Variational AutoEncoder (VAE), Generative Adversarial Networks (GAN)

Shikui Tu

Department of Computer Science and Engineering, Shanghai Jiao Tong University

2021-05-21

Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

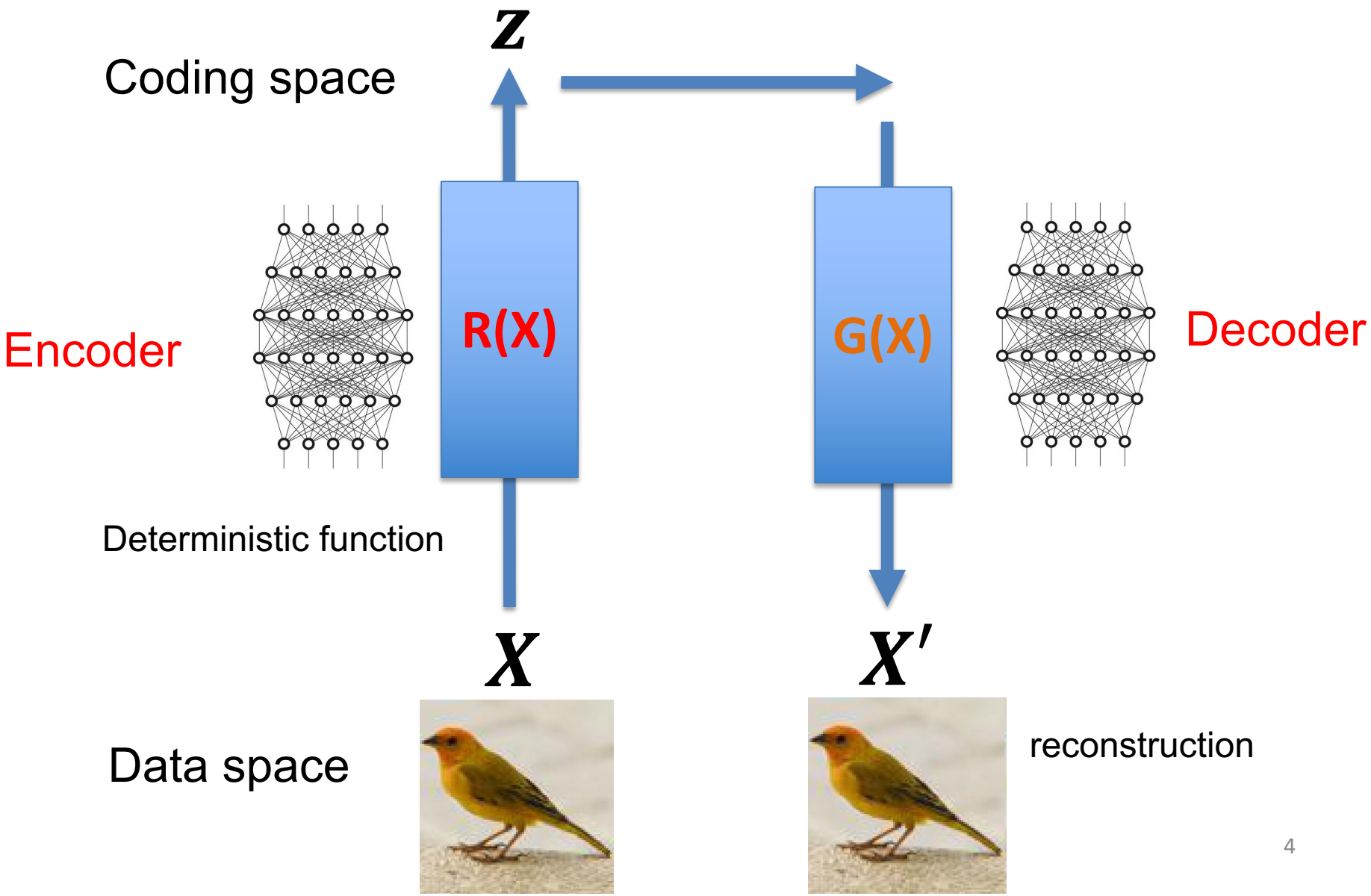
How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

Auto-encoding variational bayes
DP Kingma, M Welling - arXiv preprint arXiv:1312.6114, 2013 - arxiv.org
How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales …
☆ 99 Cited by 14057 Related articles All 28 versions »

Outline

- Related concepts
  - Autoencoder (AE)
  - Generative models
  - EM algorithm
  - Variational lower bound
- Variational Autoencoder (VAE)
- Generative Adversarial Networks (GAN)
- VAE vs. GAN

Autoencoder (AE)



A view of CNN

Output (e.g., “cat” class)

Internal representation
/ embedding / features

Input (e.g., image)

Decoder

Output layer

Softmax classifier

Layer L - 1

Encoder

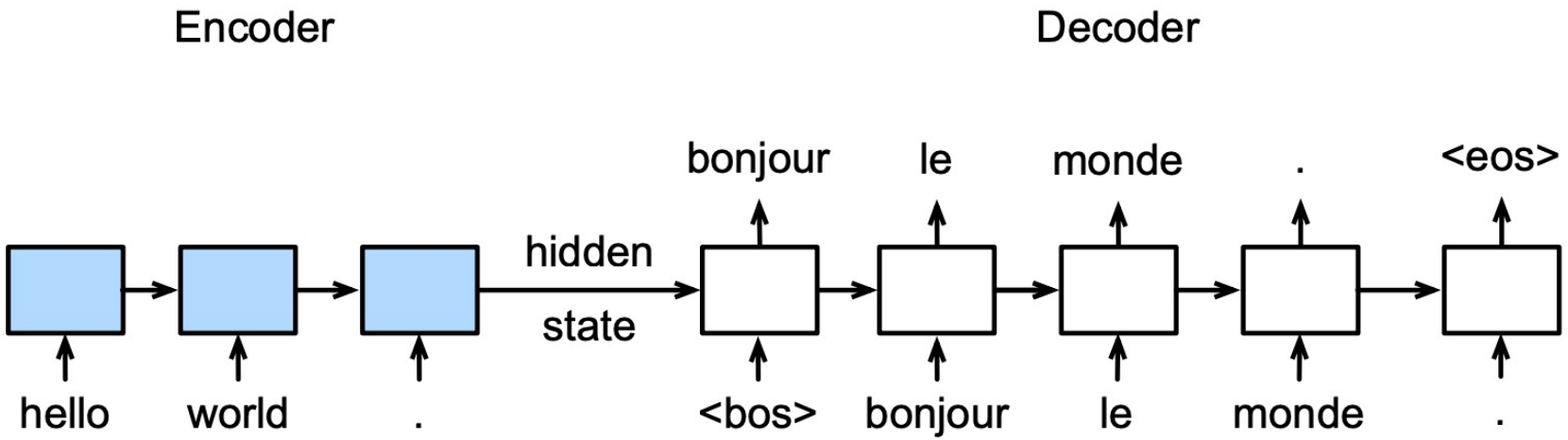
...

Feature extractor

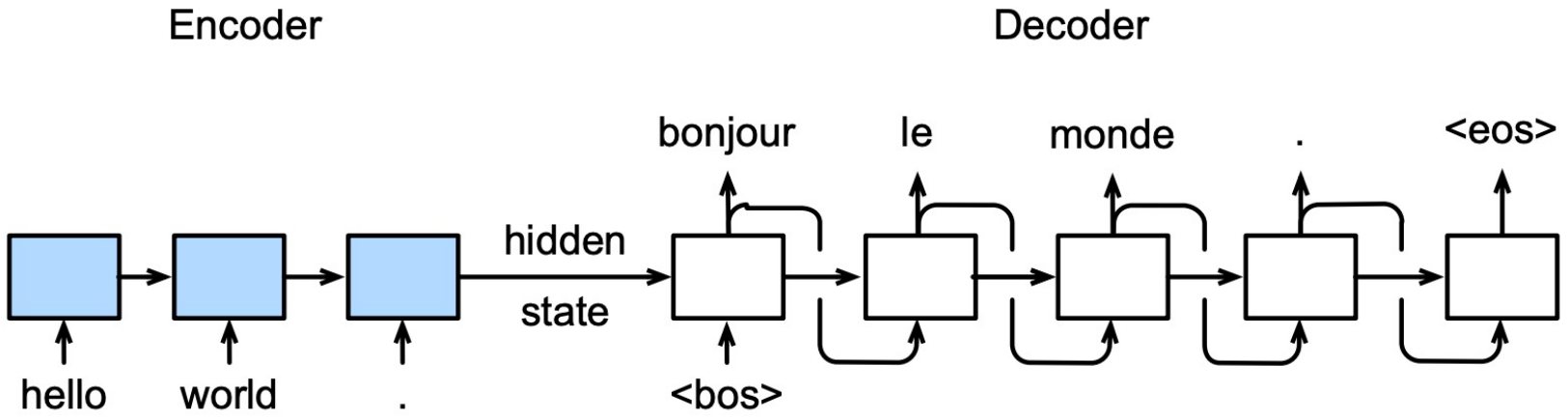
Layer 1

Sequence to sequence

Training



Testing

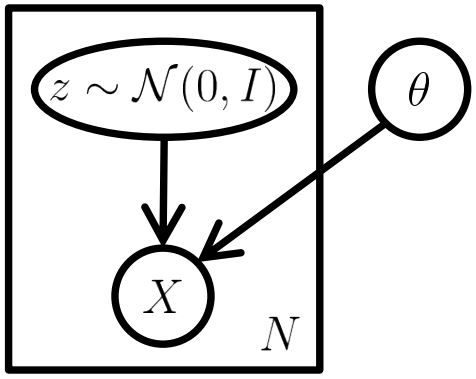


Generative models

- **Task:** Given a dataset of images $\{X_1, X_2 ...\}$ can we learn the distribution of X?
- Typically generative models implies modelling P(X).
  - **Very limited, given an image the model outputs a probability**
- **More Interested** in models which we can **sample** from.
  - Can generate random examples that follow the distribution of P(X).

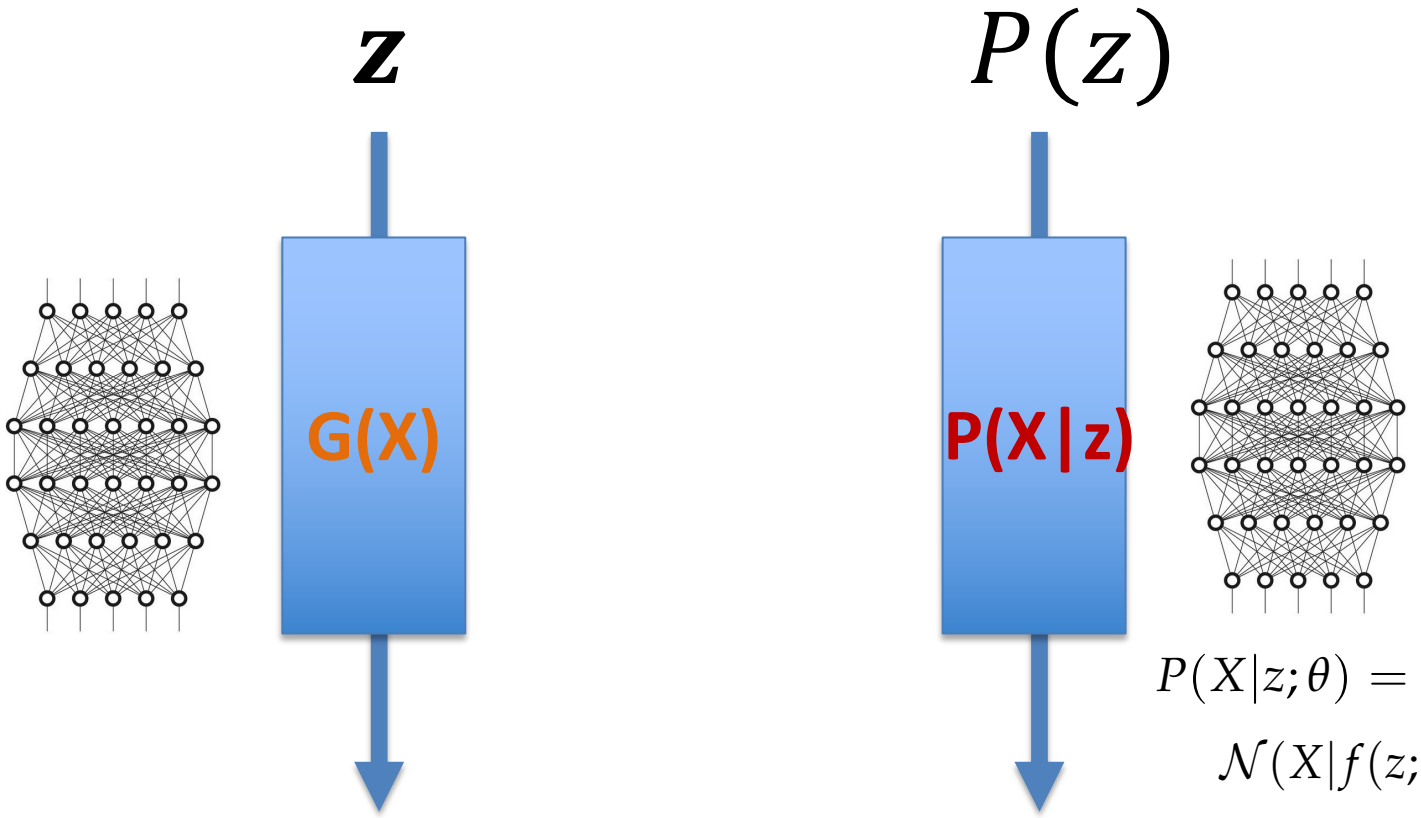
Latent variable models

- Modeling $P(X|z;\theta)$ explicitly
- $z \sim P(z)$, which we can sample from, such as a Gaussian distribution



- Then, $P(X) = \int P(X|z;\theta)P(z)dz.$
- Maximum likelihood
  - Find θ to maximize the distribution $P(X)$

From deterministic to probabilistic



$$P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$$

$$P(X) = \int P(X|z;\theta)P(z)dz.$$

One way to max $P(X)$

- Approximate with samples of z

$$P(X) \approx \frac{1}{n} \sum_{i=0}^{n} P(X|z_i)$$

- Problems
  - Need many samples of z and most of the $P(X|z) \approx 0$
  - Not computationally tractable
- Need to learn a distribution $Q(z)$, where $z \sim Q(z)$ generates $P(X|z) >> 0$.

