

# Deep Generative Models – Part I

Martin Renqiang Min  
NEC Laboratories America  
Mar 26, 2025

# Outline

- Variational Autoencoder
- Generative Adversarial Networks
- Applications

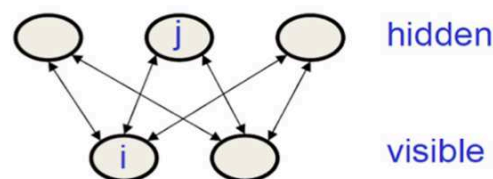
# What are Generative Models

- Generative models are probabilistic models of high-dimensional data such as text, image, video, audio, and biological sequences.
- Generative models are used to perform density estimation, anomaly detection, representation learning, dimensionality reduction, data translation, and data generation.
- Generative Models
  - Latent Variable Models such as Variational Autoencoders and Probabilistic Diffusion Models
  - Generative Adversarial Networks
  - Autoregressive models such as LLMs

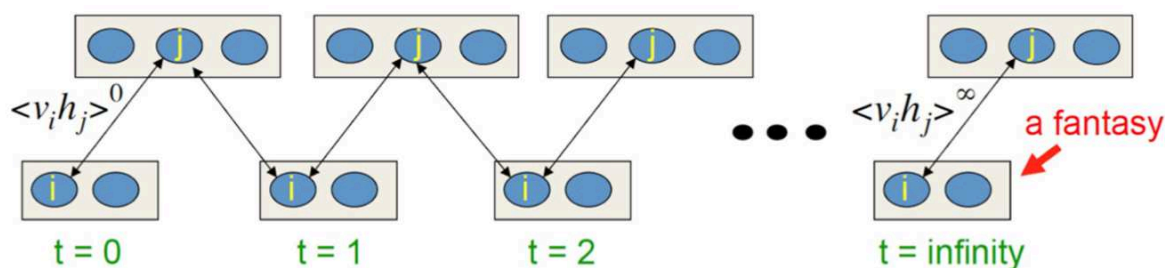
## Energy-Based Model: Undirected Probabilistic Generative Models with Latent Variables

$$p(\mathbf{v}, \mathbf{h}) \propto e^{-E(\mathbf{v}, \mathbf{h})} \quad E(v, h) = - \sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j$$

- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped.
  - So we can quickly get the exact value of :  $\langle v_i h_j \rangle_v$



$$p(h_j = 1) = \frac{1}{1 + e^{-(b_j + \sum_{i \in \text{vis}} v_i w_{ij})}}$$



Start with a training vector on the visible units. Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

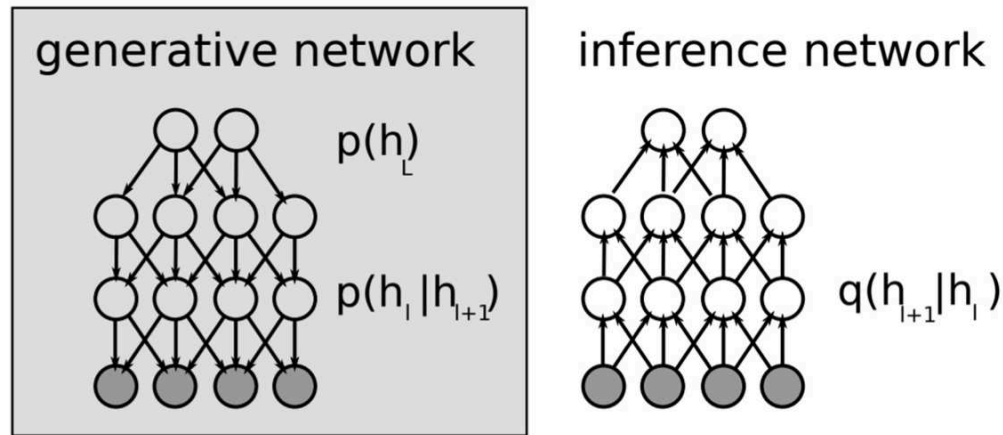
$$\Delta w_{ij} = \varepsilon (\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty)$$

Parameter-sharing RBM for Collaborative Filtering: Treat each user rating vector as a data point

			?	

# Directed Probabilistic Generative Models with Latent Variables

We want to train a directed generative model  $p$

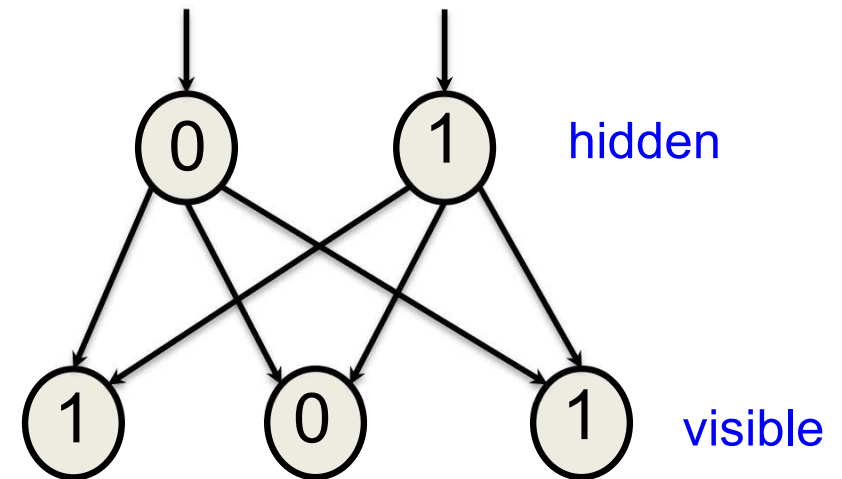


$$p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x} | \mathbf{h}_1) p(\mathbf{h}_1 | \mathbf{h}_2) \dots p(\mathbf{h}_L)$$
$$q(\mathbf{h} | \mathbf{x}) = q(\mathbf{h}_1 | \mathbf{x}) q(\mathbf{h}_2 | \mathbf{h}_1) \dots q(\mathbf{h}_L | \mathbf{h}_{L-1})$$

- Our goal is to learn the model parameters to maximize the log-probability of data  $\mathbf{x}$ 
  - Learning: learn the model parameters maximizing  $\log p(\mathbf{x})$
  - Inference: infer the hidden states from  $p(\mathbf{h} | \mathbf{x})$

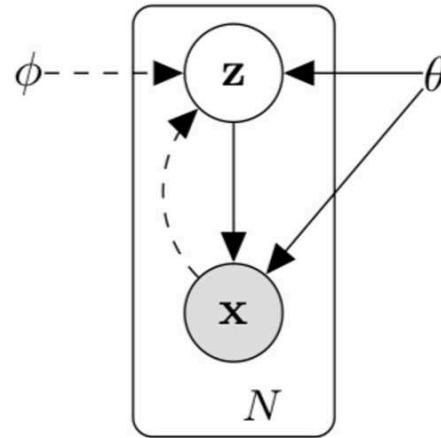
# How a Directed Probabilistic Generative Model with Latent Variables Generates Data

- We generate data in two sequential steps:
  - First pick the hidden states from their prior distribution.
  - Then pick the visible states from their conditional distribution given the hidden states.
- The probability of generating a visible vector,  $\mathbf{v}$ , is computed by summing over all possible hidden states. Each hidden state is an “explanation” of  $\mathbf{v}$ .



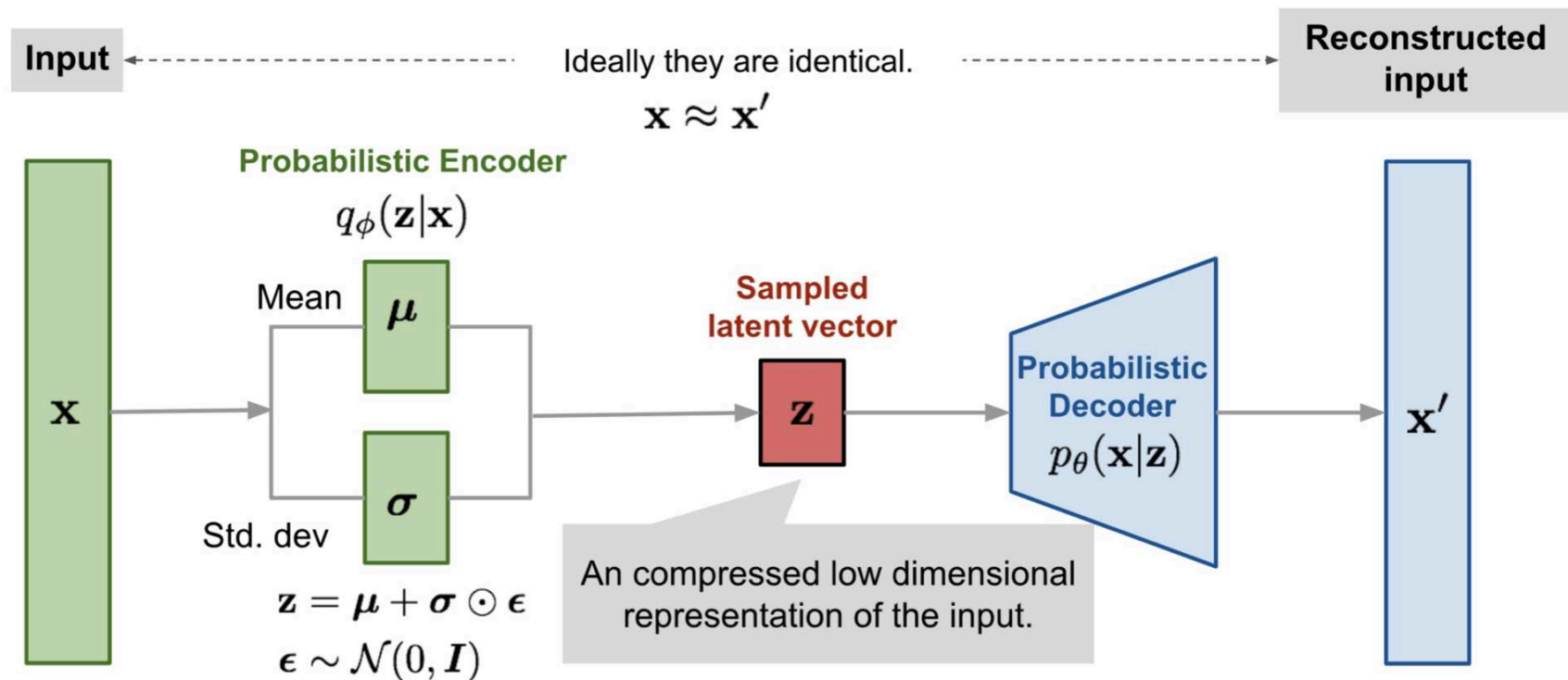
$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}) p(\mathbf{v} | \mathbf{h})$$

# Variational Autoencoder



The type of directed graphical model under consideration. Solid lines denote the generative model  $p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ , dashed lines denote the variational approximation  $q_\phi(\mathbf{z}|\mathbf{x})$  to the intractable posterior  $p_\theta(\mathbf{z}|\mathbf{x})$ . The variational parameters  $\phi$  are learned jointly with the generative model parameters  $\theta$ .

## Variational Autoencoder with an Isotropic Multivariate Gaussian Prior



Picture Credit: <https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html>



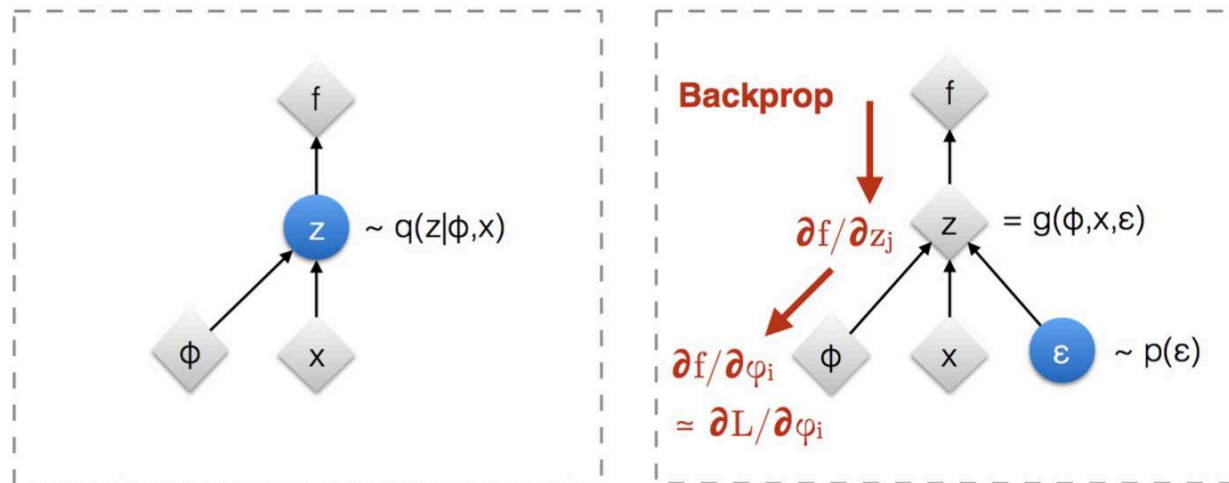
# The Reparameterization Trick Using a Deterministic Function Mapping



$$\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)} \mathbf{I})$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Original form

Reparameterised form



 : Deterministic node  
 : Random node

[Kingma, 2013]  
 [Bengio, 2013]  
 [Kingma and Welling 2014]  
 [Rezende et al 2014]

Kingma and Welling, Auto-Encoding Variational Bayes. ICLR 2014

## Variational Inference with the Reparameterization Trick

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}) = \sum_{i=1}^N \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} [-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})]$$

ELBO:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})} [\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})]$$

## Variational Autoencoder with an Isotropic Multivariate Gaussian Prior

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})} \left[ \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}) \right]$$

$$p(z) \equiv \mathcal{N}(0, I)$$

$$p(x|z) \equiv \mathcal{N}(f(z), cI) \quad f \in F \quad c > 0$$

$$f^* = \arg \max_{f \in F} \mathbb{E}_{z \sim q_x^*} (\log p(x|z))$$

$$= \arg \max_{f \in F} \mathbb{E}_{z \sim q_x^*} \left( -\frac{\|x - f(z)\|^2}{2c} \right)$$

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^J \left( 1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2 \right) + \frac{1}{L} \sum_{l=1}^L \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$$

$$\text{where } \mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)} \quad \text{and} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$$

## Training VAE Using Mini-batch Variational Inference with the Reparameterization Trick

---

**Algorithm 1** Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings  $M = 100$  and  $L = 1$  in experiments.

---

$\theta, \phi \leftarrow$  Initialize parameters

**repeat**

$\mathbf{X}^M \leftarrow$  Random minibatch of  $M$  datapoints (drawn from full dataset)

$\epsilon \leftarrow$  Random samples from noise distribution  $p(\epsilon)$

$\mathbf{g} \leftarrow \nabla_{\theta, \phi} \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M, \epsilon)$  (Gradients of minibatch estimator (8))

$\theta, \phi \leftarrow$  Update parameters using gradients  $\mathbf{g}$  (e.g. SGD or Adagrad [DHS10])

**until** convergence of parameters  $(\theta, \phi)$

**return**  $\theta, \phi$

---

## VAE for Generating MNIST Digits



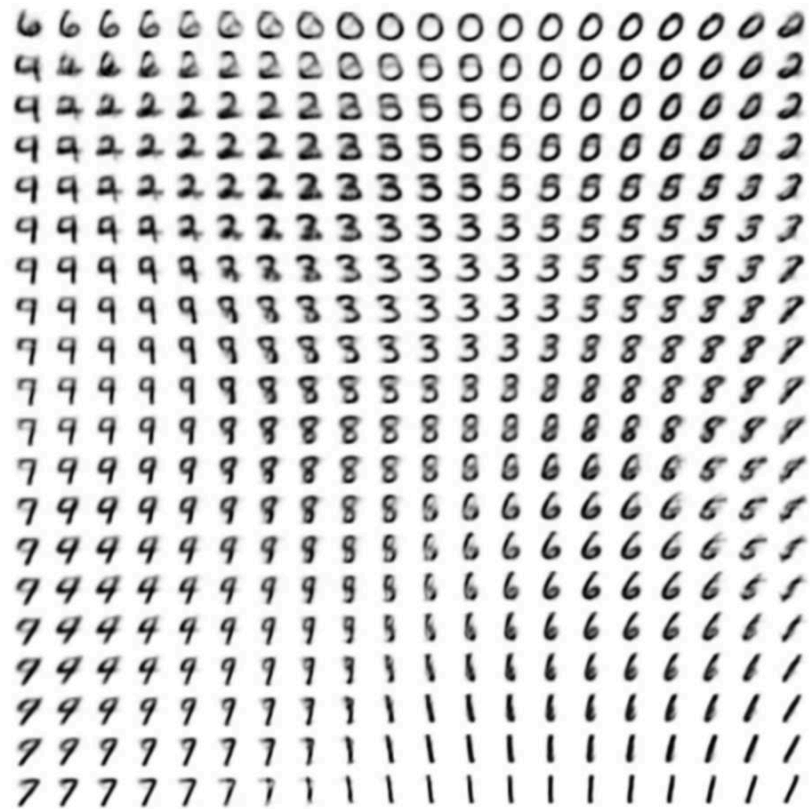
left: 1st epoch, middle: 9th epoch, right: original

