Generative Models

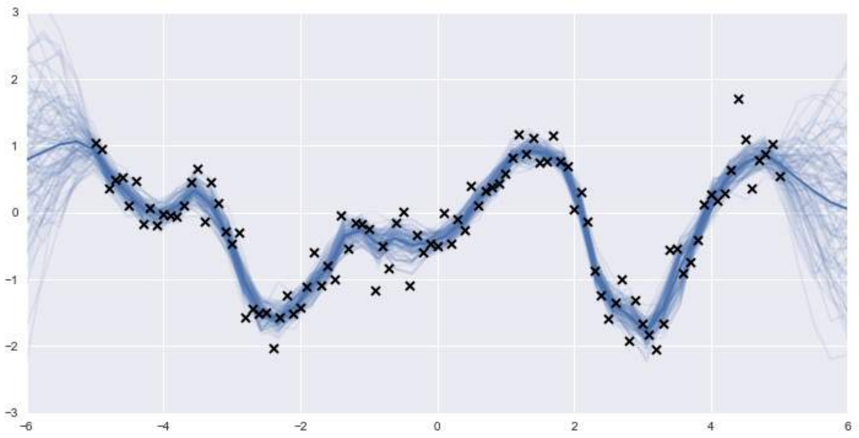
Introduction

Yingzhen Li (yingzhen.li@imperial.ac.uk)

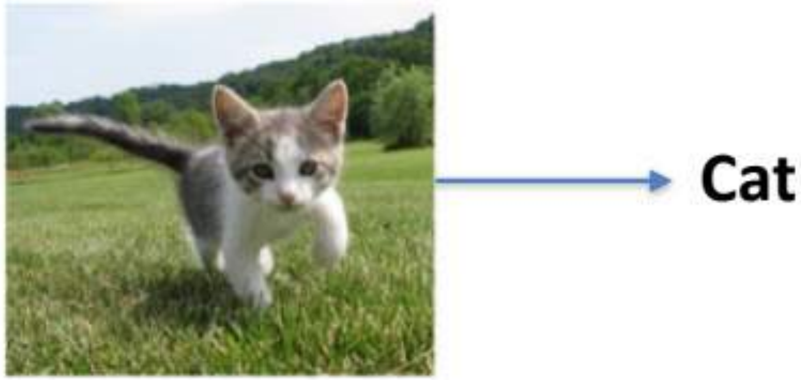
Supervised Learning

Data: $(x_1, y_1), \dots, (x_N, y_N) \sim p_{data}(x, y)$

Goal: learn a function to map $x \rightarrow y$



Regression



Classification

GRASS, CAT, TREE, SKY

Semantic Segmentation

DOG, DOG, CAT

Object detection

Most of the data are unlabelled

Unsupervised Learning

Data: $x_1, \dots, x_N \sim p_{data}(x)$ (no supervision signal)

Goal: inferring a function that describes the hidden structure of unlabelled data

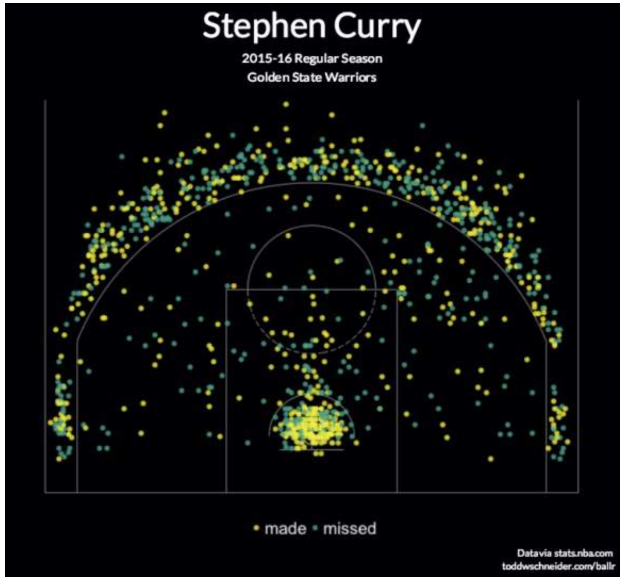
Examples:
- Probability distribution/density estimation
- Dimensionality Reduction
- Clustering

All of them can be achieved by generative modelling!

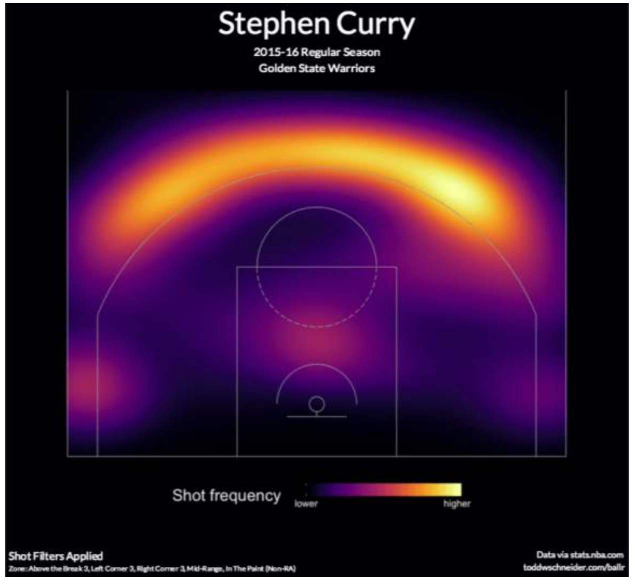
Probability distribution/density estimation

Data: $x_1, \dots, x_N \sim p_{data}(x)$

Goal: learn a distribution $p_\theta(x) \approx p_{data}(x)$ with data $x_1, \dots, x_N$

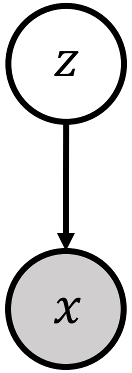


density estimation



https://www.r-bloggers.com/2016/03/ballr-interactive-nba-shot-charts-with-r-and-shiny/

Generative Latent Variable Models



- Design $p_\theta(x)$ as a generative latent variable model (LVM):

$$z \sim p_\theta(z), \qquad x \sim p_\theta(x|z)$$

$$\Rightarrow \; p_\theta(x) = \int p_\theta(x|z) p_\theta(z) dz$$

z: latent variable (unobserved) $\qquad$ x: observation variable

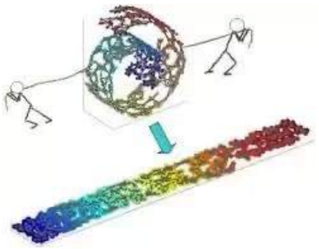
z: digit label, writing style, …
x: hand-written digit

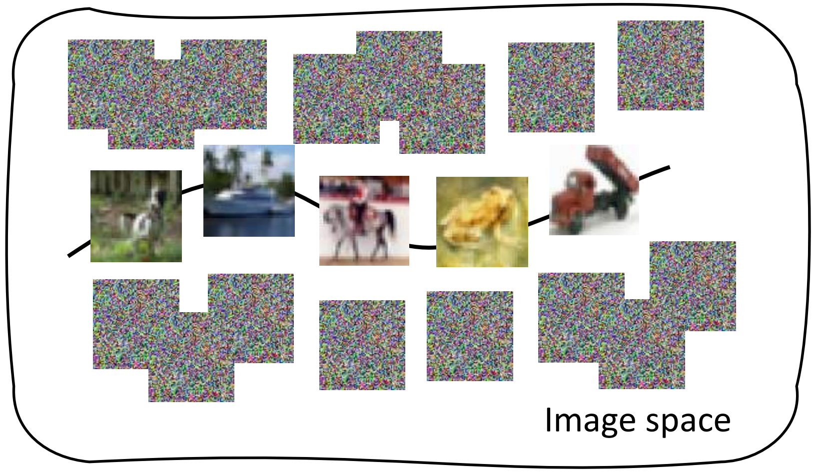
z: scene, viewing angle, lighting condition, …
x: photo image

z: semantics, sentiment, …
x: generated text

Dimensionality reduction

- High-dimensional raw data are often sparse, perhaps lying on a low-dimensional manifold:





natural images vs all RGB images

2			4	5	
5		4			1
		5		2	
	1		5		4
		4		2	
4	5		1		

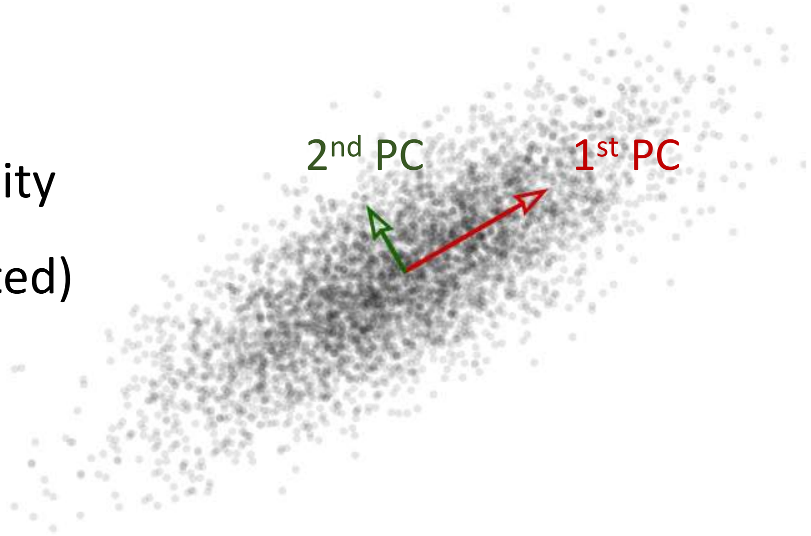
User ratings on items

Dimensionality reduction

- Principal Component Analysis (PCA):

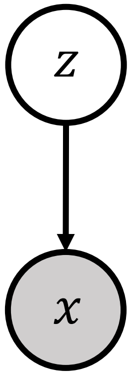
  Find principal components – orthogonal directions that capture most of the variance in the data

  - 1^{st} principal component – direction of greatest variability
  - 2^{nd} principal component – next orthogonal (uncorrelated) direction of greatest variability
  - And so on ...



Dimensionality reduction is achieved by projecting the data on the top $K < d$ principal components ($x \in R^d$)

Dimensionality reduction

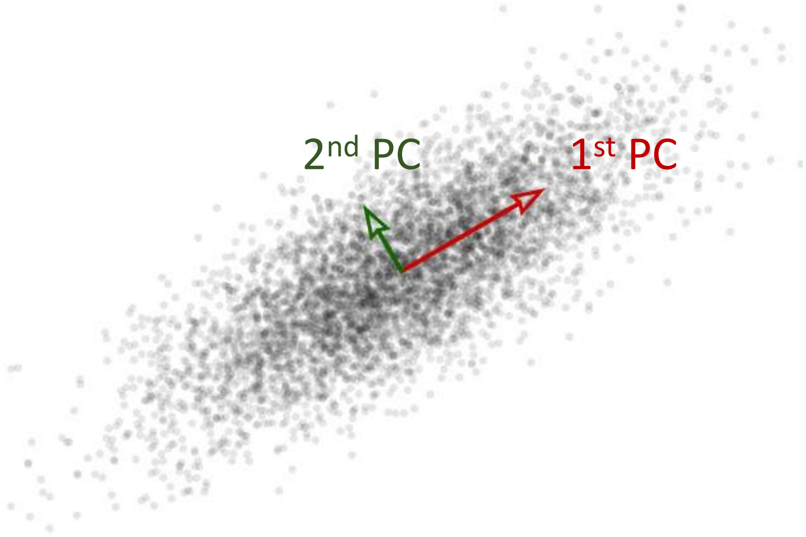


- Probabilistic Principal Component Analysis (Prob PCA):

$$p(z) = N(z; 0, I), \quad z \in R^K, \quad K < d$$

$$p_\theta(x|z) = N(x; Wz, \sigma^2 I), \quad x \in R^d$$

  - Parameters to optimize: $\theta = W \in R^{d \times K}$, with the row vectors in W orthogonal to each other
  - Trained using Maximum Likelihood
  - Optimal W contains the top K principal components – the top K eigenvectors of the data covariance matrix

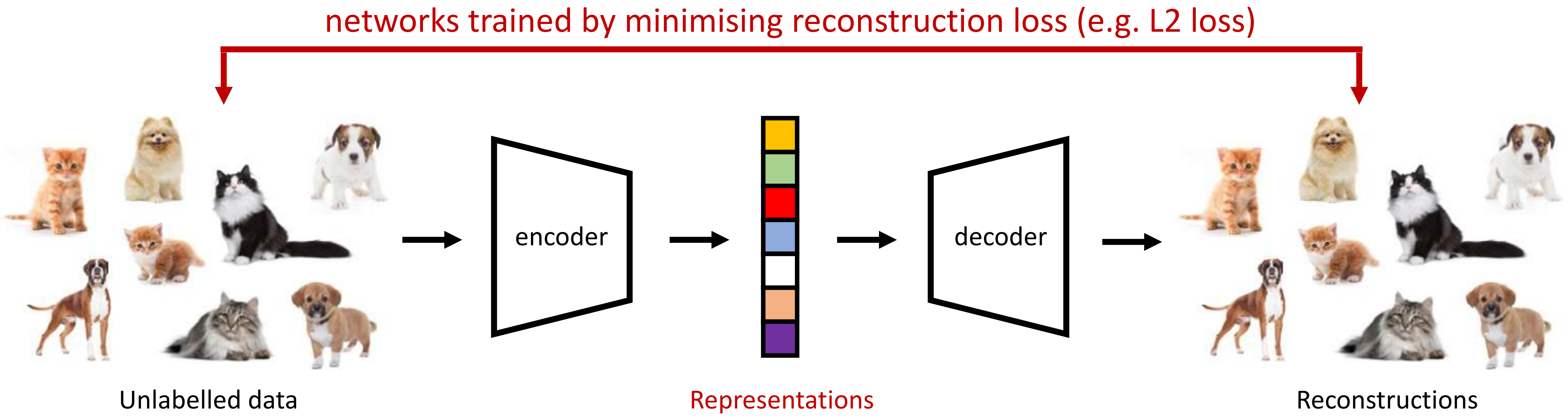


Tipping and Bishop. Probabilistic principal component analysis. Journal of the Royal Statistical Society: Series B, 1999