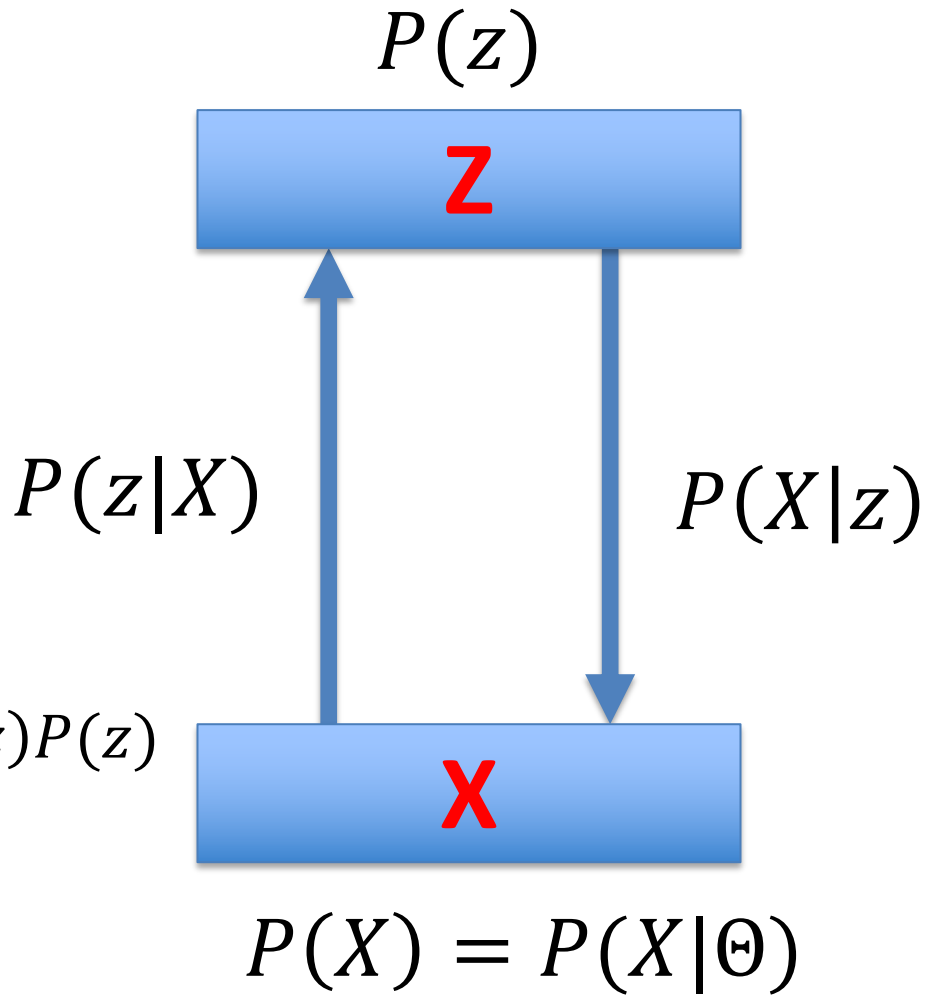
Recall EM algorithm

E-Step: Compute

$$P^{old}(z|X) = \frac{P(X|z)P(z)}{P(X)}$$

M-Step: Update Θ by

$$\max_{\Theta} \int P^{old}(z|X) \ln P(X|z)P(z)$$



Lower bound of likelihood

$$E[f(x)] \geq f(E[x])$$

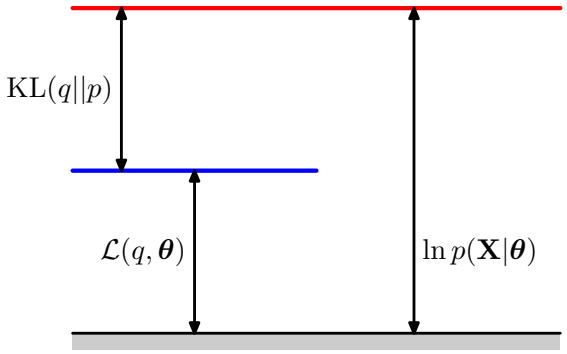
Jensen's Inequality due to convexity

$$\log(P(\mathbf{x}|\theta)) = \log(\sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y}|\theta))$$

$$\begin{aligned}
\log(P(\mathbf{x}|\theta)) &= \log(\sum_{\mathbf{y}} q(\mathbf{y}) \frac{P(\mathbf{x}, \mathbf{y}|\theta)}{q(\mathbf{y})}) \\
&\geq E_q[\log(\frac{P(\mathbf{x}, \mathbf{y}|\theta)}{q(\mathbf{y})}] \\
&\geq E_q[\log(\frac{P(\mathbf{y}|\mathbf{x}, \theta)P(\mathbf{x}|\theta)}{q(\mathbf{y})}] \\
&\geq E_q[\log(P(\mathbf{x}|\theta)] - E_q[\log(\frac{q(\mathbf{y})}{P(\mathbf{y}|\mathbf{x}, \theta)}] \\
&\geq E_q[\log(P(\mathbf{x}|\theta))] - KL(q(\mathbf{y})\|P(\mathbf{y}|\mathbf{x}, \theta)) \\
&\geq \log(P(\mathbf{x}|\theta)) - KL(q(\mathbf{y})\|P(\mathbf{y}|\mathbf{x}, \theta))
\end{aligned}$$

$$\begin{aligned}
\log(P(\mathbf{x}|\theta)) &\geq E_q[\log(\frac{P(\mathbf{x}, \mathbf{y}|\theta)}{q(\mathbf{y})}] \\
&\geq E_q[\log(P(\mathbf{x}, \mathbf{y}|\theta)] - E_q[\log(q(\mathbf{y}))] \\
&\geq E_q[\log(P(\mathbf{x}, \mathbf{y}|\theta)] + H(q(\mathbf{y}))
\end{aligned}$$

EM never decreases the likelihood



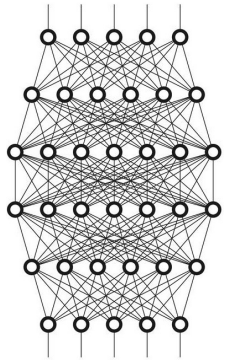
Outline

- Related concepts
- **Variational Autoencoder (VAE)**
  - Theory and algorithm
  - Applications
  - Conditional VAE
- Generative Adversarial Networks (GAN)
- VAE vs. GAN

Variational approximation

$P(z)$

$f(z|\theta)$



$P(z|X)$

Computational difficult!

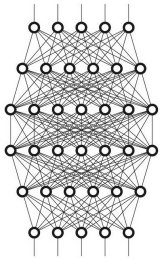
$P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$

Approximation by variational approach!

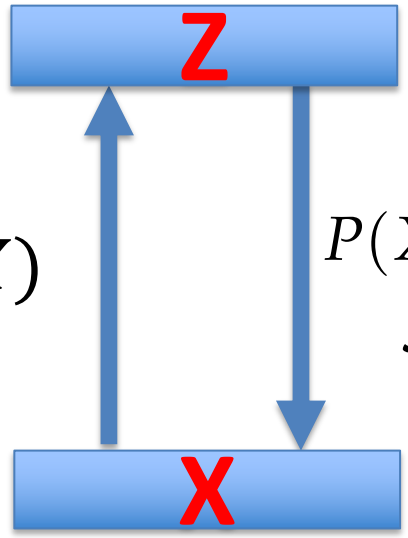
$P(X) = P(X|\Theta)$

Variational approximation

$f(z|\theta)$



$P(z)$



Approximate

$Q(z) \dashrightarrow P(z|X)$

$P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$

$P(X) = P(X|\Theta)$

$\log P(X) - \mathcal{D}\left[Q(z)||P(z|X)\right]$

KL divergence

Which form of $Q(z)$ is better?

$$\log P(X) - \mathcal{D}\left[Q(z) \| P(z|X)\right]$$

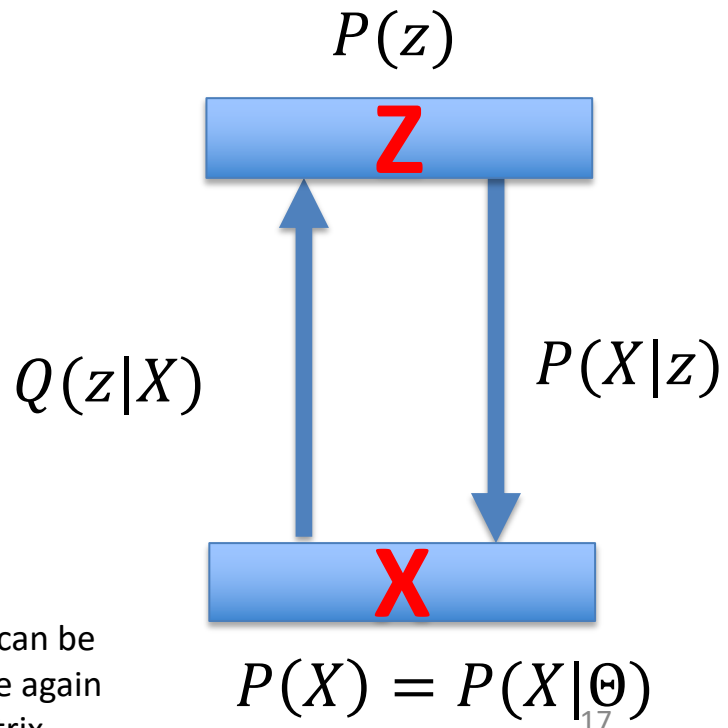
Note that X is fixed, and Q can be *any* distribution, not just a distribution which does a good job mapping X to the z's that can produce X. Since we're interested in inferring $P(X)$, it makes sense to construct a Q which *does* depend on X, and in particular, one which makes $\mathcal{D}\left[Q(z)\|P(z|X)\right]$ small:

$$\log P(X) - \mathcal{D}\left[Q(z|X) \| P(z|X)\right]$$

This term will become small, if Q is **high-capacity**, making Q produce z that can reproduce a given X.

$$Q(z|X) = \mathcal{N}(z|\mu(X;\vartheta), \Sigma(X;\vartheta))$$

where μ and Σ are arbitrary deterministic functions with parameters ϑ that can be learned from data (we will omit ϑ in later equations). In practice, μ and Σ are again implemented via neural networks, and Σ is constrained to be a diagonal matrix.



The objective function

$$\log P(X) - \mathcal{D}\left[Q(z) \| P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z) \| P(z)\right]$$

$$\log P(X) - \mathcal{D}\left[Q(z|X) \| P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X) \| P(z)\right]$$

Derivations:

- Definition of KL divergence:

$$\mathcal{D}\left[Q(z) \| P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(z|X)\right]$$

- Apply **Bayes Rule on** P(z|X) and substitute into the equation above.
  - P(z|X) = P(X|z) P(z) / P(X)
  - log (P(z|X)) = log P(X|z) + log P(z) - log P(X)
  - P(X) does not depend on z, can take it outside of $E_{z\sim Q}$

$$\mathcal{D}\left[Q(z) \| P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(X|z) - \log P(z)\right] + \log P(X)$$

Rearrange the terms

$$\mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(X|z) - \log P(z)\right] + \log P(X)$$

Rearrange the terms:

$E_{z\sim Q}$ [log Q(z) - log P(z)] = D [Q (z) || P(z)]

$$\log P(X) - \mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z)\|P(z)\right]$$

Why is this important?

$$\log P(X) - \mathcal{D}\left[Q(z) \| P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z) \| P(z)\right]$$

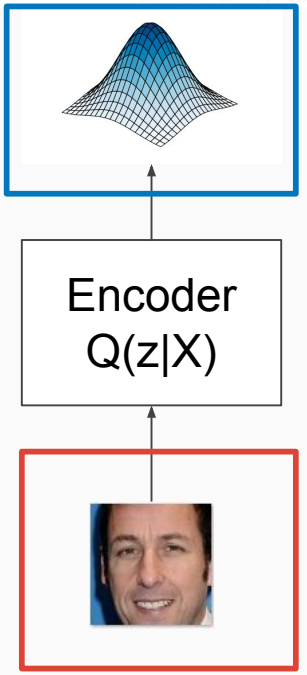
- Recall we want to **maximize P(X)** with respect to θ, which we cannot do.
- KL divergence is always > 0.
- log P(X) > log P(X) - D[Q(z) || P(z|X)].
- Maximize the **lower bound** instead.

Choosing appropriate Q

$$\log P(X) - \mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z)\|P(z)\right]$$

- **Q(z) or Q(z|X)?**
- Model Q(z|X) with a neural network.
- Assume Q(z|X) to be Gaussian, N(μ, c·I)
  - Neural network **outputs** the **mean μ**, and diagonal covariance matrix **c** · **I**.
  - **Input:** Image, **Output:** Distribution

Let's call Q(z|X) the **Encoder.**



$$Q(z|X) = \mathcal{N}(z|\mu(X;\vartheta), \Sigma(X;\vartheta))$$

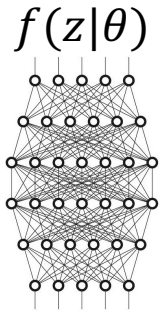
The loss function: 1st term

$$\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]$$

The first term:

- Model P(X|z) with a neural network, let f(z) be the network output.
- Assume P(X|z) to be i.i.d. **Gaussian**
  - X = f(z) + η , where η ~ N(0,I) *Think Linear Regression*
  - **Simplifies to an l_2 loss: $||X-f(z)||^2$**

$$P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$$



Optimization by stochastic gradient descent

We could use sampling to estimate $E_{z\sim Q}\left[\log P(X|z)\right]$, but getting a good estimate would require passing many samples of z through f, which would be expensive. Hence, as is standard in stochastic gradient descent, we take one sample of z and treat $P(X|z)$ for that z as an approximation of $E_{z\sim Q}\left[\log P(X|z)\right]$

Actually, we want to optimize

$$E_{X\sim D}\left[\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right]\right] = E_{X\sim D}\left[E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]\right]$$

The loss function: 2nd term

$$\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]$$

Assume $P(z) = G(z|0, I)$. Then, the second term is a KL divergence between two multivariate Gaussians.

$$\mathcal{D}[\mathcal{N}(\mu(X), \Sigma(X))\|\mathcal{N}(0, I)] = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma(X)\right) + \left(\mu(X)\right)^{\top}\left(\mu(X)\right) - k - \log\det\left(\Sigma(X)\right)\right)$$

where *k* is the dimensionality of the distribution.