Picture Credit: <http://kvfrans.com/variational-autoencoders-explained/>

## Learned 2D Manifold by VAE



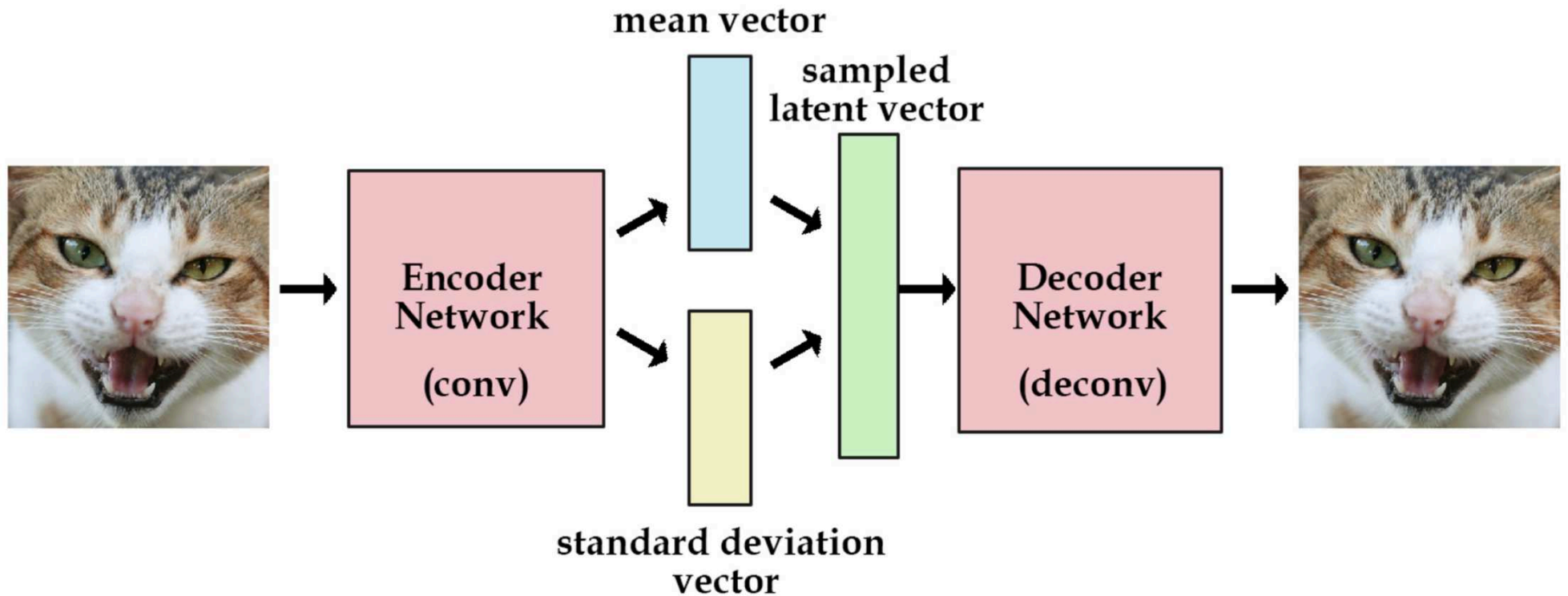
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

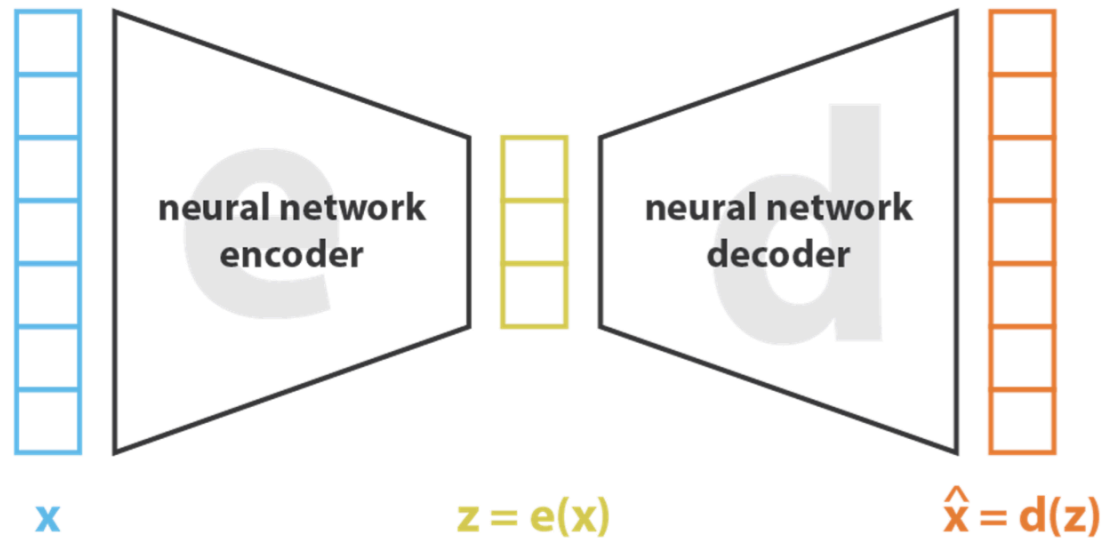


## VAE with Convolutional Neural Networks as Encoder and Decoder



Picture Credit: <http://kvfrans.com/variational-autoencoders-explained/>

# Autoencoder vs. Variational Autoencoder

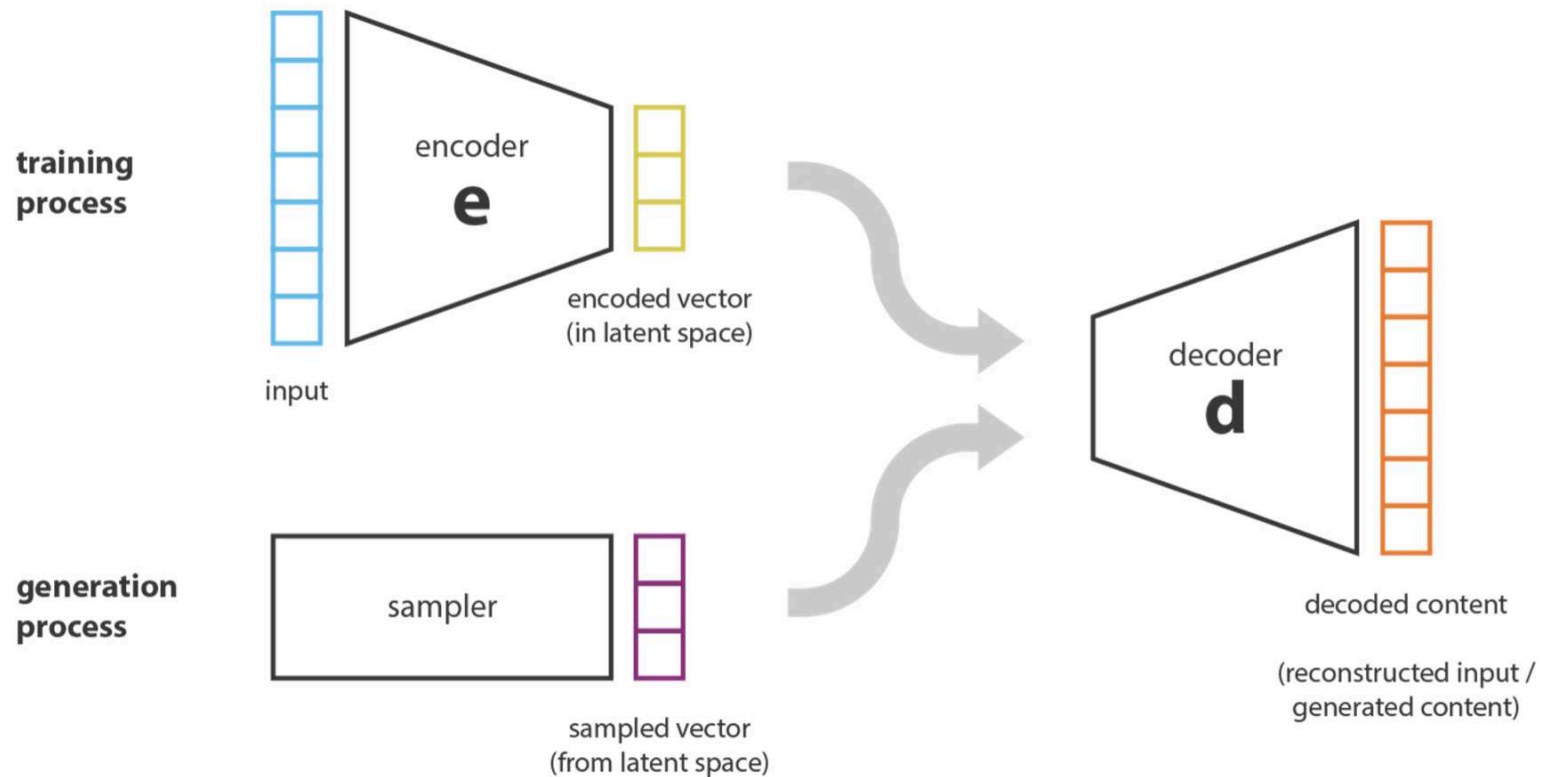


---

$$\text{loss} = ||x - \hat{x}||^2 = ||x - d(z)||^2 = ||x - d(e(x))||^2$$

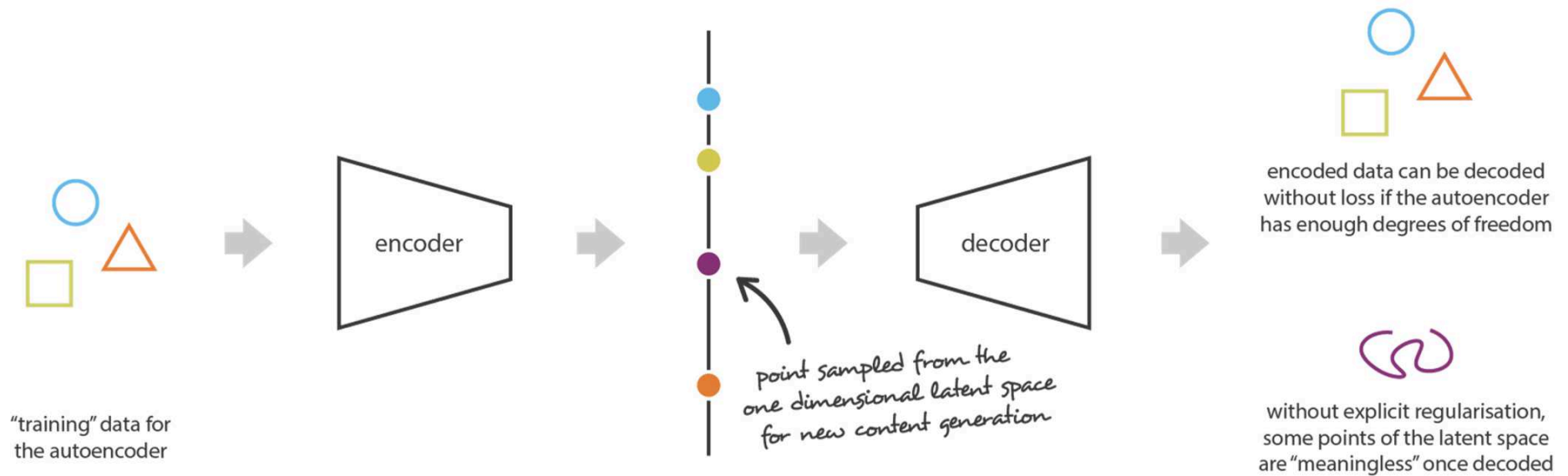


# Autoencoder vs. Variational Autoencoder

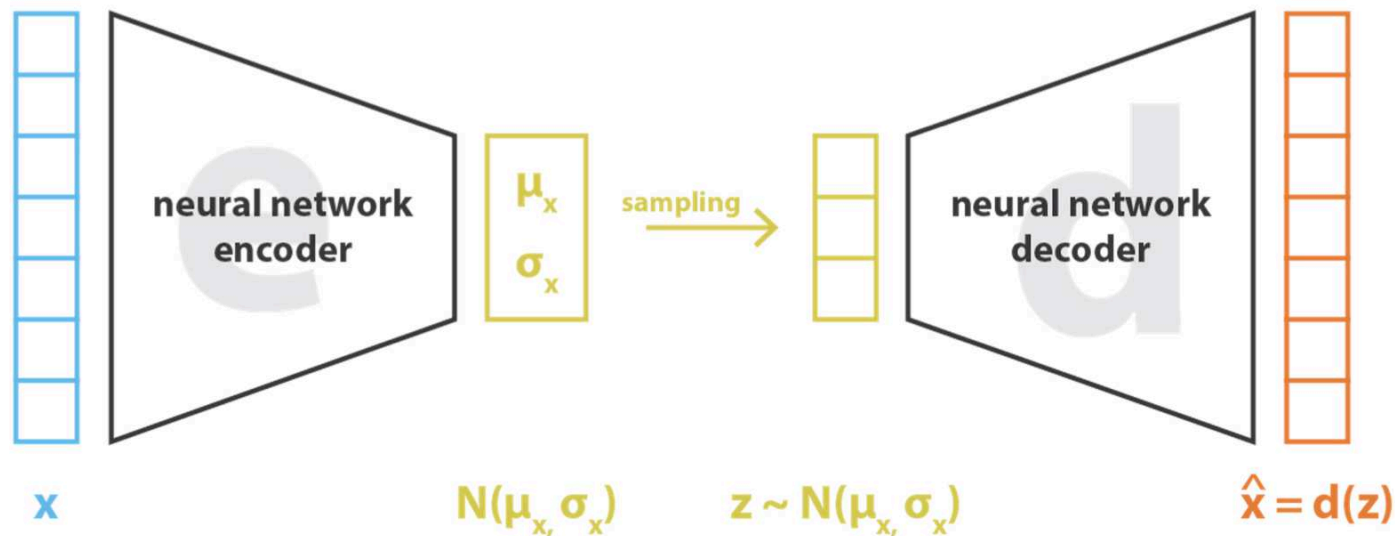


Picture Credit: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

# Autoencoder vs. Variational Autoencoder



# Autoencoder vs. Variational Autoencoder



---

$$\text{loss} = ||x - \hat{x}||^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

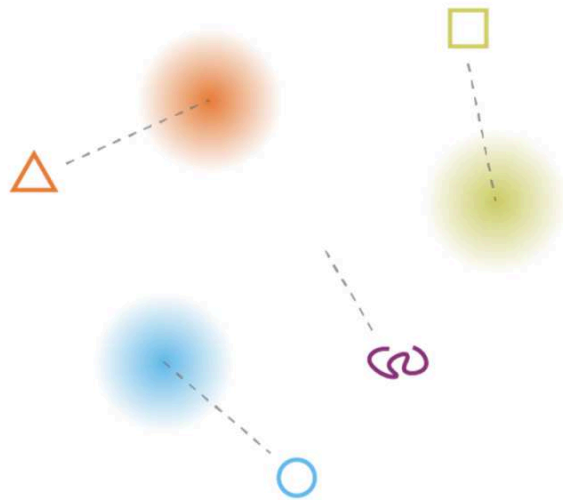
Picture Credit: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