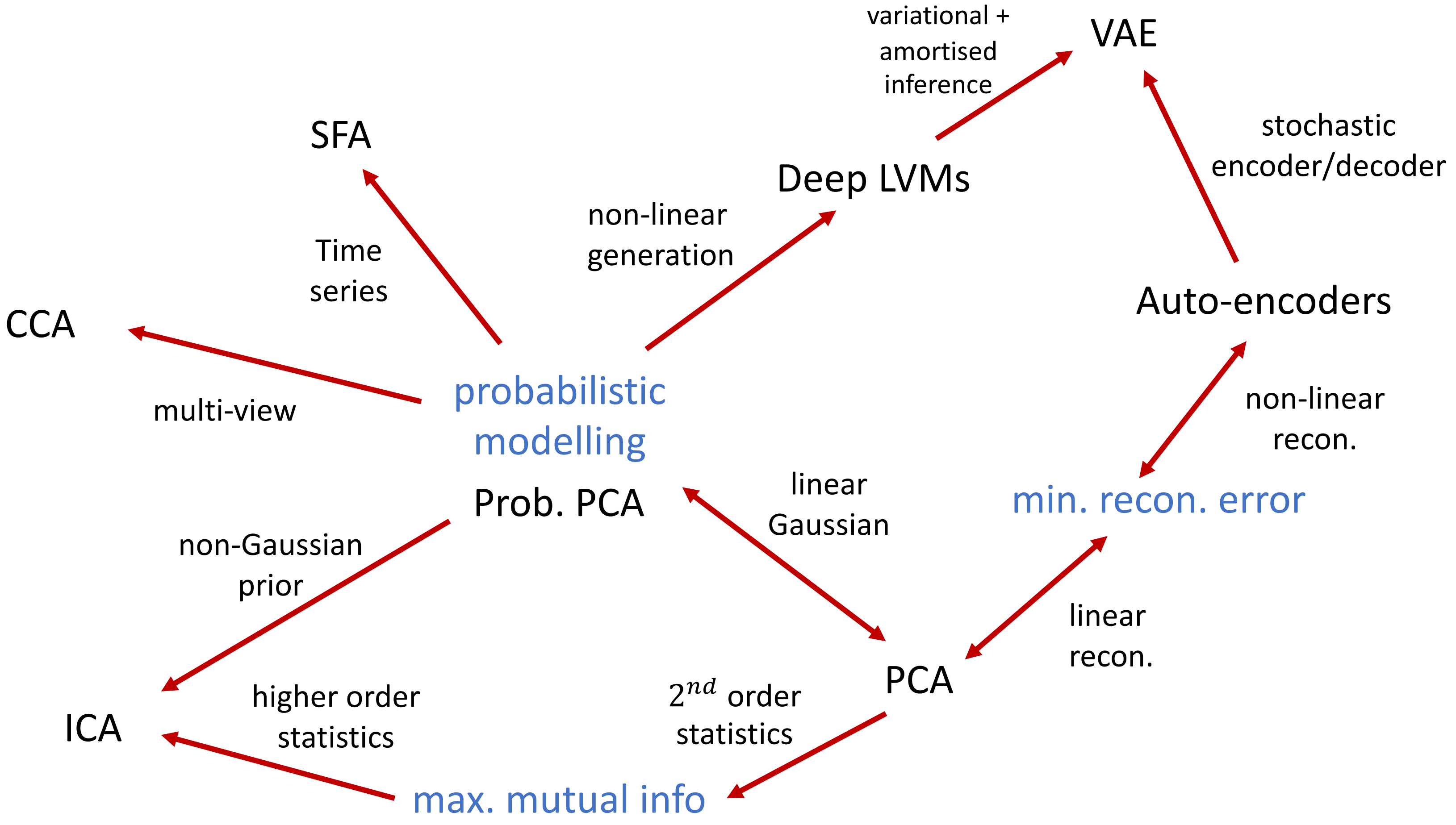
Dimensionality reduction

- Auto-encoders for dimensionality reduction:
  - Encoder network to extract data representations (often with lower dimensionality)
  - Decoder network to reconstruct data given the representations

A probabilistic version: Variational auto-encoder (VAE)



Dimensionality reduction



Clustering

- Clustering: discover “group structure”
  - grouping datapoints into several clusters
  - Datapoints in the same cluster are similar
  - Datapoints in different clusters are “dissimilar”

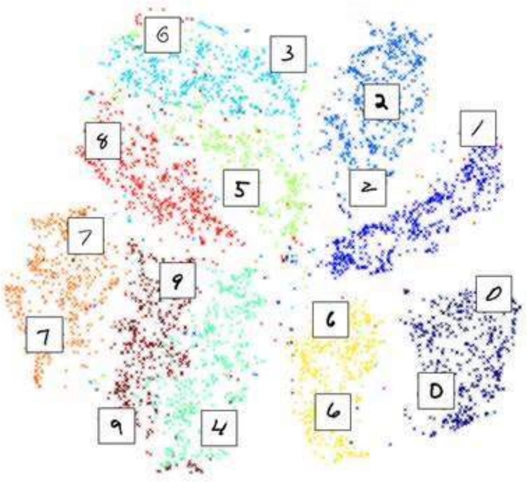
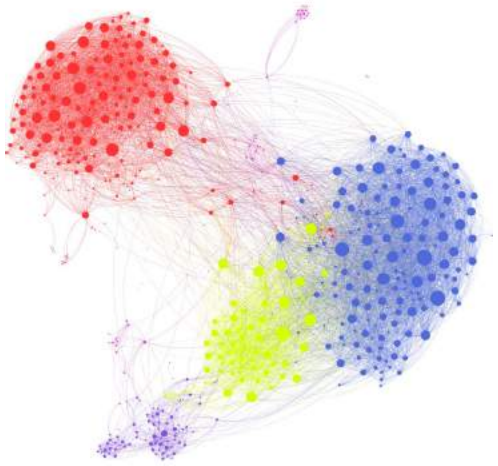
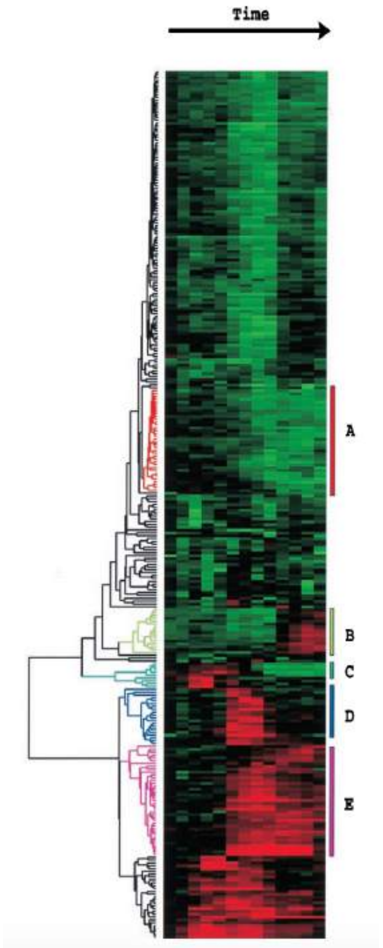


Image clustering



social network analysis



gene data analysis

Clustering

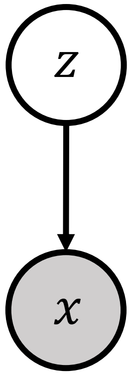
- Gaussian mixture model (GMM):

$$p_\theta(z) = Categorical(\pi),$$

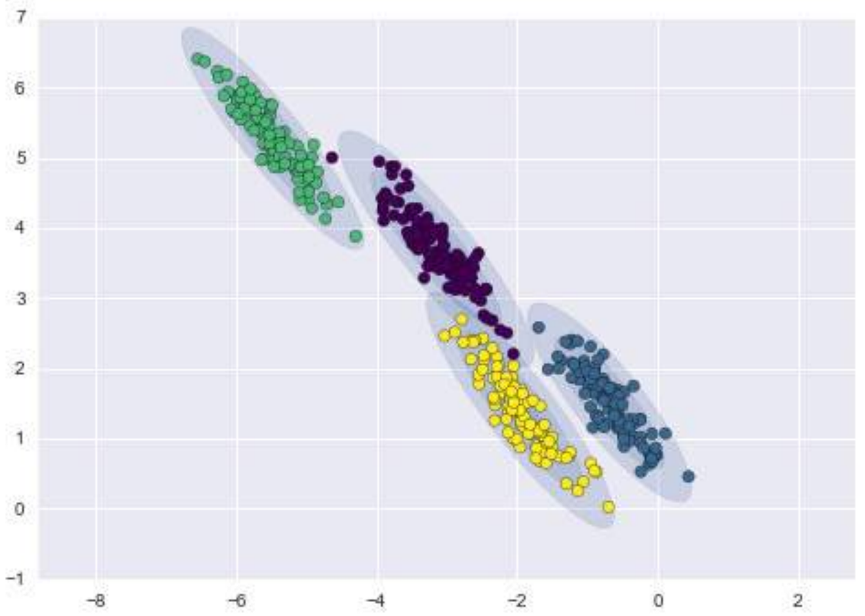
$$\pi = (\pi_1, \dots, \pi_K), \pi_i = p_\theta(z = i), \sum_{i=1}^{K} \pi_i = 1$$

$$p_\theta(x|z) = N(x; \mu_z, \Sigma_z)$$

  - $z \in \{1, \dots, K\}$: index of the Gaussian component
  - μ_z: mean of the i^{th} Gaussian component if $z = i$
  - Σ_z: Covariance matrix of the i^{th} Gaussian component if $z = i$



$\Rightarrow$ Clustering can be done by fitting a GMM model to the data

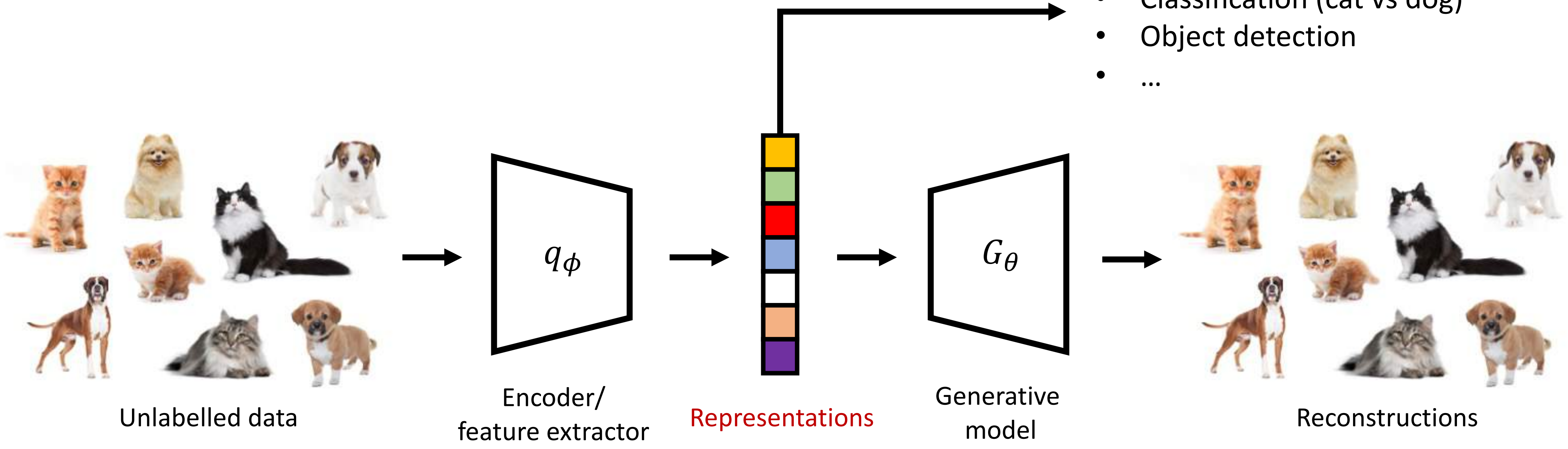


Representation learning

- Both dimensionality reduction and clustering can be viewed as representation learning
  - Hope: useful for downstream tasks

Representations used in downstream tasks:

- Classification (cat vs dog)
- Object detection
- …



Taxonomy of Generative Models

Generative models

Parametric models

Parametric LVMs

Auto-regressive models

Normalising flows

Energy-based models

…

Gaussian mixture model

VAEs (explicit distribution)

GANs (implicit distribution)

…

next videos

Non-parametric models

Kernel density estimation

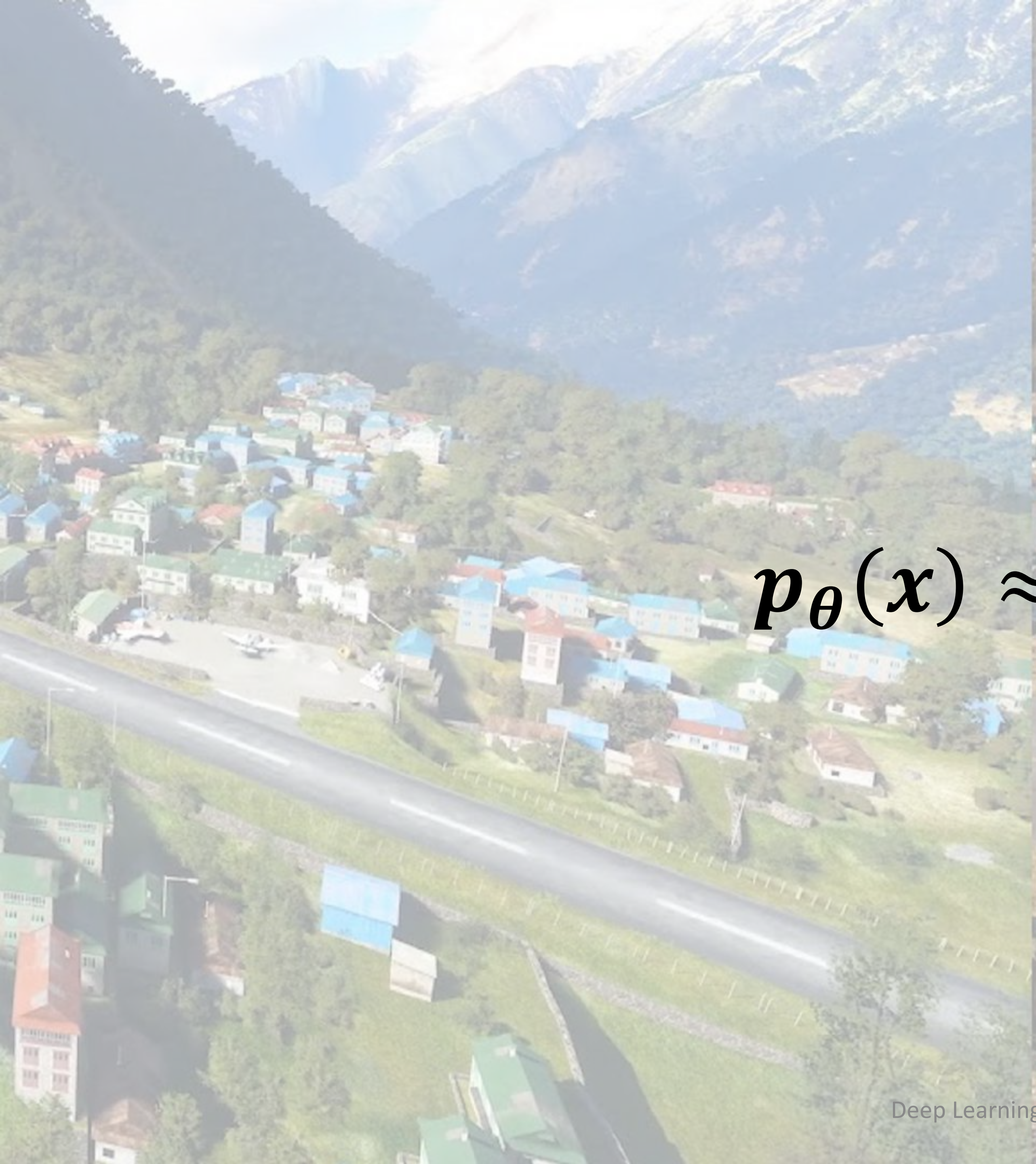
(hierarchical) Dirichlet process

…

Generative Models

VAE basics

Yingzhen Li (yingzhen.li@imperial.ac.uk)



$$p_\theta(x) \approx p_{data}(x)$$

Divergence minimisation

- Fitting the model to the data by divergence minimisation:

$$\theta^* = argmin\, D[p_{data}(x) \,||\, p_\theta(x)]$$

$D[p \,||\, q]$ is a valid divergence if and only if:

- $D[p \,||\, q] = 0 \Rightarrow p = q$
- $p = q \Rightarrow D[p \,||\, q]$
  (or $p(x) \neq$ q$(x) \Rightarrow D[p \,||\, q] > 0$)

Divergence minimisation

- An example of a valid divergence: Kullback-Leibler (KL) divergence:

$$KL[p(x) \,||\, q(x)] \;=\; E_{p(x)}\,[\log \frac{p(x)}{q(x)}]$$

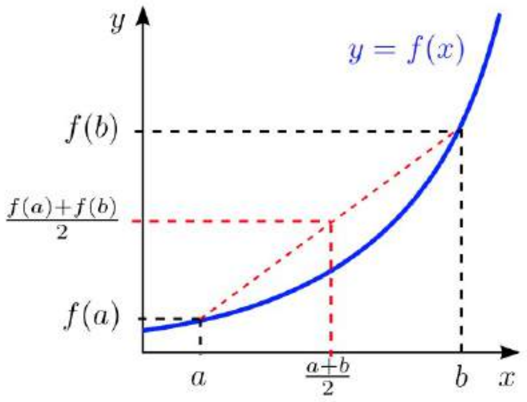
  - $p(x) = \; q(x) \Rightarrow \; KL[p||q] \;=\; 0$
  - To show $p(x) \neq \; q(x) \Rightarrow \; KL[p||q] \;>\; 0$:

$$\begin{aligned} -KL[p||q] &= -E_{p(x)}[\log \frac{p(x)}{q(x)}] \\ &= E_{p(x)}[\log \frac{q(x)}{p(x)}] \\ &\leq \log E_{p(x)}[\, q(x)/p(x) \,] \\ &= \log 1 \;=\; 0 \end{aligned}$$

(Jensen's inequality)
(equality holds iff. $p(x) = q(x)$)

Jensen's inequality:
Let f, g be two functions and f is convex, then

$$E_{p(x)}[f(g(x))] \geq \; f(E_{p(x)}[g(x)])$$



Maximum Likelihood Estimation

- Fitting the model by minimising KL:

$$\theta^* = \operatorname{argmin} KL[p_{data}(x) \,||\, p_\theta(x)]$$

$$KL[p_{data}(x) \,||\, p_\theta(x)] = E_{p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_\theta(x)}\right]$$

Constant terms w.r.t. θ

$$= -E_{p_{data}(x)}[\log p_\theta(x)] + E_{p_{data}(x)}[\log p_{data}(x)]$$

- Equivalent objective to fit the model: maximum likelihood estimation (MLE)

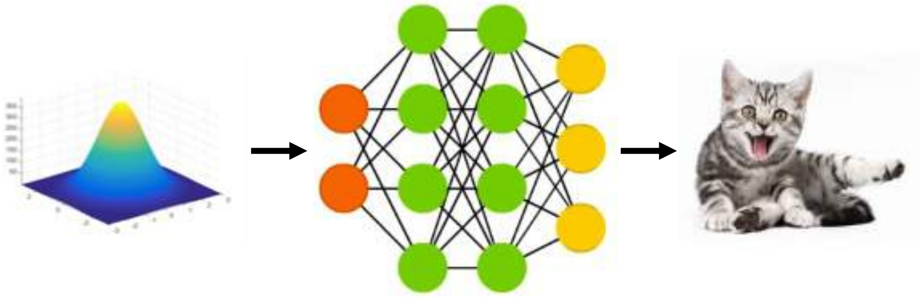
$$\theta^* = argmax\, E_{p_{data}(x)}[\log p_\theta(x)]$$

In practice: $\theta^* = argmax\, \frac{1}{N}\sum_{n=1}^{N} \log p_\theta(x),\, x_1, \ldots, x_N \sim p_{data}(x)$

Maximum Likelihood Estimation

- MLE for training latent variable models (LVM):

$$p(z) = N(z; 0, I)$$
$$p_\theta(x|z) = N(x; G_\theta(z), \sigma^2 I)$$



$$\theta^* = argmax\ E_{p_{data}(x)}[\log p_\theta(x)]$$

integrate over
all possible z

$$p_\theta(x) = \int \underline{p_\theta(x|z)} p(z) dz$$

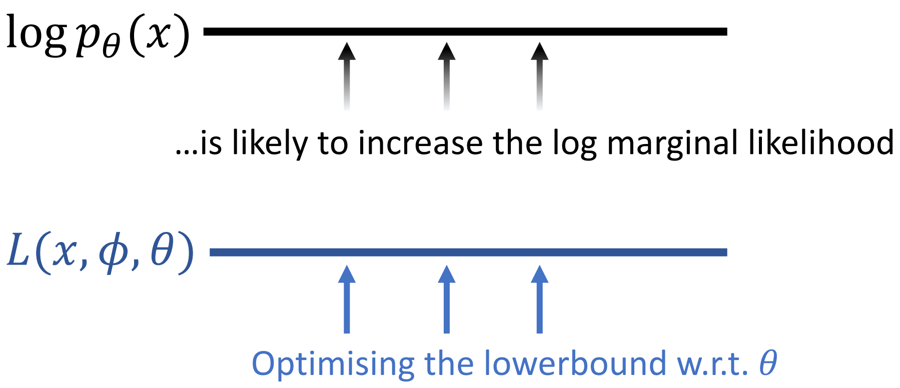
requires passing z
through a neural net

The marginal distribution $p_\theta(x)$ is intractable
$\Rightarrow$ MLE objective is intractable

Variational Auto-Encoders

- MLE for latent variable model training is intractable
- Optimising a variational lower-bound instead:

$$
\begin{aligned}
\log p_\theta(x) &= \log \int p_\theta(x|z)p(z)dz \\
&= \log \int q_\phi(z|x)\frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}dz \\
&\geq \int q_\phi(z|x)\log\frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}dz \qquad \text{(Jensen's inequality)} \\
&= E_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)\,||\,p(z)] \\
&:= L(x,\phi,\theta)
\end{aligned}
$$

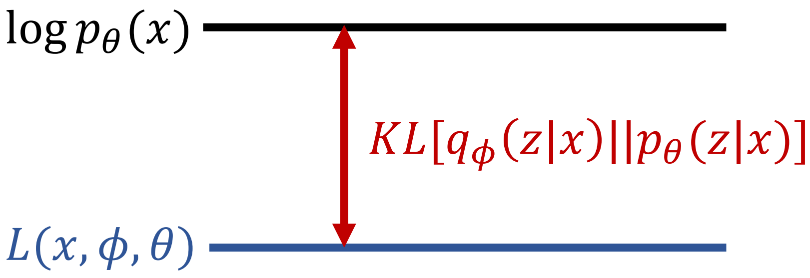


Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Variational Auto-Encoders

- An alternative way to derive the variational lower-bound:

$$
\begin{aligned}
&\log p_\theta(x) - KL[q_\phi(z|x) \,||\, p_\theta(z|x)] \\
&= \log p_\theta(x) \; - E_{q_\phi(z|x)}[\log q_\phi(z|x) \; - \log p_\theta(z|x)] \qquad \text{(Bayes' rule)} \\
&= \log p_\theta(x) \; - E_{q_\phi(z|x)}[\log q_\phi(z|x) \; - \log \frac{p_\theta(x|z)p(z)}{p_\theta(x)}] \\
&= \log p_\theta(x) + E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} - \log p_\theta(x)\right] \\
&= E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}\right] \\
&:= L(x, \phi, \theta)
\end{aligned}
$$



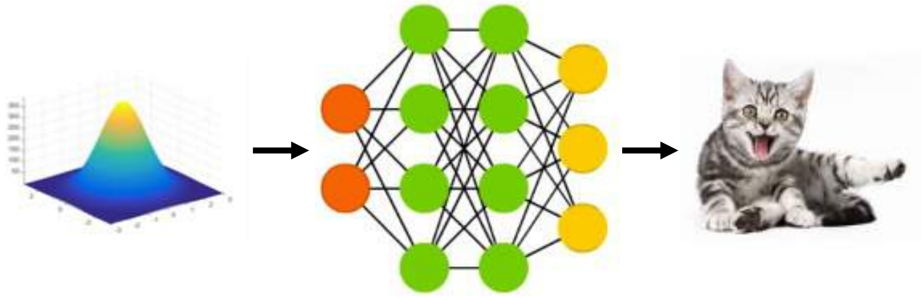
Optimising $L(x, \phi, \theta)$ w.r.t. ϕ
$\Rightarrow$ minimising $KL[q_\phi(z|x) \,||\, p_\theta(z|x)]$
$\Rightarrow$ fit $q_\phi(z|x)$ to the posterior $p_\theta(z|x)$

Variational Auto-Encoders

- The VAE objective:

$$\theta^*, \phi^* = argmax\ L(\phi, \theta)$$

$$L(\phi, \theta) := E_{p_{data}(x)}[E_{q_\phi(z|x)}[\underbrace{\log p_\theta(x|z)}_{= -\frac{1}{2\sigma^2}\|x - G_\theta(z)\|_2^2 + C}] - KL[q_\phi(z|x)\ ||\ p(z)]]$$



$$p(z) = N(z; 0, I)$$
$$p_\theta(x|z) = N(x; G_\theta(z), \sigma^2 I)$$

Ingredients of training VAEs:

- The generative model (decoder)
- The $q_\phi(z|x)$ distribution (encoder)
- The optimisation procedure

Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Designing the q distribution

- Common choice: factorized Gaussian distribution:

$$q_\phi(z|x) = N(z; \mu_\phi(x), diag(\sigma_\phi^2(x)))$$

  - $\mu_\phi(x)$ and $\sigma_\phi(x)$ are parameterized by neural networks parameterized by ϕ, for example:

$$\mu_\phi(x) = NN_{\phi_1}(x), \quad \underline{\log \sigma_\phi(x) = NN_{\phi_2}(x)}$$

to ensure the variance is non-negative

  - Analytic form for the KL regularizer: with $p(z) = N(z; 0, I)$ and $z \in R^d$

$$KL[q_\phi(z|x) \,||\, p(z)] = \frac{1}{2}(||\mu_\phi(x)||_2^2 + ||\sigma_\phi(x)||_2^2 - d - 2\langle \log \sigma_\phi(x), 1\rangle)$$

Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Reparameterisation trick

- The VAE objective:

$$\theta^*, \phi^* = argmax\ L(\phi, \theta)$$

analytic between two Gaussians

$$L(\phi, \theta) := E_{p_{data}(x)}[\underline{E_{q_\phi(z|x)}[\log p_\theta(x|z)]} - KL[q_\phi(z|x)\ ||\ p(z)]]$$

intractable expectation

- Monte Carlo (MC) estimation:

$$E_{q_\phi(z|x)}[\log p_\theta(x|z)] \approx \log p_\theta(x \mid z), \quad z \sim q_\phi(z|x)$$

differentiate to obtain (MC) gradient w.r.t. θ

how about the gradient w.r.t. ϕ?

Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Reparameterisation trick

- Monte Carlo (MC) estimation:

$$E_{q_\phi(z|x)}[\log p_\theta(x|z)] \approx \log p_\theta(x \mid z), \quad z \sim q_\phi(z|x)$$

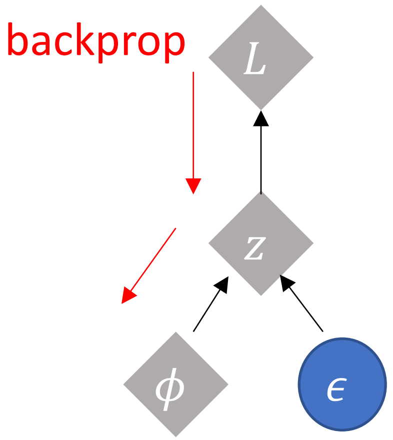
  - With Gaussian encoder:

$$z \sim q_\phi(z|x) \quad \Leftrightarrow \quad z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim N(\epsilon; 0, I)$$

  - Writing $z = T_\phi(x, \epsilon) \coloneqq \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$:

$$E_{q_\phi(z|x)}[\log p_\theta(x|z)] \approx \log p_\theta(x \mid T_\phi(x, \epsilon)), \quad \epsilon \sim N(\epsilon; 0, I)$$

differentiate to obtain (MC) gradient w.r.t. ϕ



Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Variational Auto-Encoders

- Combining all the ingredients together:

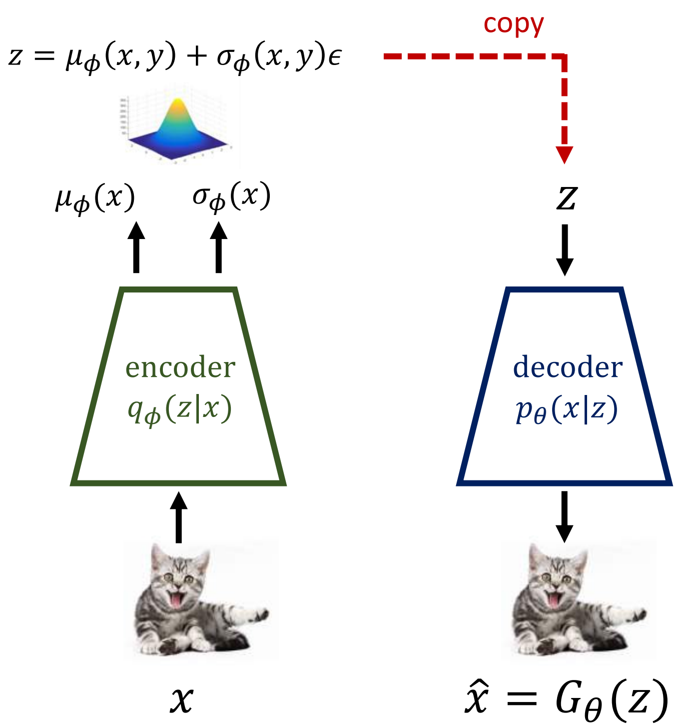
$$\theta^*, \phi^* = argmax\ L(\phi, \theta)$$