KL divergence between two multivariate Gaussians:

$$\mathcal{D}[\mathcal{N}(\mu_0, \Sigma_0)\|\mathcal{N}(\mu_1, \Sigma_1)] = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)^{\top}\Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right)$$

The loss function: all together

$$\log P(X) - \mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z)\|P(z)\right]$$

Putting it all together: $E_{z\sim Q(z|X)} \log P(X|z) \propto \|X-f(z)\|^2$

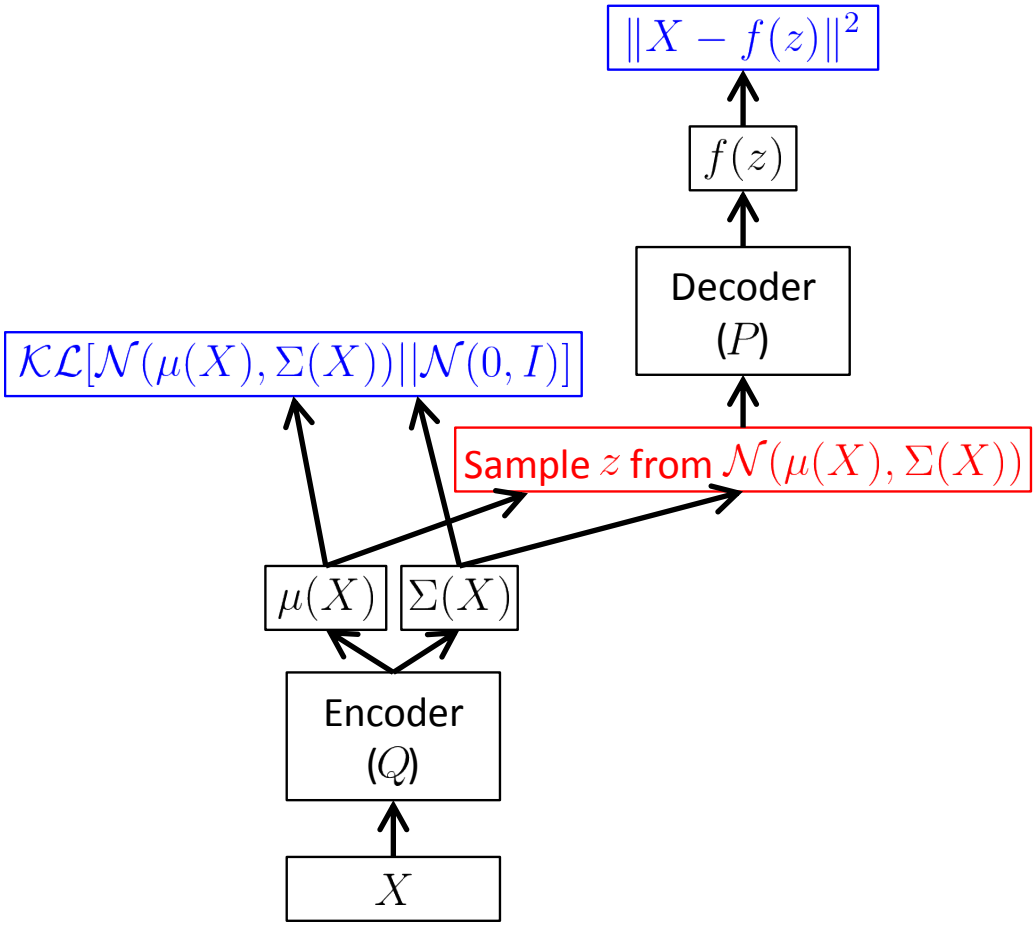
$$L = \|X - f(z)\|^2 - \lambda \cdot D[Q(z) \| P(z)]$$

, given a (X, z) pair.

Pixel difference

Regularization

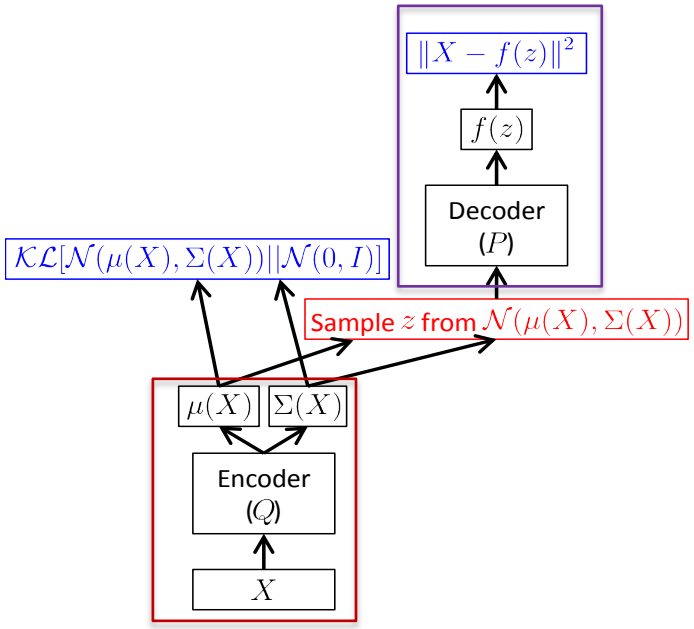
The whole architecture



How to train the model?

- Training the Decoder is easy, by standard backpropagation.

- How to train the Encoder?
  - How to apply gradient descent through samples? Not trivial.



Reparameterization trick

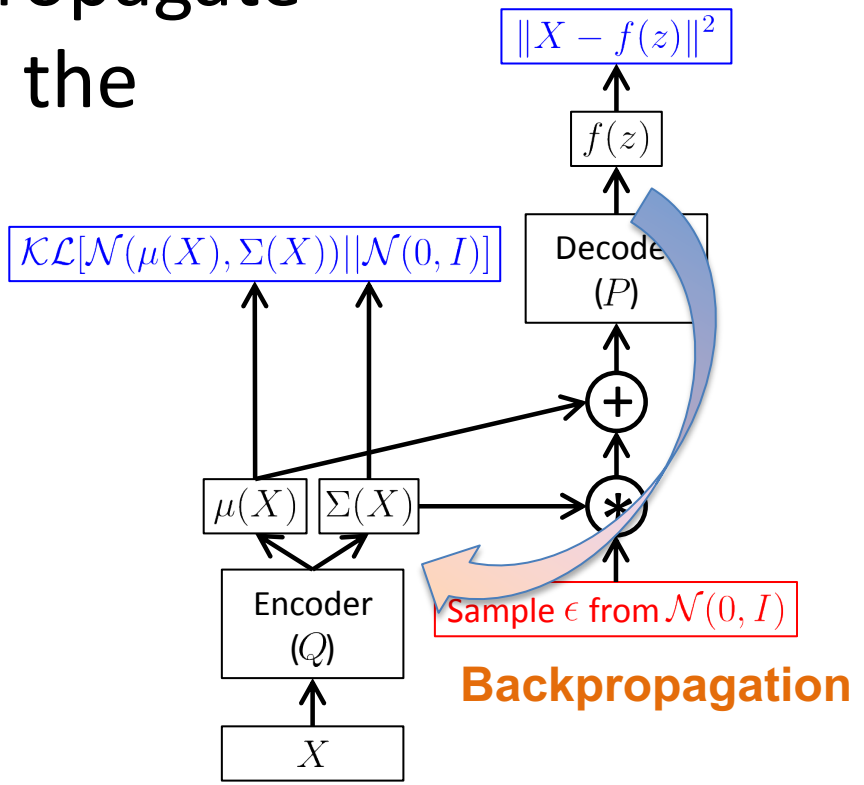
- How to effectively backpropagate through the z samples to the Encoder?

Reparameterization

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

Equivalent!

$$z = \mu + \sigma \cdot \epsilon,$$
$$\epsilon \sim \mathcal{N}(\epsilon|0,1)$$



$$E_{X\sim D}\left[E_{\epsilon\sim\mathcal{N}(0,I)}[\log P(X|z = \mu(X) + \Sigma^{1/2}(X) * \epsilon)] - \mathcal{D}\left[Q(z|X)\|P(z)\right]\right]$$

Now, we can easily backpropagate the loss to the Encoder.

The whole training process

Given a dataset of examples **X =** {X1, X2...}

Initialize parameters for Encoder and Decoder

Repeat till convergence:

$\mathbf{X^M}$ <-- Random minibatch of M examples from **X**

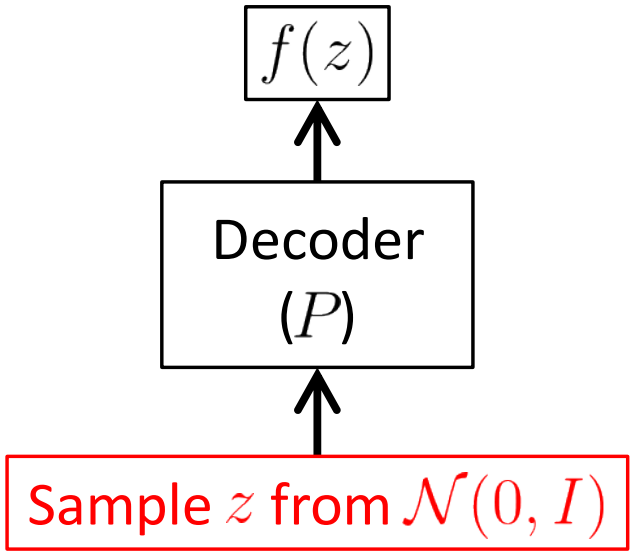
ε <-- Sample M noise vectors from N(0, I)

Compute $L(\mathbf{X^M}, \varepsilon, \theta)$ (i.e. run a forward pass in the neural network)

Gradient descent on *L* to updated Encoder and Decoder.

Testing the learned model

At test time, when we want to generate new samples, we simply input values of $z \sim \mathcal{N}(0, I)$ into the decoder. That is, we remove the "encoder," including the multiplication and addition operations that would change the distribution of z.



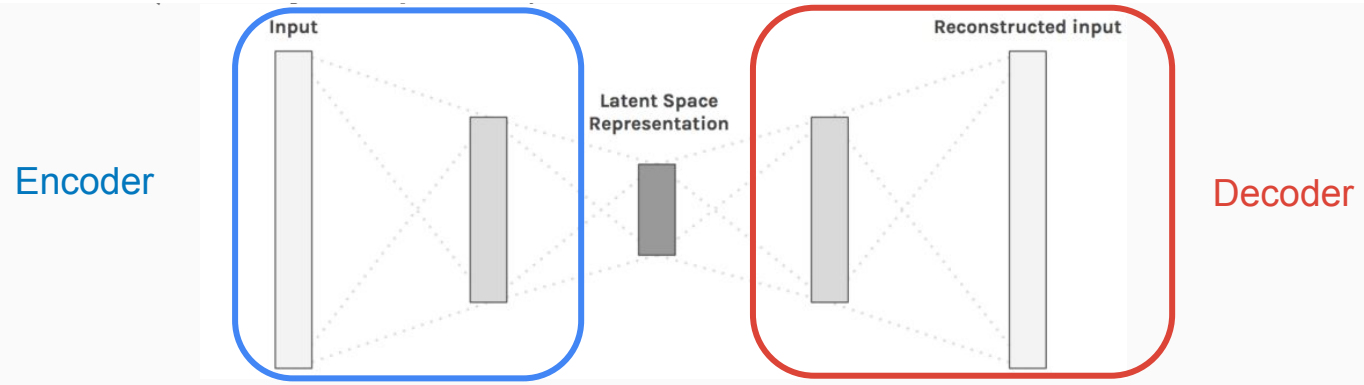
No good quantitative metric to evaluate the generated samples (e.g., images), and it relies on human visual inspection usually.

Outline

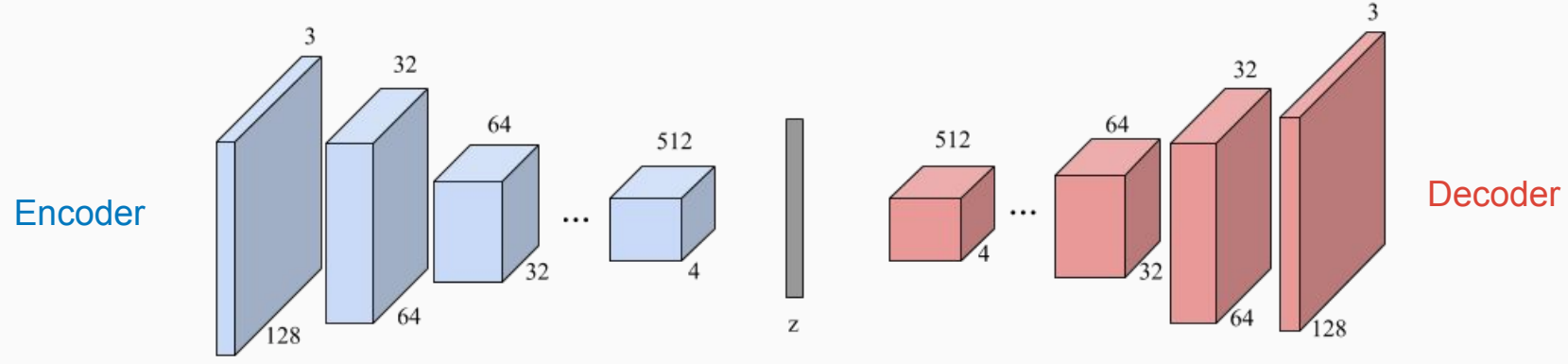
- Related concepts
- **Variational Autoencoder (VAE)**
  - Theory and algorithm
  - Applications
  - Conditional VAE
- Generative Adversarial Networks (GAN)
- VAE vs. GAN