# Autoencoder vs. Variational Autoencoder

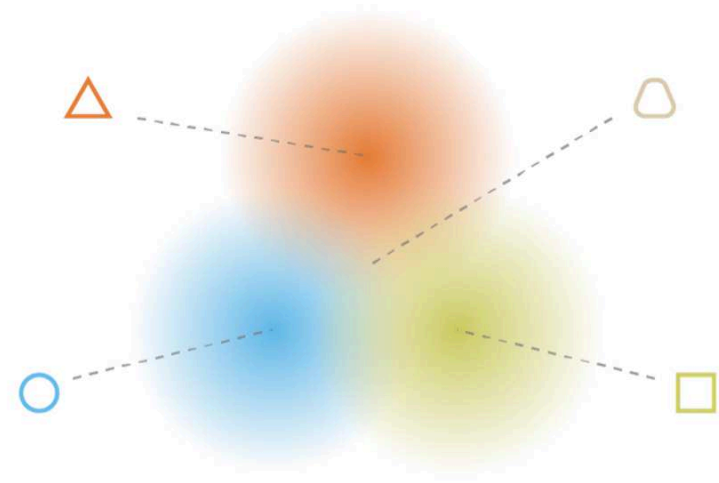


Picture Credit: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

# Autoencoder vs. Variational Autoencoder



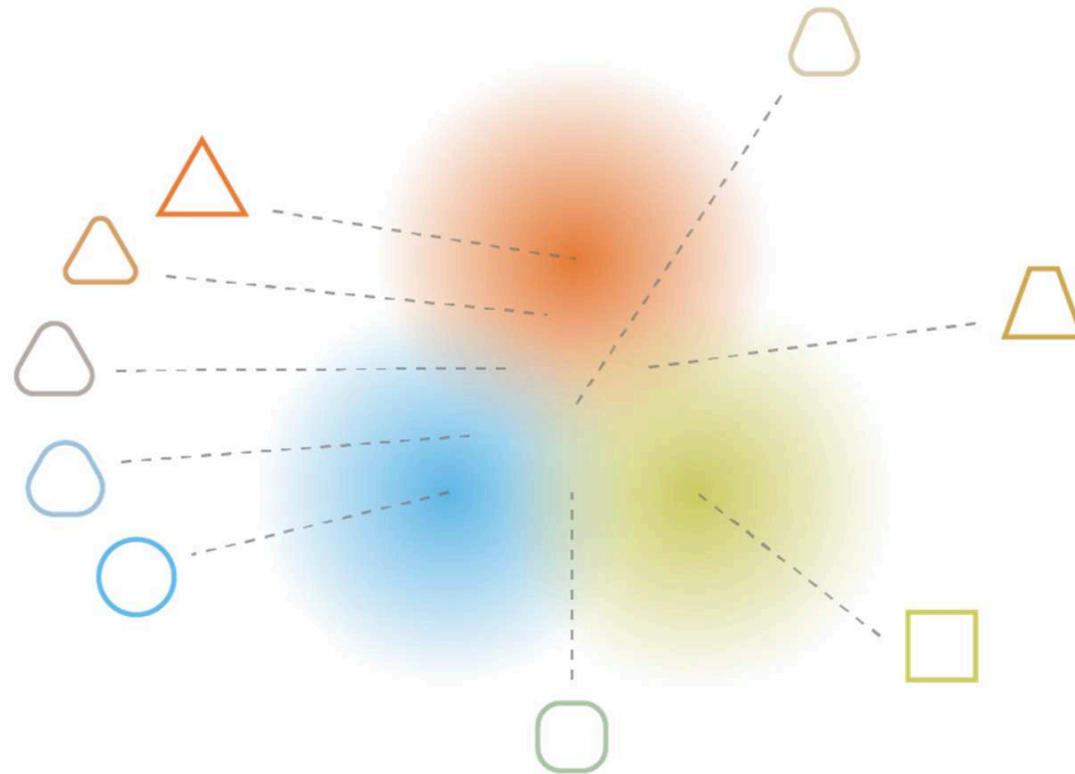
what can happen without regularisation



what we want to obtain with regularisation

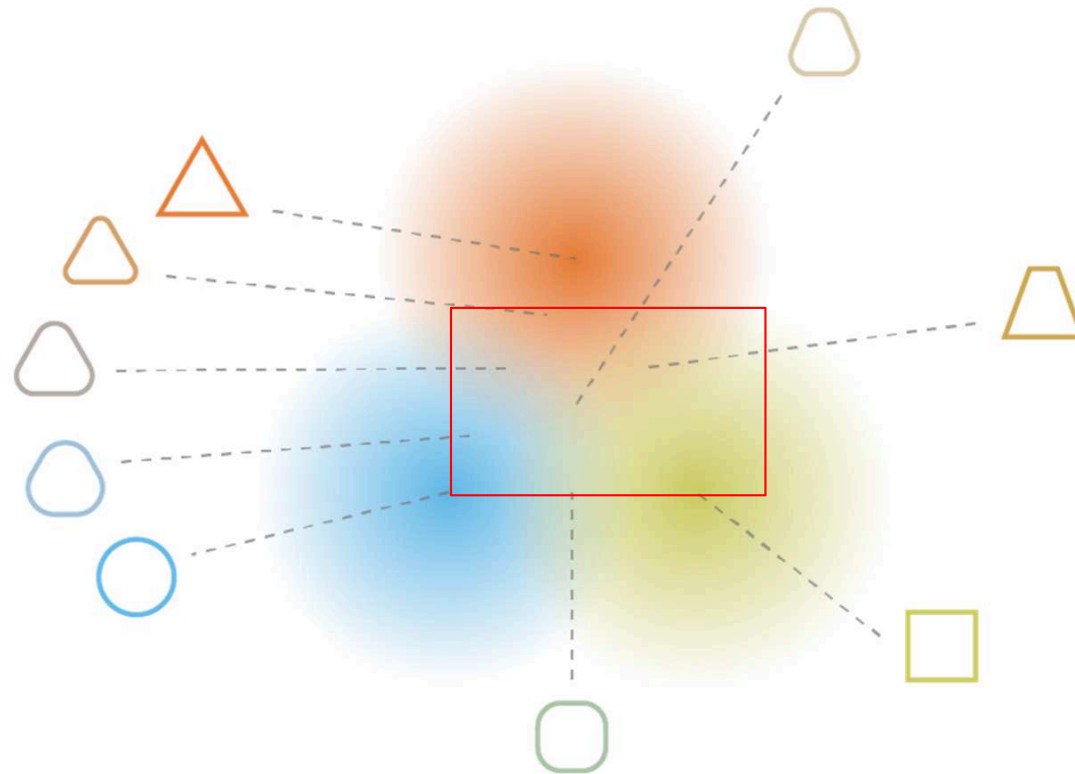
Picture Credit: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

# Autoencoder vs. Variational Autoencoder



Picture Credit: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

# Problems of VAE: Overlapping Latent Space



Picture Credit: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

# Conditional VAE

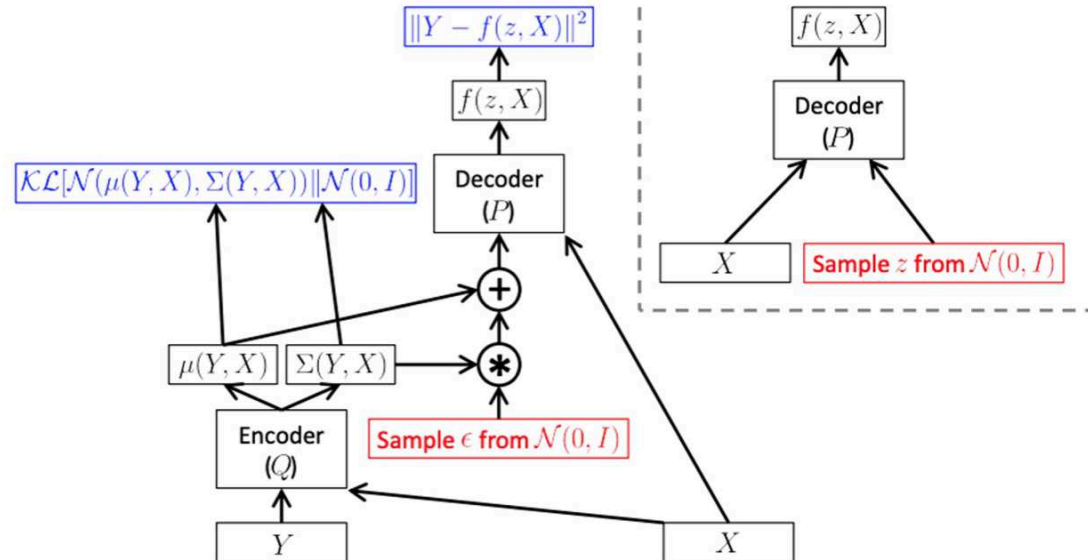


Figure 6: Left: a training-time conditional variational autoencoder implemented as a feedforward neural network, following the same notation as Figure 4. Right: the same model at test time, when we want to sample from  $P(Y|X)$ .

Picture Credit: <https://arxiv.org/pdf/1606.05908.pdf>



## Conditional VAE

$$\log p_{\theta}(\mathbf{y}|\mathbf{x}) \geq -KL(q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{y})||p_{\theta}(\mathbf{z}|\mathbf{x})) + \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{y})} [\log p_{\theta}(\mathbf{y}|\mathbf{x}, \mathbf{z})]$$

and the empirical lower bound is written as:

$$\tilde{\mathcal{L}}_{\text{CVAE}}(\mathbf{x}, \mathbf{y}; \theta, \phi) = -KL(q_{\phi}(\mathbf{z}|\mathbf{x}, \mathbf{y})||p_{\theta}(\mathbf{z}|\mathbf{x})) + \frac{1}{L} \sum_{l=1}^L \log p_{\theta}(\mathbf{y}|\mathbf{x}, \mathbf{z}^{(l)}),$$

$\mathbf{z}^{(l)} = g_{\phi}(\mathbf{x}, \mathbf{y}, \epsilon^{(l)})$ ,  $\epsilon^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  and  $L$  is the number of samples.

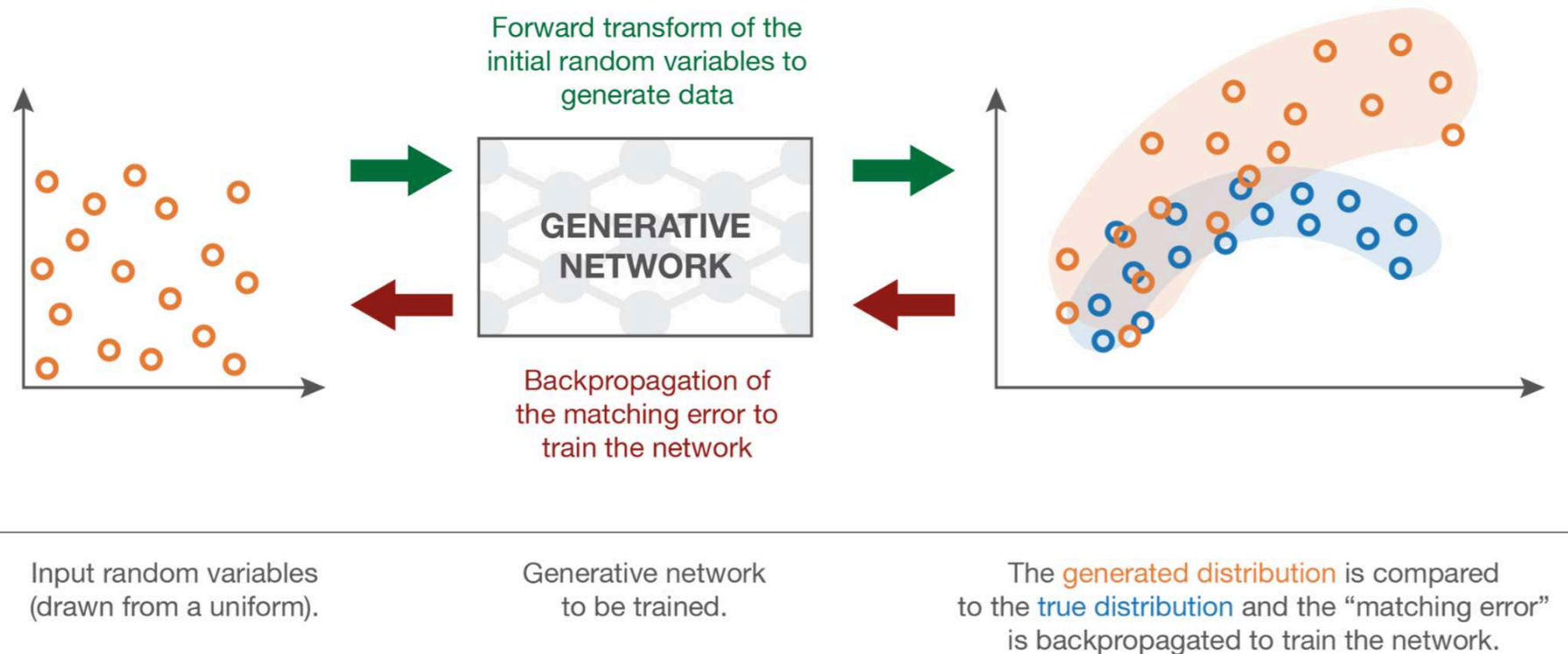
# The Reparameterization Trick in VAE

$$\begin{aligned} p(z) &\equiv \mathcal{N}(0, I) \\ p(x|z) &\equiv \mathcal{N}(f(z), cI) \quad f \in F \quad c > 0 \end{aligned}$$

Let's forget about variational inference for maximizing  $\log p(x)$  but focus on the probability distribution of  $p(x|z)$  itself, we can easily sample from  $p(x|z)$ , which leads to a nice GENERATIVE model and transforms a simple Gaussian distribution to a complex data distribution  $p_g(x)$  through a one-to-one mapping  $f: z \rightarrow x$

A direct approach to aligning our generated data distribution  $p_g(x)$  with real data distribution  $p_r(x)$  is to perform moment matching, for e.g., minimizing maximum mean discrepancy in a high-dimensional feature space induced by a kernel (kernel MMD).

# Generative Moment Matching Networks



Picture Credit: <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29>

Li *et al.*, Generative Moment Matching Networks. ICML 2015.

# An Indirect Approach for Comparing Distributions

$$p(z) \equiv \mathcal{N}(0, I)$$

$$p(x|z) \equiv \mathcal{N}(f(z), cI) \quad f \in F \quad c > 0$$

- Transform a simple Uniform/Gaussian distribution  $p(z)$  to a complex data distribution  $p_g(x)$  through a one-to-one mapping  $f: z \rightarrow x$
- An indirect approach is to assume that we have an oracle discriminator that can perfectly discriminates whether or not a data point is from the real data distribution. We can make use of this oracle discriminator to improve our generative network such that our generated data distribution perfectly aligns with the real data distribution.
- In practice, we don't have this oracle discriminator, but we can treat it as a deep neural network and learn it from data.

# Generative Adversarial Network (GAN)

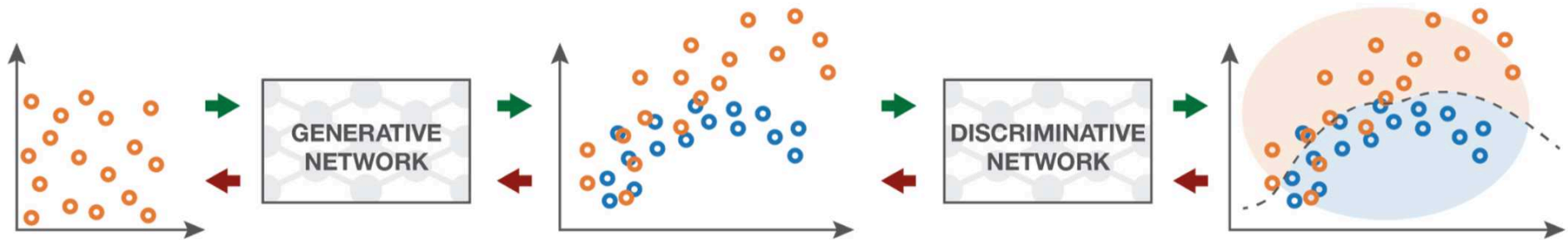
- The goal of the discriminator  $D$  is to discriminate whether a sample comes from the real data distribution (training data) or the generated data distribution (generated data).
- The goal of the generator  $G$  is to transform a simple (e.g., Gaussian, Uniform) distribution to a real data distribution such that the generated sample will fool the discriminator.
- This is a minmax two-player game. In a global optimum,  $D$  will output  $\frac{1}{2}$  everywhere and  $p_g(x) = p_r(x)$

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

# Generative Adversarial Network (GAN)

■ Forward propagation (generation and classification)

■ Backward propagation (adversarial training)



Input random variables.

