# Deep Generative Models

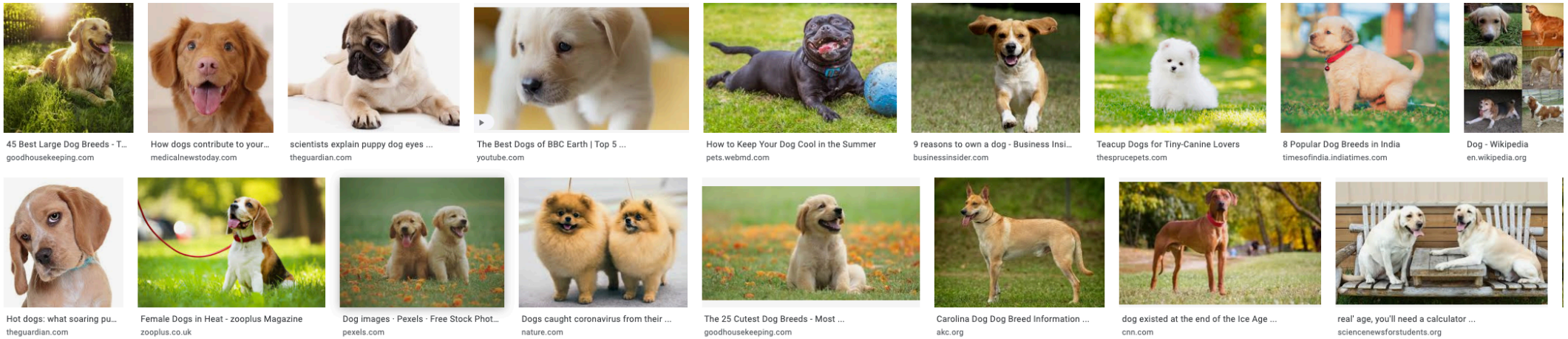
ITM528 Deep Learning

Taemoon Jeong

# Generative Model

What does it mean to learn a generative model?

# Generative Model




Google Search: Dog

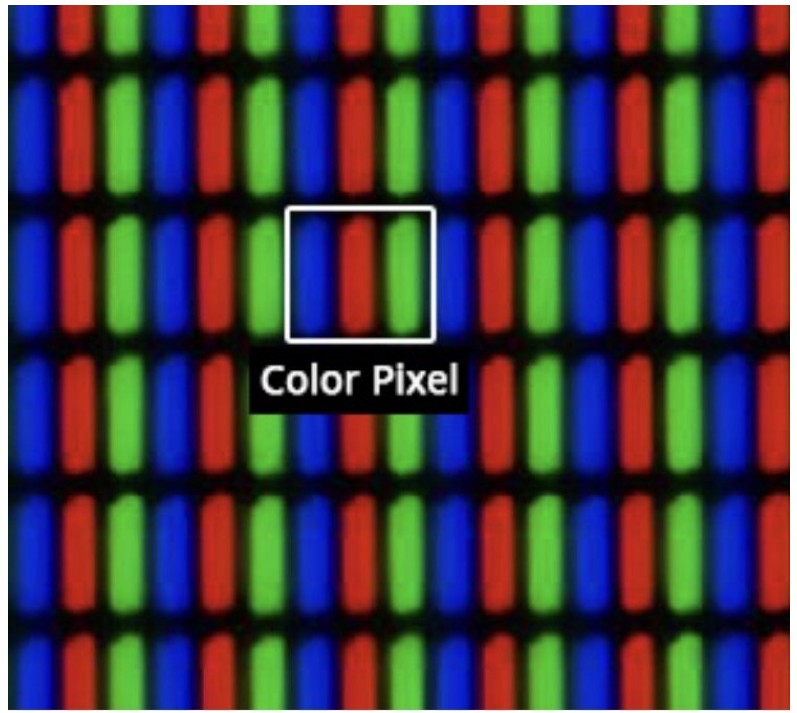
- Suppose that we are given images of dogs.
- We aim to learn a probability distribution $p(x)$ such that
  - we can sample $\tilde{x} \sim p(x)$ where $\tilde{x}$ looks like a dog (aka **generation**)
  - $p(x)$ should be high if $x$ looks like a dog and low otherwise (aka **density estimation**)
- Then, how can we represent $p(x)$?

# Basic Discrete Distributions

- Bernoulli distribution: (biased) coin flip
  - $D = \{\text{Heads}, \text{Tails}\}$
  - Specify $P(X = \text{Heads}) = p$. Then $P(X = \text{Tails}) = 1 - p$.
    - The number of parameters is one.
  - Denote $X \sim \text{Ber}(p)$
- Categorical distribution: (biased) m-sided dice
  - $D = \{1, \ldots, m\}$
  - Specify $P(Y = i) = p_i$ such that $\sum_{i=1}^{m} = 1$.
    - The number of parameters is $m - 1$.
  - Denote $Y \sim \text{Cat}(p_1, \ldots, p_m)$

# Example

- Let's model **a single pixel** of an RGB image.
  - $(r, g, b) \sim P(R, G, B)$
  - Number of possible cases?
    - $256 \times 256 \times 256$
      - Each color channel is independent and has $256$ possible cases.
  - How many parameters do we need?
    - $256 \times 256 \times 256 - 1$
      - We can treat a single pixel as a huge dice with $256 \times 256 \times 256$ sides.




https://www.quora.com/Why-cant-we-color-a-pixel-partially

# (Conditional) Independence

# Example

- Suppose we have a set of images with $n$ binary pixels (binary images)
  - Number of possible cases?
    - $2 \times 2 \times \cdots \times 2 = 2^n$
  - How many parameters do we need?
    - $2^n - 1$
    - If we have an image with $28 \times 28 = 784$ pixels, the number of parameters is approximately $2^{784} \approx 10^{236}$ (where the number of total atoms in the universe is approximately $10^{82}$).

# Structure Through Independence

- What if each pixel $X_1, \ldots, X_n$ is independent?
  - Then, $P(X_1, \ldots X_n) = P(X_1)P(X_2)\cdots P(X_n)$.
  - Number of possible cases?
    - $2 \times 2 \times \cdots \times 2 = 2^n$
    - It is, of course, the same as before.
  - How many parameters do we need?
    - $n$
  - The total $2^n$ entries can be described by just $n$ numbers. What does it mean?

# Probability Rules

- Three important rules in probability theory
  - Chain rule

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$$

  - Bayes' theorem

$$p(x \mid y) = \frac{p(x, y)}{p(y)} = \frac{p(y \mid x)p(x)}{p(y)}$$

  - Conditional independence

$$\text{If } x \perp y \mid z, \text{ then } p(x \mid y, z) = p(x \mid z)$$

# Chain Rule

- Using the chain rule,

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_1, X_2) \cdots P(X_n \mid X_1, \cdots, X_{n-1})$$

  where $X_i \sim \text{Ber}(p)$.

- How many parameters do we need?
  - $P(X_1)$: 1 parameter
  - $P(X_2 \mid X_1)$: 2 parameters (one per $P(X_2 \mid X_1 = 0)$ and one per $P(X_2 \mid X_1 = 1)$)
  - $P(X_3 \mid X_1, X_2)$: 4 parameters
  - Hence, the total number of parameters becomes:

$$1 + 2 + 2^2 + \cdots + 2^{n-1} = 2^n - 1$$

  - This was the same as before. Why?

# Conditional Independence

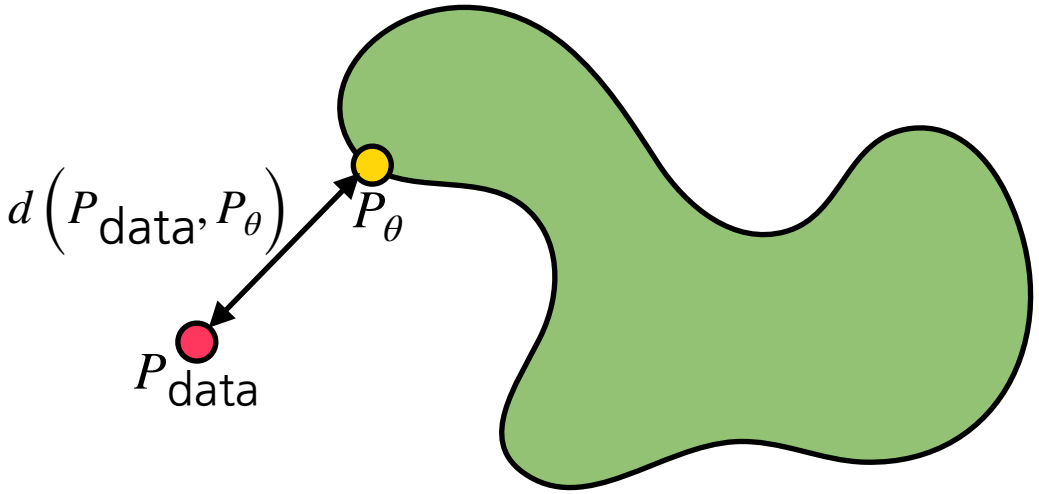
- Now, suppose that $X_{i+1} \perp X_1, \ldots, X_{i-1} \mid X_i$ (aka a Markov assumption), then

$$p(X_1, \ldots, X_n) = p(X_1)p(X_2 \mid X_1)p(X_3 \mid X_2) \cdots p(X_n \mid X_{n-1})$$

- How many parameters do we need?
  - $P(X_1)$: 1 parameter
  - $P(X_2 \mid X_1)$: 2 parameters
  - $P(X_3 \mid X_2)$: 2 parameters
  - Hence, the total number of parameters becomes $2n - 1$.
- By simply utilizing the Markov assumption, we get an exponential reduction in the number of parameters! $((2^n - 1) \rightarrow (2n - 1))$
- **Autoregressive models** leverage this conditional independence.

# Maximum Likelihood Learning

# Learning Generative Models




- Given a training set of examples, we can cast the generative model learning process as finding the **best approximating density** model from the **model family** (a set of all possible solutions).
- One important question to address is

  "How can we evaluate the **goodness** of the approximation?"