Common VAE architecture

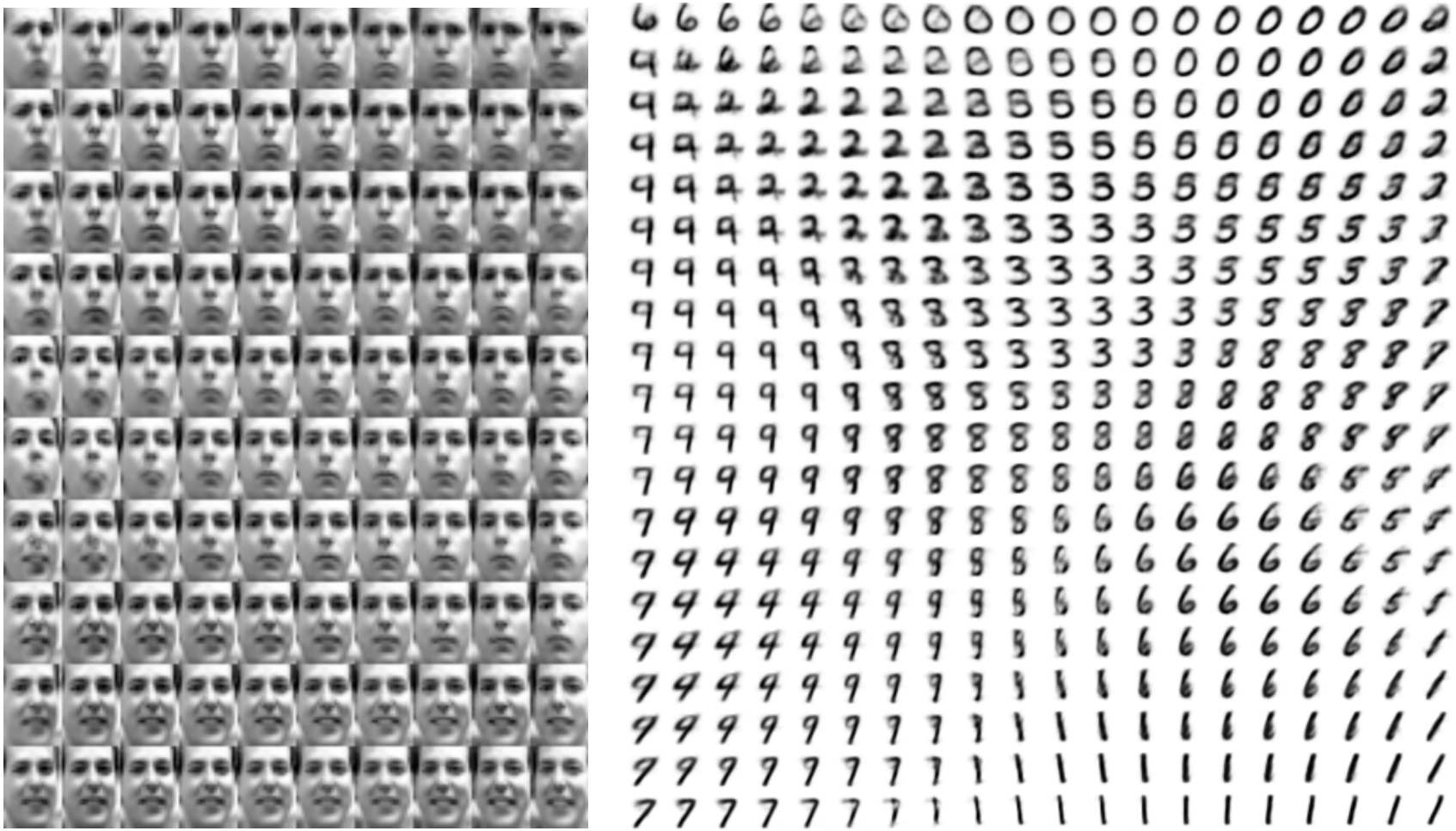
- Fully connected network



- Convolution and deconvolution network



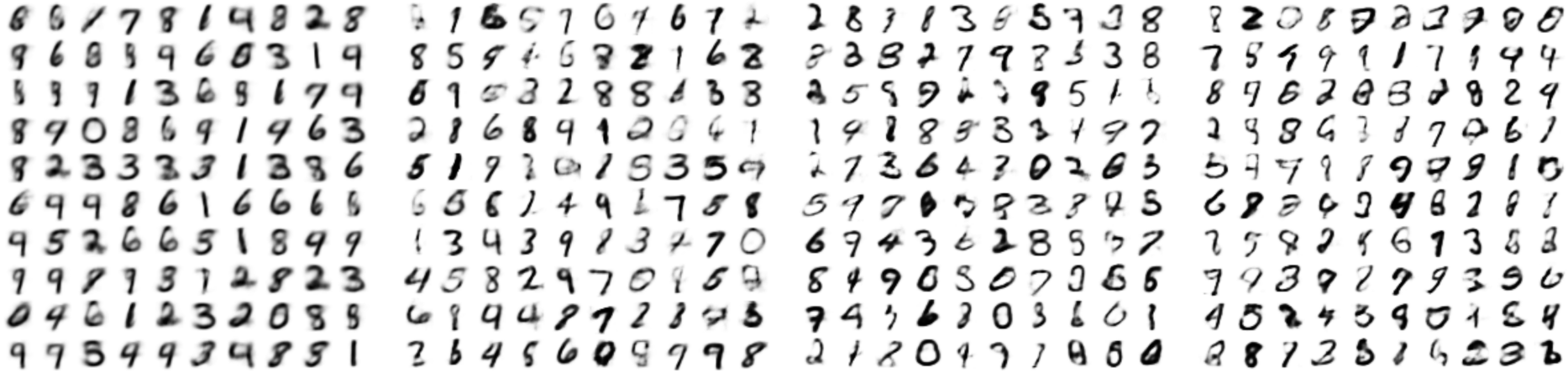
Disentangle latent factors by VAE



Visualization of learned data manifold for generative models with two-dimensional latent space.
http://www.dpkingma.com/sgvb_mnist_demo/demo.html

Different dimensionalities of latent space

Random samples from learned generative models of MNIST for different dimensionalities of latent space.



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

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. ICLR, 2014.

Control latent factors for generation

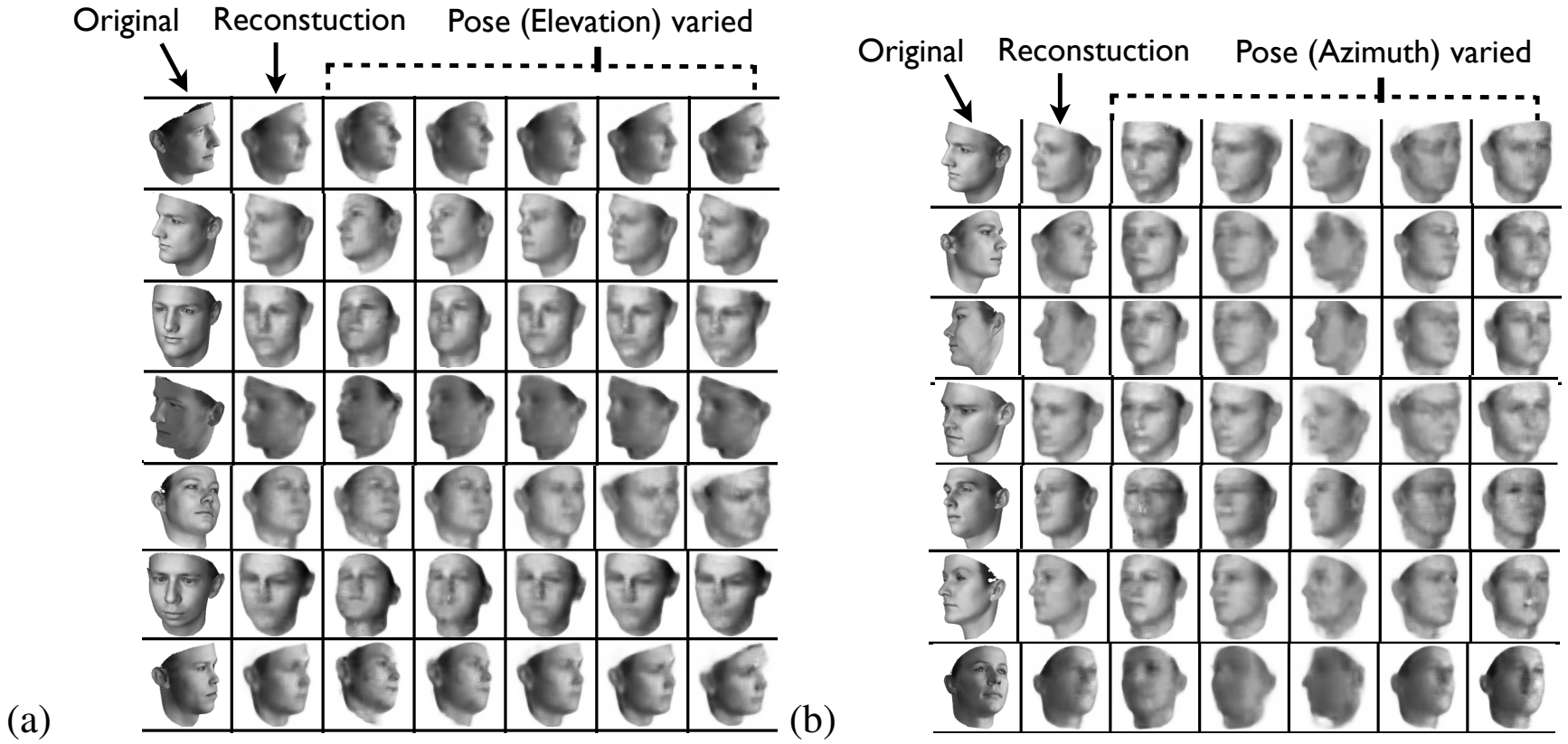


Figure 4: **Manipulating pose variables:** Qualitative results showing the generalization capability of the learned DC-IGN decoder to rerender a single input image with different pose directions. **(a)** We change the latent $z_{elevation}$ smoothly from -15 to 15, leaving all 199 other latents unchanged. **(b)** We change the latent $z_{azimuth}$ smoothly from -15 to 15, leaving all 199 other latents unchanged.

Tejas D Kulkarni, William F. Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015.

Manipulating rotation

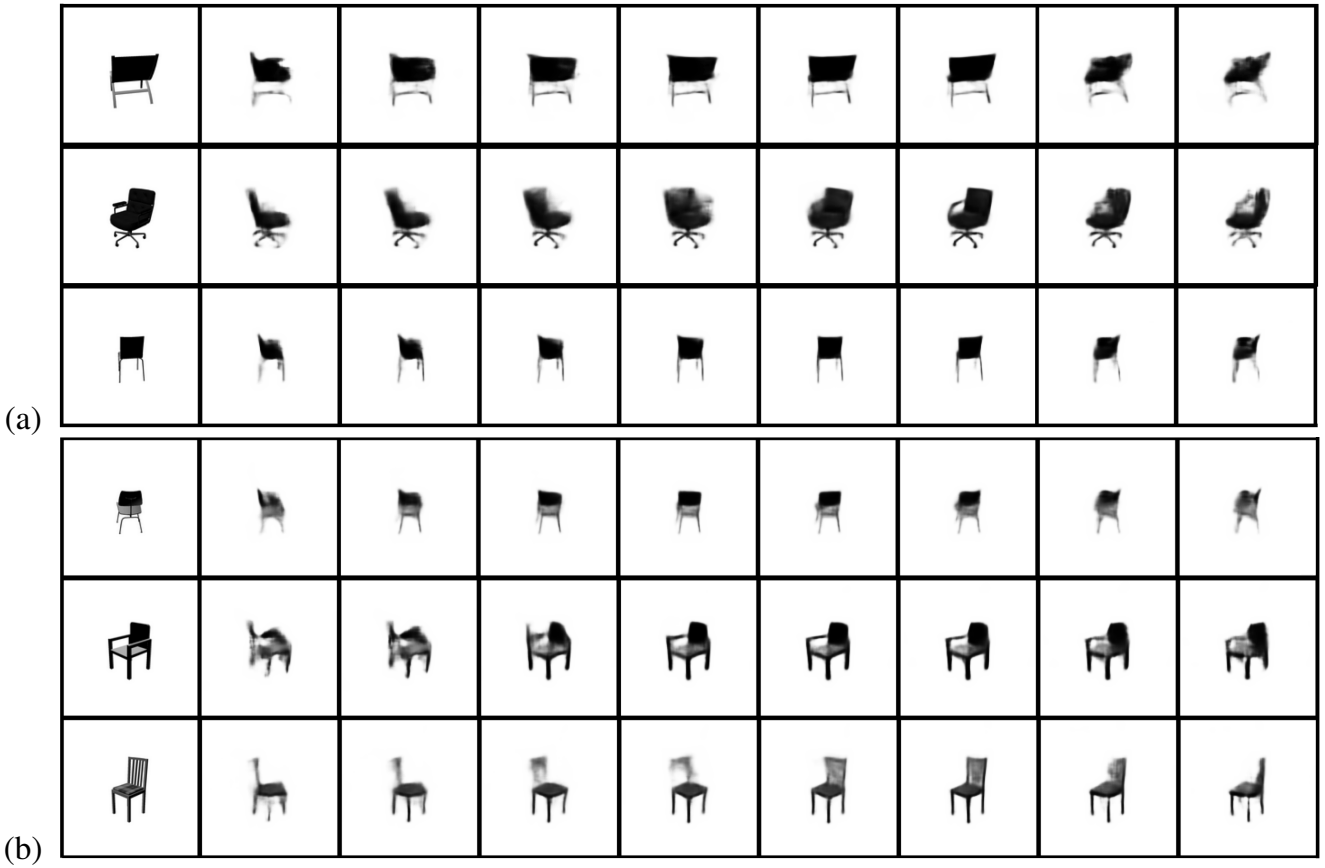


Figure 7: **Manipulating rotation:** Each row was generated by encoding the input image (leftmost) with the encoder, then changing the value of a single latent and putting this modified encoding through the decoder. The network has never seen these chairs before at any orientation. **(a)** Some positive examples. Note that the DC-IGN is making a conjecture about any components of the chair it cannot see; in particular, it guesses that the chair in the top row has arms, because it can't see that it doesn't. **(b)** Examples in which the network extrapolates to new viewpoints less accurately.

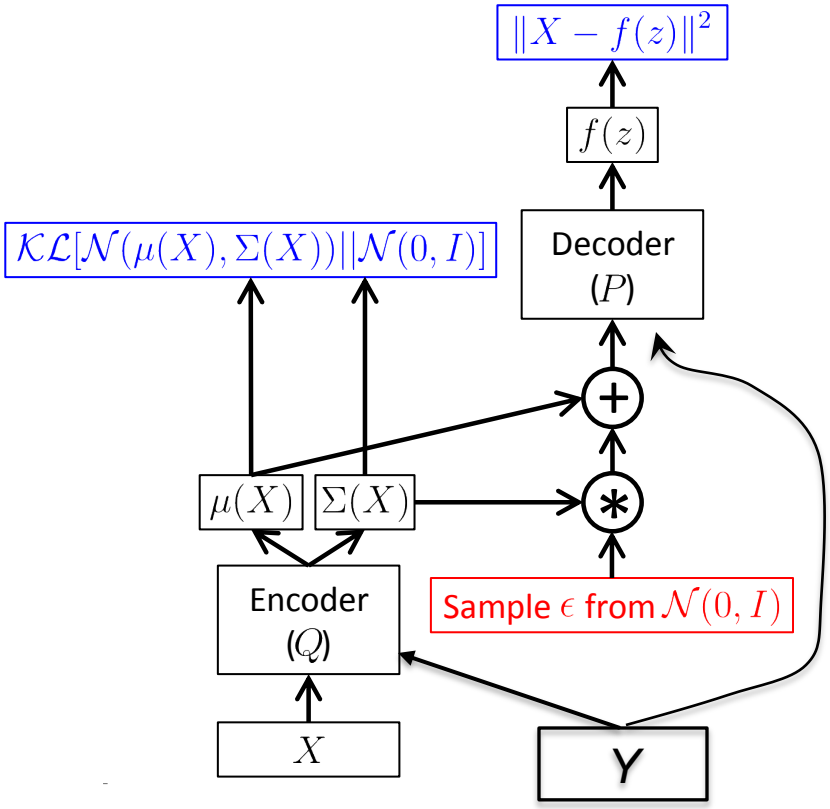
Outline

- Related concepts
- **Variational Autoencoder (VAE)**
  - Theory and algorithm
  - Applications
  - Conditional VAE
- Generative Adversarial Networks (GAN)
- VAE vs. GAN