The generative network is trained to **maximise** the final classification error.

The **generated distribution** and the **true distribution** are not compared directly.

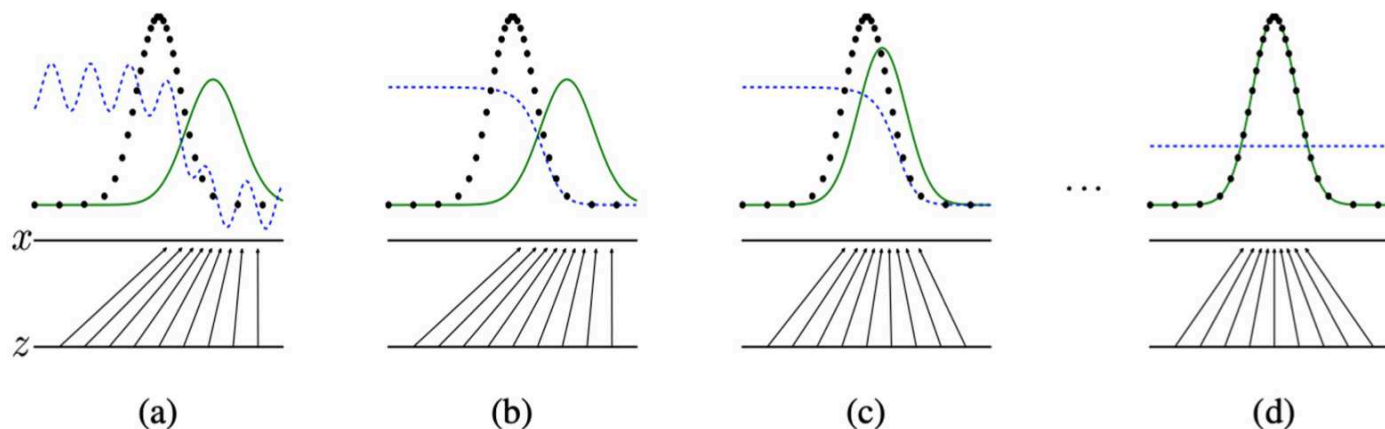
The discriminative network is trained to **minimise** the final classification error.

The classification error is the basis metric for the training of both networks.

Picture Credit: <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29>

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

# Generative Adversarial Network (GAN)



Generative adversarial nets are trained by simultaneously updating the discriminative distribution ( $D$ , blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line)  $p_x$  from those of the generative distribution  $p_g$  ( $G$ ) (green, solid line). The lower horizontal line is the domain from which  $z$  is sampled, in this case uniformly. The horizontal line above is part of the domain of  $x$ . The upward arrows show how the mapping  $x = G(z)$  imposes the non-uniform distribution  $p_g$  on transformed samples.  $G$  contracts in regions of high density and expands in regions of low density of  $p_g$ . (a) Consider an adversarial pair near convergence:  $p_g$  is similar to  $p_{\text{data}}$  and  $D$  is a partially accurate classifier. (b) In the inner loop of the algorithm  $D$  is trained to discriminate samples from data, converging to  $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$ . (c) After an update to  $G$ , gradient of  $D$  has guided  $G(z)$  to flow to regions that are more likely to be classified as data. (d) After several steps of training, if  $G$  and  $D$  have enough capacity, they will reach a point at which both cannot improve because  $p_g = p_{\text{data}}$ . The discriminator is unable to differentiate between the two distributions, i.e.  $D(x) = \frac{1}{2}$ .

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

# Optimal D of Generative Adversarial Networks

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

if  $y = a \log(y) + b \log(1 - y)$ , the optimal  $y$  is

$$\Rightarrow y^* = \frac{a}{a + b}$$

Optimize  $D(x) = p_r(x) \log D(x) + p_g(x) \log(1 - D(x))$ , we get

$$\Rightarrow D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

$$y = a \log(y) + b \log(1 - y)$$

$$y' = \frac{a}{y} - \frac{b}{1 - y}$$

$$\frac{a}{y^*} = \frac{b}{1 - y^*}$$

$$\frac{1 - y^*}{y^*} = \frac{b}{a}$$

$$\frac{1}{y^*} = \frac{a + b}{a}$$

$$y^* = \frac{a}{a + b}$$

Find optimal  $y^*$  by setting  $y' = 0$ .



# Optimal Solution of Generative Adversarial Networks

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))].$$

*With  $p = q$ , the optimal value for  $D$  and  $V$  is*

$$D^*(x) = \frac{p}{p + q} = \frac{1}{2}$$

$$\begin{aligned} \min_G \max_D V(D, G) &= \mathbb{E}_{x \sim p_r(x)} [\log \frac{1}{2}] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - \frac{1}{2})] \\ &= -2 \log 2 \end{aligned}$$

# Training Algorithm of GAN

---

**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:

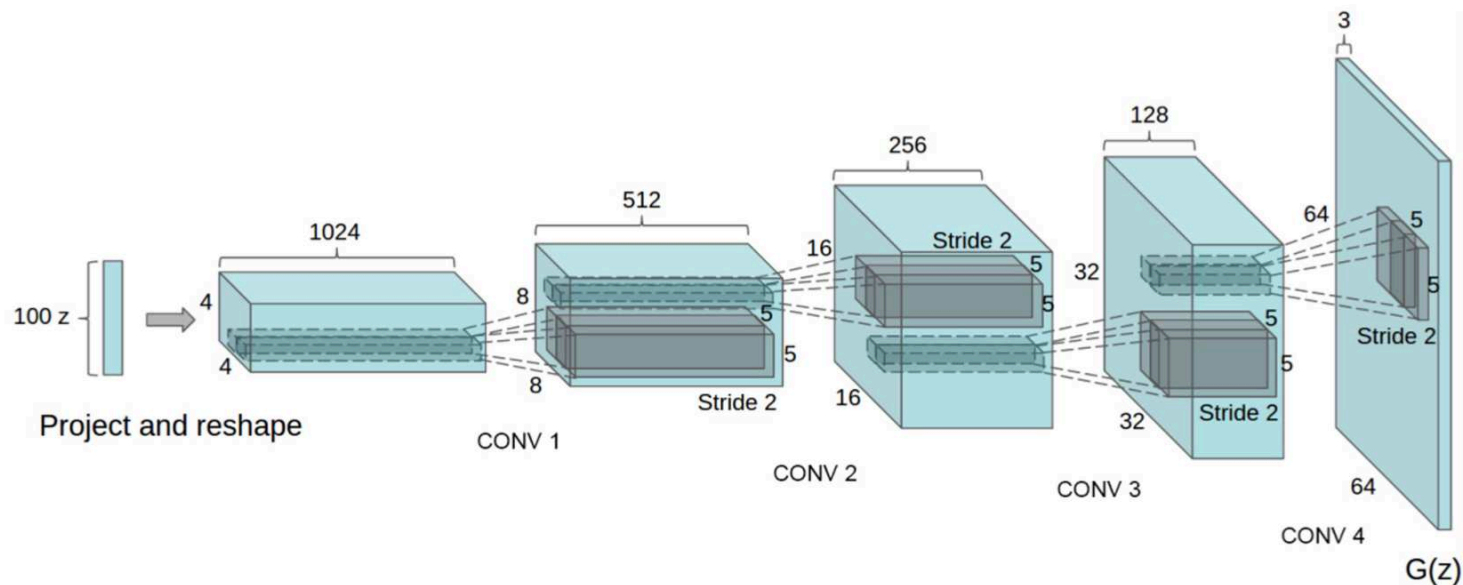
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left( 1 - D(G(\mathbf{z}^{(i)})) \right).$$

**end for**

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

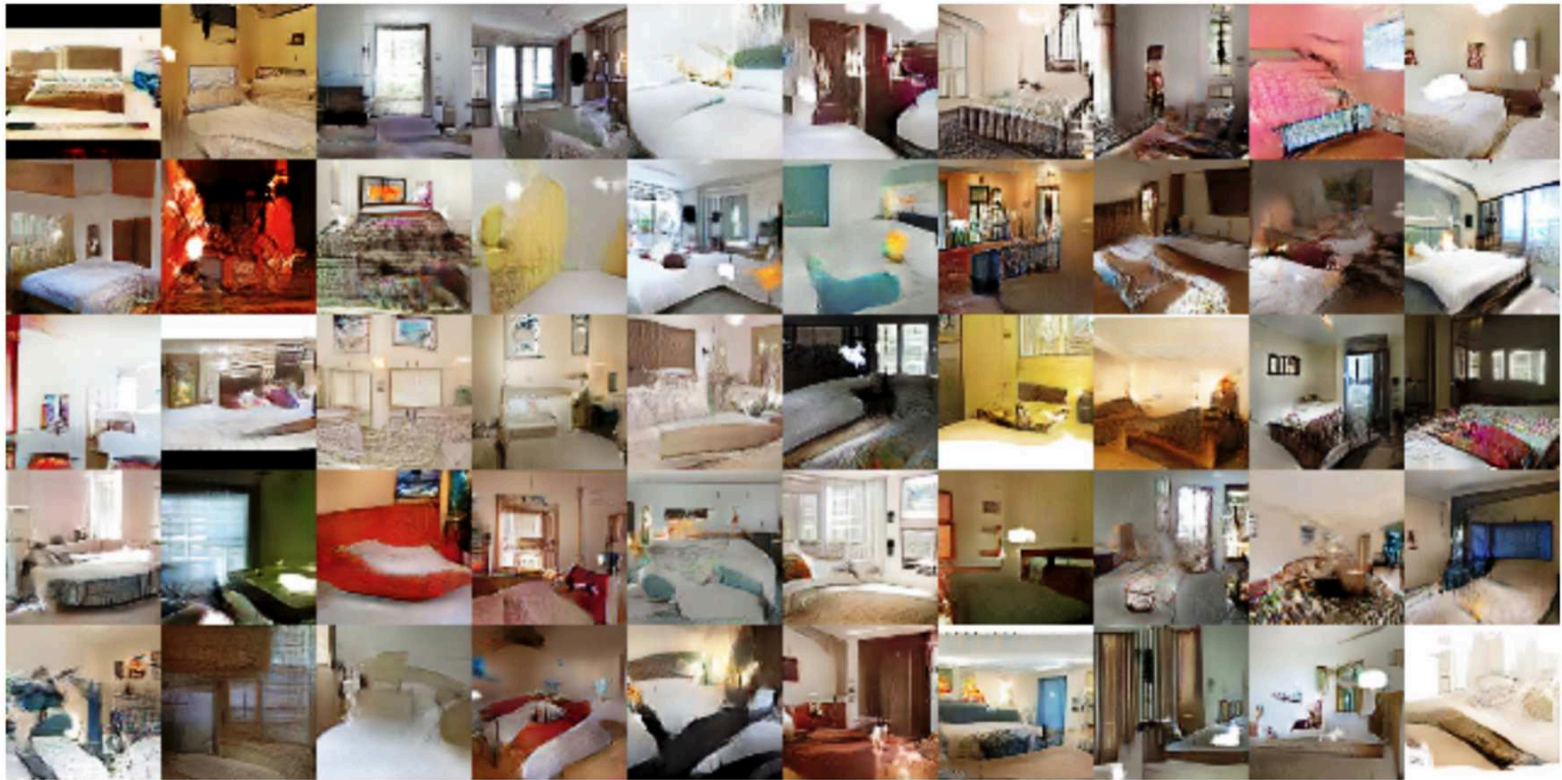
---

# Deep Convolutional GAN (DCGAN): CNN Generator



DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution  $Z$  is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a  $64 \times 64$  pixel image. Notably, no fully connected or pooling layers are used.

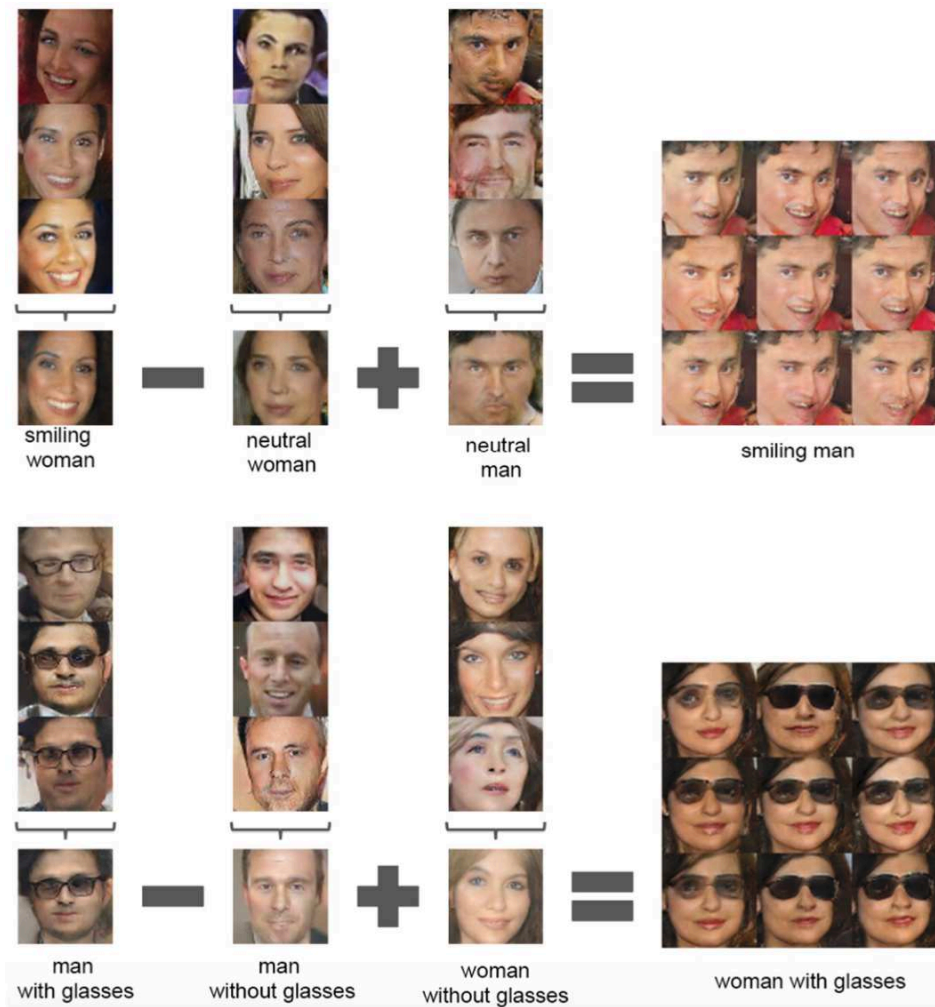
# Generated Samples of DCGAN



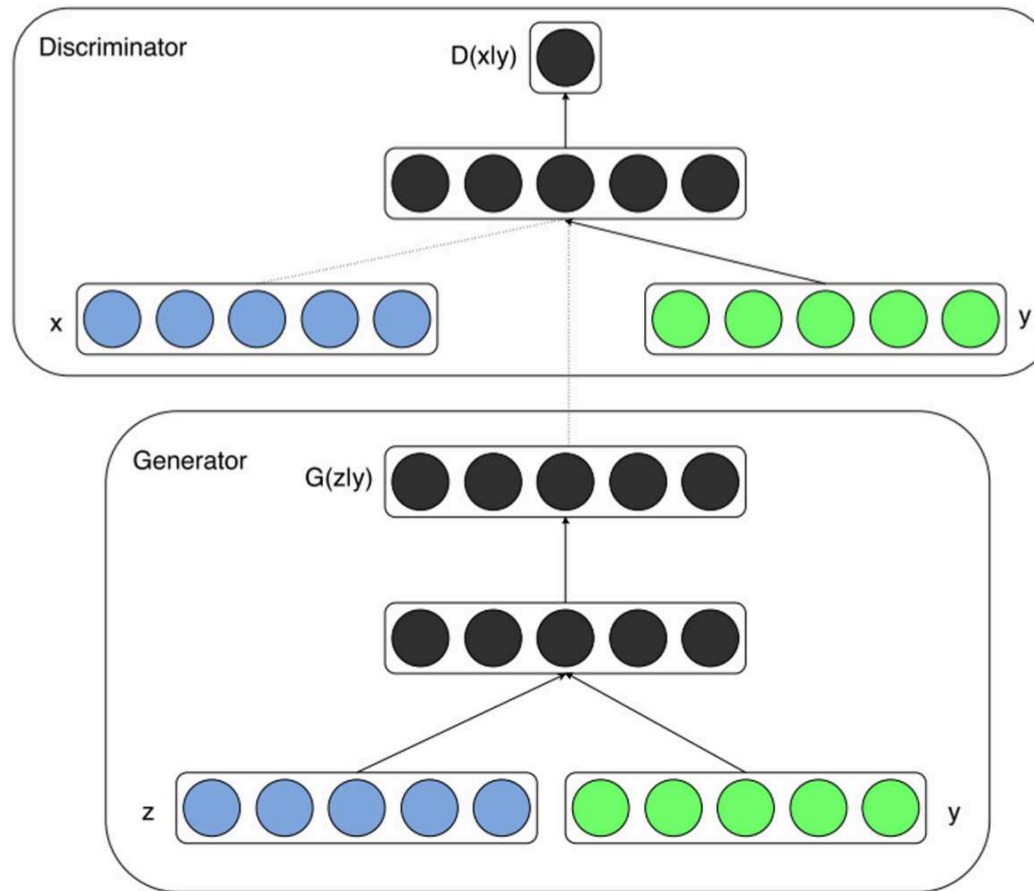
Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.