# Learning Generative Models

- One widely used metric for computing the similarity between distributions is KL divergence:

$$D_{KL}(P_{\text{data}}(x)\|P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log \frac{P_{\text{data}}(x)}{P_\theta(x)}\right] = \sum_x P_{\text{data}}(x)\log \frac{P_{\text{data}}(x)}{P_\theta(x)}$$

- We can simplify this with

$$D_{KL}(P_{\text{data}}(x)\|P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log \frac{P_{\text{data}}(x)}{P_\theta(x)}\right] = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log P_{\text{data}}(x)\right] - \mathbb{E}_{x\sim P_{\text{data}}}\left[\log P_\theta(x)\right]$$

- When it comes to optimizing $P_\theta$, the first term does not depend on $\theta$, and minimizing the KL divergence becomes equivalent to **maximizing the expected log-likelihood**:

$$\arg\min_\theta D_{KL}(P_{\text{data}}(x)\|P_\theta(x)) = \arg\max_\theta \mathbb{E}_{x\sim P_{\text{data}}}\left[\log P_\theta(x)\right]$$

# Maximum Likelihood Learning

- The maximum likelihood learning of a generative model:

$$\arg\max_{\theta} \mathbb{E}_{x \sim P_{\text{data}}} \left[ \log P_\theta(x) \right]$$

- One can approximate the expected log-likelihood with the empirical log-likelihood:

$$\mathbb{E}_{x \sim P_{\text{data}}} \left[ \log P_\theta(x) \right] = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log P_\theta(x)$$

- The **maximum likelihood learning** is then

$$\arg\max_{\theta} \mathbb{E}_{x \sim P_{\text{data}}} \left[ \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log P_\theta(x) \right]$$

# Empirical Risk Minimization

- For the maximum likelihood learning, **empirical risk minimization (ERM)** is often used.
- However, **ERM** often suffers from the overfitting.
  - Extreme case: the model remembers all training data

$$P(x) = \frac{1}{|\mathscr{D}|} \sum_{i=1}^{|\mathscr{D}|} \delta(x, x_i)$$

- To achieve a better generalization performance, we typically restrict the **hypothesis space** (model family) of distributions that we search over.
  - For example, $P_\theta(x) = \mathcal{N}(x; \mu, \Sigma)$ where $\theta = \{\mu, \Sigma\}$.
- However, it may also deteriorate the performance of the generative model.

# Maximum Likelihood Learning

- Usually, the maximum likelihood learning is prone to under-fitting as we often use simple parametric distributions such as spherical Gaussians.

- What about other ways of measuring the similarity?
  - **KL divergence**: it leads to maximum likelihood learning (variational autoencoder or diffusion models).
  - **Jensen-Shannon divergence**: it leads to adversarial learning (generative adversarial networks).
  - **Wasserstein distance**: it leads to Wasserstein autoencoder (adversarial autoencoder).

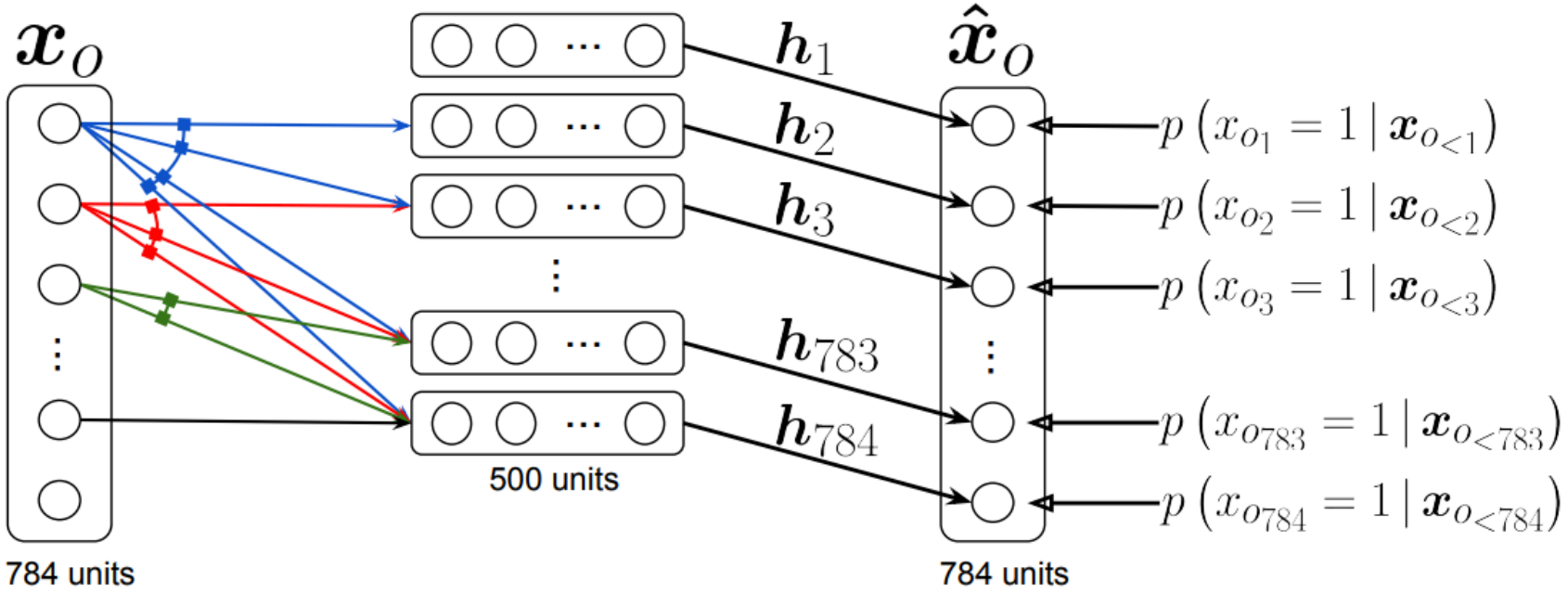
# Autoregressive Models

"The Neural Autoregressive Distribution Estimator," 2011

# Autoregressive Model

- Suppose we have $28 \times 28$ binary images.
- Our goal is to learn $P(X) = P(X_1, \ldots, P_{784})$ over $X \in \{0,1\}^{784}$.
- Then, how can we parametrize $P(X)$?
  - Let's use the chain rule to factor the joint distribution into conditionals.
  - In other words,
    - $P(X_{1:784}) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_2) \cdots P(X_{784} \mid X_{783})$.
    - This is called an autoregressive model.
    - Note that we need an ordering (e.g., raster scan order) of random variables.

# NADE: The Neural Autoregressive Distribution Estimator




- The probability distribution of $i$-th pixel is

$$p(x_i \mid x_{1:i-1}) = \sigma(\alpha_i h_i + b_i) \text{ where } h_i = \sigma(W_{<i} x_{1:i-1} + c).$$

# NADE: The Neural Autoregressive Distribution Estimator

- NADE (or most autoregressive models) is an explicit model that can not only sample plausible outputs but also compute the density of the given inputs.
- How can we compute the probability density of the given image?
  - Suppose we have a binary image with $784$ binary pixels.
  - Then, the joint probability is computed as follows:
    - $p(x_1, \ldots, x_{784}) = p(x_1)p(x_2 \mid x_1) \cdots p(x_{784} \mid x_{783})$ where each conditional probability $p(x_i \mid x_{i-1})$ is computed independently.
    - We can easily compute the log probability:

$$\log p(x_1, \ldots, x_{784}) = \log p(x_1) + \log p(x_2 \mid x_1) + \log p(x_{784} \mid x_{783})$$

- Since we factor the joint distribution into conditionals, we may use more complex probability distributions, such as a **mixture of Gaussians**.

# Summary

- The autoregressive models have two major strengths:
- Easy to **sample** from
  - Sample $\bar{x}_0 \sim p(x_0)$
  - Sample $\bar{x}_1 \sim p(x_1 \mid x_0 = \bar{x}_0)$
  - ⋯ and so forth (in a sequential manner, with a predefined order)
- Easy to **compute** the probability $p(x = \bar{x})$
  - Compute $p(x_0 = \bar{x}_0)$
  - Compute $p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0)$
  - ⋯ and so forth (sum their logarithms)
  - Ideally, we can compute all these terms in parallel (hence fast).
- Easy to be extended to **continuous** random variables:
  - Use a mixture of Gaussians.

# Latent Variable Models

"Auto-Encoding Variational Bayes," 2013

"Adversarial Autoencoders," 2015

"Wasserstein Auto-Encoders," 2017

# Question

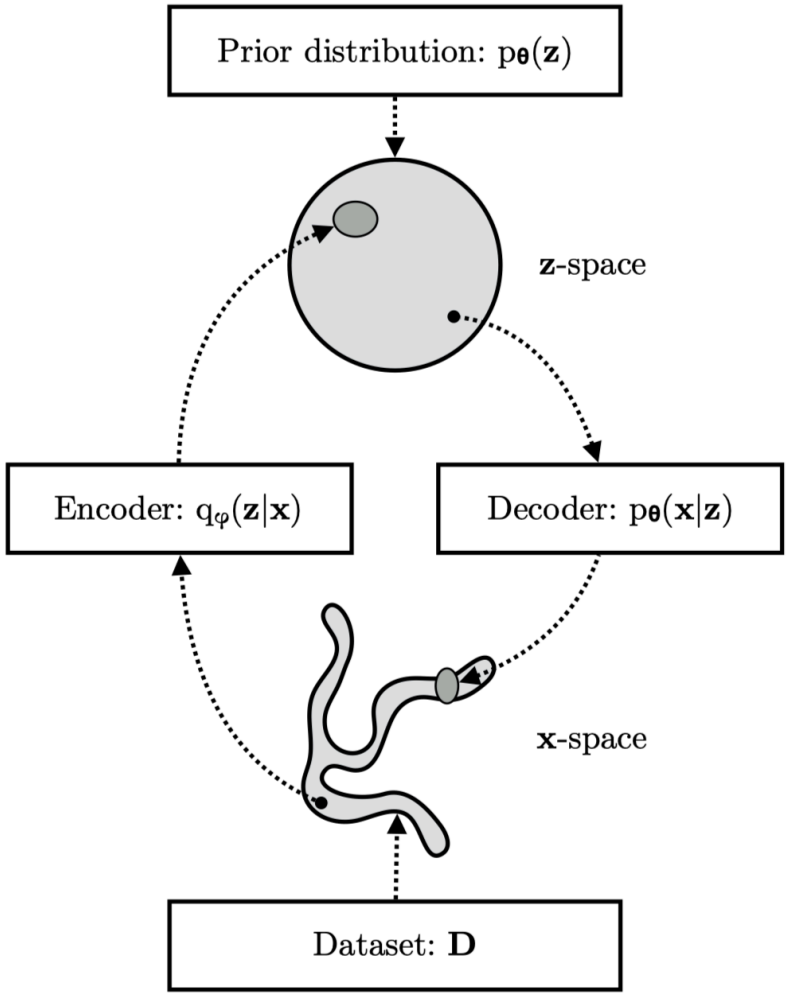
Is an **auto-encoder** a generative model?