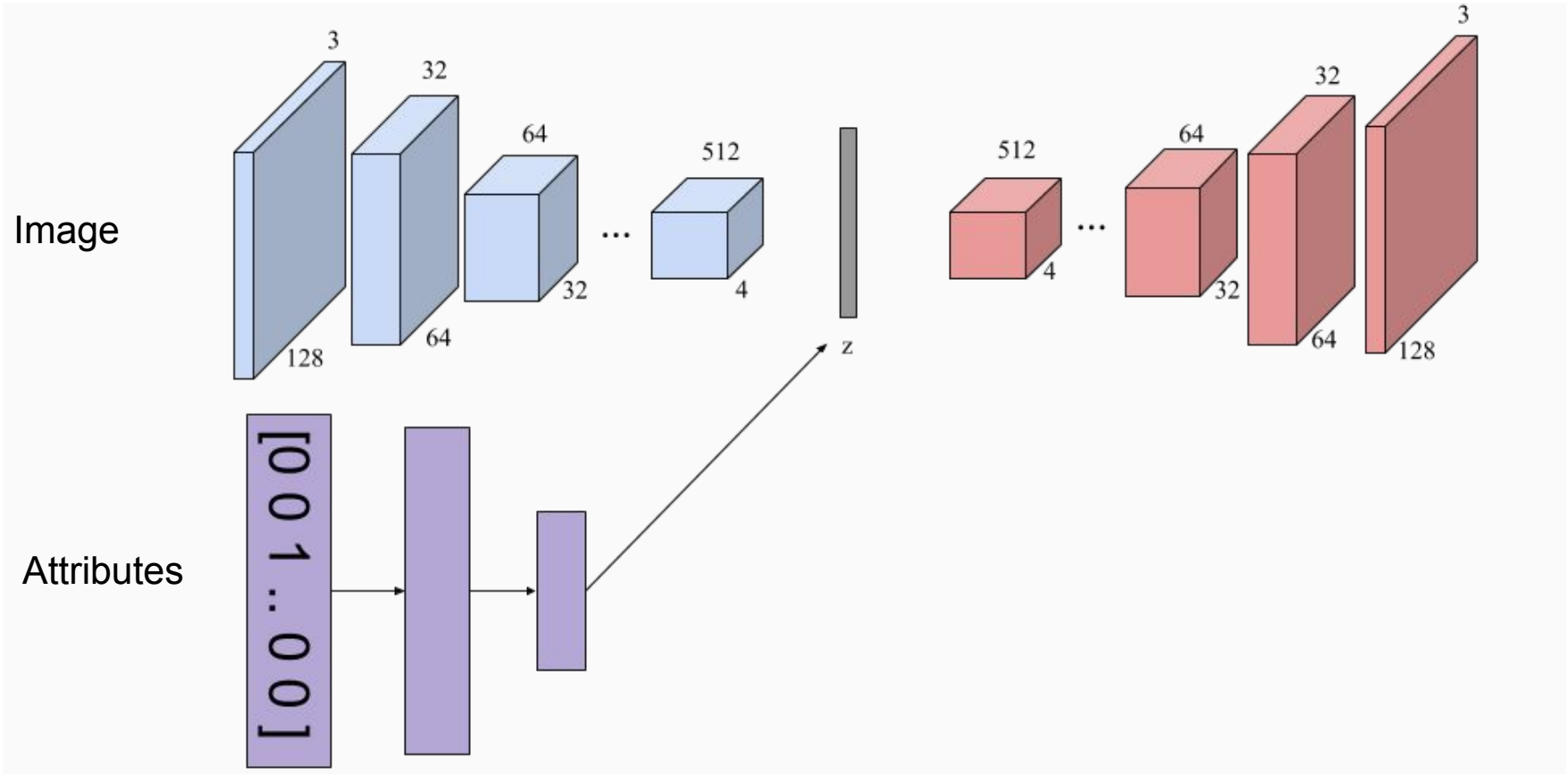
Conditional VAE

What if we have **labels**? (e.g. digit labels or attributes) Or other inputs we wish to condition on **(Y)**.

- **NONE** of the derivation changes.
- Replace all **P(X|z) with P(X|z,Y)**.
- Replace all **Q(z|X) with Q(z|X,Y)**.
- Go through the same KL divergence procedure, to get the same lower bound.

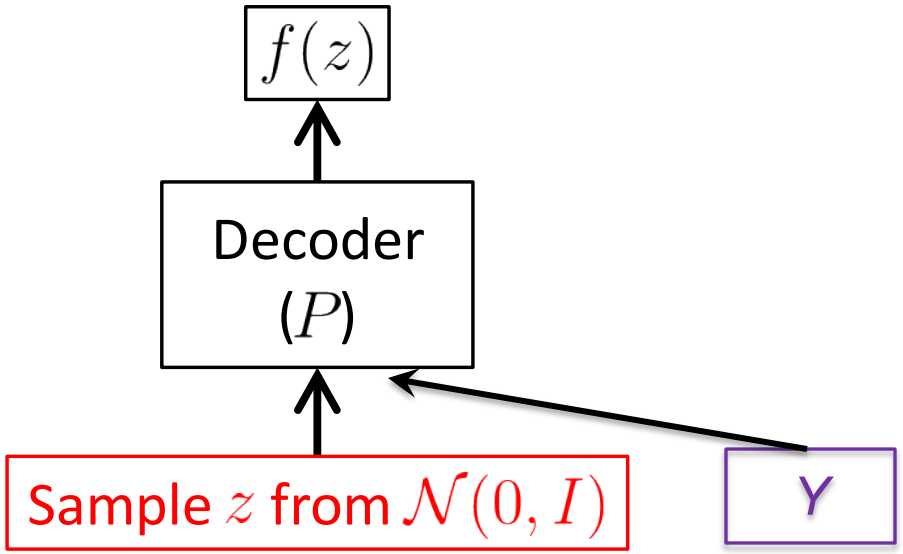


Common architecture for CVAE

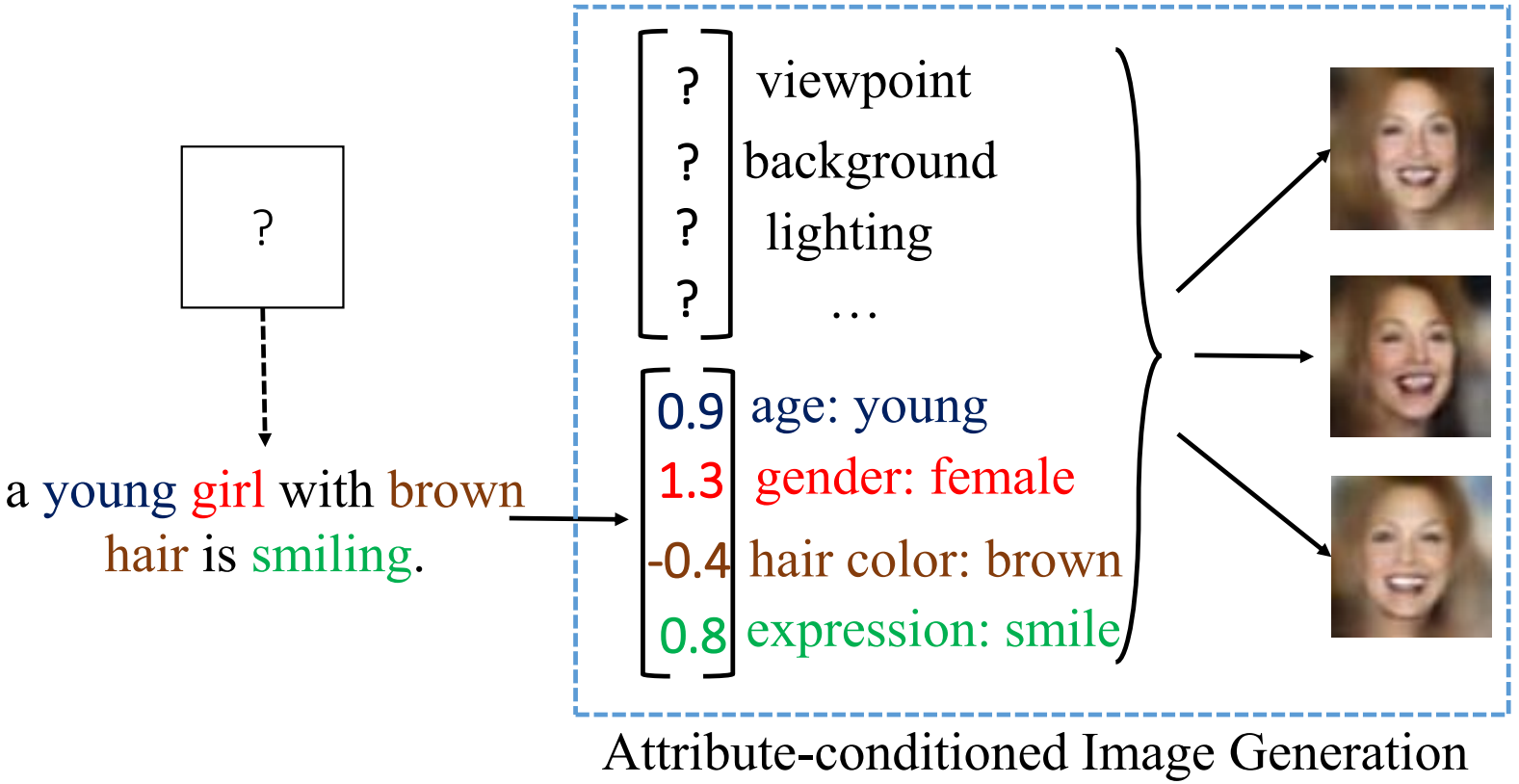


Testing CVAE

- Again, remove the Encoder as test time
- Sample z ~ *N(0,I)* **and** input a **desired Y** to the Decoder.



Conditioned image generation



Yan et al., Attribute2Image: Conditional Image Generation from Visual Attributes, in ECCV 2016

Attribute-conditioned image progression

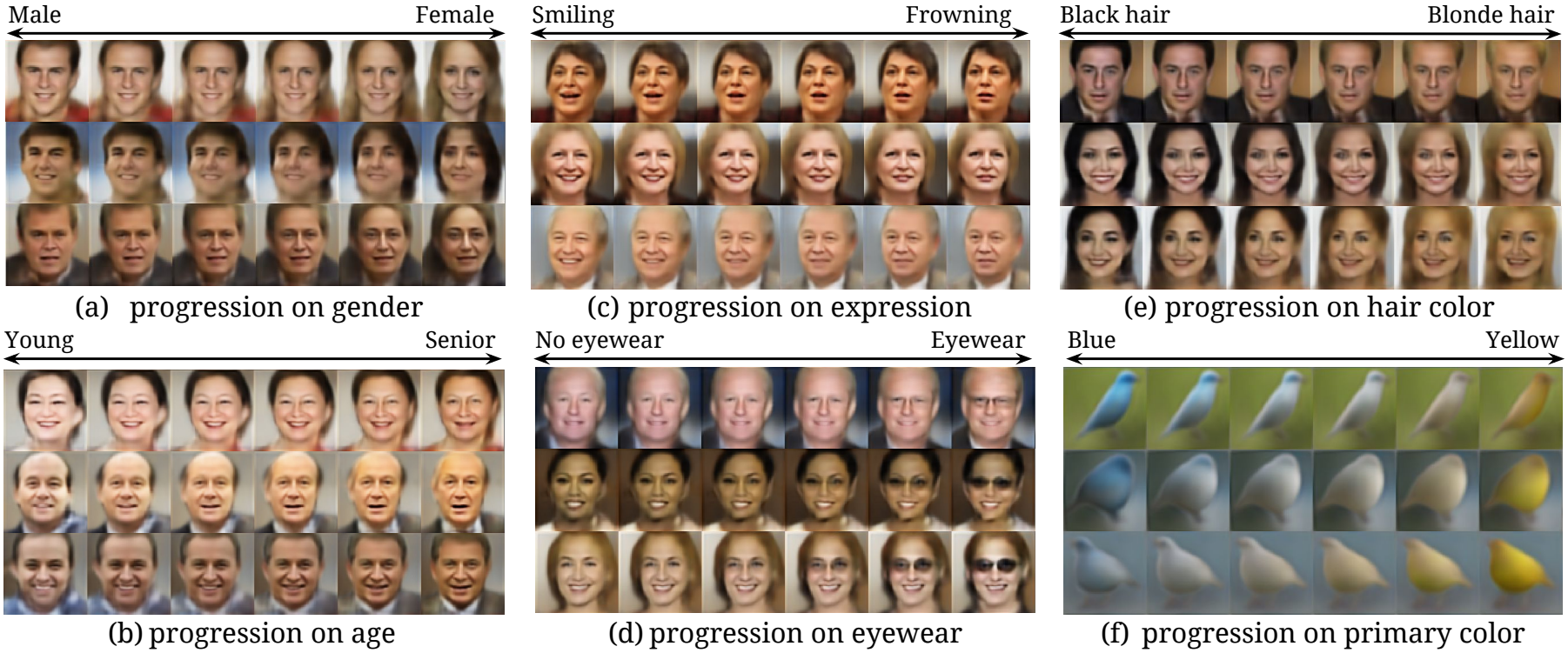


Fig. 4. Attribute-conditioned image progression. The visualization is organized into six attribute groups (e.g., "gender", "age", "facial expression", "eyewear", "hair color" and "primary color (blue vs. yellow)"). Within each group, the images are generated from $p_\theta(x|y,z)$ with $z \sim \mathcal{N}(0, I)$ and $y = [y_\alpha, y_{rest}]$, where $y_\alpha = (1-\alpha)\cdot y_{min} + \alpha \cdot y_{max}$. Here, y_{min} and y_{max} stands for the minimum and maximum attribute value respectively in the dataset along the corresponding dimension.