# Latent Vector (z) Manipulation Results of DCGAN

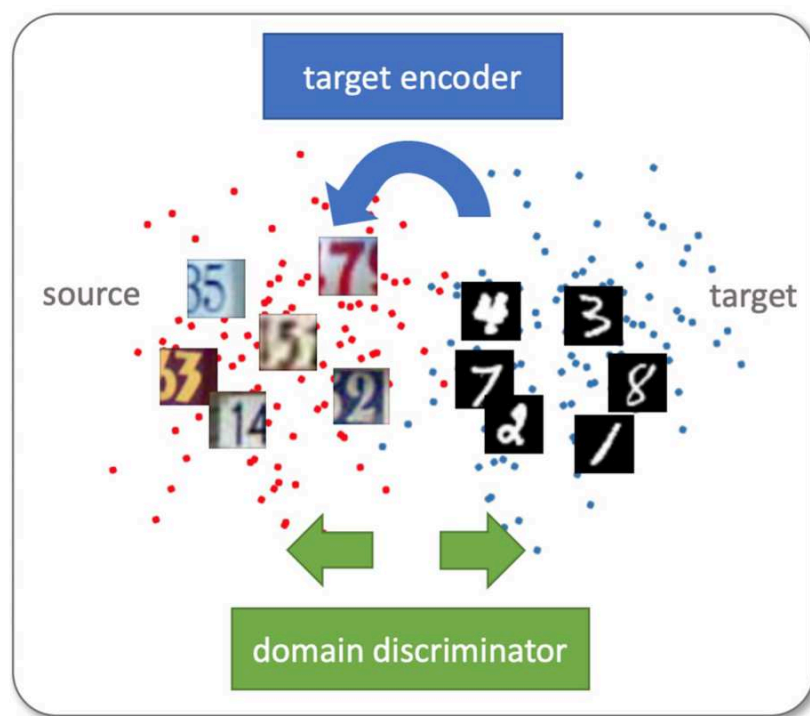


# Conditional GAN



<https://arxiv.org/pdf/1411.1784.pdf>

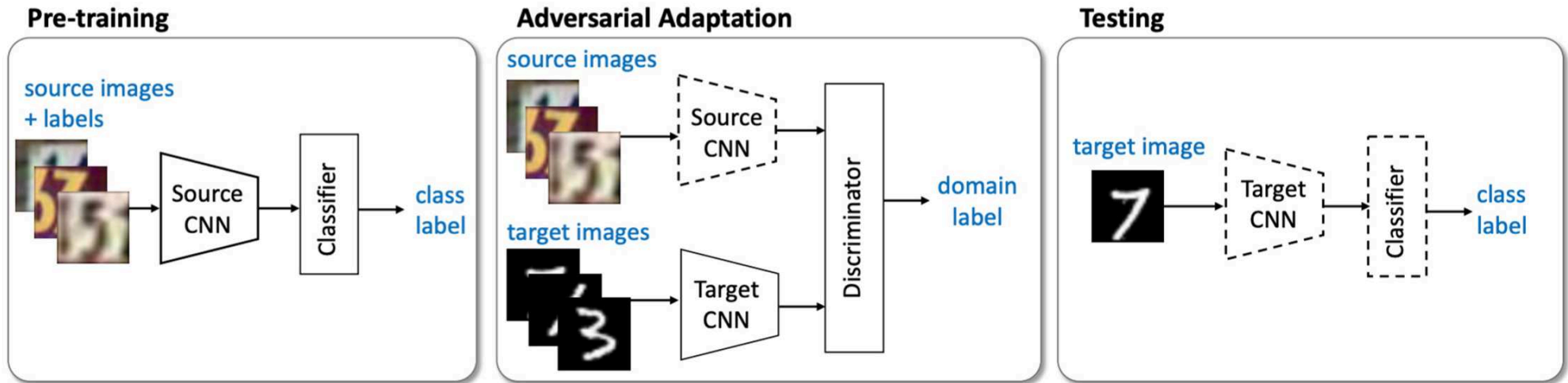
# Domain Adaptation



We have a lot of (labeled) training data in a source domain, and we plan to deploy our learned model in the source domain to a target domain that has a different data distribution from the one in the source domain.

Picture Credit: Tzeng *et al.*, Adversarial Discriminative Domain Adaptation, CVPR 2017.

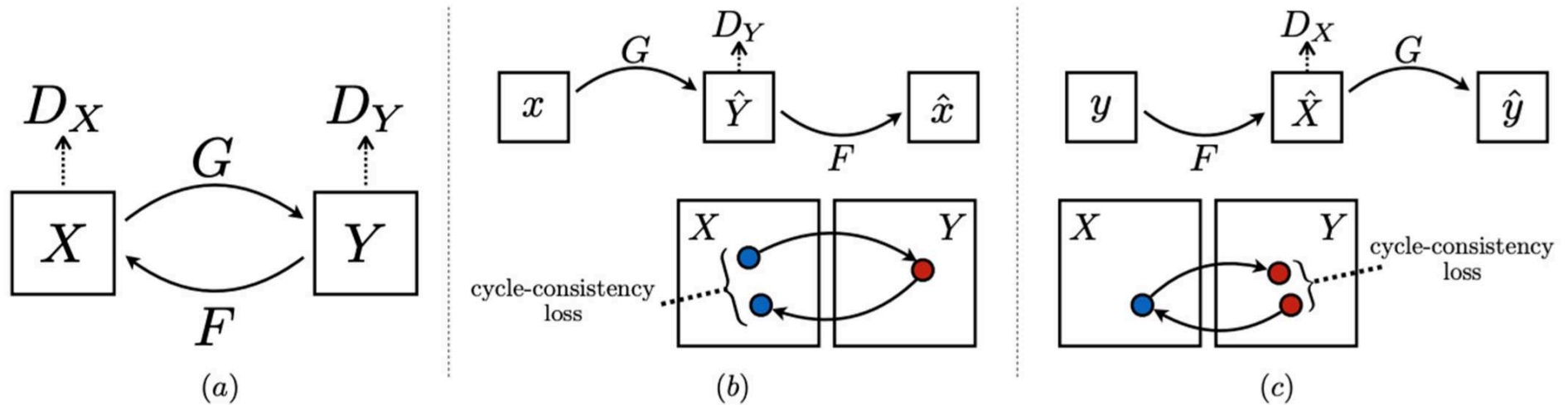
# Adversarial Feature Learning for Domain Adaptation



An overview of our proposed Adversarial Discriminative Domain Adaptation (ADDA) approach. We first pre-train a source encoder CNN using labeled source image examples. Next, we perform adversarial adaptation by learning a target encoder CNN such that a discriminator that sees encoded source and target examples cannot reliably predict their domain label. During testing, target images are mapped with the target encoder to the shared feature space and classified by the source classifier. Dashed lines indicate fixed network parameters.



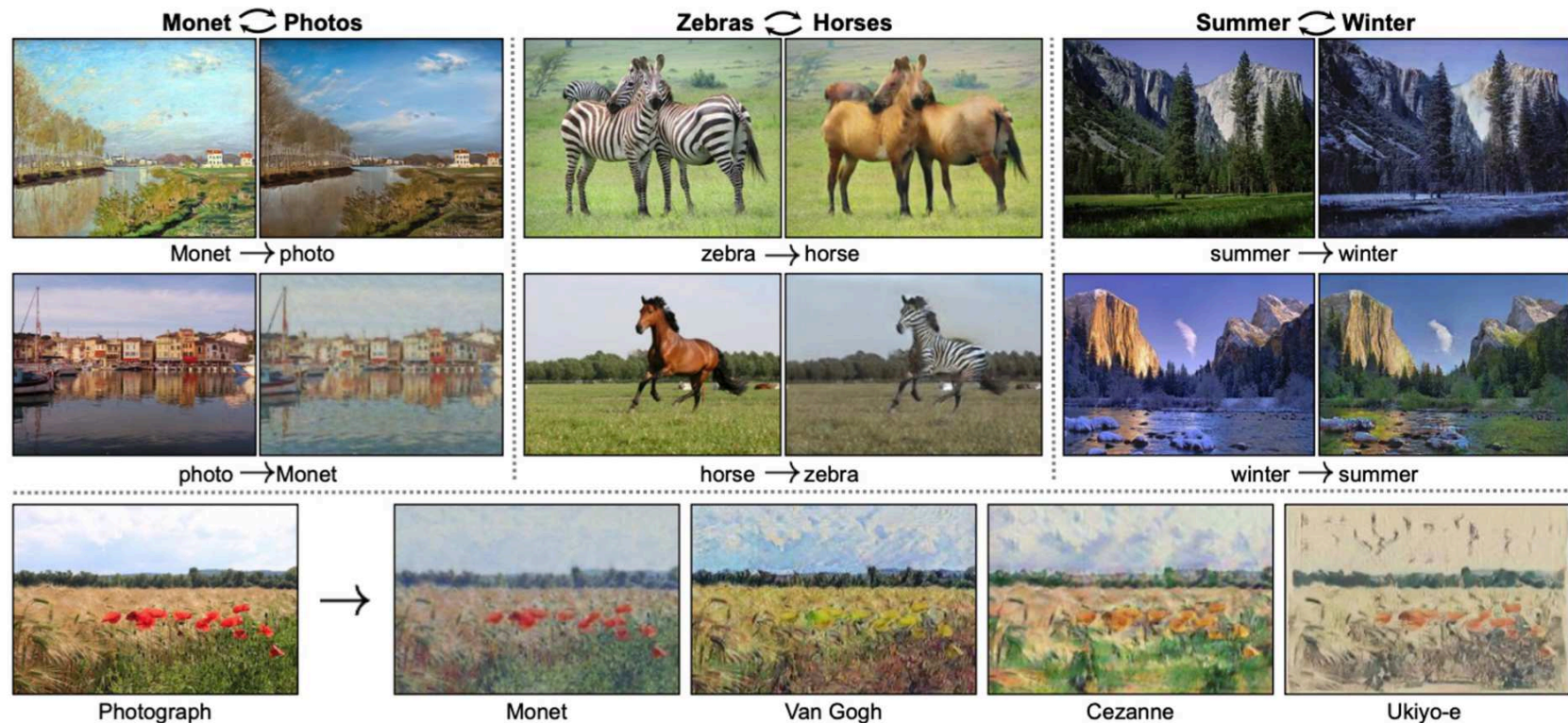
# CycleGAN



(a) Our model contains two mapping functions  $G : X \rightarrow Y$  and  $F : Y \rightarrow X$ , and associated adversarial discriminators  $D_Y$  and  $D_X$ .  $D_Y$  encourages  $G$  to translate  $X$  into outputs indistinguishable from domain  $Y$ , and vice versa for  $D_X$  and  $F$ . To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss:  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ , and (c) backward cycle-consistency loss:  $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017.

# CycleGAN Results



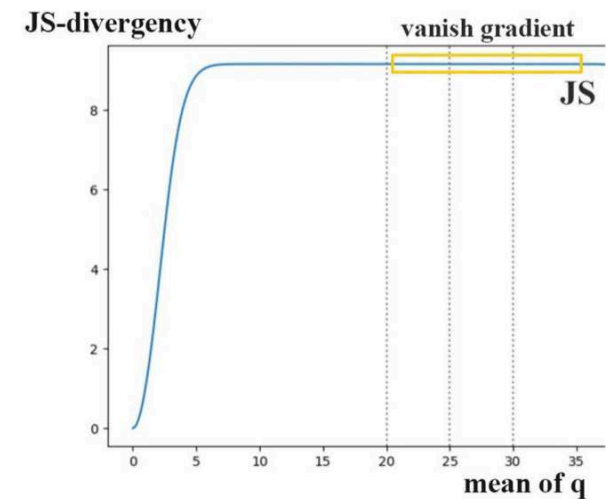
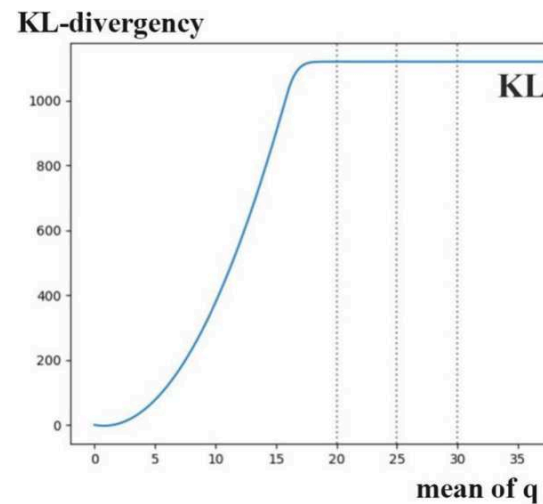
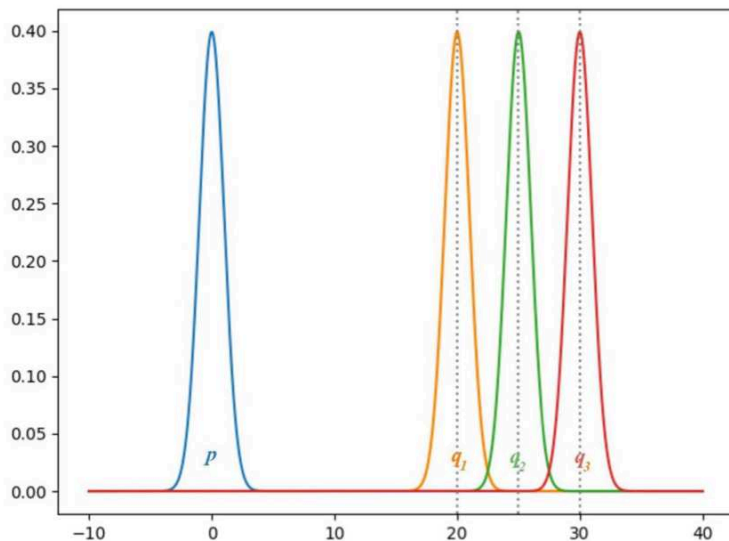
Given any two unordered image collections  $X$  and  $Y$ , our algorithm learns to automatically “translate” an image from one into the other and vice versa: (*left*) Monet paintings and landscape photos from Flickr; (*center*) zebras and horses from ImageNet; (*right*) summer and winter Yosemite photos from Flickr. Example application (*bottom*): using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017.