Reconstruction loss

$$L(\phi,\theta) := E_{p_{data}(x)}\{ -\boxed{E_{N(\epsilon;0,I)}\left[\frac{1}{2\sigma^2} \parallel \underbrace{G_\theta\left(T_\phi(x,\epsilon)\right)}_{\text{stochastic auto-encoder}} - x \parallel_2^2\right]}$$

$$\underbrace{-KL[q_\phi(z|x) \parallel p(z)]\}}_{\text{KL regularizer}}$$

KL regularizer
to make q closer to the prior and prevent $\sigma_\phi(x) \to 0$



$$p_\theta(x|z) = N(x; G_\theta(z), \sigma^2 I)$$

Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

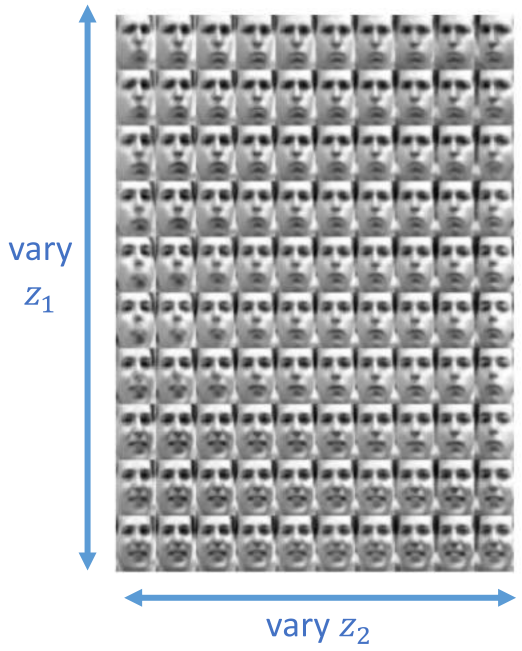
Generating data from the VAE

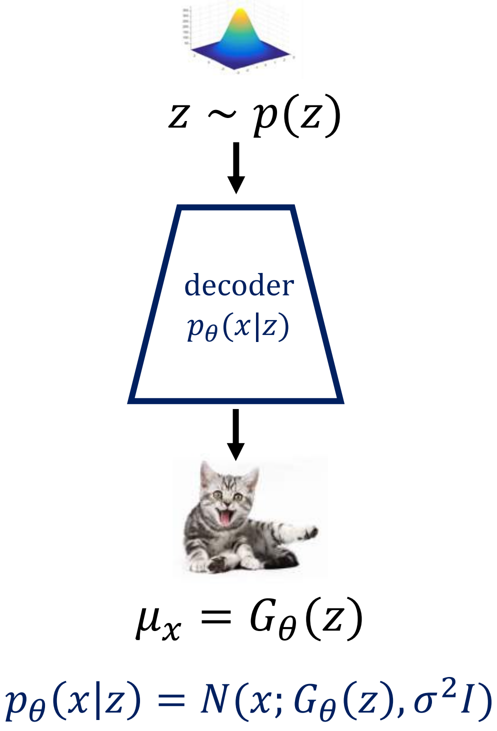
- Once trained, sample new images from the model:

$$z \sim p(z), \quad x \sim p(x|z)$$

Different z dimensions encode different info:

- z_1: facial expressions
- z_2: head pose





Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Variational Auto-Encoders

Practical implementation for solving (pseudo code):

$$\max_{\theta,\phi} E_{p_{data}(x)}[E_{q_\phi(z|x)}[\underline{\log p_\theta(x|z)}] - KL[q_\phi(z|x) \,||\, p(z)]]$$

$$= -\frac{1}{2\sigma^2}\|x - G_\theta(z)\|_2^2 + C$$

- Initialise θ, ϕ, learning rates γ, choose total iteration T for SGD
- For $t = 1, \dots, T$
  - $x_1, \dots, x_M \sim p_{data}(x)$

  # encoder: performing (approximate) posterior inference
  - Compute $\mu_\phi(x_m), \sigma_\phi(x_m)$ for $m = 1, \dots, M$
  - $z_m = \mu_\phi(x_m) + \sigma_\phi(x_m) \odot \epsilon_m,\ \epsilon_m \sim N(0, I)$ # reparam. trick

  # Decoder: reconstructing data
  - $\hat{x}_m = G_\theta(z_m)$ for $m = 1, \dots, M$

  # update neural network parameters
  - $L = \frac{1}{M}\sum_{m=1}^{M}[-\frac{1}{2\sigma^2}\|x_m - \hat{x}_m\|_2^2 - \underline{KL[q_\phi(z_m|x_m) \,||\, p(z_m)]}]$
  - $(\theta, \phi) \leftarrow (\theta, \phi) + \gamma \nabla_{(\theta,\phi)} L$

can use the analytic KL form
or estimated by Monte Carlo

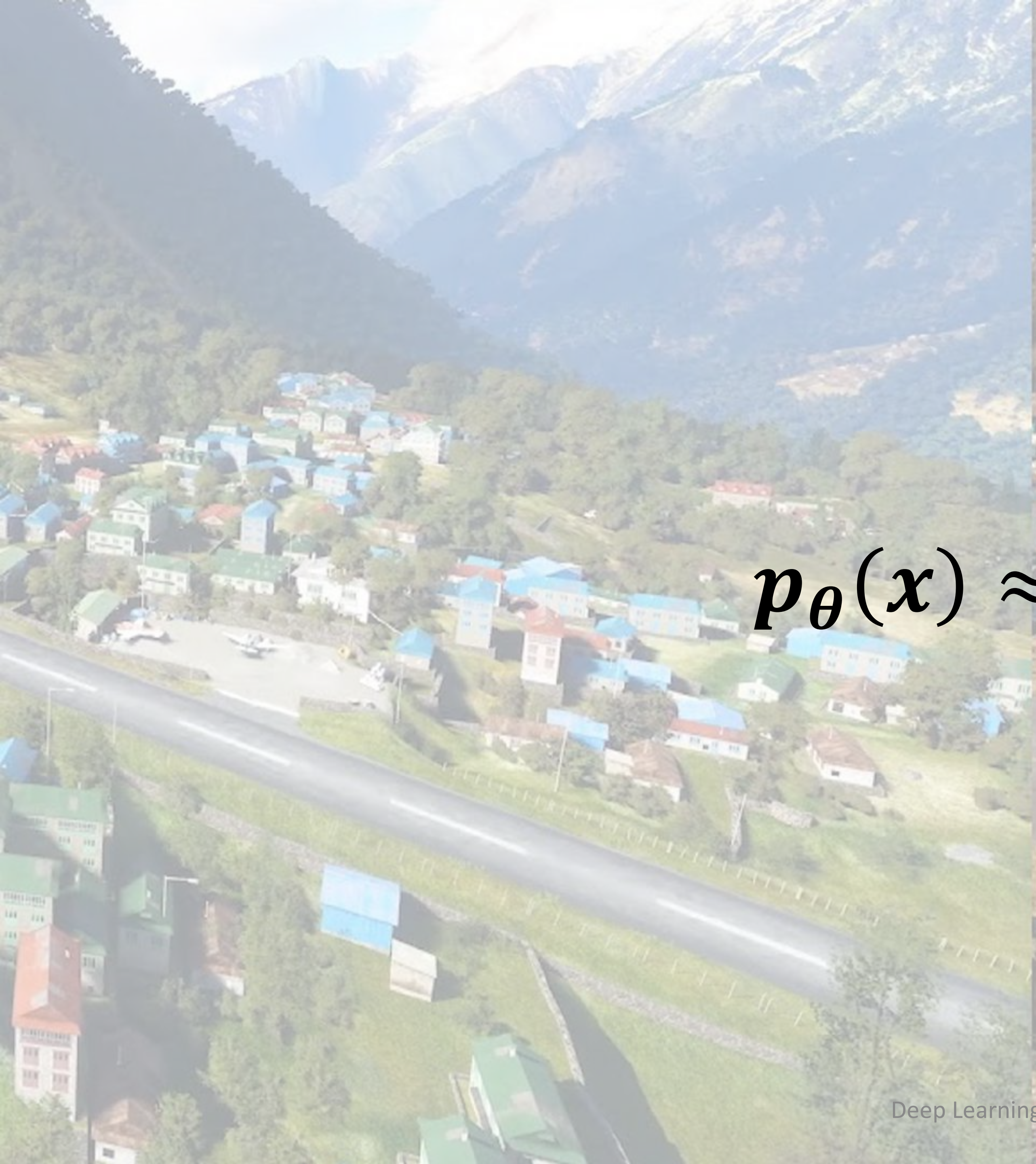
A practical trick: KL annealing

Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014
Rezende et al. Stochastic backpropagation and approximate inference in deep generative model. ICML 2014

Generative Models

GAN basics

Yingzhen Li (yingzhen.li@imperial.ac.uk)



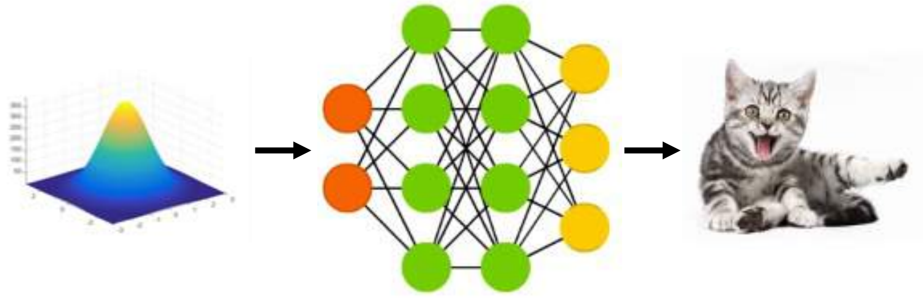
$$p_\theta(x) \approx p_{data}(x)$$

Divergence minimisation

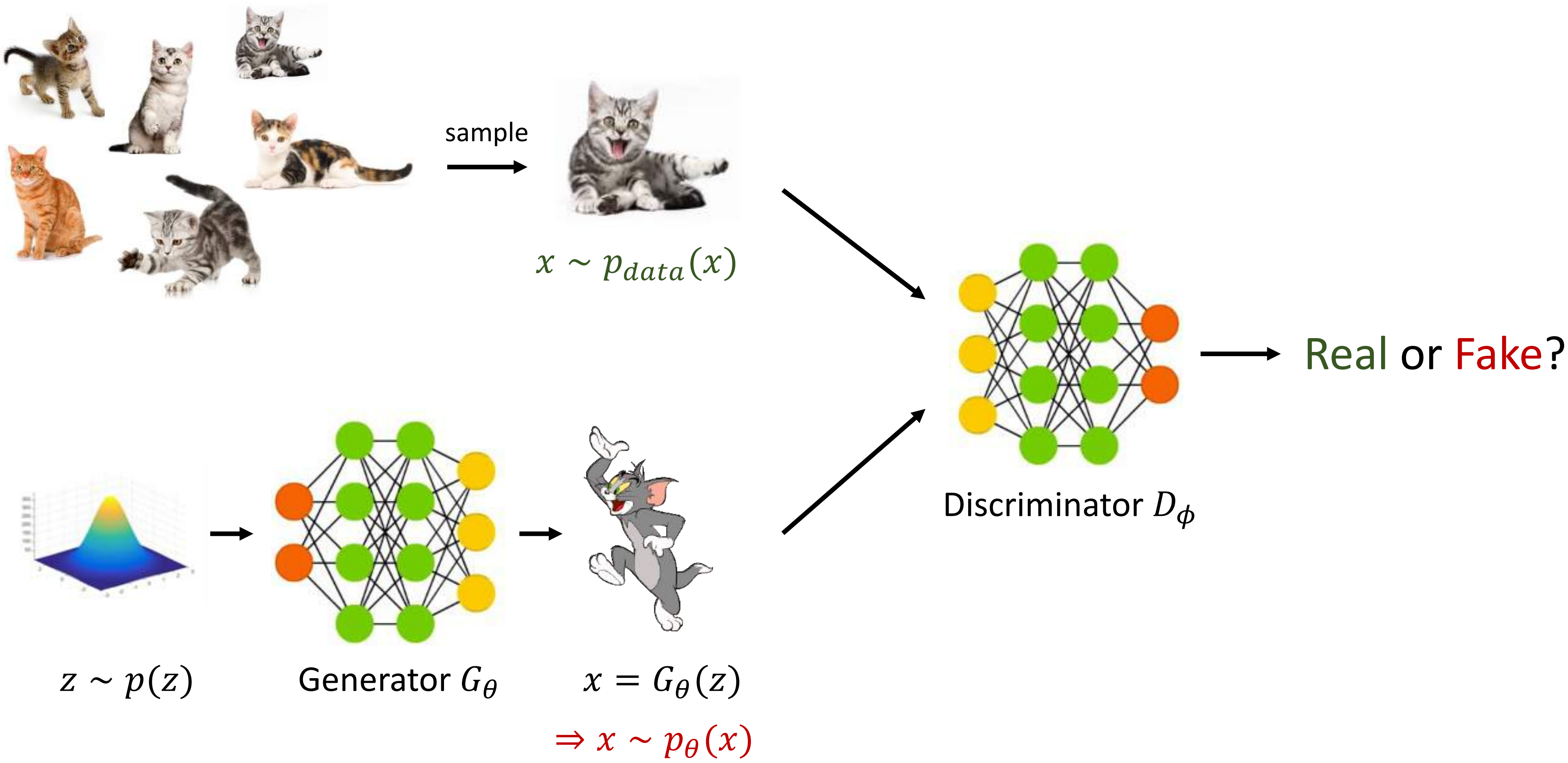
- Fitting the model to the data by divergence minimisation:

$$\theta^* = argmin\ D[p_{data}(x)\ ||\ p_\theta(x)]$$

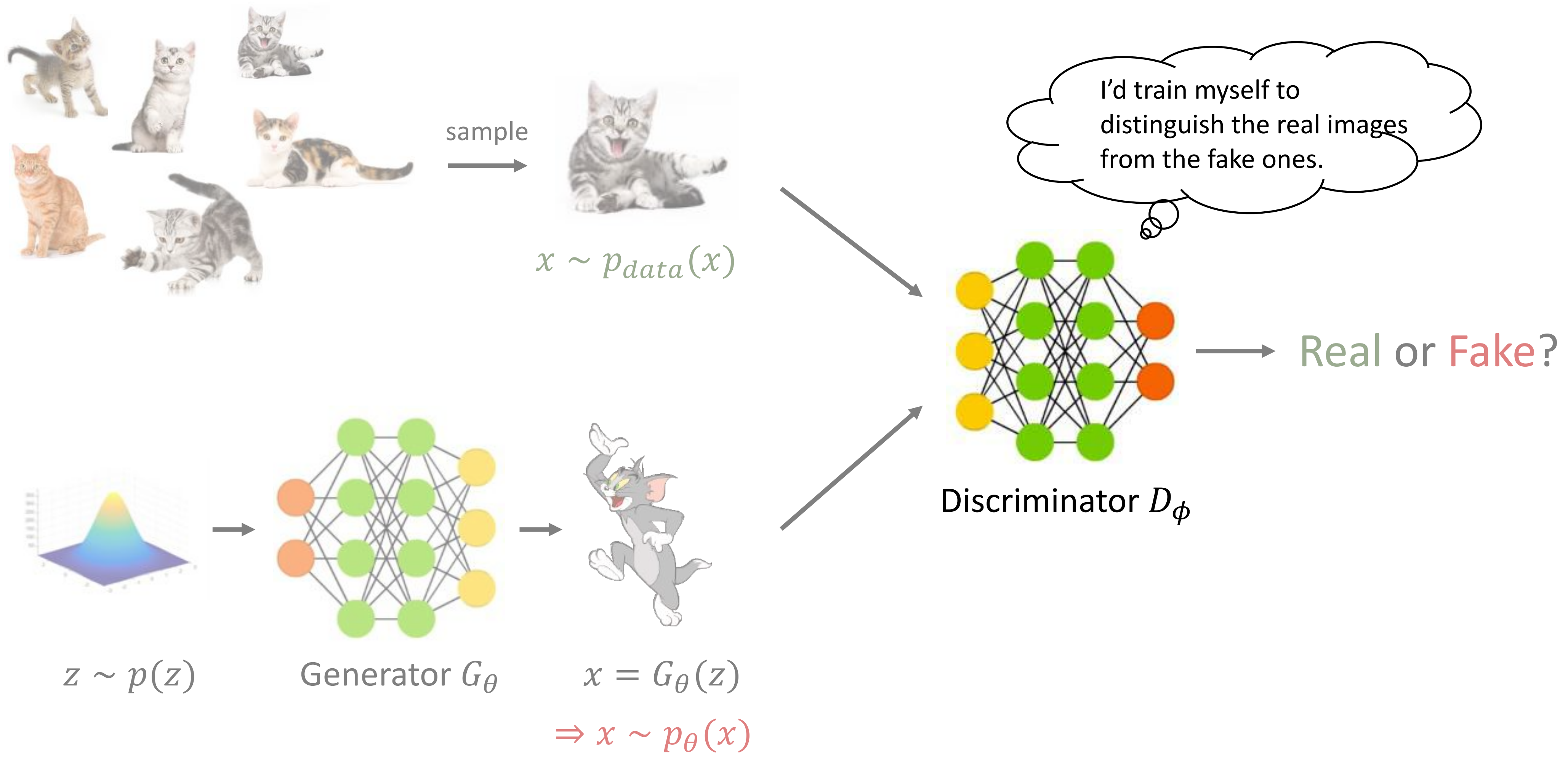
- VAE: variational maximum likelihood training
  - Objective: MLE is equivalent to minimizing $KL[p_{data}(x)\ ||\ p_\theta(x)]$
  - For LVMs, $\log p_\theta(x) = \log \int p_\theta(x|z)p(z)dz$ is intractable
    - $\Rightarrow$ variational lower-bound $L(x, \phi, \theta) \leq \log p_\theta(x)$
    - maximise $E_{p_{data}(x)}[L(x, \phi, \theta)]$ instead



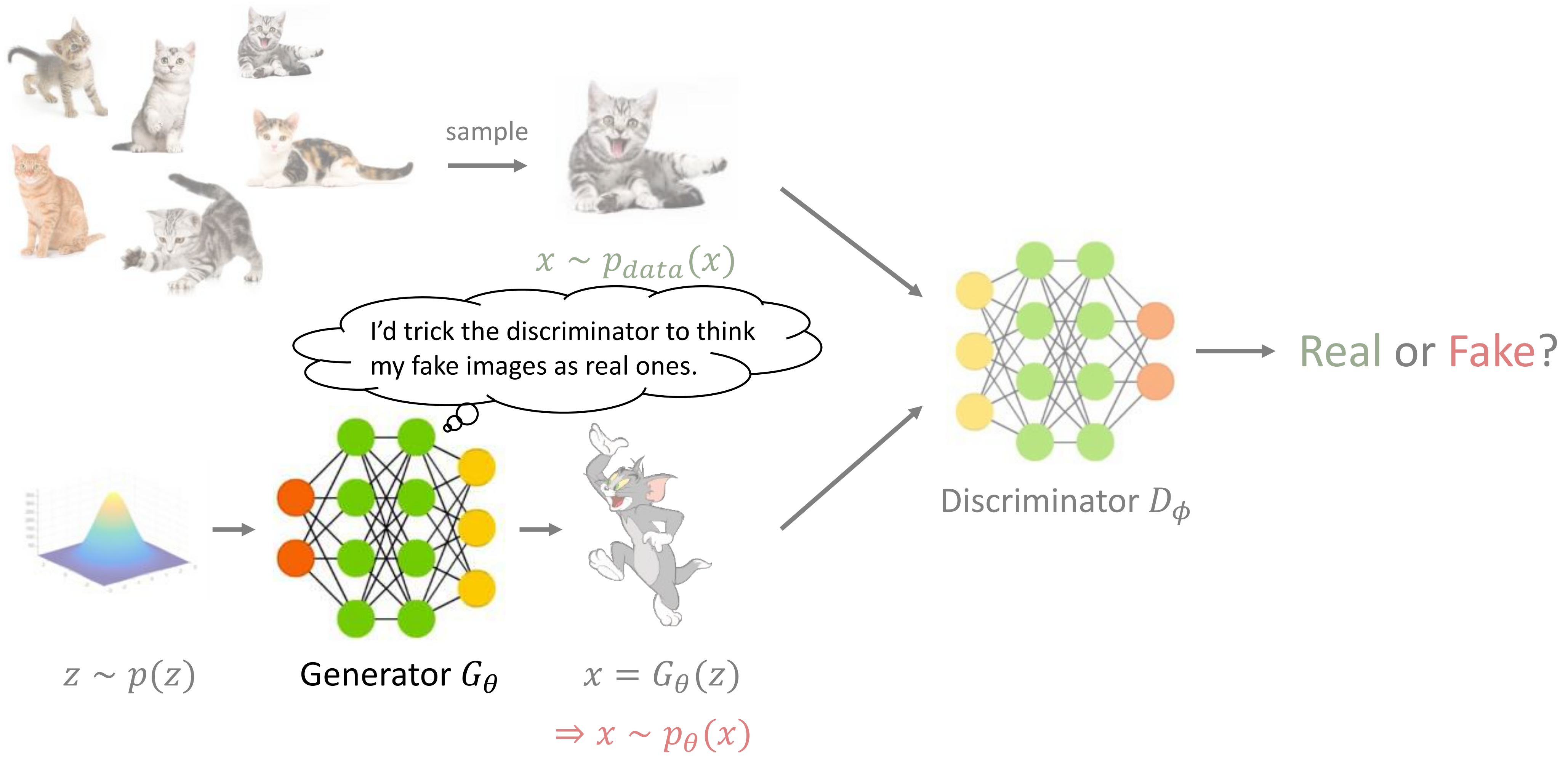
Generative adversarial networks (GANs)



Generative adversarial networks (GANs)



Generative adversarial networks (GANs)



Generative adversarial networks (GANs)

- Two-player game objective:

$$\min_{\theta} \max_{\phi} L(\theta, \phi) \coloneqq E_{p_{data}(x)}\left[\log D_{\phi}(x)\right] + E_{p_{\theta}(x)}[\log(1 - D_{\phi}(x))]$$

$$D_{\phi}(x) \coloneqq P(x\ is\ real), \quad 1 - D_{\phi}(x) = P(x\ is\ fake)$$

  - With fixed θ: training D_{ϕ} as the classifier of the following binary classification task with maximum likelihood (i.e. negative cross-entropy):

$$y = 1 \text{ if } x \sim p_{data}(x), \quad \text{else} \quad y = 0 \text{ if } x \sim p_{\theta}(x)$$

  - With fixed ϕ: training G_{θ} to minimize the log-probability of $x \sim p_{\theta}(x)$ being classified as “fake data” by D_{ϕ}

Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014

Generative adversarial networks (GANs)

- Solving the two-player game objective:

$$\min_{\theta} \max_{\phi} L(\theta, \phi) := E_{p_{data}(x)}[\log D_{\phi}(x)] + E_{p_{\theta}(x)}[\log(1 - D_{\phi}(x))]$$

  - Assume the discriminator network D_{ϕ} has infinite capacity: with fixed θ

$$\phi^* := \max_{\phi} L(\theta, \phi) \text{ satisfies } D_{\phi^*}(x) = \frac{p_{data}(x)}{p_{data}(x)+p_{\theta}(x)}$$

  - Plug-in the optimal discriminator (θ dependant) to the objective:

$$L(\theta, \phi^*(\theta)) = E_{p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_{\theta}(x)}\right] + E_{p_{\theta}(x)}\left[\log \frac{p_{\theta}(x)}{p_{data}(x) + p_{\theta}(x)}\right]$$

$$= KL[p_{data}(x) \,||\, \tilde{p}(x)] + KL[p_{\theta}(x) \,||\, \tilde{p}(x)] - 2\log 2$$

$$\tilde{p}(x) := \frac{1}{2} p_{data}(x) + \frac{1}{2} p_{\theta}(x)$$

$$= 2\, JS[p_{data}(x) \,||\, p_{\theta}(x)] - 2\log 2$$

Jensen-Shannon divergence between $p_{data}(x)$ and $p_{\theta}(x)$

$JS[p_{data} \,||\, p_{\theta}] = 0 \iff p_{\theta}(x) = p_{data}(x)$

Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014

Generative adversarial networks (GANs)

- Optimising GANs in practice: a double-loop algorithm
  - Inner loop: with fixed θ, optimise ϕ for a few gradient ascent iterations:

$$\max_{\phi} E_{p_{data}(x)}[\log D_\phi(x)] + E_{p_\theta(x)}[\log(1 - D_\phi(x))]$$

  - Outer loop: with fixed ϕ from the inner loop, optimize θ by ONE gradient descent step:

$$\min_{\theta} E_{p_\theta(x)}[\log(1 - D_\phi(x))]$$

  - In practice the expectations $E_{p_{data}(x)}[\cdot]$ and $E_{p_\theta(x)}[\cdot]$ are estimated by mini-batches:

$$E_{p_{data}(x)}[\log D_\phi(x)] \approx \frac{1}{M}\sum_{m=1}^{M} \log D_\phi(x_m), x_m \sim p_{data}(x)$$

$$E_{p_\theta(x)}\left[\log\left(1 - D_\phi(x)\right)\right] \approx \frac{1}{K}\sum_{k=1}^{K} \log\left(1 - D_\phi\left(G_\theta(z_k)\right)\right), z_k \sim p(z)$$

Loop over until convergence

Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014

Generative adversarial networks (GANs)

Practical implementation for solving $\min_{\theta} \max_{\phi} E_{p_{data}(x)}[\log D_{\phi}(x)] + E_{p_{\theta}(x)}[\log(1 - D_{\phi}(x))]$ (pseudo code):

- Initialise θ, ϕ, learning rates γ_D, γ_G, SGD outer-/inner-loop iterations T, K
- For $t = 1, \dots, T$

  # update discriminator

  - For $i = 1, \dots, K$
    - $z_1, \dots, z_M \sim p(z)$
    - $x_1, \dots, x_M \sim p_{data}(x)$
    - $\phi \leftarrow \phi + \gamma_D \nabla_{\phi} [\frac{1}{M} \sum_{m=1}^{M} \log D_{\phi}(x_m) + \frac{1}{M} \sum_{m=1}^{M} \log(1 - D_{\phi}(G_{\theta}(z_m)))]$

  # update generator

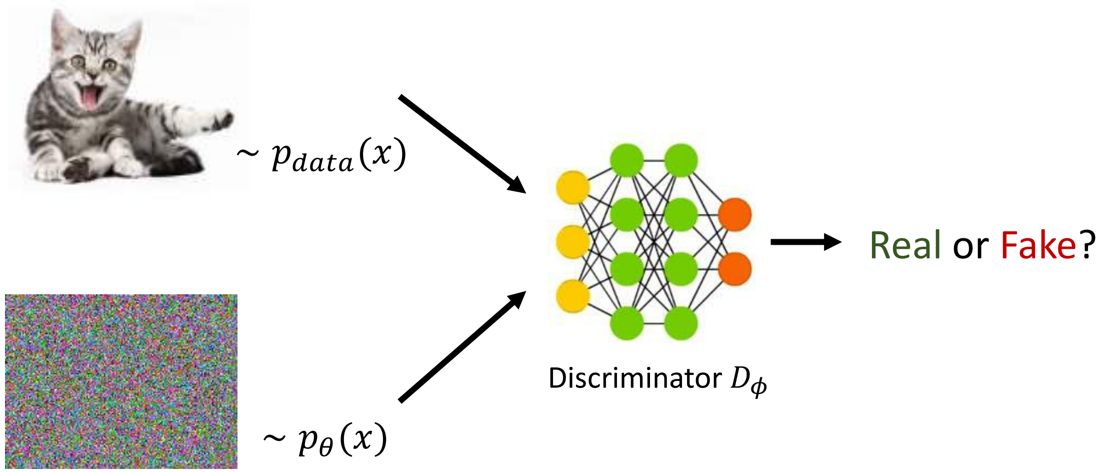
  - $z_1, \dots, z_J \sim p(z)$
  - $\tilde{x}_j = G_{\theta}(z_j), j = 1, \dots, J$
  - $\theta \leftarrow \theta - \gamma_G \nabla_{\theta} \frac{1}{J} \sum_{j}^{J} \log\left(1 - D_{\phi}(\tilde{x}_j)\right)$

Learning rates γ_D, γ_G & inner-loop iterations K need to be chosen carefully!
(otherwise training may be unstable)

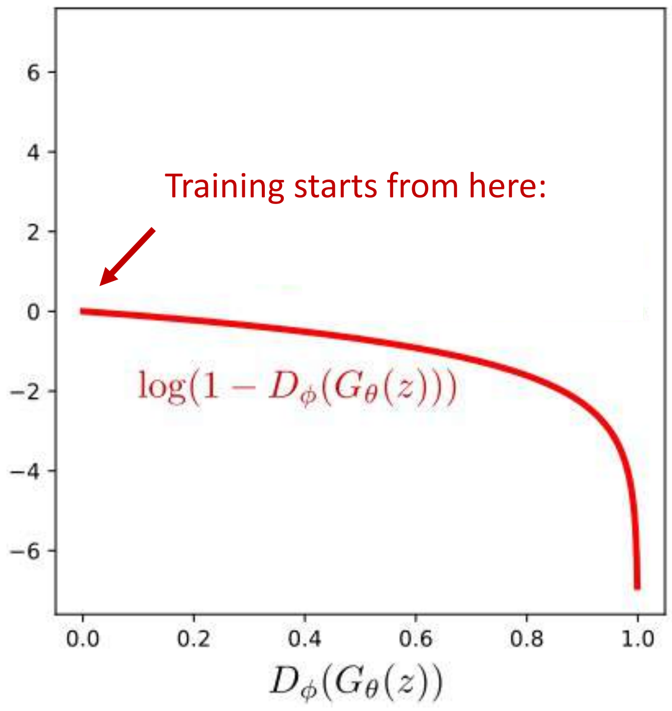
Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014