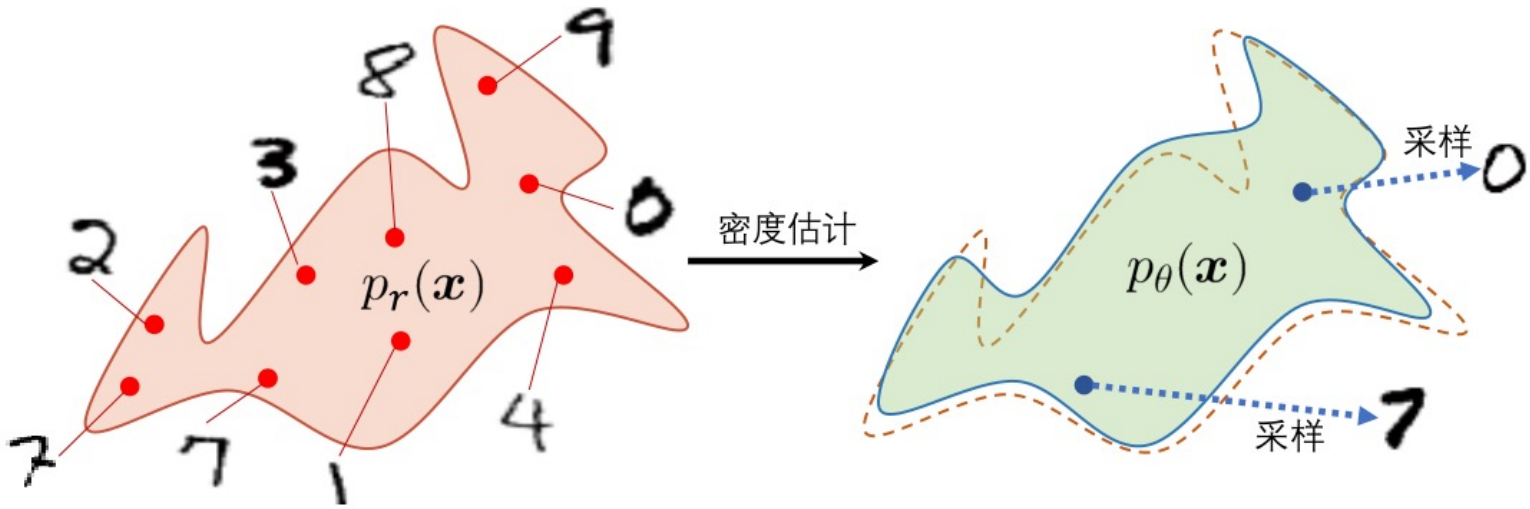
Yan et al., Attribute2Image: Conditional Image Generation from Visual Attributes, in ECCV 2016

Outline

- Related concepts
- Variational Autoencoder (VAE)
- **Generative Adversarial Networks (GAN)**
- VAE vs. GAN

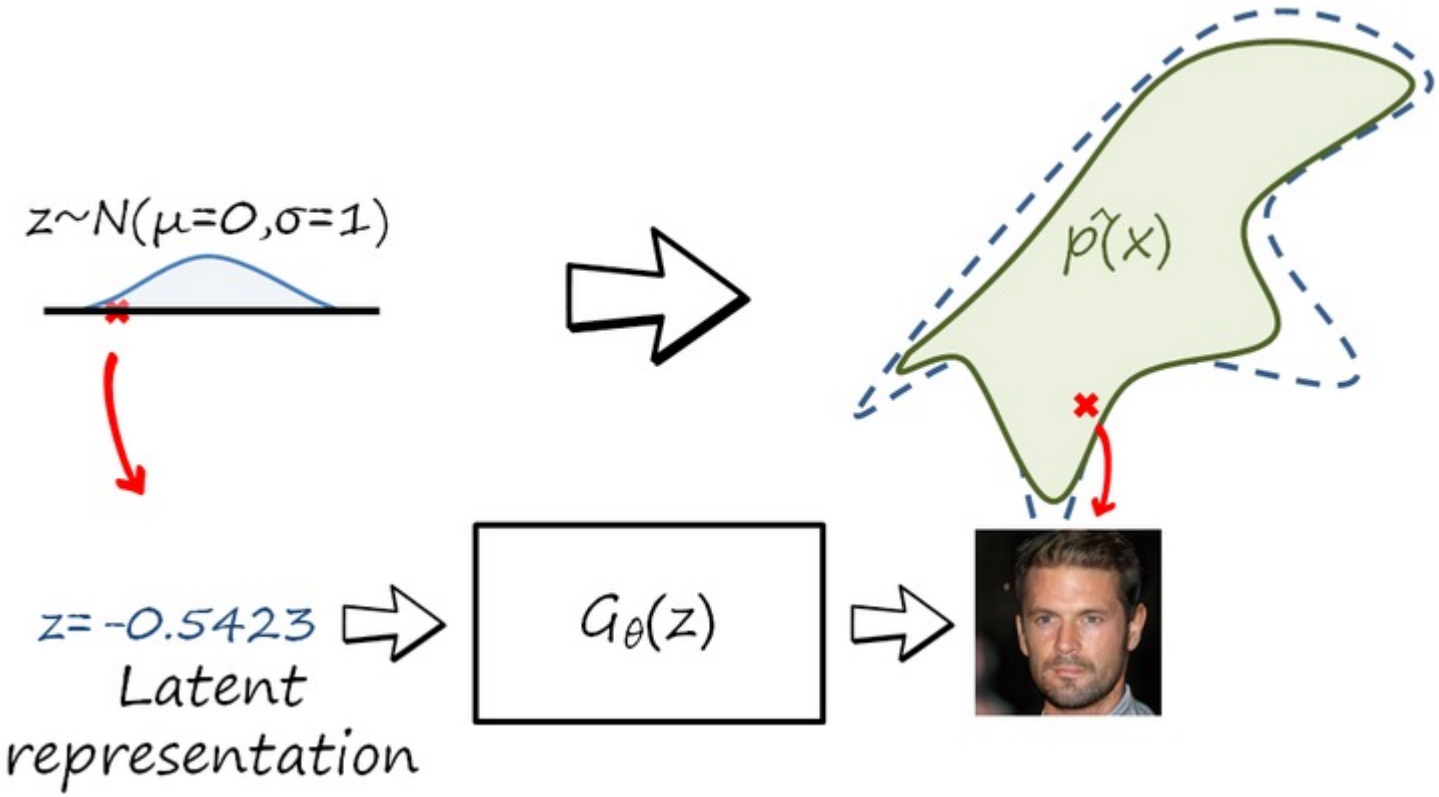
Generative models (1)

- **Task:** Given a dataset of images $\{X_1, X_2 ...\}$ can we learn the distribution of X?
  - modelling $P(X|\boldsymbol{\Theta})$ and estimate $\boldsymbol{\Theta}$
  - Sample from $P(X|\boldsymbol{\Theta})$



Generative models (2)

- **More Interested** in models which we can **sample** from.
  - Can generate random examples that follow the distribution of P(X).



Generation

Image Generation

$\begin{bmatrix} 0.3 \\ -0.1 \\ \vdots \\ -0.7 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.1 \\ \vdots \\ 0.7 \end{bmatrix} \begin{bmatrix} -0.3 \\ 0.1 \\ \vdots \\ 0.9 \end{bmatrix}$

In a specific range

→ NN Generator →

Sentence Generation

$\begin{bmatrix} 0.3 \\ -0.1 \\ \vdots \\ -0.7 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.1 \\ \vdots \\ 0.2 \end{bmatrix} \begin{bmatrix} -0.3 \\ 0.1 \\ \vdots \\ 0.5 \end{bmatrix}$

→ NN Generator →

How are you?

Good morning.

Good afternoon.

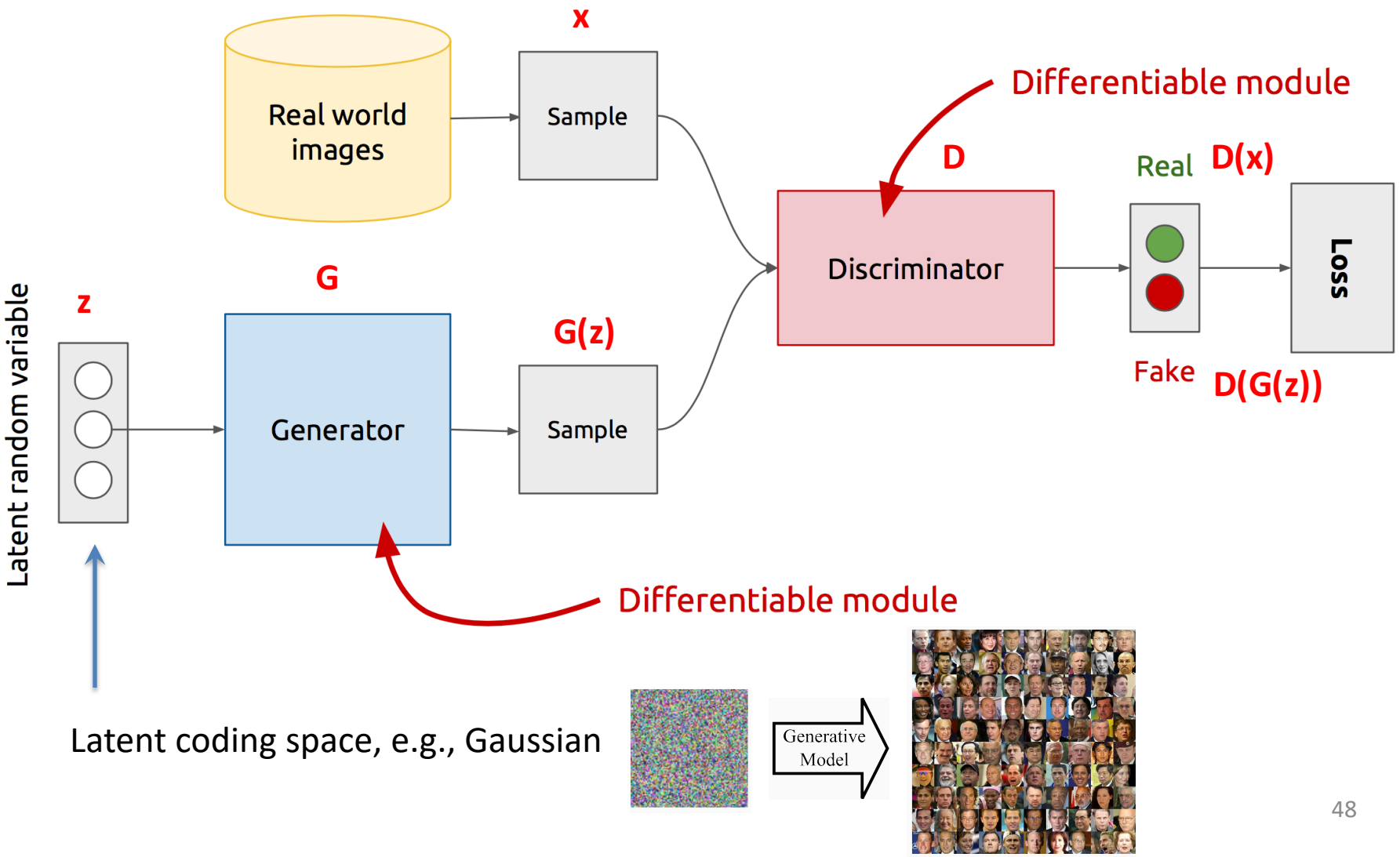
Generative Adversarial Nets

Ian J. Goodfellow[*], **Jean Pouget-Abadie**[†], **Mehdi Mirza**, **Bing Xu**, **David Warde-Farley**, **Sherjil Ozair**[‡], **Aaron Courville**, **Yoshua Bengio**[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Generative Adversarial Networks (GAN)



Basic idea of using Discriminator

Generator (student)

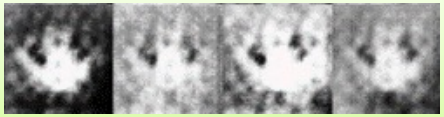
Discriminator (teacher)