# GAN Minimizes JS-Divergence to Update G

$$D_{KL}(P||Q) = \sum_{x=1}^N P(x) \log \frac{P(x)}{Q(x)} \quad \text{for VAE}$$

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2}) \quad \text{for GAN}$$

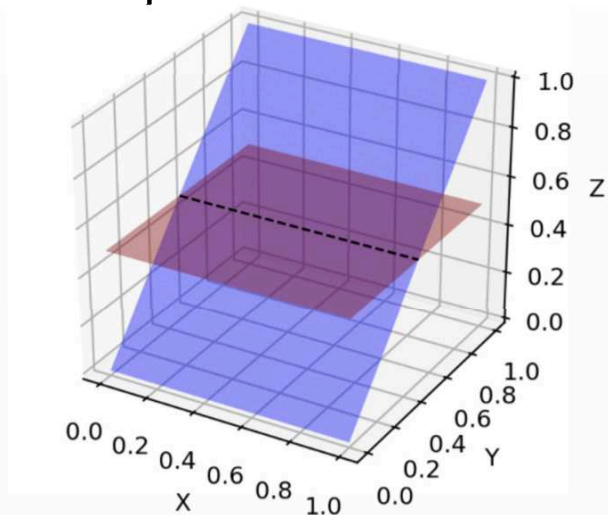
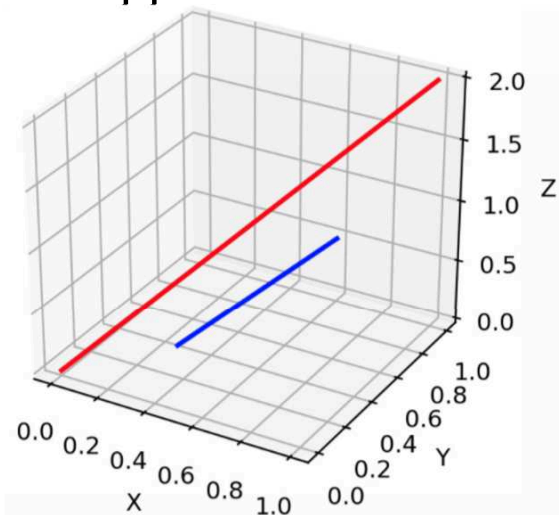


Picture Credit: [https://medium.com/@jonathan\\_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490](https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490)

## Real Image/Video Data is often Supported in a Low-D Manifold

For e.g. MNIST digits, ImageNet Images, Videos, although the pixel space is very high-dimensional.

It's easy to find a perfect discriminator to separate high-dimensional data supported in low-dimensional space.



$$-\nabla_{\theta_g} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \rightarrow \mathbf{0}$$

original GAN generator's gradient

Picture Credit: <https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html>

# Problems of GAN

The minmax training of GAN doesn't necessarily converge in practice:

If we have a perfect discriminator in the beginning, the gradient of the loss function with respect to generator parameters is close to zero and the learning is very slow

If we have a very bad discriminator, we don't get much useful feedback from the discriminator.

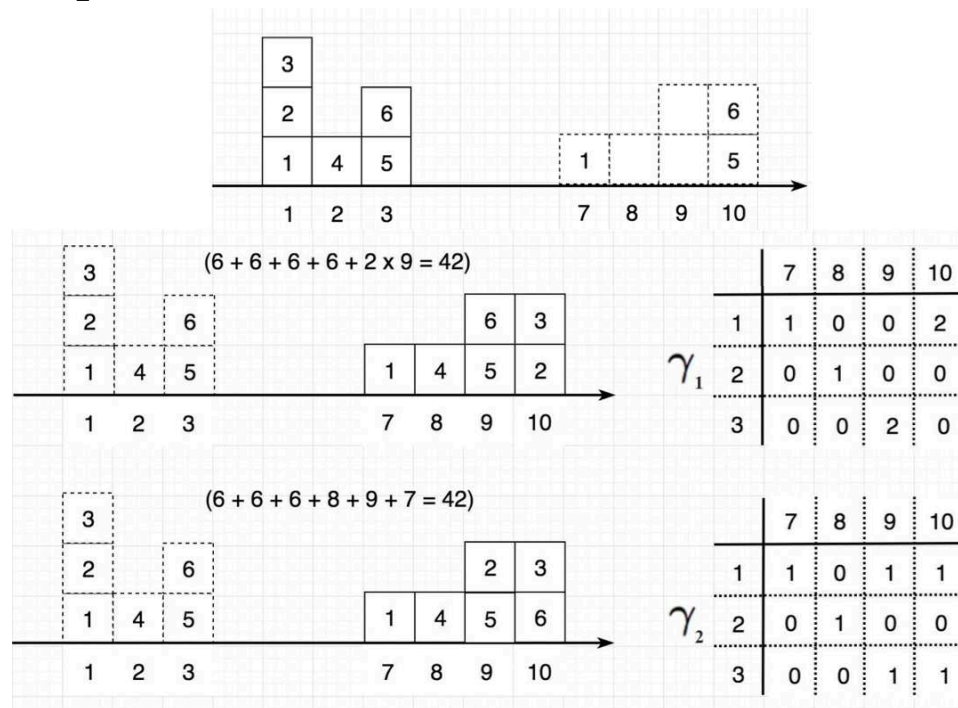
Training can be unstable.

Mode collapse: the generator only generates a subset of training data distribution modes to fool the discriminator and fails to explore other modes.



# Wasserstein Distance

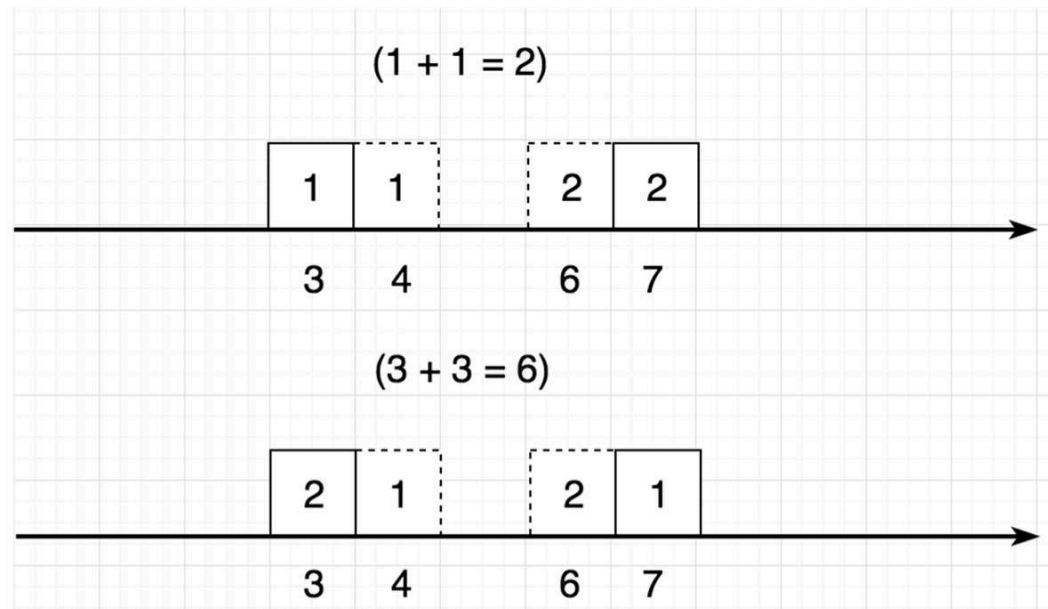
The Wasserstein distance of  $\mathbf{p}$  and  $\mathbf{q}$  is the minimum cost of transporting mass in converting the shape of a data distribution  $\mathbf{q}$  to the shape of a data distribution  $\mathbf{p}$ . It is also called Optimal Transport Cost or Earth Mover Distance.



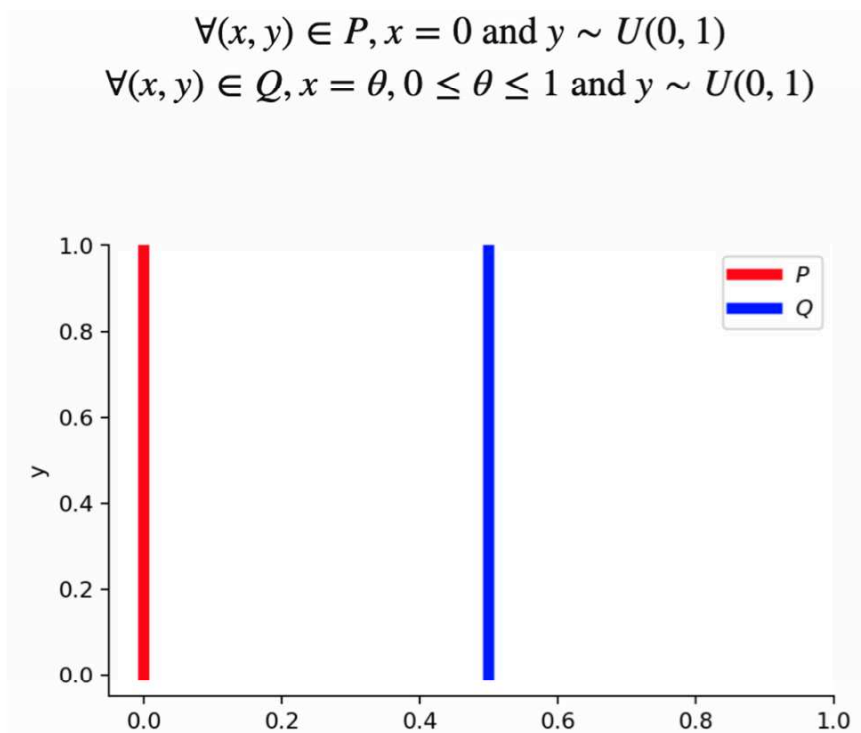
Picture Credit: [https://medium.com/@jonathan\\_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490](https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490)

# Wasserstein Distance

The Wasserstein distance of  $\mathbf{p}$  and  $\mathbf{q}$  is the minimum cost of transporting mass in converting the shape of a data distribution  $\mathbf{q}$  to the shape of a data distribution  $\mathbf{p}$ . It is also called Optimal Transport Cost or Earth Mover Distance.



# Comparing Wasserstein Distance with KLD and JSD



When  $\theta \neq 0$ :

$$D_{KL}(P||Q) = \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q||P) = \sum_{x=\theta, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{JS}(P, Q) = \frac{1}{2} \left( \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=\theta, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} \right) = \log 2$$

$$W(P, Q) = |\theta|$$

But when  $\theta = 0$ , two distributions are fully overlapped:

$$D_{KL}(P||Q) = D_{KL}(Q||P) = D_{JS}(P, Q) = 0$$

$$W(P, Q) = 0 = |\theta|$$

Picture Credit: <https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html>



## Wasserstein GAN (WGAN) Minimizing Wasserstein Distance between $p_g$ and $p_r$

Using the Kantorovich-Rubinstein duality, we can simplify the calculation to

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

Arjovsky *et al.*, Wasserstein Generative Adversarial Networks. ICML 2017.

## WGAN vs. GAN

**Discriminator/Critic**

**Generator**

**GAN**

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

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

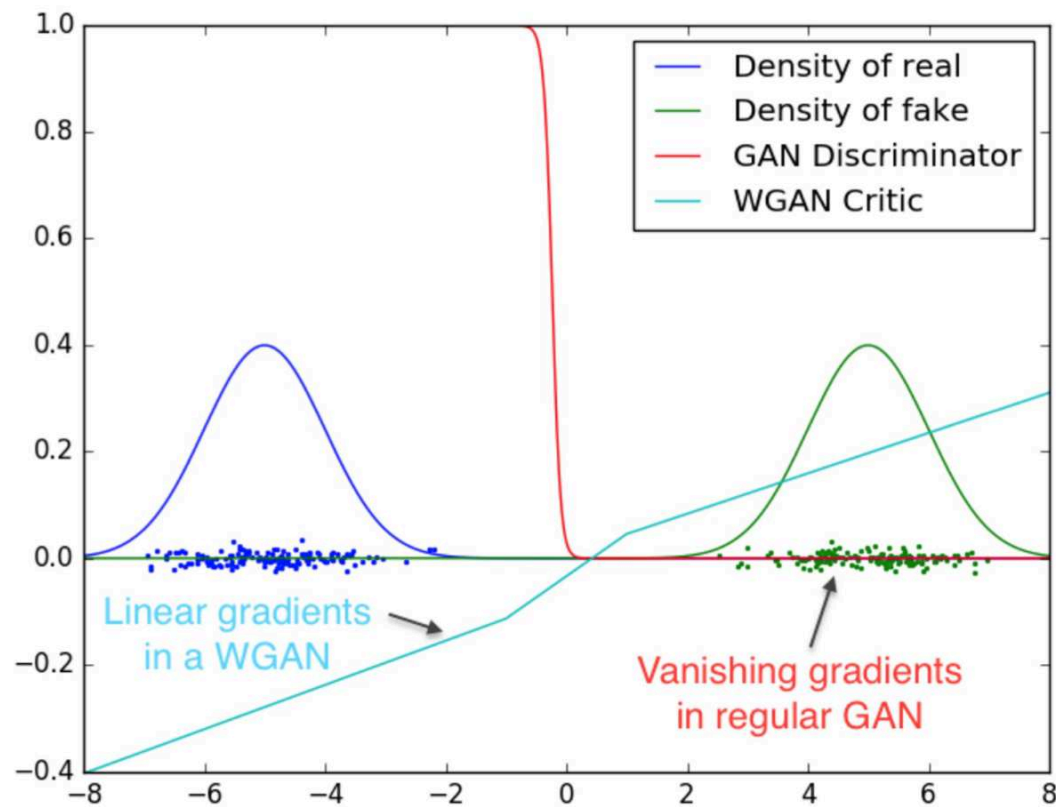
**WGAN**

$$\nabla_w \frac{1}{m} \sum_{i=1}^m [f(\mathbf{x}^{(i)}) - f(G(\mathbf{z}^{(i)}))]$$

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f(G(\mathbf{z}^{(i)}))$$

In WGAN, we have a critic with a scalar output without log

## WGAN vs. GAN



Arjovsky *et al.*, Wasserstein Generative Adversarial Networks. ICML 2017.