# Variational Auto-Encoder

- The objective is simple.

$$\text{Maximize } p_\theta(\mathbf{x})$$

# Variational Auto-Encoder




D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

# Variational Auto-Encoder

$$
\begin{aligned}
\underbrace{\log p_\theta(x)}_{\text{Maximum Likelihood Learning} \uparrow} &= \int_z \underbrace{q_\phi(z|x)}_{\text{For any } q_\phi} \log p_\theta(x) dz \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x)p_\theta(z|x)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \\
&= \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{\text{ELBO} \uparrow} + \underbrace{D_{KL}\left(q_\phi(z|x)\|p_\theta(z|x)\right)}_{\text{Variational Gap} \downarrow}
\end{aligned}
$$

# Variational Auto-Encoder

$$\underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{\text{ELBO}\uparrow} = \int \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} q_\phi(z|x)dz$$

$$= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[p_\theta(x|z)\right]}_{\text{Reconstruction Term}} - \underbrace{D_{KL}\left(q_\phi(z|x)\|p(z)\right)}_{\text{Prior Fitting Term}}$$

D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

# Variational Auto-Encoder

$$\underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{\text{ELBO}\uparrow} = \int \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} q_\phi(z|x)dz$$

$$= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[p_\theta(x|z)\right]}_{\text{Reconstruction Term}} - \underbrace{D_{KL}\left(q_\phi(z|x)\|p(z)\right)}_{\text{Prior Fitting Term}}$$

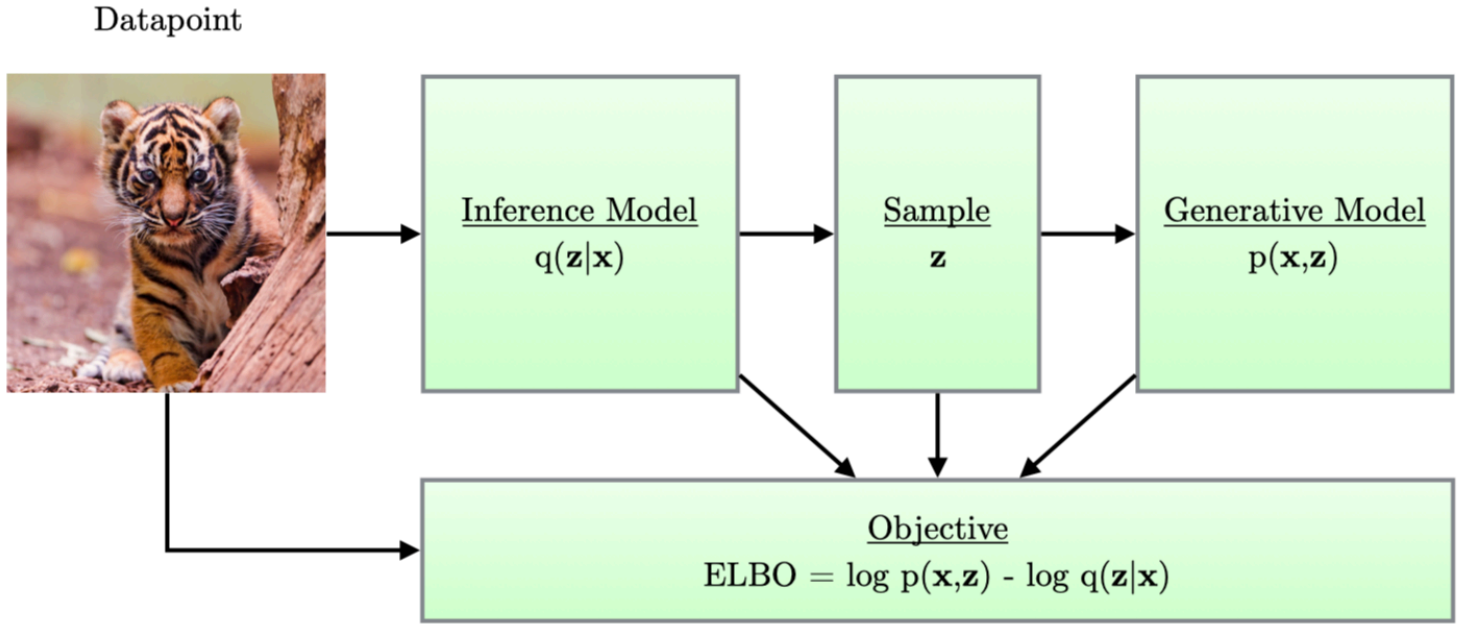
- Key Limitation
  - It is an **intractable** model (hard to evaluate likelihood).
  - The prior fitting term should be differentiable, hence it is hard to use diverse latent prior distributions.
  - In most cases, we use an isotropic Gaussian where we have a closed-form for the prior fitting term.

$$\underbrace{D_{KL}\left(q_\phi(z|x)\|\mathcal{N}(0,I)\right)}_{\text{Prior Fitting Term}} = \frac{1}{2}\sum_{i=1}^{D}\left(\sigma_{z_i}^2 + \mu_{z_i}^2 - \log(\sigma_{z_i}^2) - 1\right)$$

D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

# Variational Auto-Encoder

- Key Limitation
  - It is an **intractable** model (hard to evaluate likelihood).
  - The prior fitting term should be differentiable, hence it is hard to use diverse latent prior distributions.
  - In most cases, we use an isotropic Gaussian where we have a closed-form for the prior fitting term.




D. Kingma, "Variational Inference and Deep Learning: A New Synthesis," Ph.D. Thesis

# Re-parametrization Trick

- Want to compute a gradient with respect to $\phi$ of

$$E_{q(z;\phi)}[r(z)] = \int q(z;\phi) r(z) dz$$

  where $z$ is continuous.

- Suppose $q(z;\phi) = \mathcal{N}(\mu, \sigma^2)$ is Gaussian with parameters $\phi = (\mu, \sigma)$. These are equivalent ways of sampling:
  - Sample $z \sim q_\phi(z)$
  - Sample $\epsilon \sim \mathcal{N}(0,1)$, then $z = \mu + \sigma\epsilon = g(\epsilon;\phi)$
- Using this equivalence, we compute the expectation in two ways:

$$E_{z\sim q(z;\phi)}[r(z)] = E_{\epsilon\sim\mathcal{N}(0,1)}[r(g(\epsilon;\phi))] = \int p(\epsilon) r(\mu + \sigma\epsilon) d\epsilon$$

$$\nabla_\phi E_{q(z;\phi)}[r(z)] = \nabla_\phi E_\epsilon[r(g(\epsilon;\phi))] = E_\epsilon[\nabla_\phi r(g(\epsilon;\phi))]$$

- It is easy to approximate the gradient by

$$E_\epsilon[\nabla_\phi r(g(\epsilon;\phi))] \approx \frac{1}{k}\sum_k \nabla_\phi r(g(\epsilon^k;\phi)) \text{ where } \epsilon^k \sim \mathcal{N}(0,1)$$

# Amortization

$$\max_{\theta} l(\theta; \mathcal{D}) \geq \max_{\theta, \phi^1, \phi^2, \cdots} \sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i; \theta, \phi^i)$$

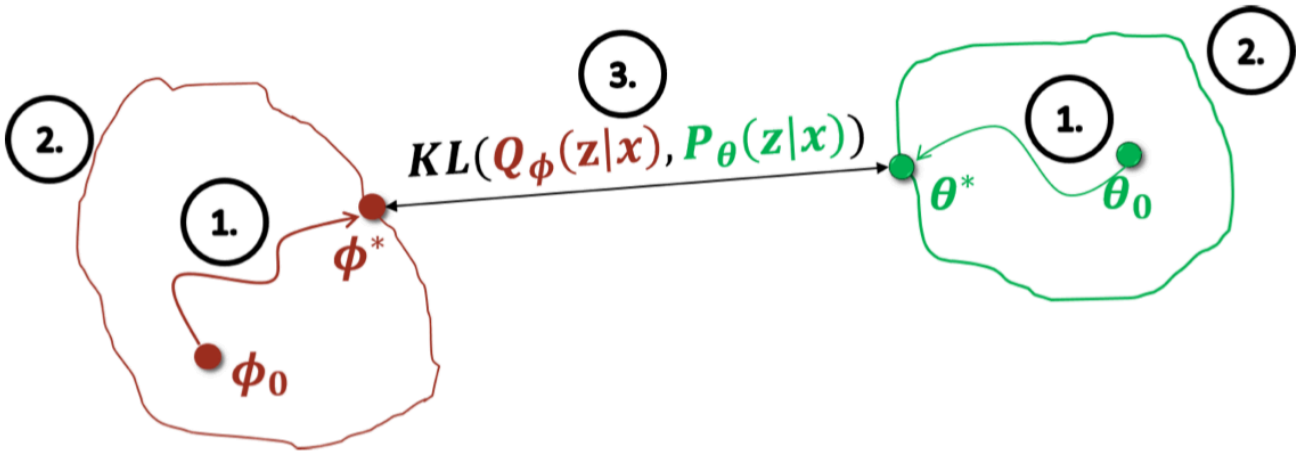
- So far, we have used a set of variational parameters (e.g., $\mu$ and $\sigma$) $\phi^i$ for **each** data point $x^i$.
  - In other words, we usually estimate $\phi^i = (\mu_i, \sigma_i^2)$ for each $x^i$.
- However it does not scale to large datasets.

- **Amortization**: Now we learn a **single** parametric function $f_\lambda$ that maps each $x$ to a set of (good) variational parameters (i.e., $f_\lambda : x^i \mapsto \phi^i$)
  - For example, if $q(z|x^i)$ are Gaussians with different means and variances, we learn a single neural network $f_\lambda$ mapping $x^i$ to $(\mu^i, \sigma^i)$.
  - This is an encoder network in VAE.
- We approximate the posteriors $q(z|x^i)$ using the distribution $q_\lambda(z|x)$.

# Summary of Latent Variable Models

1. Combine simple models to get a more flexible one (e.g., mixture of Gaussians)
2. Directed model permits ancestral sampling (efficient generation):
    1. $z \sim p(z)$
    2. $x \sim p(x \mid z; \theta)$
3. However, log-likelihood is generally intractable, hence learning is difficult.
4. Joint learning of a (decoder) model ($\theta$) and an amortized inference component ($\phi$) to achieve tractability via ELBO optimization
5. Latent representation for any $x$ can be inferred via $q_\phi(z \mid x) = q(z; f_\lambda(x))$ (aka an encoder model).