Generative adversarial networks (GANs)

- Practical strategy for training the generator G_θ:
  - At the beginning, generated image quality is bad



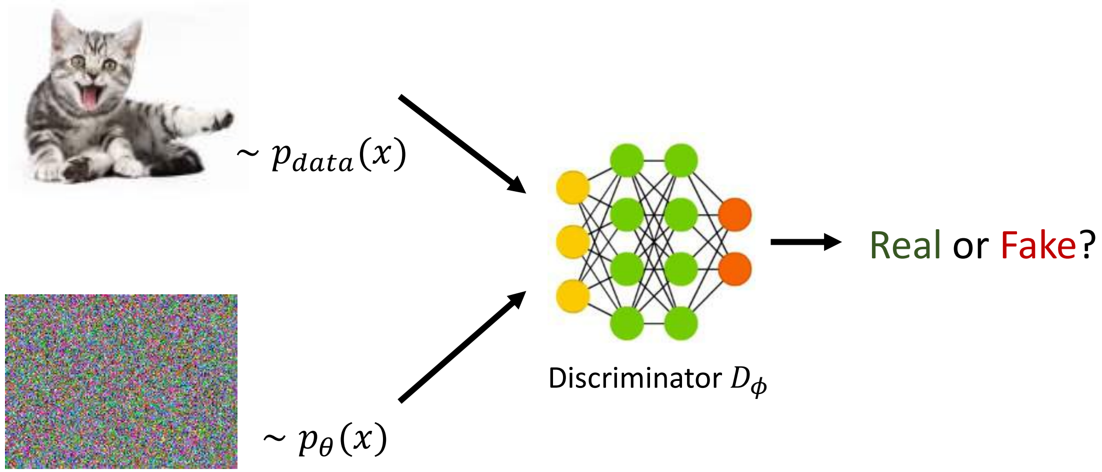
$\Rightarrow$ Discriminator can classify fake images correctly with high confidence: $D_\phi\big(G_\theta(z)\big) \approx 0$

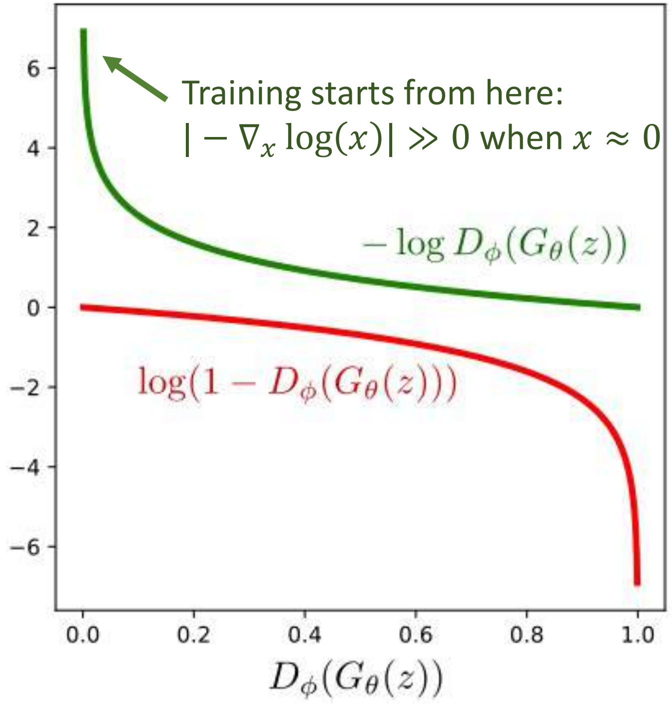


Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014

Generative adversarial networks (GANs)

- Practical strategy for training the generator G_θ:
  - At the beginning, generated image quality is bad





⇒ Use an alternative "non-saturate" loss:

$$\min_\theta -E_{p_\theta(x)}[\log D_\phi(x)]$$

"maximizing the probability of making wrong decisions on fake data"

Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014

Wasserstein GAN

- Discriminator can be used to score the provided inputs

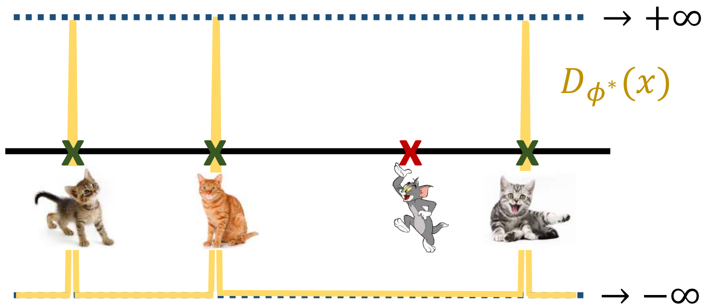
$$\min_{\theta} \max_{\phi} \underline{E_{p_{data}(x)}[D_\phi(x)]} - \underline{E_{p_\theta(x)}[D_\phi(x)]}$$

Discriminator should assign high scores to data inputs and low scores to fake inputs

- Assume the discriminator network D_ϕ has infinite capacity: a trivial solution

$$D_{\phi^*}(x) = +\infty \text{ if } x \sim p_{data}(x) \text{ else } D_{\phi^*}(x) = -\infty$$

No useful gradient info for generator learning!



Arjovsky et al. Wasserstein Generative Adversarial Networks. ICML 2017
Gulrajani et al. Improvedtraining of Wasserstein GANs. NeurIPS 2017

Wasserstein GAN

- Regularised discriminator can be used to score the provided inputs

$$\min_{\theta} \max_{\phi} \underline{E_{p_{data}(x)}[D_\phi(x)]} - \underline{E_{p_\theta(x)}[D_\phi(x)]} \text{ subject to } \| D_\phi(\cdot)\|_L \leq 1$$

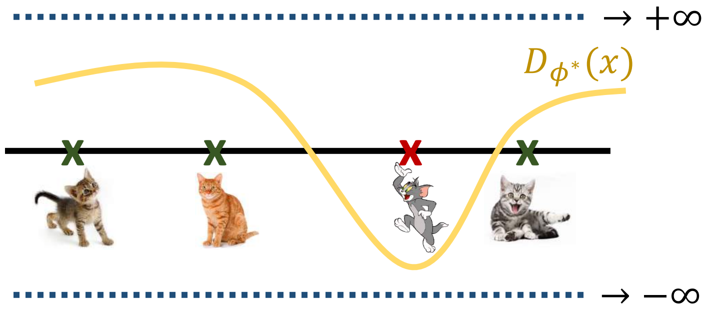
Discriminator should assign high scores to data inputs and low scores to fake inputs
At the same time, discriminator should be smooth to provide useful gradient for learning G_θ

- $\| D_\phi(\cdot)\|_L \leq 1$ is the Lipschitz continuity constraint

$$\|\nabla_x D_\phi(x)\|_2 \leq 1 \text{ for all } x$$

- Equivalent to minimising the Wasserstein distance :

$$W_2(p_{data}(x), p_\theta(x)) := \sup_{\phi: \| D_\phi(\cdot)\|_L \leq 1} E_{p_{data}(x)}[D_\phi(x)] - E_{p_\theta(x)}[D_\phi(x)]$$



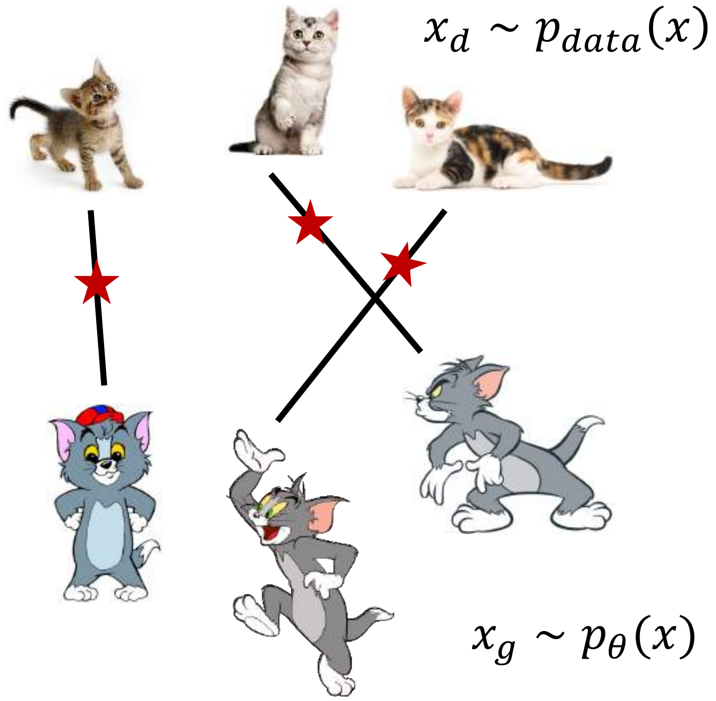
Arjovsky et al. Wasserstein Generative Adversarial Networks. ICML 2017
Gulrajani et al. Improvedtraining of Wasserstein GANs. NeurIPS 2017

Wasserstein GAN

- Practical implementation: WGAN-GP

Regulariser to enforce the Lipschitz continuity constraint

$$\min_{\theta}\max_{\phi} E_{p_{data}(x)}[D_{\phi}(x)] - E_{p_{\theta}(x)}[D_{\phi}(x)] + \boxed{\lambda E_{\hat{p}(x)}[(\|\nabla_x D_{\phi}(x)\|_2 - 1)^2]}$$



- $\hat{p}(x)$ is defined by the following sampling procedure:

$$x_d \sim p_{data}(x)$$
$$x_g \sim p_{\theta}(x)$$
$$\alpha \sim Uniform([0,1])$$
$$x = \alpha x_d + (1-\alpha)x_g$$

- Training strategy is similar to the original GAN
    - Double-loop algorithm
    - Minibatch sampling

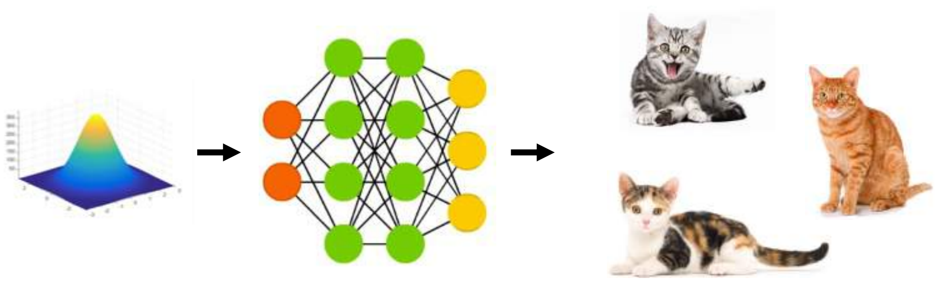
Arjovsky et al. Wasserstein Generative Adversarial Networks. ICML 2017
Gulrajani et al. Improvedtraining of Wasserstein GANs. NeurIPS 2017

Generative Models

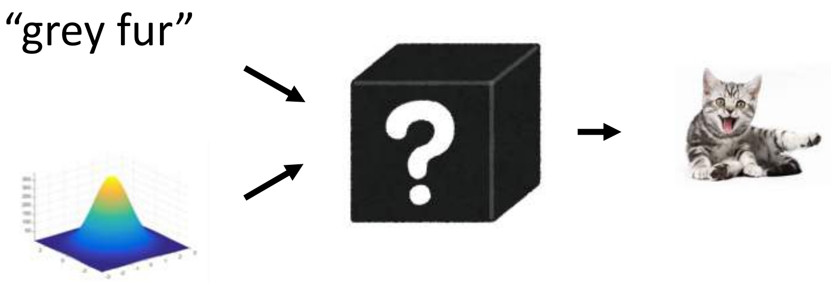
Advances & Applications

Yingzhen Li (yingzhen.li@imperial.ac.uk)

Conditional latent variable models



(unconditional) latent variable models

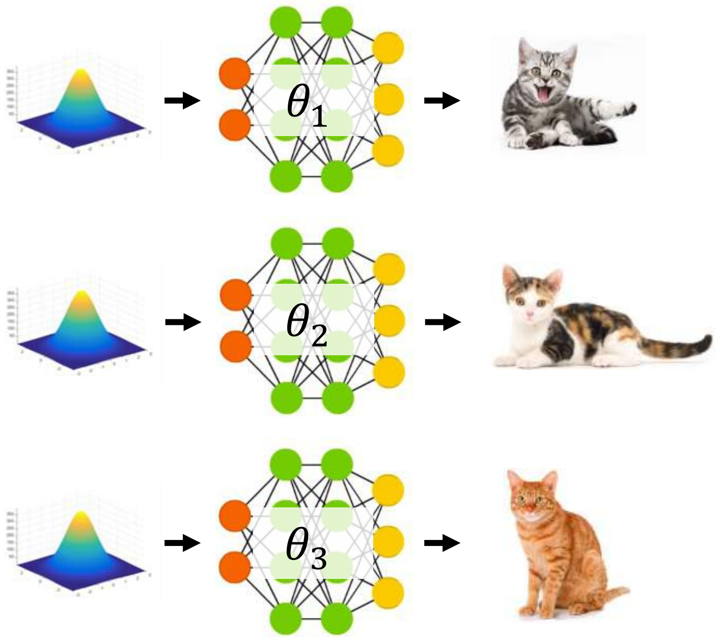


How to construct conditional LVMs?