# Training Algorithm of WGAN

---

**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:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  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
```

---

# Text2Video: Goals and Challenges

Build a conditional generative model to generate videos from text capturing different contextual semantics of natural language descriptions

Capable of capturing both static content and dynamic motion features of videos

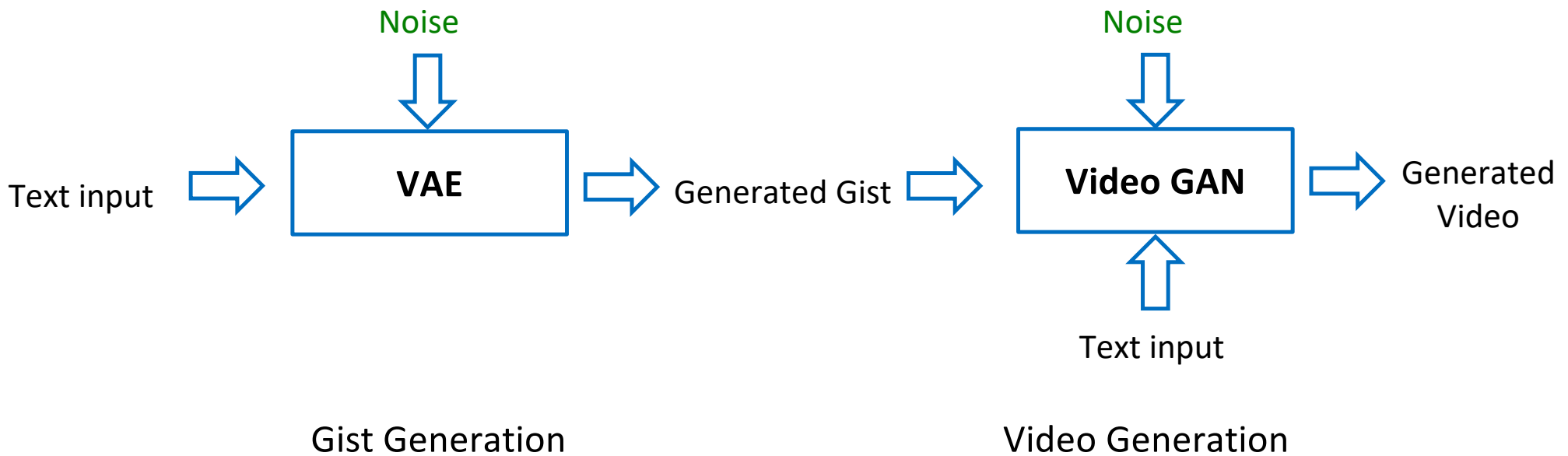
## Challenges

- It's hard to condition on text, a big gap
- It is hard to build powerful video generator
- No publicly available dataset

How? Integrating VAE and GAN

# Model Overview

- We introduce an intermediate step called 'Gist' Generation.
- The model is trained end-to-end.



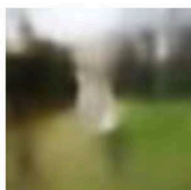
# Generated Video Samples

Text input

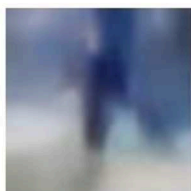
Generated gist

Generated video

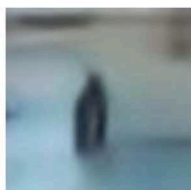
Play golf on grass



Play golf on snow



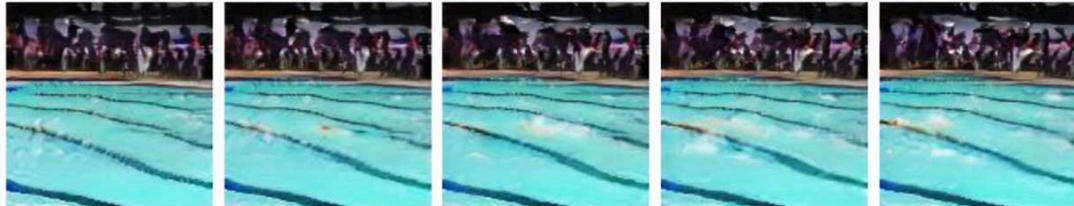
Play golf on water



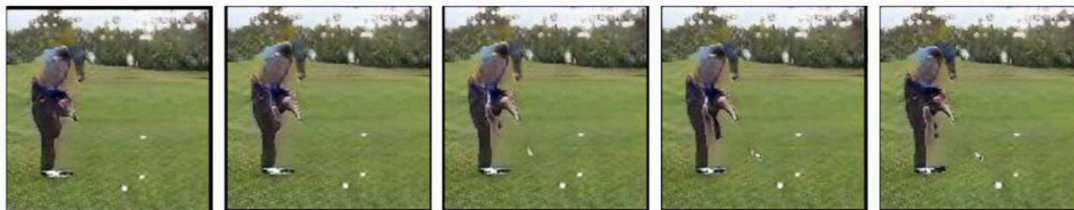


# Generated Videos

*People  
swimming in the  
pool*



*Play golf on  
grass*

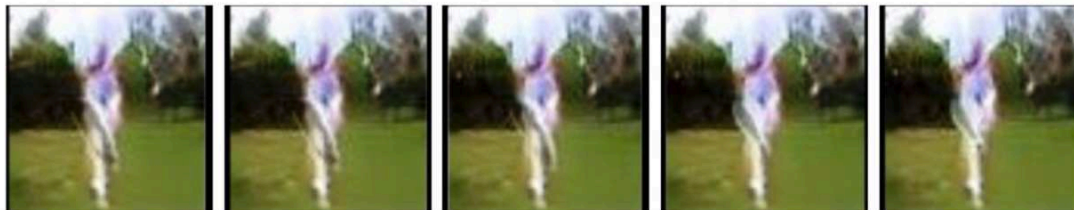


*A boat sailing  
in the sea*



**TFGAN  
(Ours)**

*Play golf on  
grass*

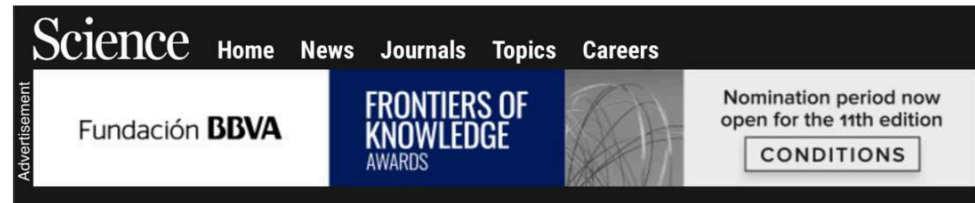


**Li et al.  
(2018)**

**Previous  
Model**



# Media Reports from Science, MIT Technology Review, Communications of ACM, etc.



SHARE



796



46



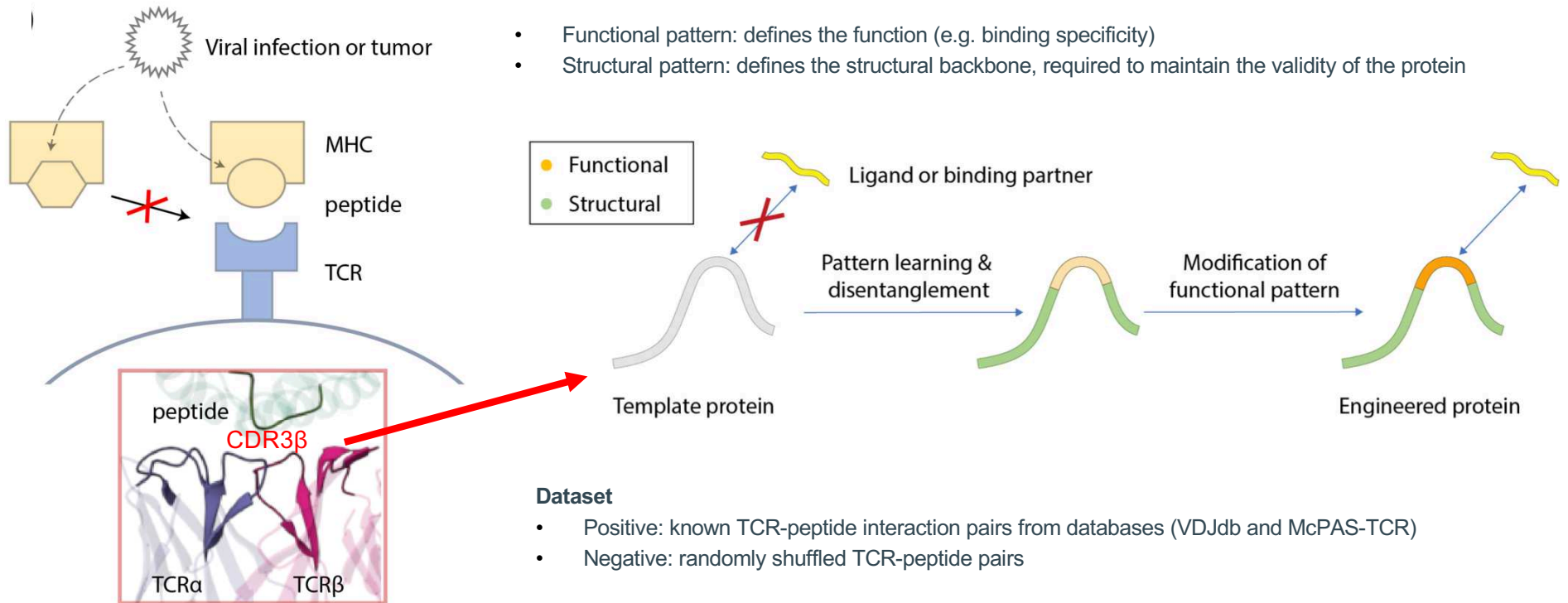
Artificial intelligence is moving into movie production. SHAREGRID/UNSPLASH

## New algorithm can create movies from just a few snippets of text

By [Matthew Hutson](#) | Feb. 23, 2018, 4:35 PM

Li, Min, *et al.*, AAAI 2018

# Disentangled Wasserstein Autoencoder for T-Cell Receptor Engineering (NeurIPS 2023)

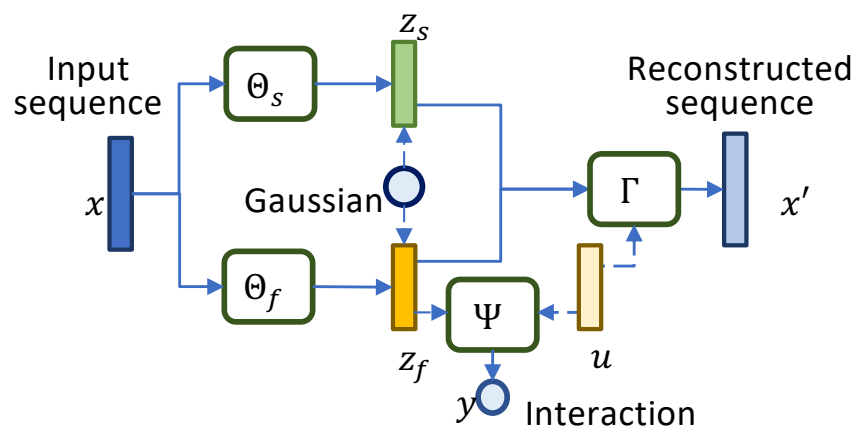


## Dataset

- Positive: known TCR-peptide interaction pairs from databases (VDJdb and McPAS-TCR)
- Negative: randomly shuffled TCR-peptide pairs

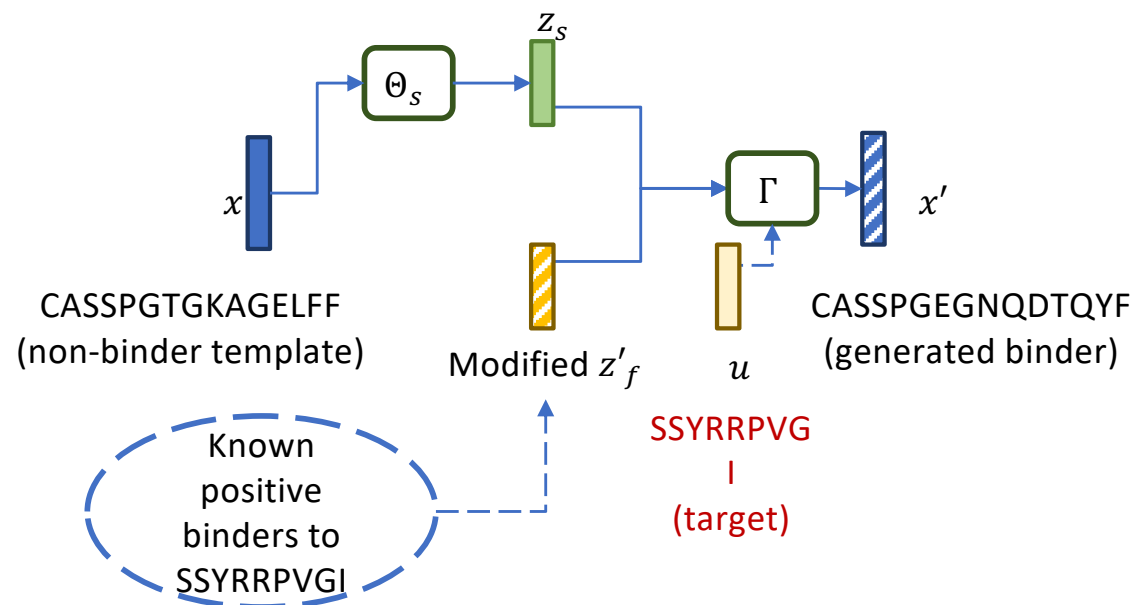
Objective: modify the binding specificity given a TCR CDR3 $\beta$  and a target peptide, by only altering its functional pattern.

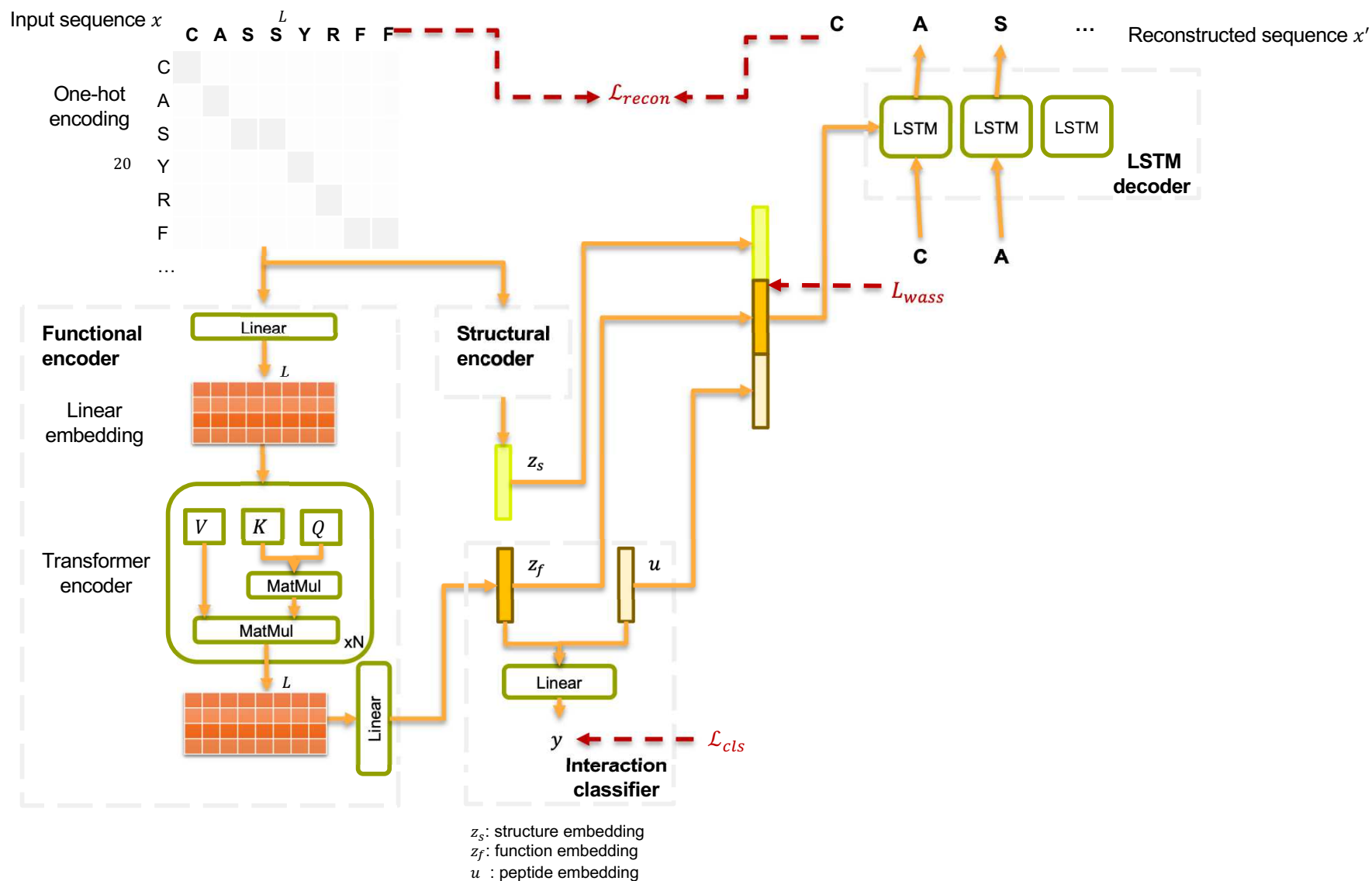
## Training



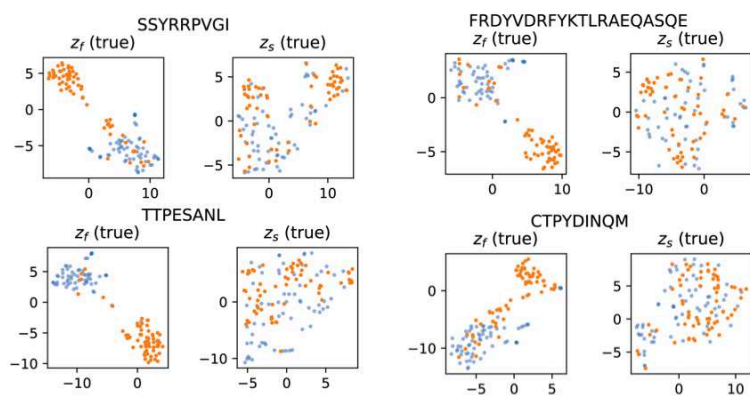
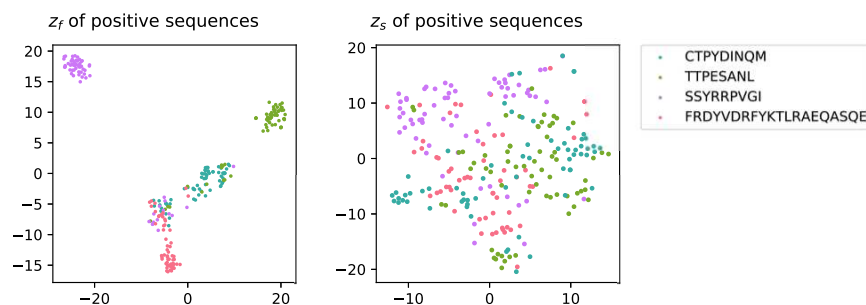
- Reconstruction loss  $\mathcal{L}_{recon}(x, x')$
- Classification loss  $\mathcal{L}_{cls}(\Psi(u, z_f), y)$
- Wasserstein loss  $L_{wass}((z_f, z_s), N(0, I))$

## Optimization





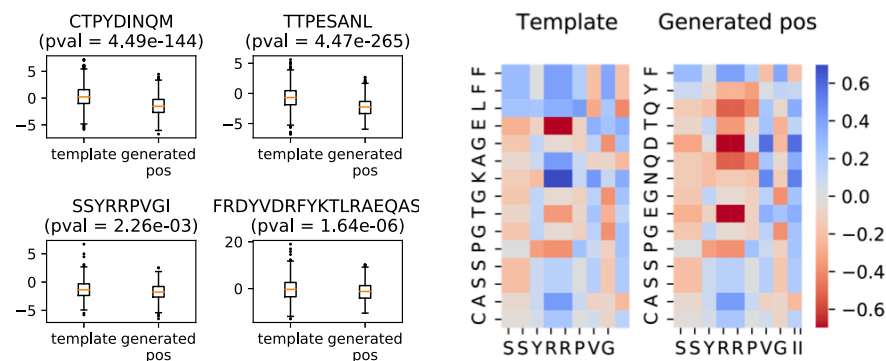
## Disentanglement analysis



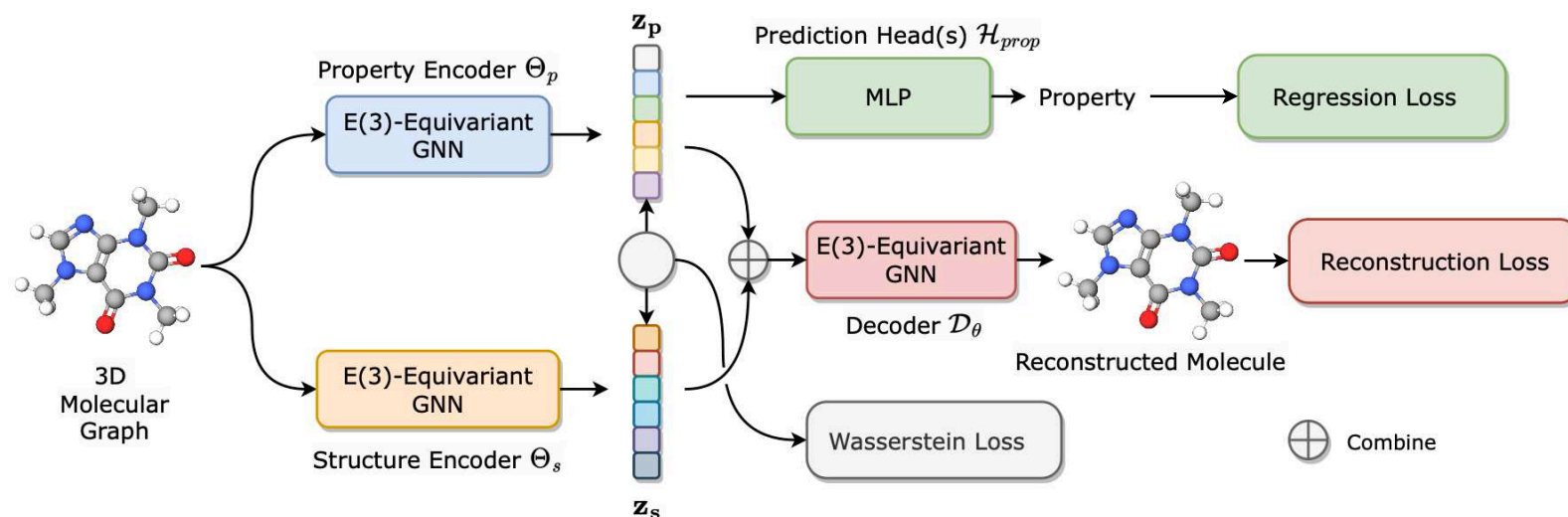
## TCR engineering results

	%valid	#mut/len	%positive valid ↑
TCR-dWAE	0.61±0.06	0.49±0.05	<b>0.23±0.02</b>
TCR-dVAE	0.64±0.05	0.4±0.02	0.16±0.01
greedy	0.02±0.0	0.34±0.0	0.02±0.0
genetic	0.02±0.0	NA	0.02±0.0
naive rm	0.03±0.0	0.35±0.01	0.0±0.0
MCTS	0.0±0.0	NA	0.0±0.0
TCR-dWAE (null)	0.85±0.01	0.45±0.07	0.04±0.03
original	0.92	NA	0.01

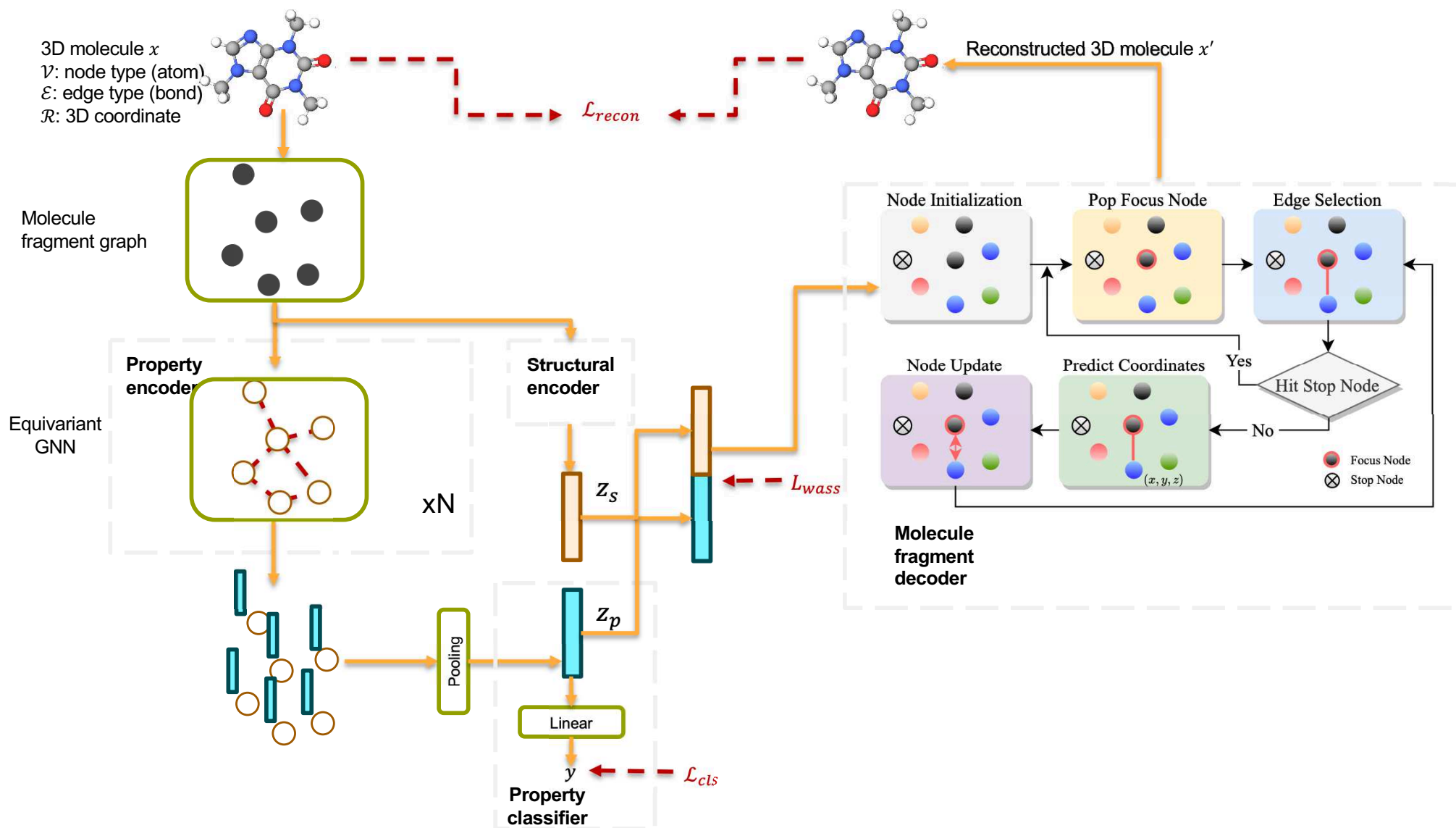
## Miyazawa-Jerningen energy



# Learning Disentangled Equivariant Representation for Explicitly Controllable 3D Molecule Generation (AAAI 2025)



- Reconstruction loss  $\mathcal{L}_{recon}(x, x') = \mathcal{L}_{nodeType} + \mathcal{L}_{edge} + \mathcal{L}_{coords}$
- Classification loss  $\mathcal{L}_{cls}(\Psi(z_p), y)$
- Wasserstein loss  $L_{wass}((z_p, z_s), N(0, I))$



---

Algorithm 1: 3D Molecule Graph Reconstruction in E3WAE

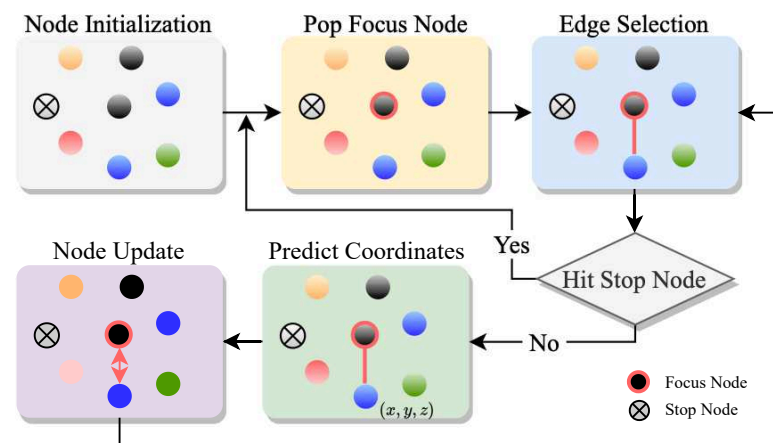
---