# Research Directions




1. Better optimization techniques
2. More expressive approximating (variational) families
3. Alternate loss functions

# Adversarial Auto-encoder

- Let $x$ be the input and $z$ be the latent vector.
- Let $p(z)$ be the prior distribution, $q(z|x)$ be an encoding distribution, and $p(x|z)$ be the decoding distribution.
- Also let $p_d(x)$ be the data distribution, and $p(x)$ be the model distribution.
- The **aggregated posterior distribution** $q(z)$ is defined by the encoder function $q(z|x)$:

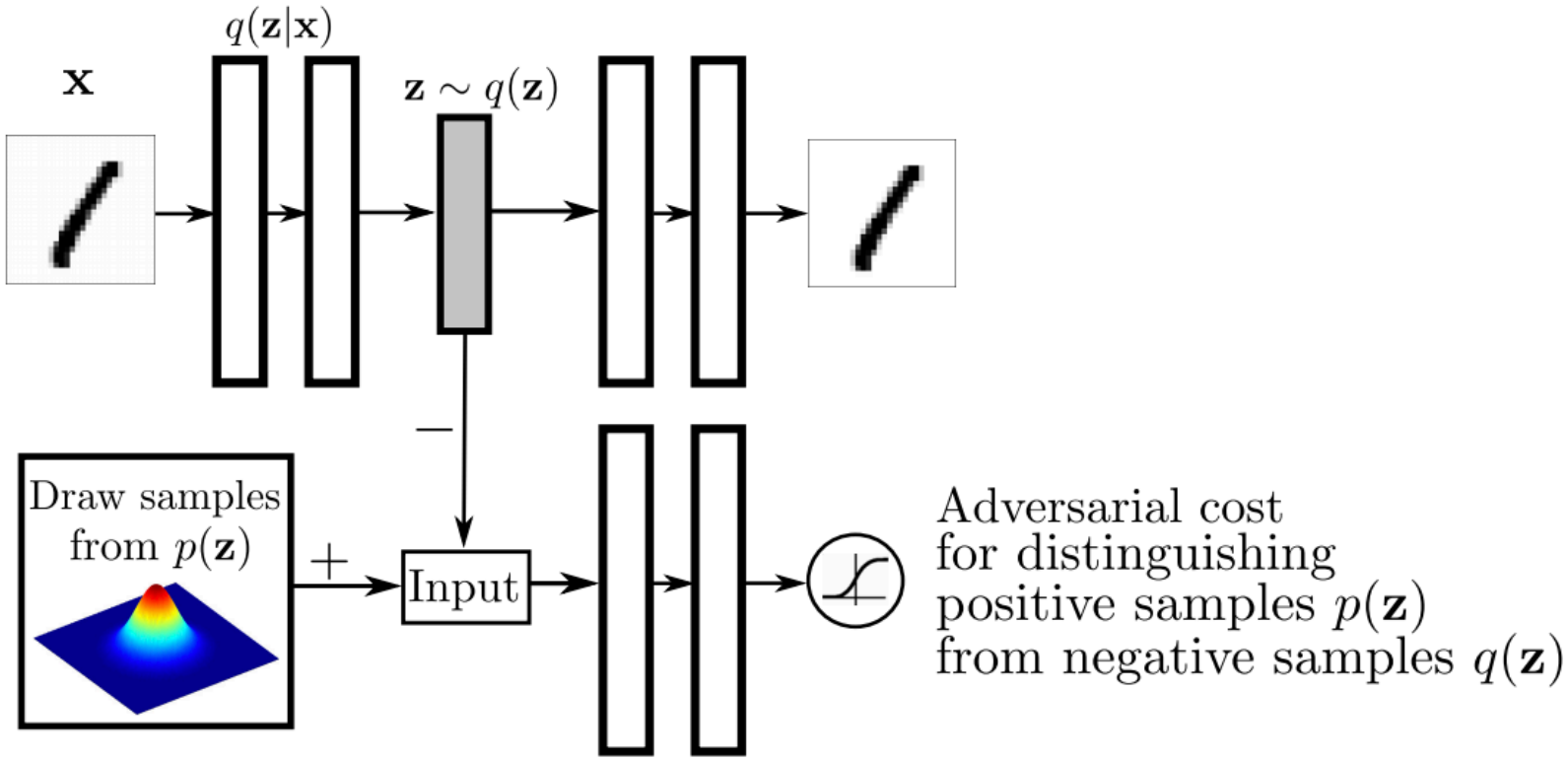
$$q(z) = \int_x q(z|x)p_d(x)dx$$

- The adversarial encoder (AAE) is an autoencoder that is **regularized by matching** the aggregated posterior, $q(z)$, to an arbitrary prior, $p(z)$.
- To achieve this, an adversarial network is attached on top of the latent vector of the autoencoder that guides $q(z)$ to match $p(z)$.

https://arxiv.org/abs/1511.05644

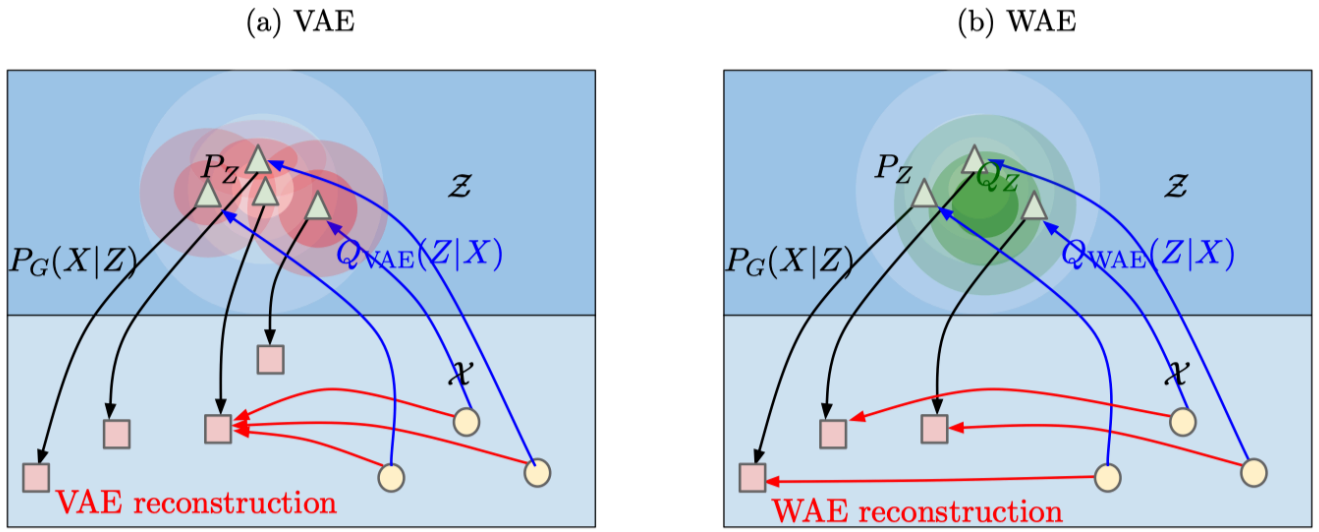
# Adversarial Auto-encoder

- AAE allows us to **use any arbitrary latent prior distributions** that we can sample.




https://arxiv.org/abs/1511.05644

# Wasserstein Auto-encoders




- Both **VAE** and **WAE** minimize two terms: the reconstruction cost and the regularizer penalizing discrepancy between $P_Z$ and distribution induced by the encoder $Q$.
- VAE forces $Q(Z|X=x)$ to match $P_Z$ for all the different input examples $x$ drawn from $P_X$.
- This is illustrated on picture (a), where every single red ball is forced to match $P_Z$ depicted as the white shape. Red balls start **intersecting**, which leads to problems with reconstruction.
- In contrast, WAE forces the continuous mixture $Q_Z = \int Q(Z|X)dP_X$ to match $P_Z$, as depicted with the green ball in picture (b). As a result, latent vectors of different examples get a chance to **stay far way** from each other, promoting a better reconstruction.

https://arxiv.org/abs/1711.01558

# Wasserstein Auto-encoders

- Optimal transport and its dual formulations
  - A rich class of divergences between probability distributions is induced by the optimal transport (OT) problem.

$$W_c(P_X, P_G) = \inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} E_{(X,Y)\sim\Gamma}[c(X, Y)]$$

where $c(x, y) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is any measurable cost function and $\mathcal{P}(X \sim P_X, Y \sim P_G)$ is a set of all joint distributions of $(X, Y)$ with marginals $P_X$ and $P_G$, respectively.

  - A particularly interesting case is when $(\mathcal{X}, d)$ is a metric space and $c(x, y) = d^p(x, y)$ for $p \geq 1$. In this case, $W_c$ is called the $p$-Wasserstein distance.

https://arxiv.org/abs/1711.01558

# Wasserstein Auto-encoders

**Theorem 1.** *For $P_G$ as defined above with deterministic $P_G(X|Z)$ and any function $G: \mathcal{Z} \to \mathcal{X}$*

$$\inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} \mathbb{E}_{(X,Y)\sim\Gamma}\left[c(X,Y)\right] = \inf_{Q:\, Q_Z = P_Z} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\left[c(X, G(Z))\right],$$

*where $Q_Z$ is the marginal distribution of $Z$ when $X \sim P_X$ and $Z \sim Q(Z|X)$.*

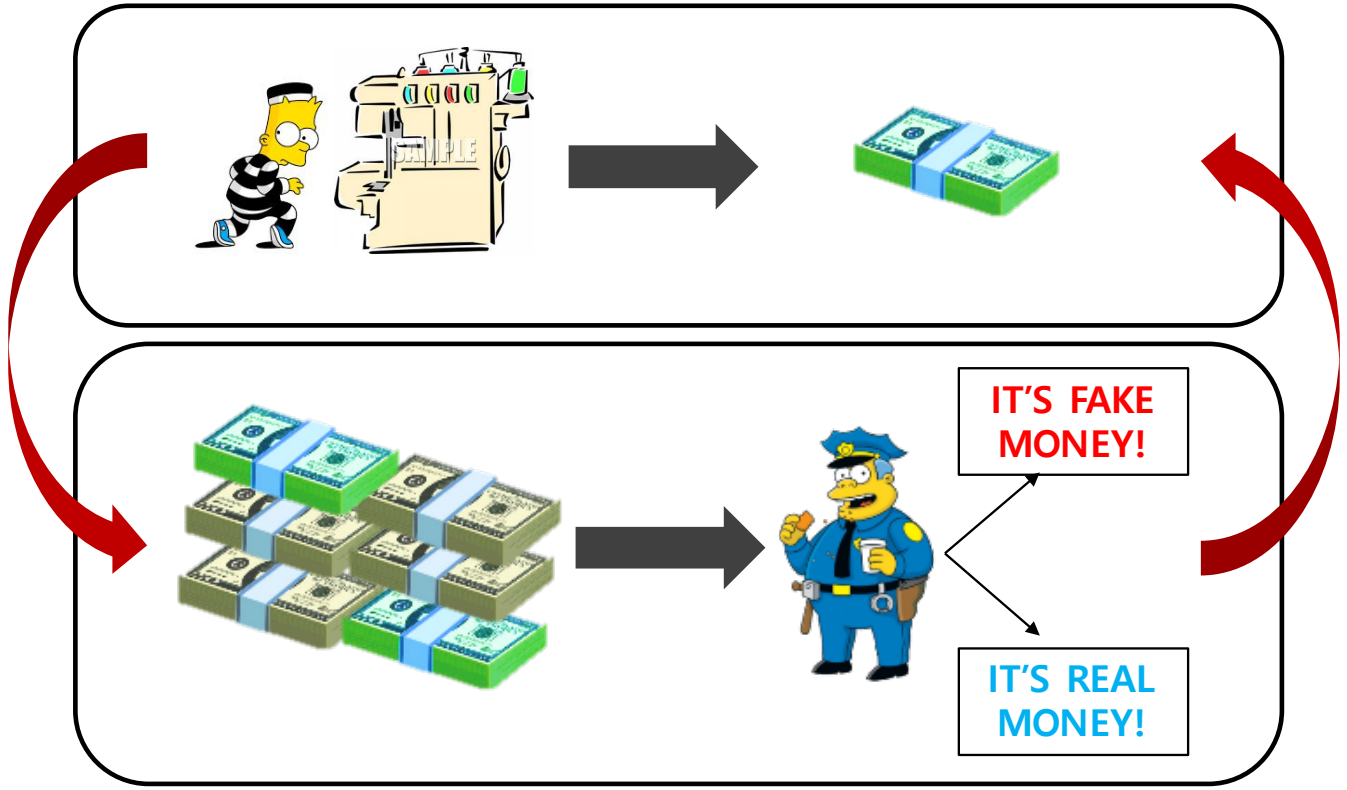
$$D_{\mathrm{WAE}}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\left[c(X, G(Z))\right] + \lambda \cdot \mathcal{D}_Z(Q_Z, P_Z), \quad \text{(WAE objective)}$$

- What it means is that
  1. We can reformulate the **optimal transport problem** into the **WAE objective** using its dual formulations.
  2. The WAE objective contains a reconstruction loss and an arbitrary divergence between the aggregated posterior $Q_Z = \int Q(Z|X)dP_X$ and the prior $P_Z$.
  3. If we use Jensen-Shannon divergence for $D_Z$ (i.e., $D_Z(Q_Z, P_Z) = D_{JS}(Q_Z, P_Z)$, and use the adversarial training to estimate it, it becomes the **adversarial auto-encoder**.

