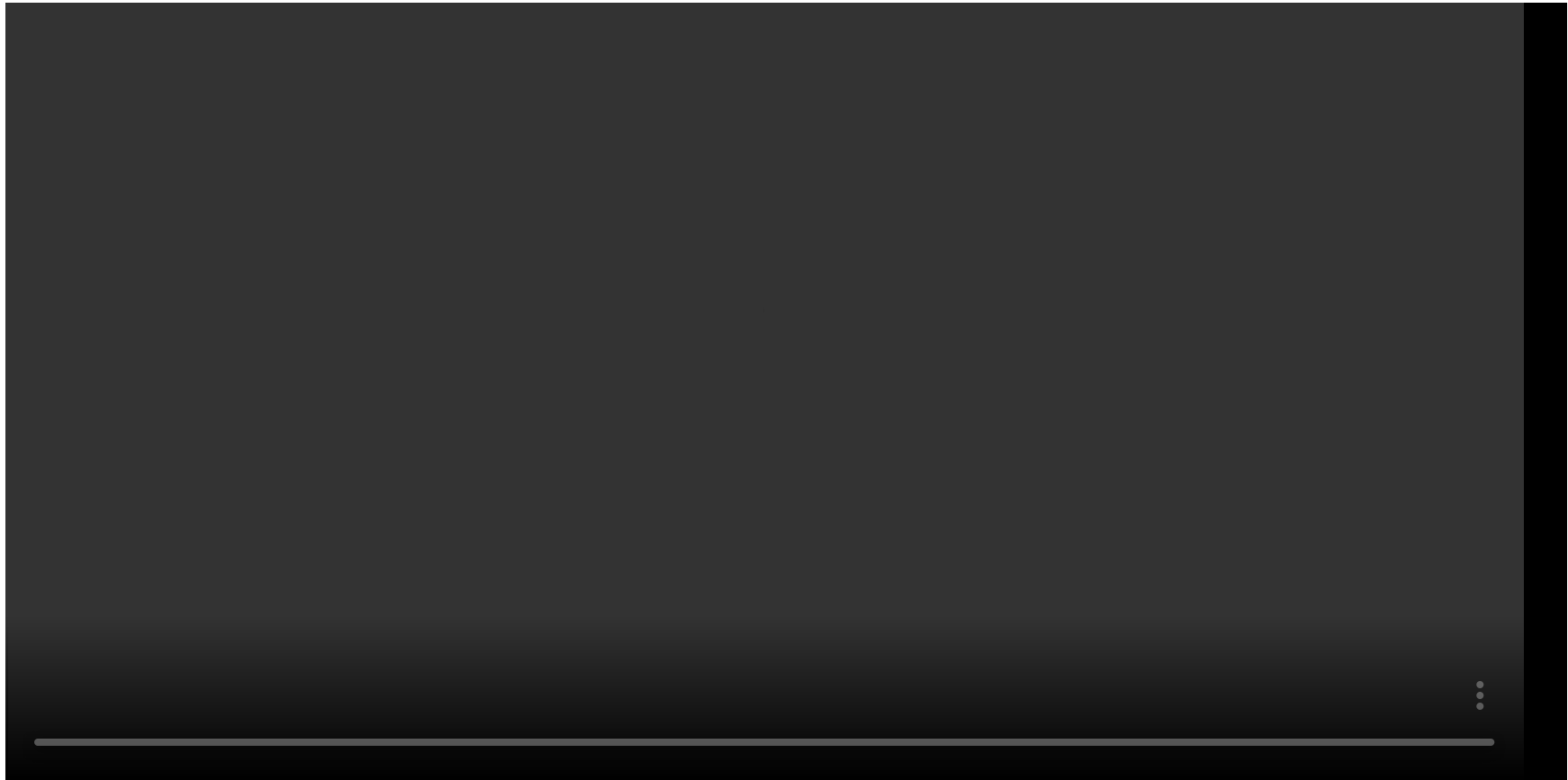


Figure 1 of "Diffusion Models Beat GANs on Image Synthesis", <https://arxiv.org/abs/2105.05233>

- Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, M. Norouzi: **Image Super-Resolution via Iterative Refinement** <https://arxiv.org/abs/2104.07636>



- Alex Nichol et al.: **GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models** <https://arxiv.org/abs/2112.10741>



“a hedgehog using a calculator”



“a corgi wearing a red bowtie and a purple party hat”



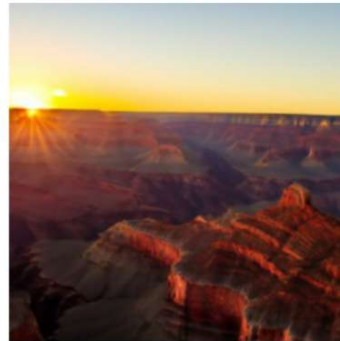
“robots meditating in a vipassana retreat”



“a fall landscape with a small cottage next to a lake”



“a surrealist dream-like oil painting by salvador dalí of a cat playing checkers”



“a professional photo of a sunset behind the grand canyon”



“a high-quality oil painting of a psychedelic hamster dragon”



“an illustration of albert einstein wearing a superhero costume”

Figure 1 of "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models", <https://arxiv.org/abs/2112.10741>

Diffusion-Based Text-Conditional Image Generation



“zebras roaming in the field”



“a girl hugging a corgi on a pedestal”



“a man with red hair”



“a vase of flowers”



“an old car in a snowy forest”



“a man wearing a white hat”

Figure 2 of "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models", <https://arxiv.org/abs/2112.10741>

- Laurent Dinh, David Krueger, Yoshua Bengio: **NICE: Non-linear Independent Components Estimation** <https://arxiv.org/abs/1410.8516>
- Laurent Dinh, Jascha Sohl-Dickstein, Samy Bengio: **Density estimation using Real NVP** <https://arxiv.org/abs/1605.08803>
- Diederik P. Kingma, Prafulla Dhariwal: **Glow: Generative Flow with Invertible 1x1 Convolutions** <https://arxiv.org/abs/1807.03039>



Figure 1 of "Glow: Generative Flow with Invertible 1x1 Convolutions", <https://arxiv.org/abs/1807.03039>