```

1: Input: Latent variables  $\mathbf{z}_h, \mathbf{z}_v$ 
2: Initialize:
    $\{x_i\}_{i=1, \dots, n} \leftarrow \text{NodeTypes}(\mathbf{z}_h, \mathbf{z}_v)$ 
   Queue  $Q \leftarrow \emptyset$ 
    $Q.\text{push}(\text{RandomSelect}(\{1, \dots, n\}))$ 
   3D Graph  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{R}\}$ , where  $\mathcal{V} \leftarrow \emptyset, \mathcal{E} \leftarrow \emptyset, \mathcal{R} \leftarrow \emptyset$ 
3: while  $Q \neq \emptyset$  do
4:    $f \leftarrow Q.\text{pop}()$ 
5:   Add node  $f$  to graph  $\mathcal{G}$ 
6:   isStopNode  $\leftarrow \text{False}$ 
7:   while not isStopNode do
8:      $(i, \text{isStopNode}) \leftarrow \text{PredictEdge}(f, \mathcal{G})$ 
9:     if not isStopNode and FirstLink( $i$ ) then
10:       $\mathbf{r}_u = (x_u, y_u, z_u) \leftarrow \text{PredictCoords}(i)$ 
11:       $\mathcal{V} \leftarrow \mathcal{V} \cup \{i\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(f, i)\},$ 
12:       $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathbf{r}_i\}$ 
13:       $Q.\text{push}(i)$ 
14:    end if
15:    MarkVisited( $f$ )
16:  end while
17: end while
18: Return: Reconstructed  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ 

```

---





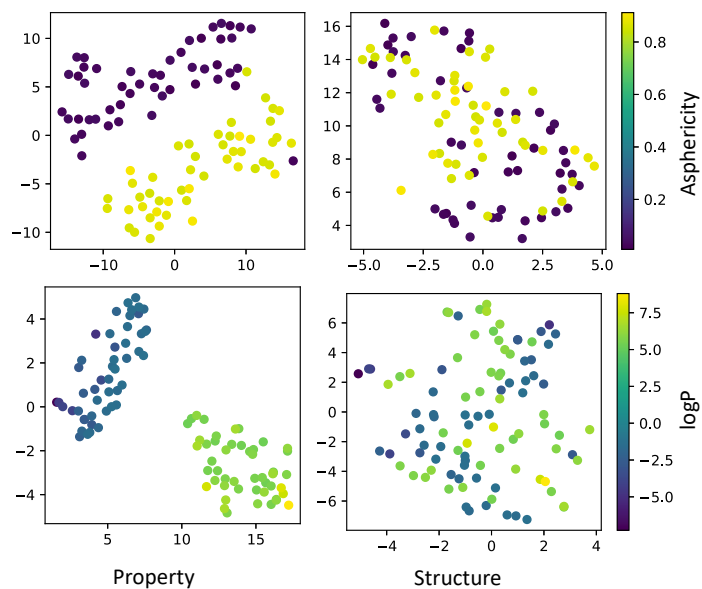
## Property-guided generation

		Isolated molecule			
		Asphericity	QED	SAS	logP
		MSE/MAE	MSE/MAE	MSE/MAE	MSE/MAE
GEOM	EDM	0.626/0.455	0.113/0.285	12.395/3.385	5.704/1.903
	HierDiff	0.176/0.406	0.120/0.289	2.618/1.347	<b>4.124/1.572</b>
	Ours	<b>0.095/0.246</b>	<b>0.072/0.221</b>	<b>1.563/1.002</b>	4.490/1.630
Cross-Docked 2020	EDM	0.109/0.274	0.147/0.309	11.244/3.205	5.715/1.957
	HierDiff	0.107/0.268	0.089/0.278	2.364/1.468	5.752/1.897
	Ours	<b>0.100/0.259</b>	<b>0.062/0.205</b>	<b>2.356/1.243</b>	<b>4.244/1.644</b>

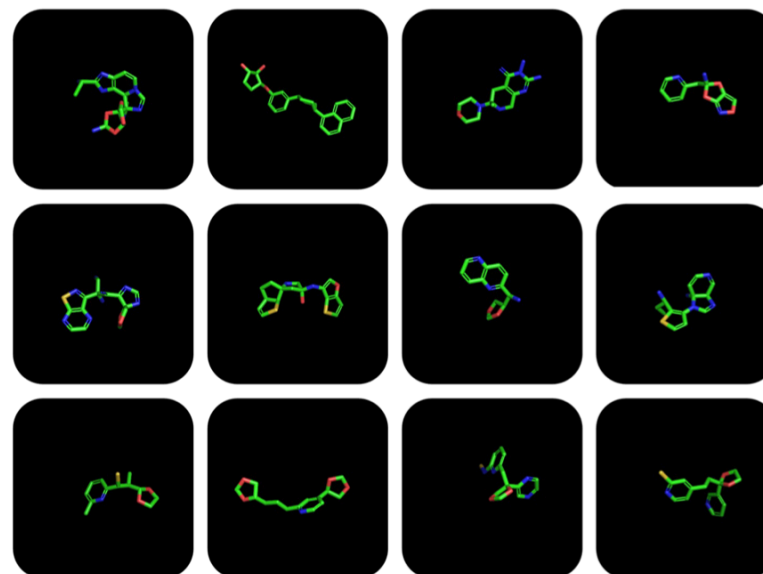
Ligand

	Asphericity			QED			SAS			logP		
	Vina	MSE	MAE	Vina	MSE	MAE	Vina	MSE	MAE	Vina	MSE	MAE
HierDiff	-4.253	0.125	0.296	-4.429	0.113	0.275	-5.053	2.364	1.620	-4.293	5.938	1.855
TargetDiff	-5.742	0.117	0.288	-5.706	0.112	0.307	-5.479	3.369	1.604	-5.501	4.509	1.946
Ours	<b>-5.891</b>	<b>0.102</b>	<b>0.271</b>	<b>-5.866</b>	<b>0.086</b>	<b>0.247</b>	<b>-5.940</b>	<b>2.358</b>	<b>1.529</b>	<b>-5.827</b>	<b>4.376</b>	<b>1.794</b>

## Disentanglement analysis



## Generated molecules



The End