# Generative Adversarial Network

"Generative Adversarial Networks," 2014

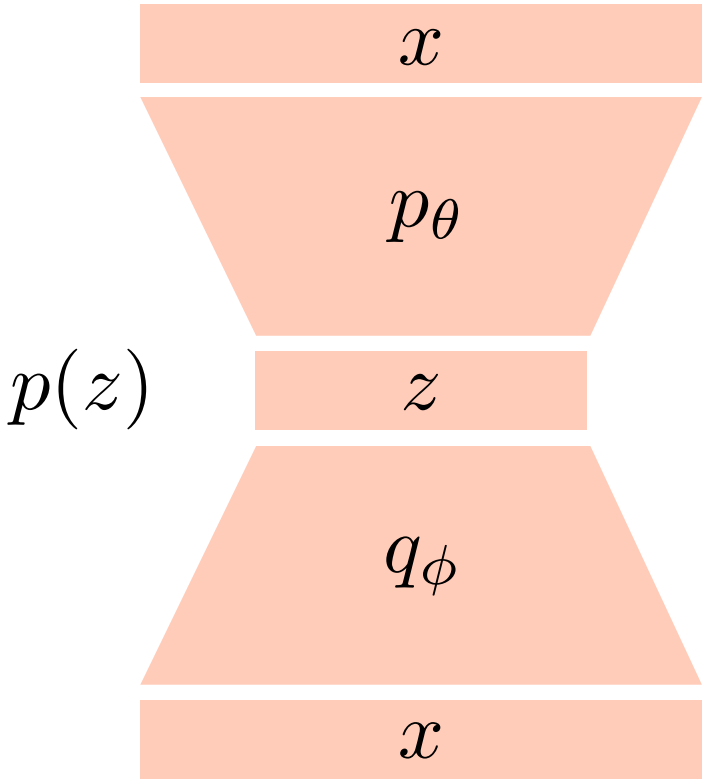
# GAN



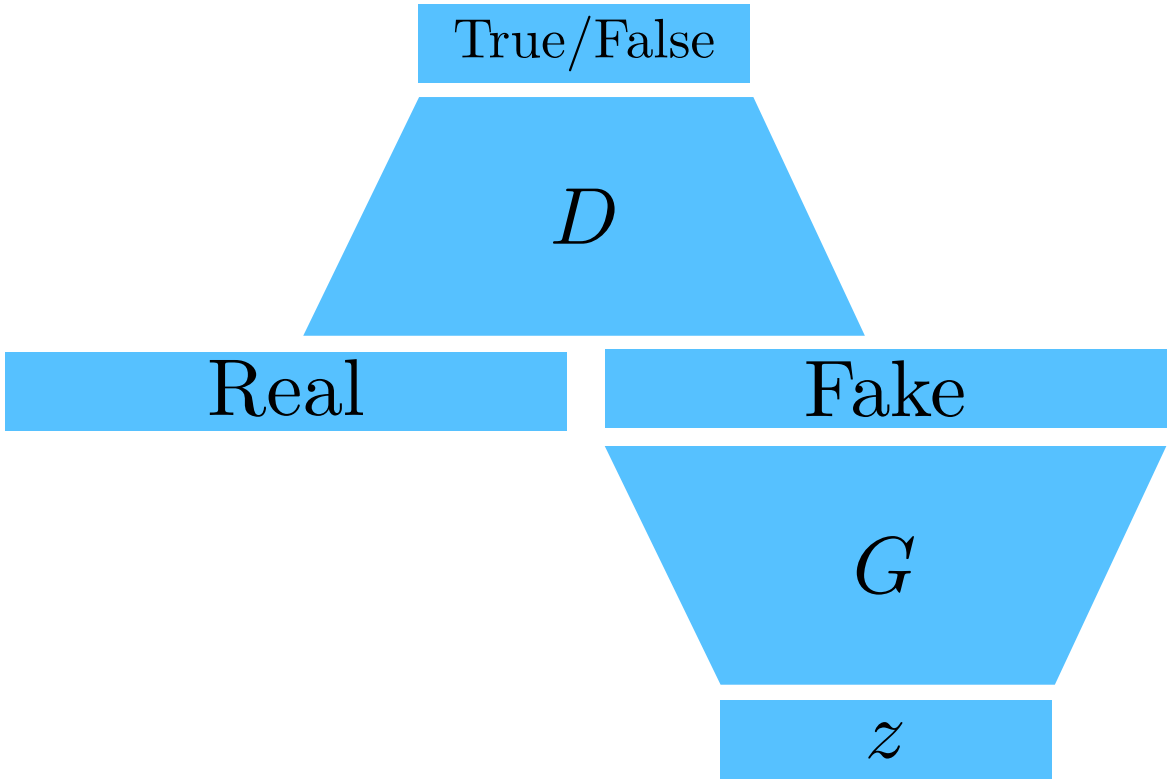

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

https://arxiv.org/abs/1406.2661

# GAN vs. VAE




Variational Auto-encoder




Generative Adversarial Networks

# GAN Objective

- GAN is a  two player minimax game between **generator** and **discriminator**.
  - **Discriminator** objective

$$\max_D V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G}[\log(1 - D(\mathbf{x}))]$$

  - The **optimal discriminator** is

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

https://arxiv.org/abs/1406.2661

# GAN Objective

- GAN is a  two player minimax game between **generator** and **discriminator**.
  - **Generator** objective

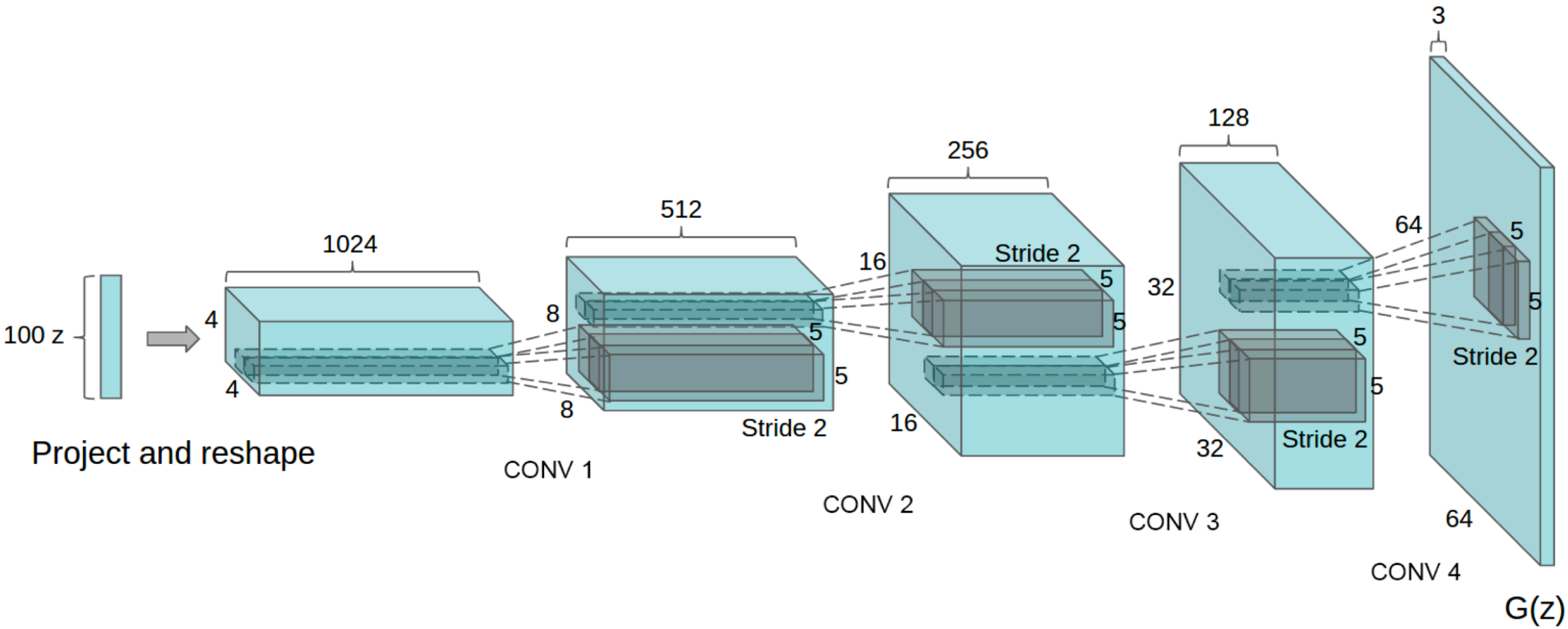
$$\min_G V(G,D) = E_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + E_{\mathbf{x}\sim p_G}[\log(1 - D(\mathbf{x}))]$$