Generator v1

Discriminator v1

沒有兩個圈

Generator v2



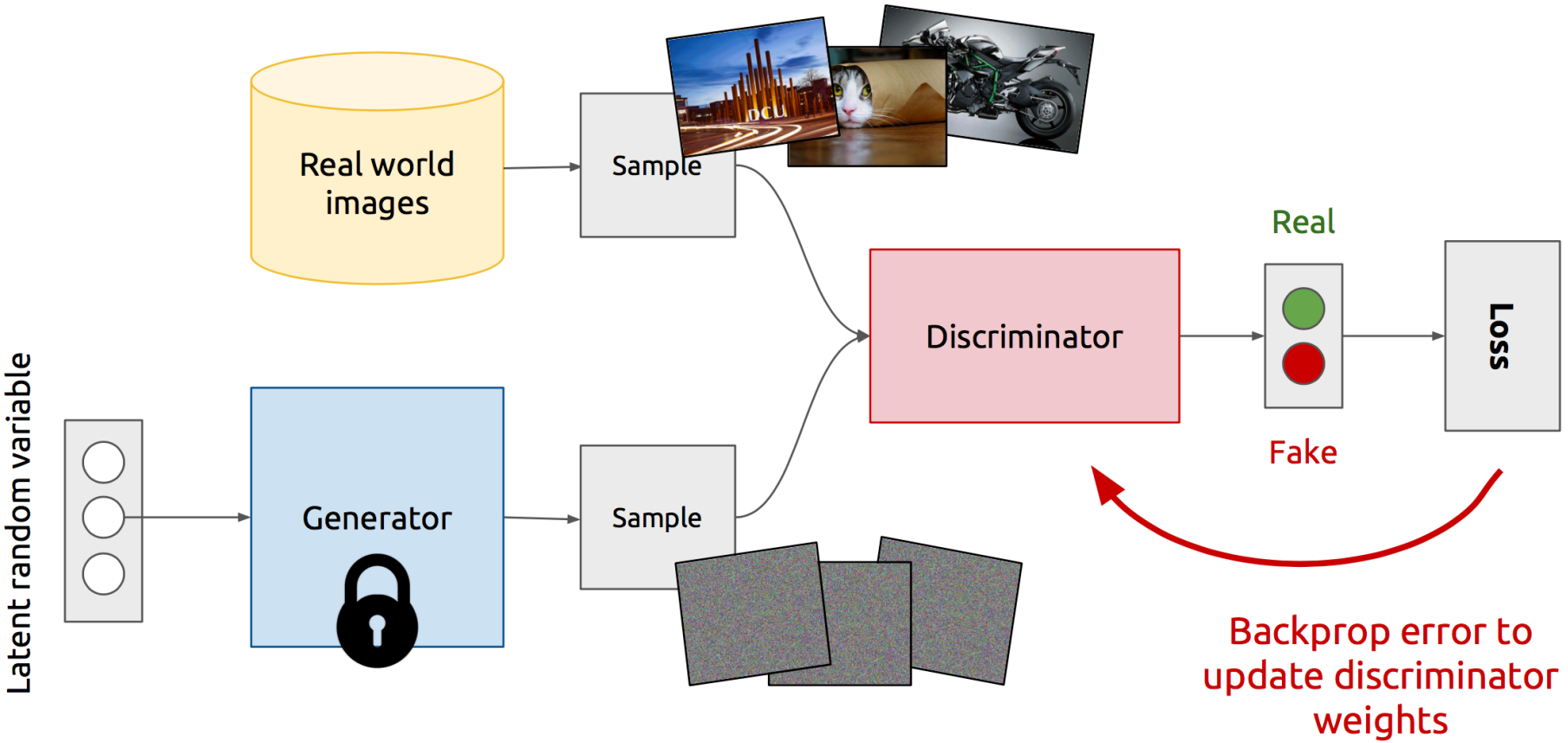
Discriminator v2

沒有彩色

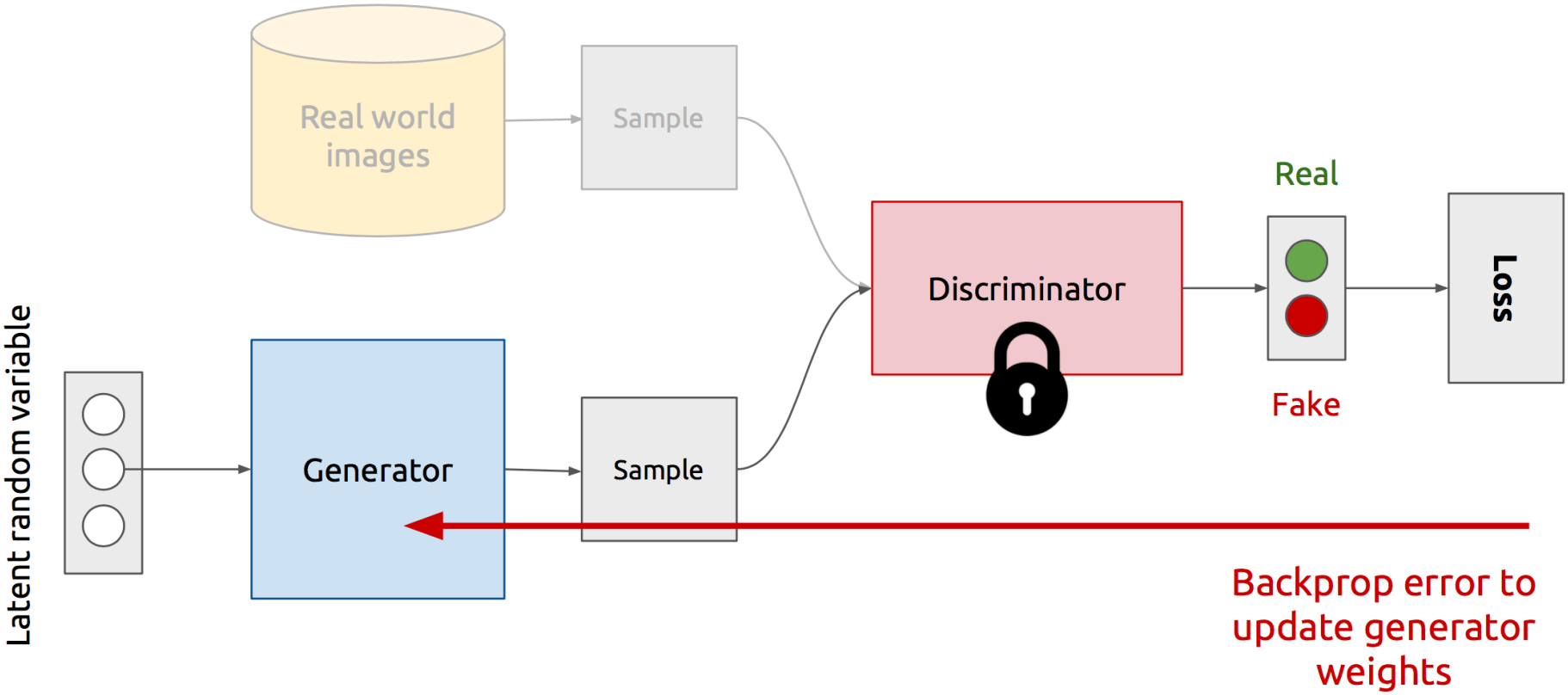
Generator v3

We can train the Generator and Discriminator from data!

Training discriminator



Training generator



The training algorithm of GAN

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

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.

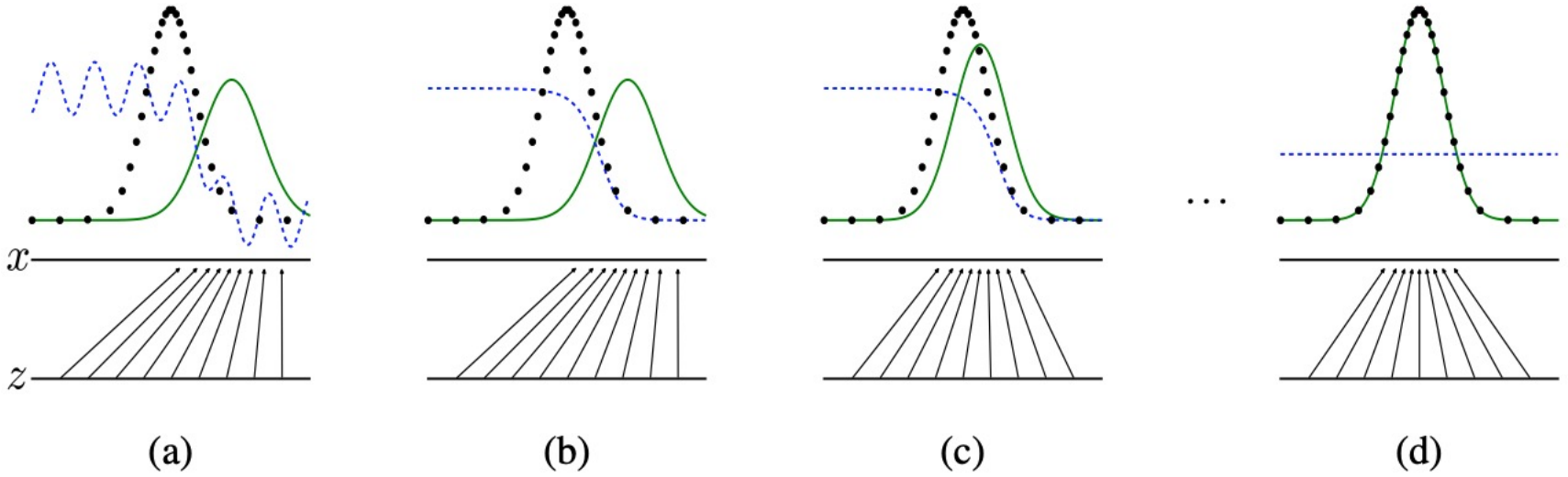


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ 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_{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 $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ 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. $D(\boldsymbol{x}) = \frac{1}{2}$.

A list of all named GANs

https://github.com/hindupuravinash/the-gan-zoo

GAN

ACGAN

BGAN

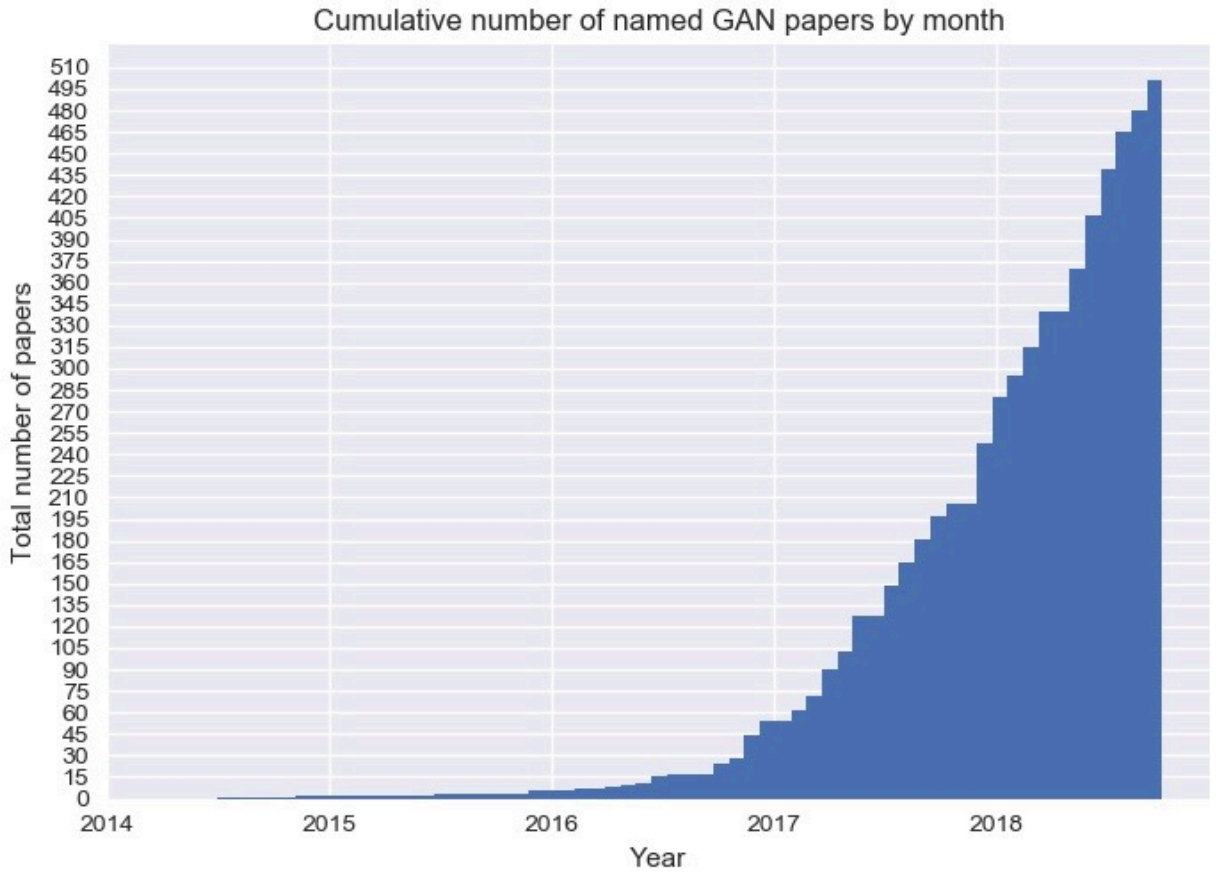
CGAN

DCGAN

EBGAN

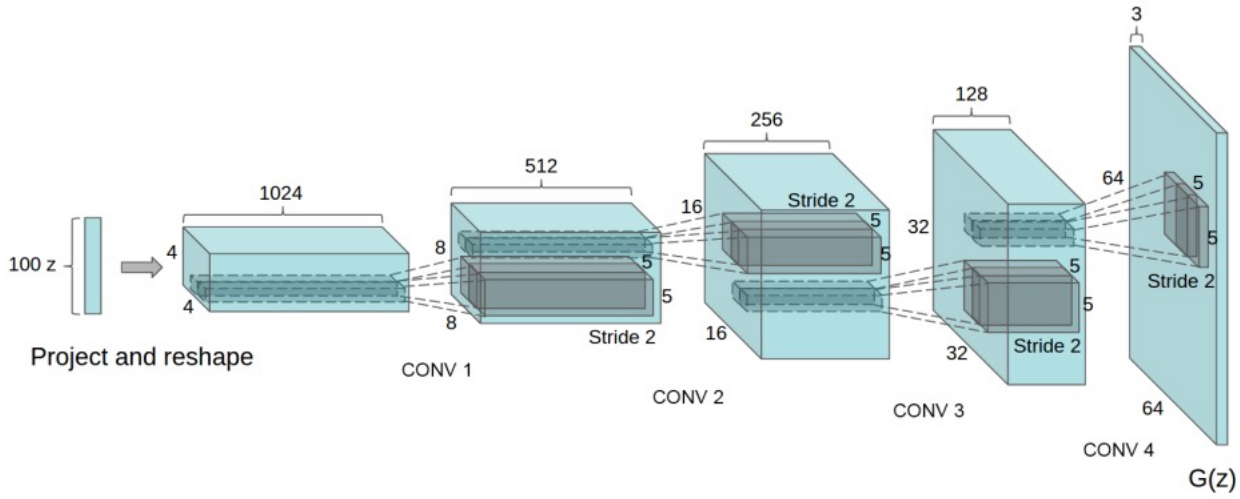
fGAN

GoGAN



Deep Convolutional GANs (DCGAN)

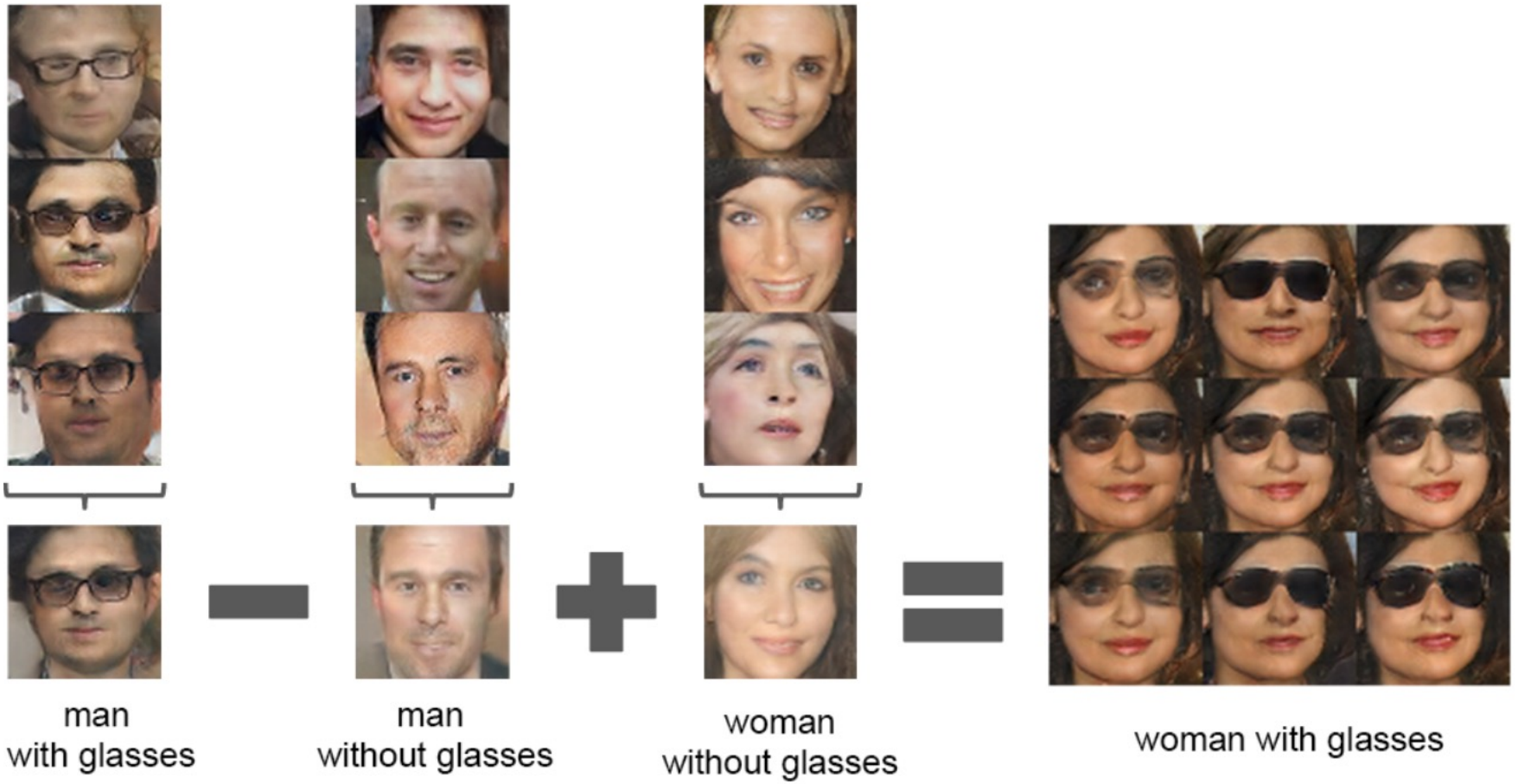
- 判别网络是一个传统的深度卷积网络，但使用了带步长的卷积来实现下采样操作，不用最大汇聚（pooling）操作。
- 生成网络使用一个特殊的深度卷积网络来实现使用微步卷积来生成64×63大小的图像。



Deep Convolutional GANs (DCGAN)

Figure 3: 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.