  - Plugging in the **optimal discriminator**, we get

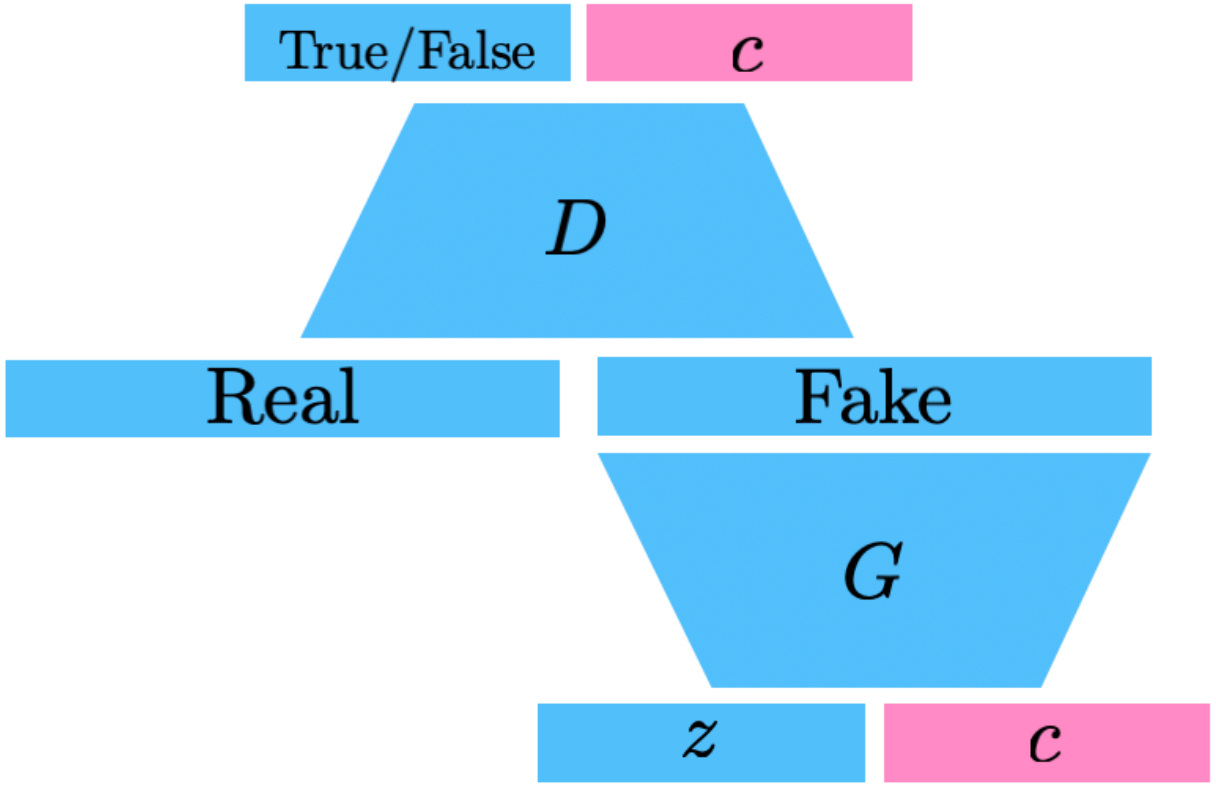
$$\begin{aligned} V(G, D_G^*(\mathbf{x})) &= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}\right] \\ &= E_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] + E_{\mathbf{x}\sim p_G}\left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_G(\mathbf{x})}{2}}\right] - \log 4 \\ &= \underbrace{D_{KL}\left[p_{\text{data}}, \frac{p_{\text{data}}+p_G}{2}\right] + D_{KL}\left[p_G, \frac{p_{\text{data}}+p_G}{2}\right]}_{2\times\text{Jenson-Shannon Divergence (JSD)}} - \log 4 \\ &= 2D_{JSD}[p_{\text{data}}, p_G] - \log 4 \end{aligned}$$

https://arxiv.org/abs/1406.2661

# DCGAN

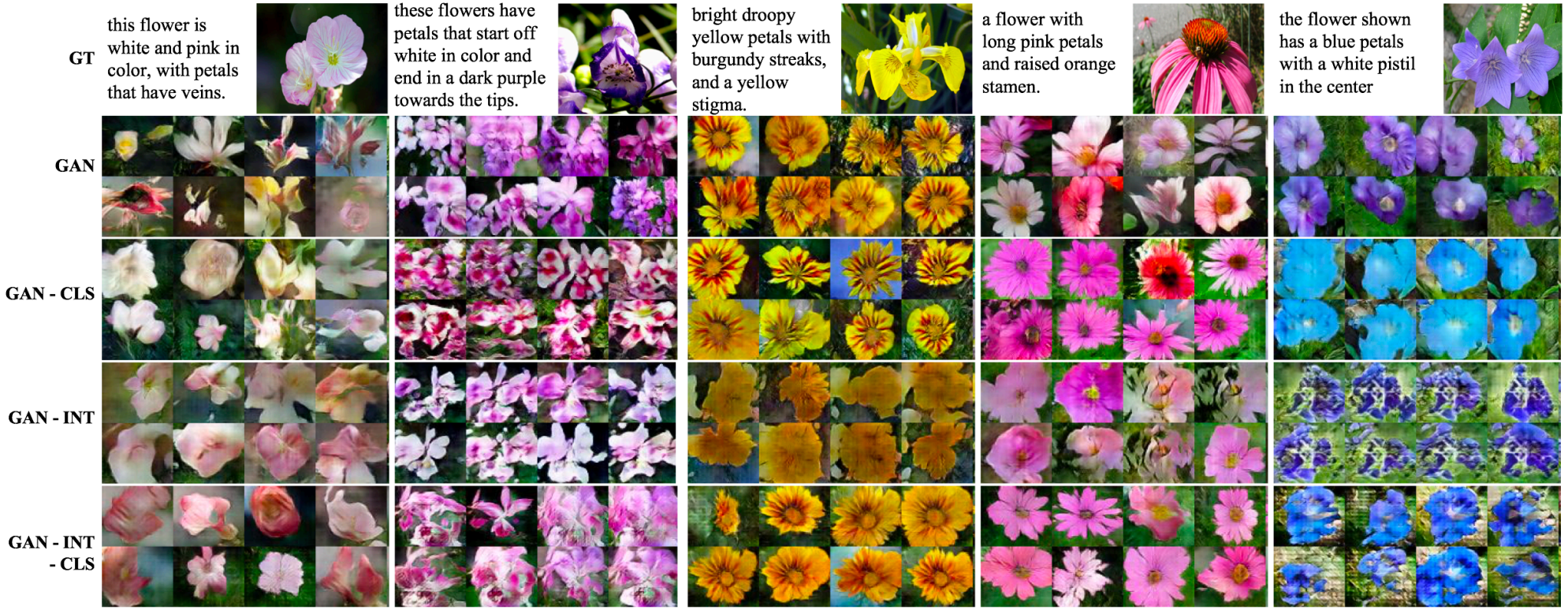

100 z
Project and reshape
1024
4
4
CONV 1
512
8
8
Stride 2
CONV 2
256
16
16
Stride 2
CONV 3
128
32
32
Stride 2
CONV 4
3
64
64
Stride 2
5
G(z)

# Info-GAN



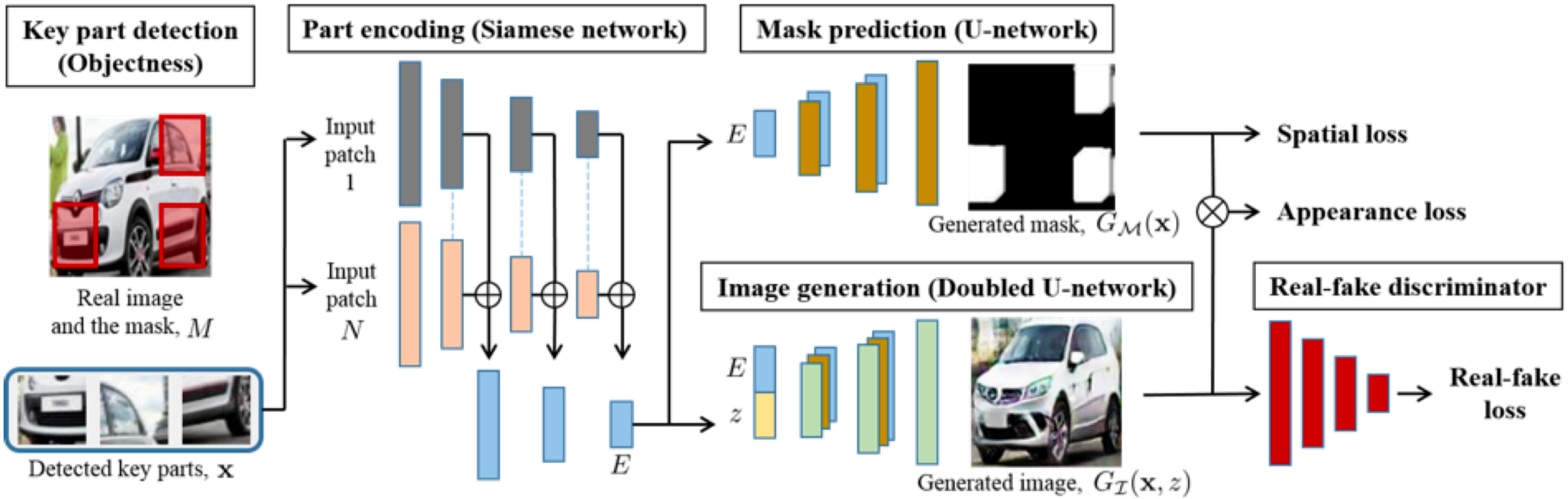

https://arxiv.org/abs/1606.03657

# Text2Image



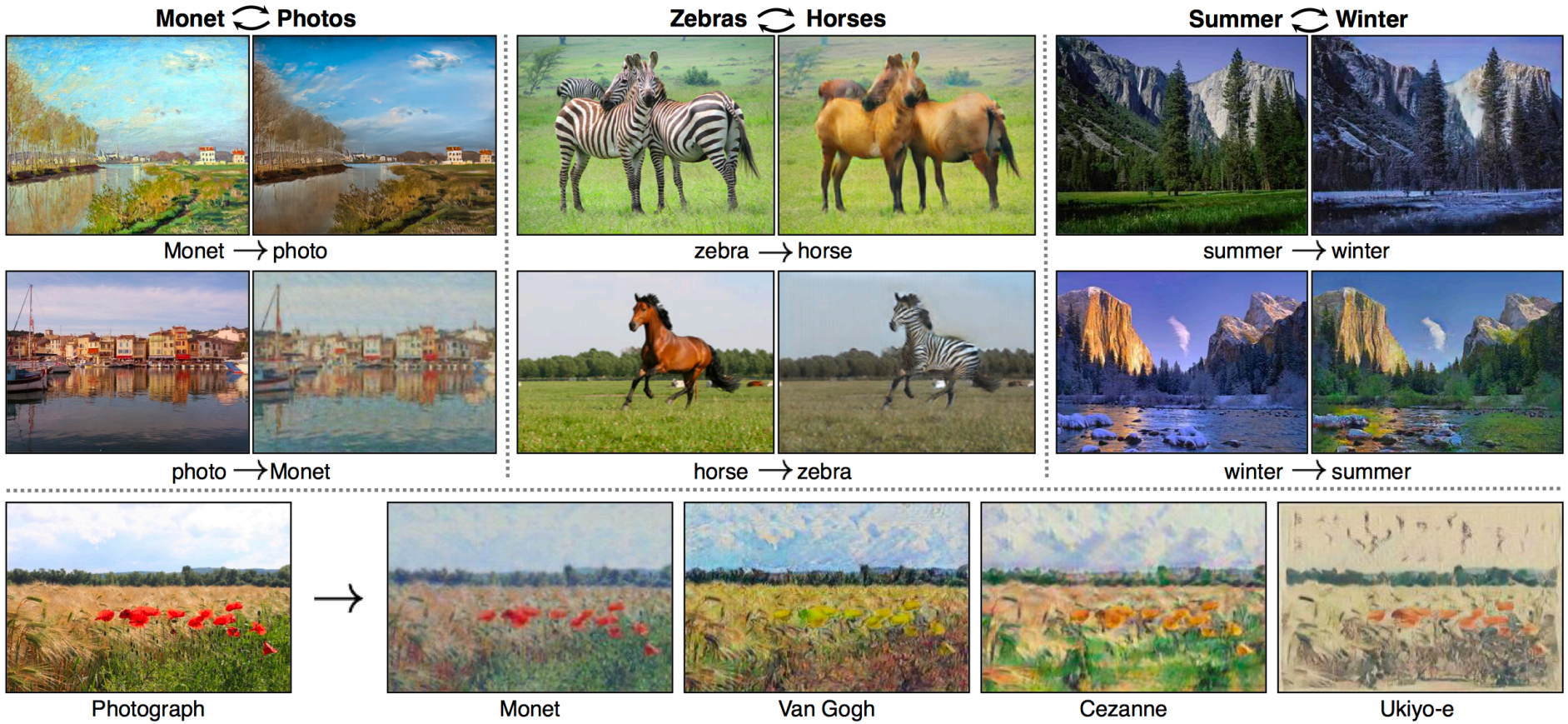

https://arxiv.org/abs/1605.05396

# Puzzle-GAN




https://arxiv.org/abs/1703.10730

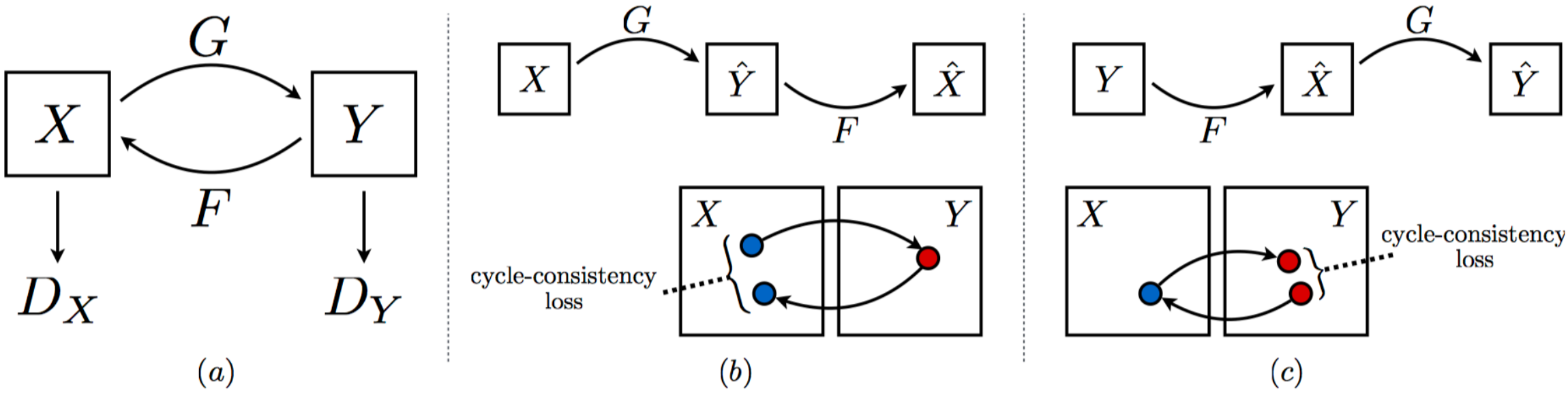
# CycleGAN




https://arxiv.org/abs/1703.10593

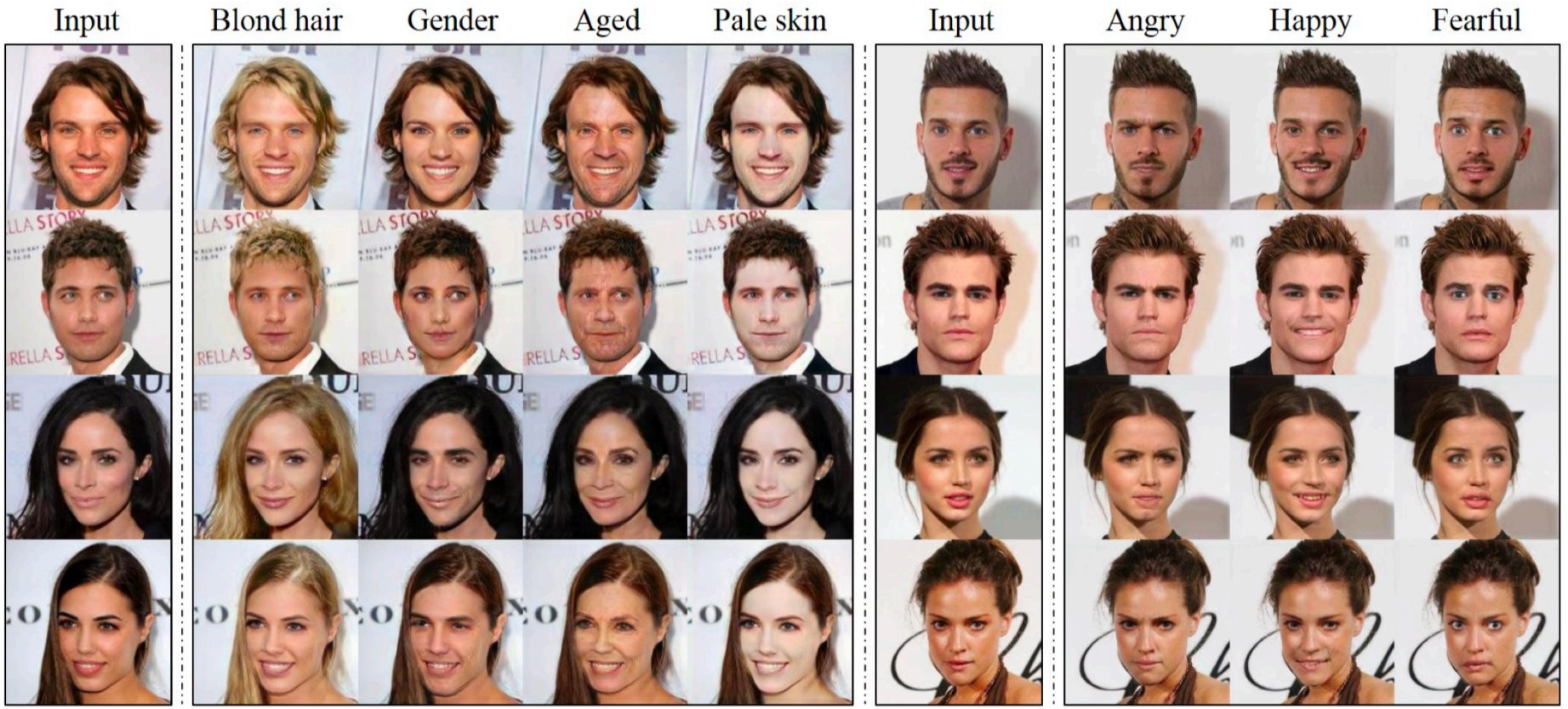
# CycleGAN

- Cycle-consistency Loss




https://arxiv.org/abs/1703.10593

# Star-GAN

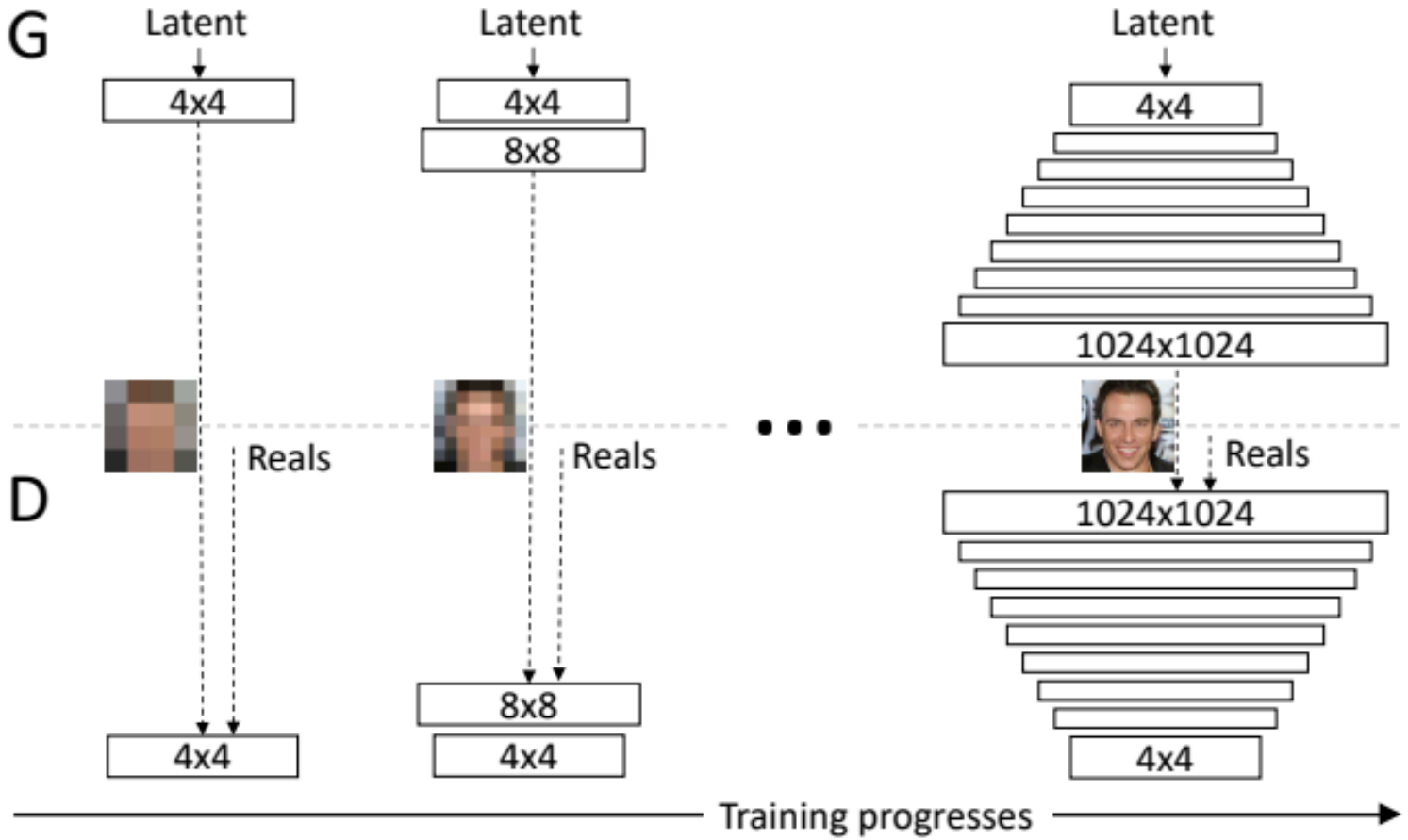



https://arxiv.org/abs/1711.09020

# Progressive-GAN

https://arxiv.org/abs/1710.10196

# Progressive-GAN

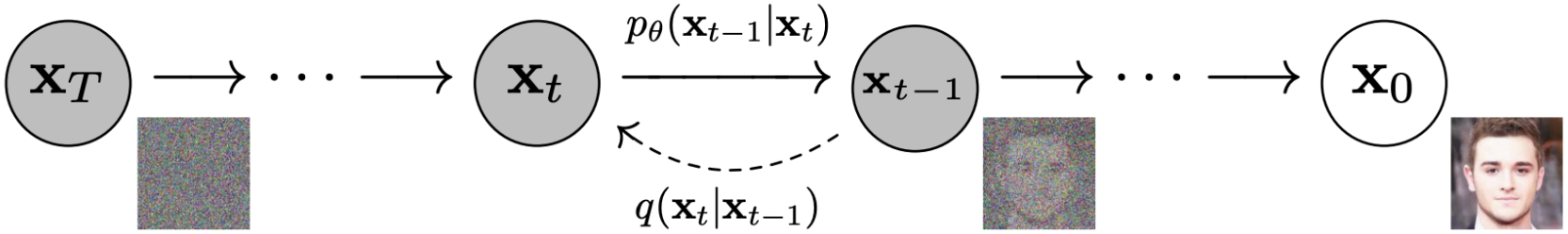



https://arxiv.org/abs/1710.10196

# DDPM

"Denoising Diffusion Probabilistic Models," 2020

# Diffusion Model




- The forward diffusion process is defined as

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t|\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