Deep Convolutional GANs (DCGAN)

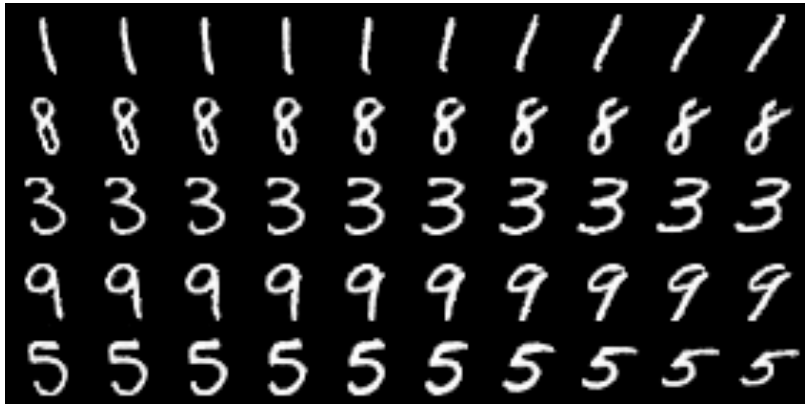


For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y . The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples.

Manipulating latent codes by InfoGAN

(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)

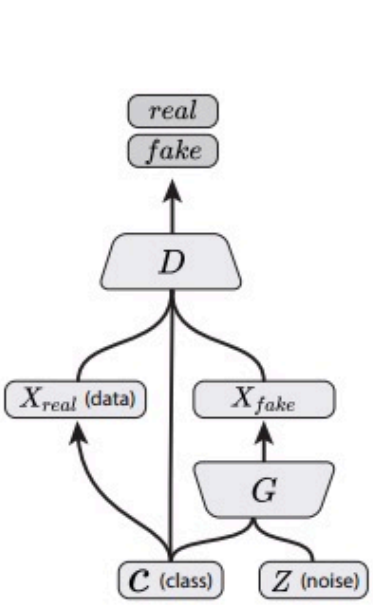


(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

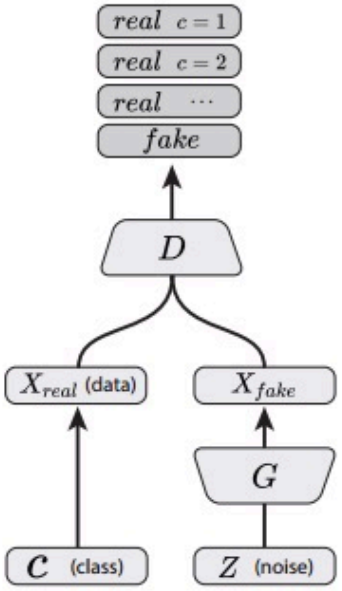
(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

InfoGAN (NIPS 2016)

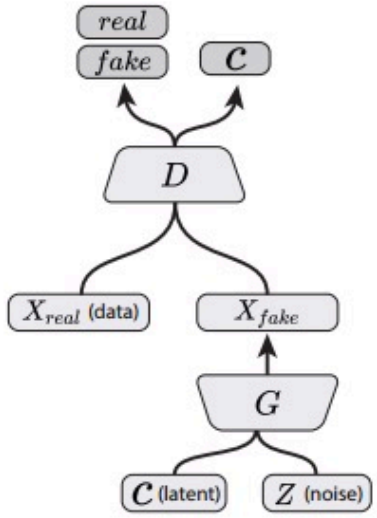
AC-GAN



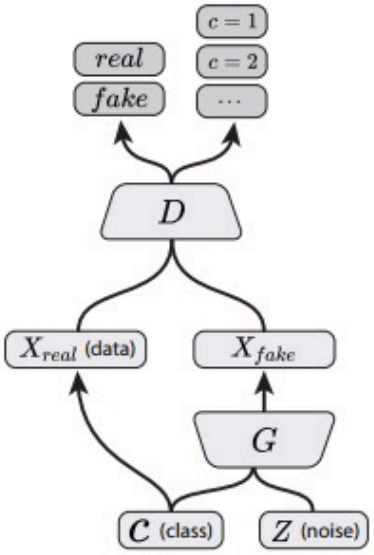
Conditional GAN
(Mirza & Osindero, 2014)



Semi-Supervised GAN
(Odena, 2016; Salimans, et al., 2016)



InfoGAN
(Chen, et al., 2016)



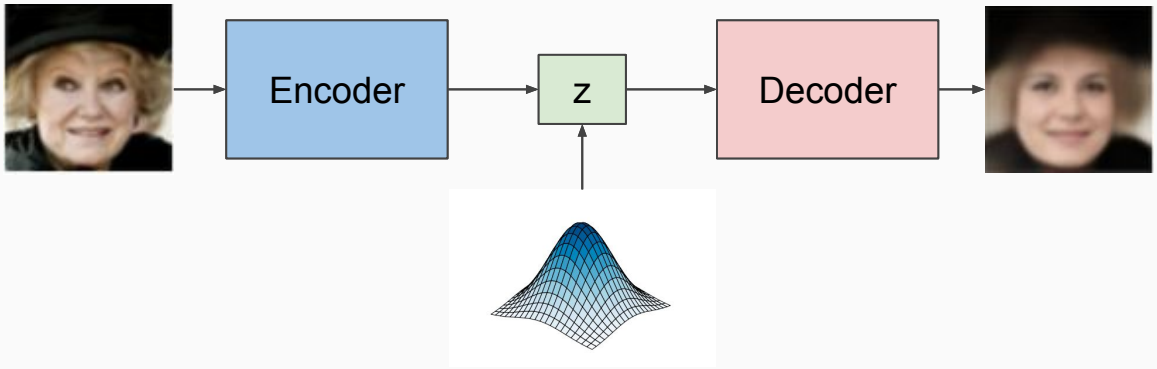
AC-GAN
(Present Work)

https://arxiv.org/abs/1610.09585
https://github.com/clvrai/ACGAN-PyTorch

Outline

- Related concepts
- Variational Autoencoder (VAE)
- Generative Adversarial Networks (GAN)
- **VAE vs. GAN**

VAE vs. GAN

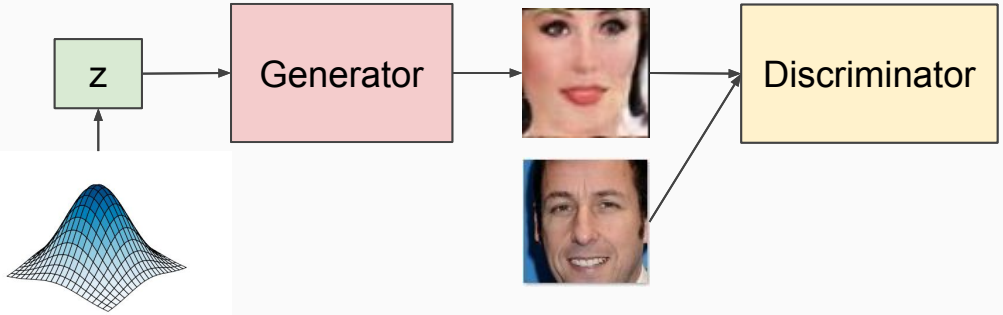


VAE

✓: Given an X **easy** to find z.
✓: Interpretable probability P(X)

X: Usually outputs blurry Images

GAN

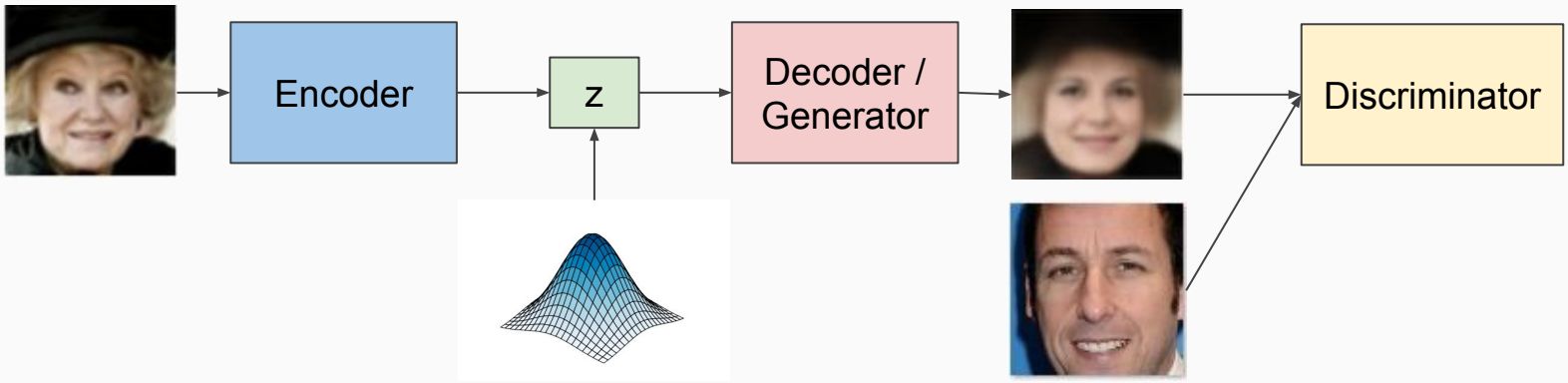


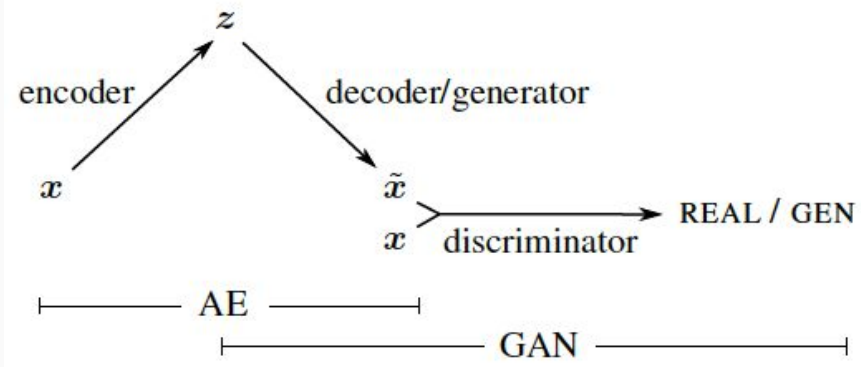
✓: Very sharp images

X: Given an X **difficult** to find z. (Need to backprop.)

✓/X: No explicit P(X).

GAN + VAE: combination



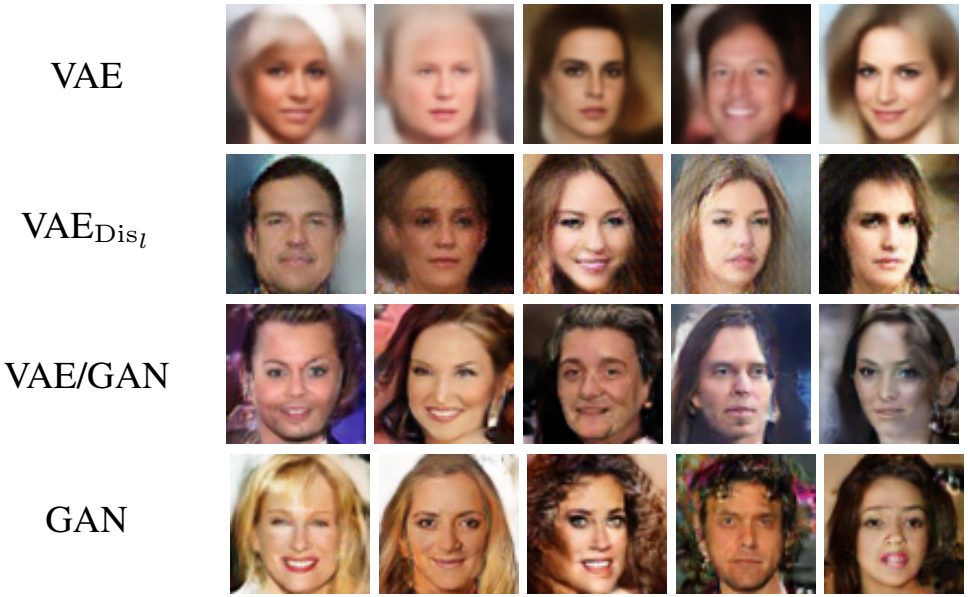


KL Divergence L_2 Difference

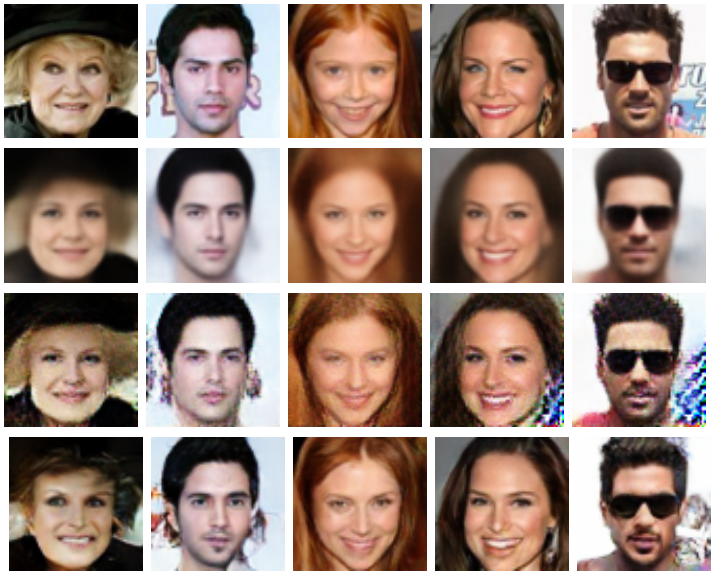
$$\mathcal{L} = \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l} + \mathcal{L}_{\text{GAN}}$$

Larsen et al., Autoencoding beyond pixels using a learned similarity metric, in ICML 2016

Results by VAE vs. GAN



Samples from different generative models



Reconstructions from different autoencoders

VAE_{Disl} : Train a GAN first, then use the discriminator of GAN to train a VAE
VAE/GAN: GAN and VAE trained together

Larsen et al., Autoencoding beyond pixels using a learned similarity metric, in ICML 2016

Thank you!