- Then, the jump-diffusion process can be derived as

$$\begin{aligned} x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}z_{t-1}, \quad \text{where } \alpha_t = 1-\beta_t \text{ and } z_t \sim \mathcal{N}(0, I) \\ &= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}z_{t-2}) + \sqrt{1-\alpha_t}z_{t-1} \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{z}_{t-2} \\ &\vdots \\ &= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{z}_0, \quad \text{where } \bar{\alpha}_t = \prod_{s=1}^{t}\alpha_s \end{aligned}$$

- Hence, $q(x_t|x_0) = \mathcal{N}\left(x_0; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I\right)$.

# Maximum Likelihood Learning

$$
\begin{aligned}
\underbrace{\mathbb{E}_{x_0 \sim q(x_0)}\left[-\log p_\theta(x_0)\right]}_{\text{Maximum Likelihood Learning} \downarrow} &= \mathbb{E}_{x_0 \sim q(x_0)}\left[-\log \int p_\theta(x_0, x_1, \ldots, x_T) dx_1 dx_2 \ldots dx_T\right] \\
&= \mathbb{E}_{x_0 \sim q(x_0)}\left[-\log \int p_\theta(x_{0:T}) \frac{q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)} dx_{1:T}\right] \\
&= \mathbb{E}_{x_0 \sim q(x_0)}\left[-\log \mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right]\right] \\
&\leq \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\log \frac{p(x_T) p_\theta(x_{T-1}|x_T) p_\theta(x_{T-2}|x_{T-1}) \cdots p_\theta(x_0|x_1)}{q(x_1|x_0) q(x_2|x_1) \cdots q(x_T|x_{T-1})}\right] \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[-\log p(x_T) - \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\right] \\
&\vdots \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[-\log \frac{p(x_T)}{q(x_T|x_0)} - \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} - \log p_\theta(x_0|x_1)\right] \\
&= \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\sum_{t=2}^{T} \underbrace{D_{KL}\left(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t)\right)}_{L_{t-1} \downarrow}\right] + C
\end{aligned}
$$

# Reverse Posterior

- Using the jump-forward process, $q(x_t|x_0) = \mathcal{N}\left(x_0; \sqrt{\bar{\alpha}_t}x_0,\ (1-\bar{\alpha}_t)I\right)$, we can derive the conditioned reverse posterior distribution, $q(x_{t-1}|x_t, x_0)$ as follows:

$$
\begin{aligned}
q(x_{t-1}|x_t, x_0) &= \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} \\
&= \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)} \\
&\vdots \\
&= \mathcal{N}\left(x_{t-1};\ \underbrace{\frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{(1-\bar{\alpha}_t)}x_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{\alpha_t}x_t}_{\tilde{\mu}_{t-1}(x_t, x_0)},\ \underbrace{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}I}_{\tilde{\sigma}^2_{t-1}}\right)
\end{aligned}
$$

# Loss Function (1/2)

- Recall that $L_{t-1} = D_{KL}\left(q(x_{t-1}\,|\,x_t, x_0)\|p_\theta(x_{t-1}\,|\,x_t)\right)$.
- Our learning objective is to optimize $p_\theta(x_{t-1}\,|\,x_t)$ that best approximates the reverse posterior:

$$q(x_{t-1}\,|\,x_t, x_0) = \mathcal{N}\left(x_{t-1};\ \underbrace{\frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{(1-\bar{\alpha}_t)}x_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{\alpha_t}x_t}_{\tilde{\mu}_{t-1}(x_t,x_0)},\ \underbrace{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}I}_{\tilde{\sigma}^2_{t-1}}\right)$$

- Assuming that the standard deviation of $p_\theta(x_{t-1}\,|\,x_t)$ is constant and the mean is $\mu_\theta(x_t, t)$, we get the following objective:

$$L_{t-1} = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\|\tilde{\mu}_{t-1}(x_t, x_0) - \mu_\theta(x_t, x_0)\|_2^2\right] + C$$

- Note that $\tilde{\mu}_{t-1}(x_t, x_0)$ can be parametrized by $x_t$ and $\epsilon \sim \mathcal{N}(0,I)$ using $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$:

$$\tilde{\mu}_t(x_t, \epsilon) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

# Loss Function (2/2)

- Plugging $\tilde{\mu}_t(x_t, \epsilon) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$ into $L_{t-1} - C$, we get

$$L_{t-1} - C = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,I)}\left[\frac{1}{2\sigma_t^2}\left\| \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \mu_\theta(x_t, t)\right\|_2^2\right]$$

$$\text{where } x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

- Now, if we parametrize $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$, aka denoising objectives, then, we get

$$L_{t-1} - C = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,I)}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right] \text{ where } x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

# DDPM Training objective

**Algorithm 1** Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

**Algorithm 2** Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

# DDPM Constants (1/2)

- Forward Diffusion Process

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t;\ \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$$

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I\right) \text{ where } \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

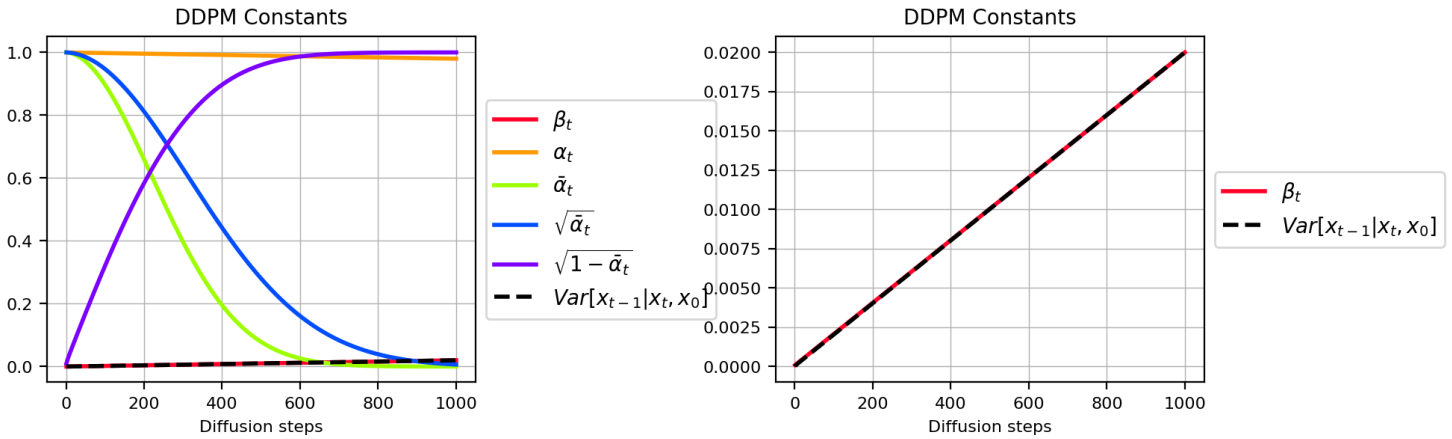
- Reverse Diffusion Posterior

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(x_{t-1}; \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{(1-\bar{\alpha}_t)}x_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{\alpha_t}x_t, \frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\right)$$

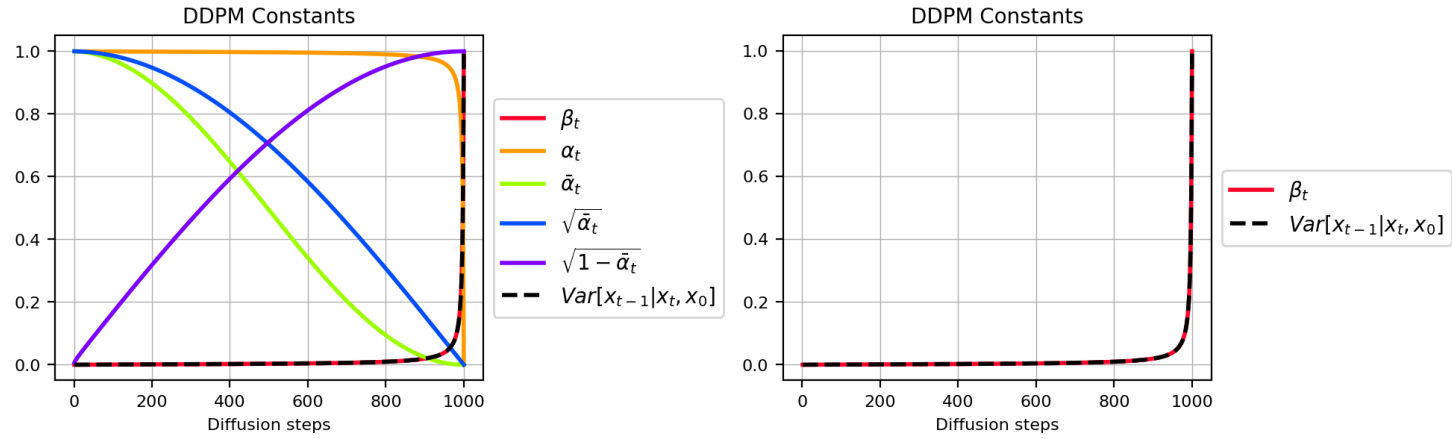
- Practically, scheduling $\{\beta_t\}_{t=1}^{T}$ is important.

# DDPM Constants (2/2)

- Linear scheduling




- Cosine scheduling




서울과학기술대학교
SEOUL NATIONAL UNIV. OF SCIENCE & TECHNOLOGY
1910