Conditional latent variable models

- Goal: learn a generative model $p_\theta(x|y)$
  - x: data to be generated (e.g. an image)
  - y: label/info that the generation process is conditioned on (e.g. fur colour)

- Idea 1: if $y \in \{1, \dots, C\}$, train a set of models
  $$p_\theta(x|y=c) = p_{\theta_c}(x) = \int p_{\theta_c}(x|z)p(z)dz$$
  - Parameter inefficient: need to train C networks
  - Cannot generalise to continuous y



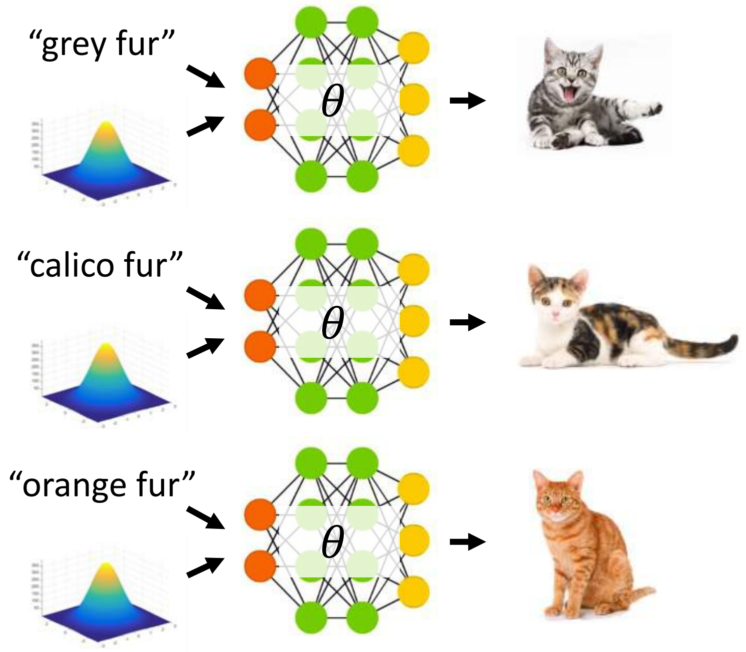
Conditional latent variable models

- Goal: learn a generative model $p_\theta(x|y)$
  - x: data to be generated (e.g. an image)
  - y: label/info that the generation process is conditioned on (e.g. fur colour)

- Idea 2: make (z, y) as the input of the network

$$p_\theta(x|y=c) = \int p_\theta(x|z, y=c)p(z)dz$$

  - Parameter ~~inefficient~~ efficient
  - Can~~not~~ generalise to continuous y
  - Disentangled the learned representation z from the label info y



Conditional VAEs

- Training the conditional LVM:

model: $p_\theta(x|y) = \int p_\theta(x|z,y)p(z)dz,$ data: $\{(x_n, y_n)\}_{n=1}^N \sim p_{data}(x,y)$

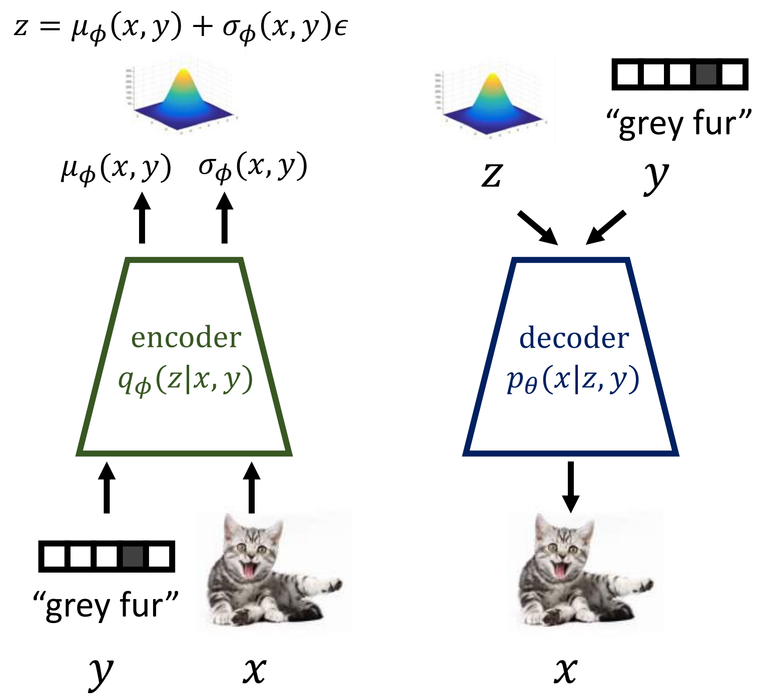
  - Maximum Likelihood training (MLE):

$$\max_\theta E_{p_{data}(x,y)} [\log p_\theta(x|y)]$$

  - (conditional) variational lower-bound:

$$\log p_\theta(x|y) \geq E_{q_\phi(z|x,y)}[\log p_\theta(x|z,y)] - KL[q_\phi(z|x,y) \| p(z)]$$
$$:= L(x, y, \phi, \theta)$$

$\Rightarrow$ maximise $E_{p_{data}(x,y)}[L(x,y,\phi,\theta)]$ w.r.t. ϕ, θ



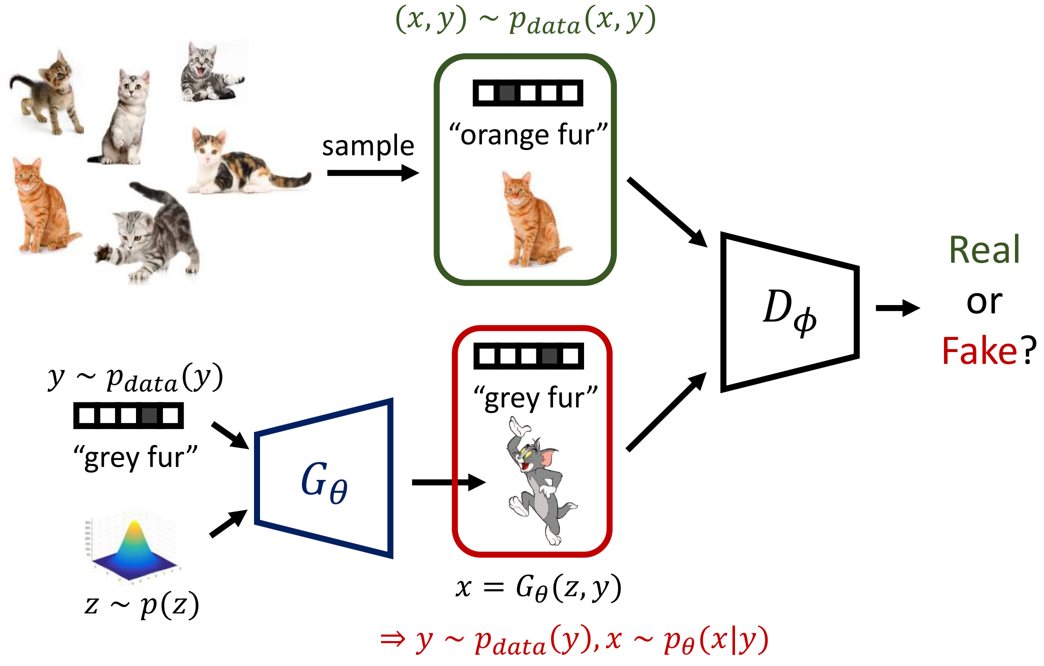
Conditional GANs

- Training the conditional LVM:

model: $p_\theta(x|y) = \int p_\theta(x|z,y)p(z)dz,$ data: $\{(x_n, y_n)\}_{n=1}^N \sim p_{data}(x,y)$

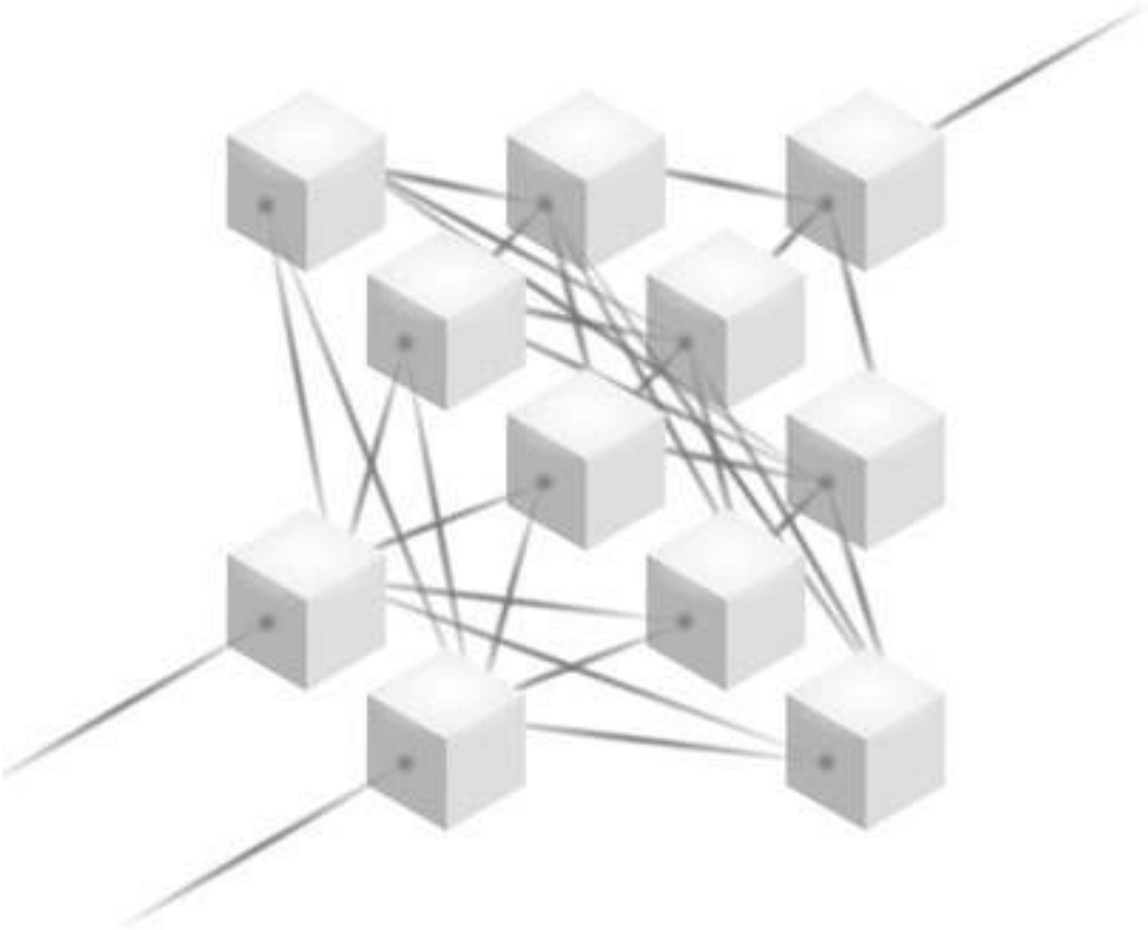
- Adversarial training:
  - Label $(x_n, y_n) \sim p_{data}(x,y)$ as "real"
  - Label $(G_\theta(z,y), y), z \sim p(z)$ as "fake"
  - For fake data, sample label $y \sim p_{data}(y)$

$$\min_\theta \max_\phi \underbrace{E_{p_{data}(x,y)}[\log D_\phi(x,y)]}_{\text{"real"}} + \underbrace{E_{p(z)p_{data}(y)}[\log(1 - D_\phi(G_\theta(z,y),y))]}_{\text{"fake"}}$$

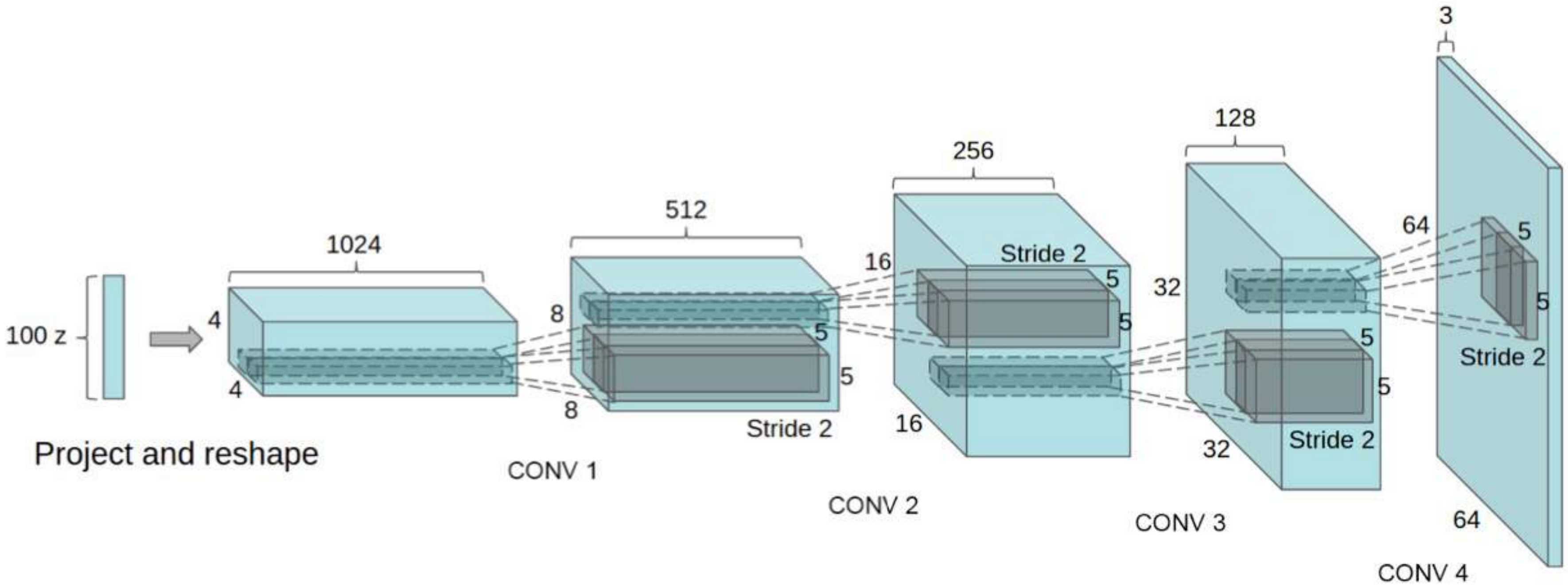


Mirza and Osindero. Conditional Generative Adversarial Networks. arXiv:1411.1784

Generative Model Architecture Design



DCGAN

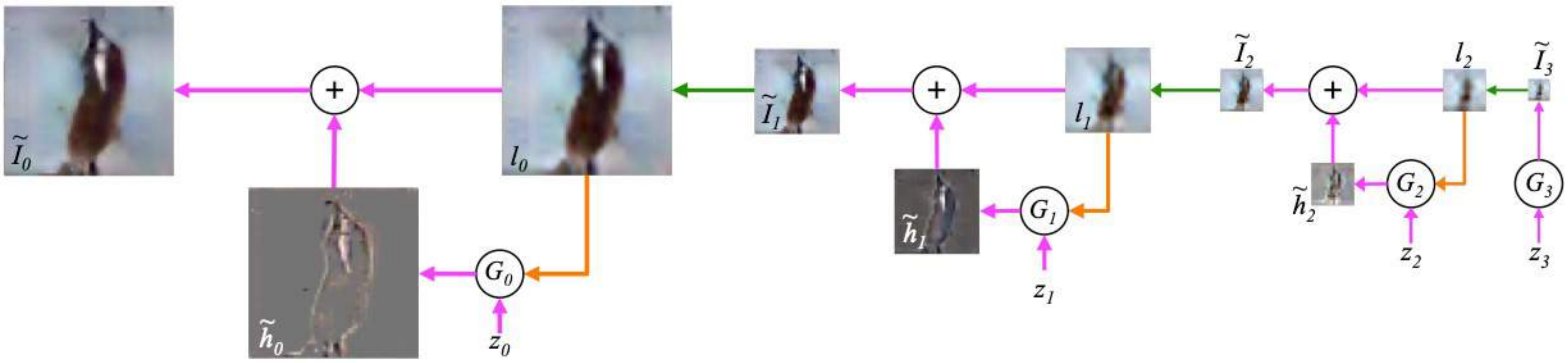


Tricks used in the DCGAN architecture & training:

- Replace pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm.
- Remove fully connected hidden layers for deeper architectures.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. ICLR 2016

LAPGAN



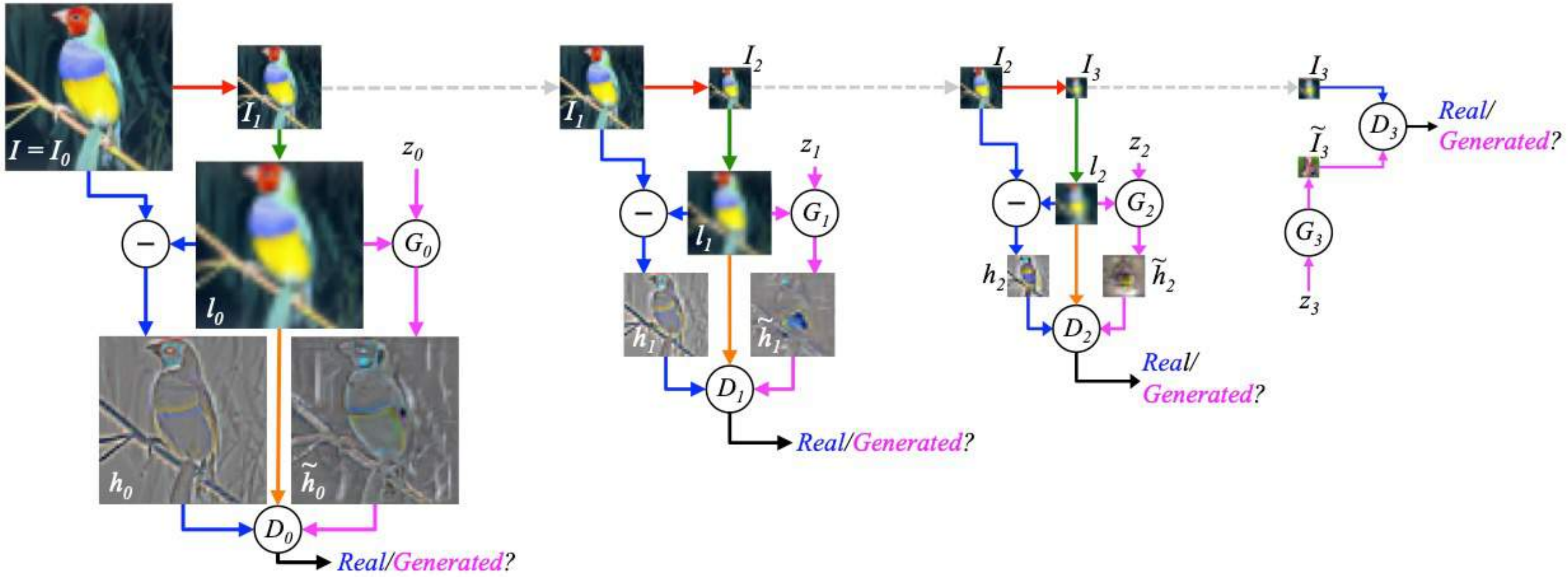
Generator's multi-scale architecture:

- Start generation for low-resolution images: $\tilde{I}_3 = G_3(z_3)$
- Generate higher-resolution images conditioned on the lower-resolution ones:

$$l_i = upscale(\tilde{I}_{i+1}), \qquad \tilde{I}_i = l_i + G_i(z_i, l_i)$$

Denton et al. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. NeurIPS 2015

LAPGAN



LAPGAN's discriminator design:

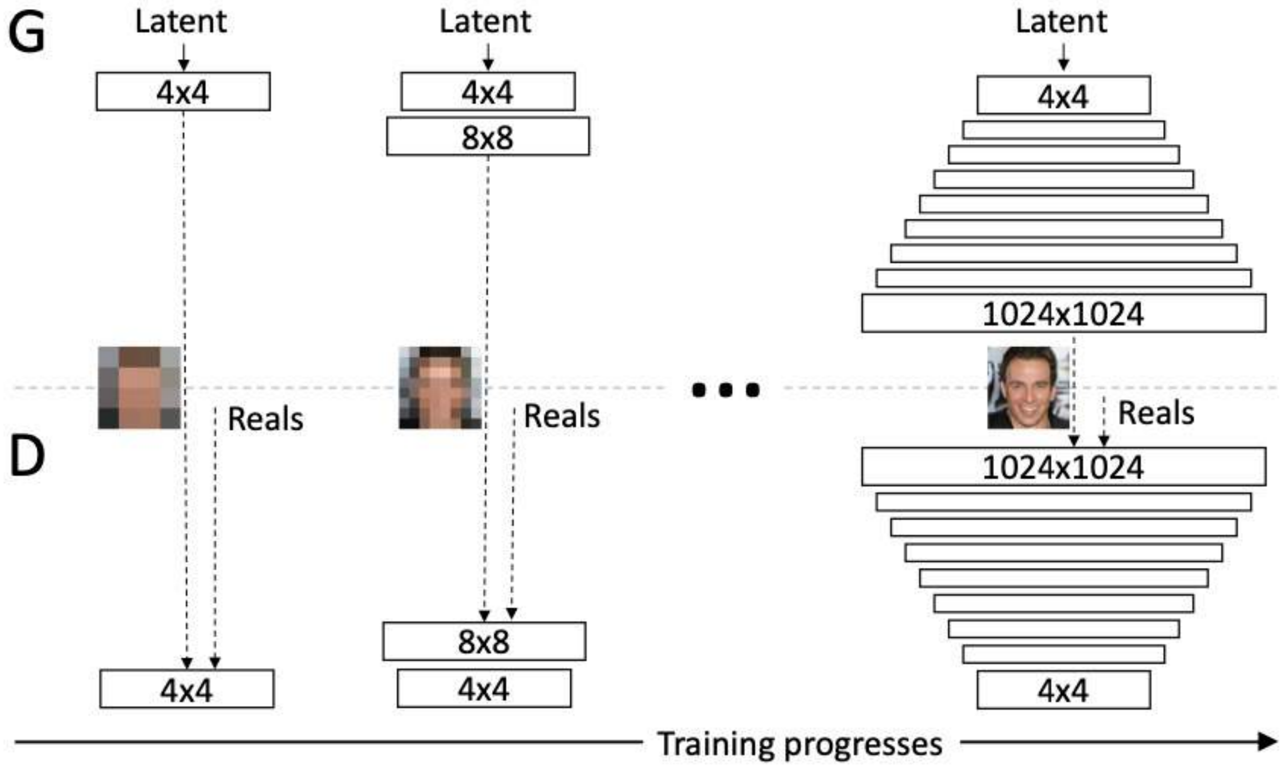
- Multiple discriminators in use, paired with generators at different resolutions
- At each resolution, check whether the "fine details" generated by G_i matches the real ones:
  - "real" input: $h_i = I_i - l_i, l_i = upscale(I_{i+1}), I_{i+1} = downscale(I_i)$
  - "fake" input: $\tilde{h}_i = G_i(z_i, l_i)$

Denton et al. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. NeurIPS 2015

Progressive GAN

Progressively building GAN generator and discriminator:

- High-res images downscaled to get training data of low resolutions
- Train a GAN starting from 4x4 images
- Add new layers into generator and discriminator
- Adapt old & new layers by GAN training with 8x8
- Continue with 16x16, 32x32…



Karras et al. Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR 2018

StyleGAN

Disentangling different sources of randomness:

- Latent variable z is transformed to "style" representation w
- This "style" w controls generation at every resolutions
- Fine details generated with noise at different scales

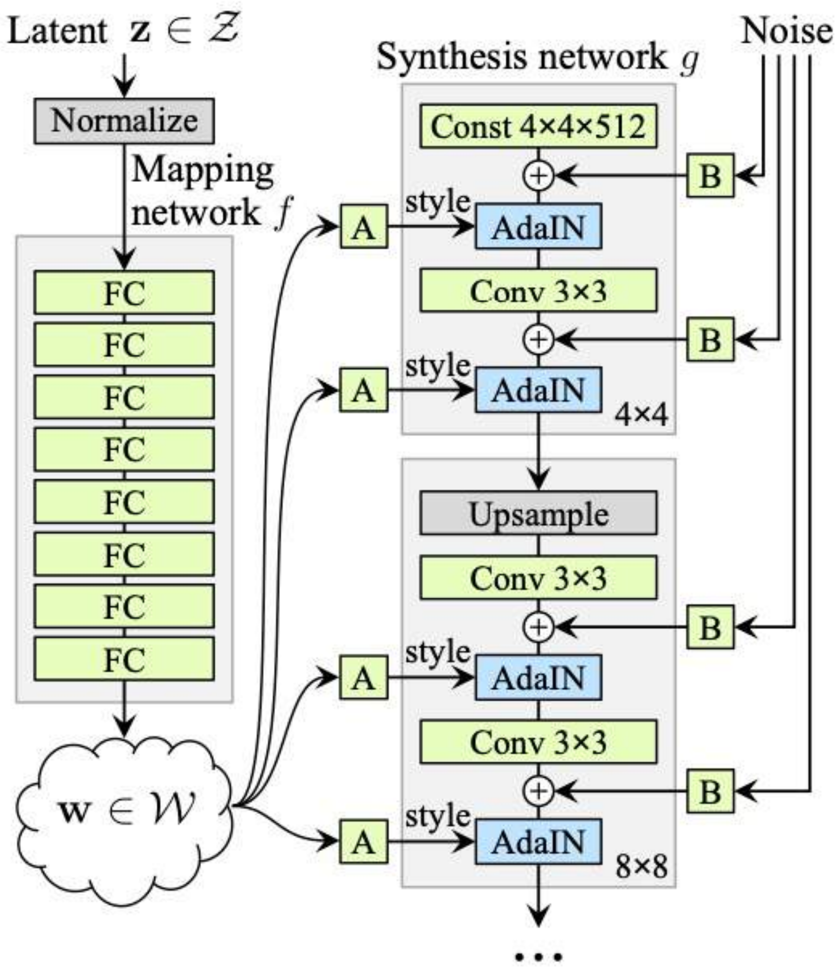
$$y = (y_s, y_b) = A(w)$$

$$x = \text{"upscaled last block output"} + B(\epsilon)$$

$$AdaIN(x_i, y) = y_{s,i} \boxed{\frac{x_i - \mu(x_i)}{\sigma(x_i)}} + y_{b,i}$$

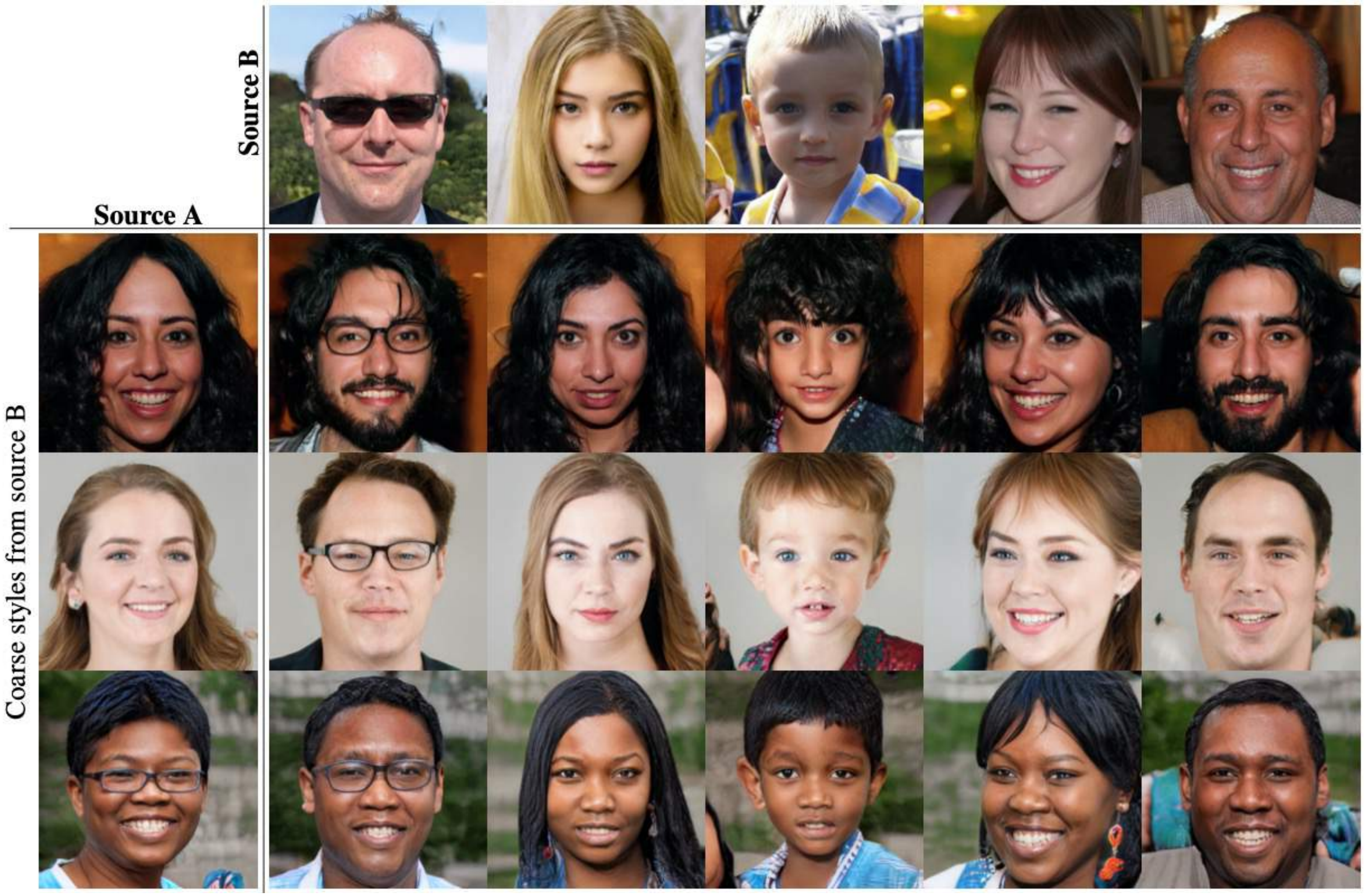
Channel index

normalised feature map for each channel



Karras et al. A Style-Based Generator Architecture for Generative Adversarial Networks. CVPR 2019

StyleGAN

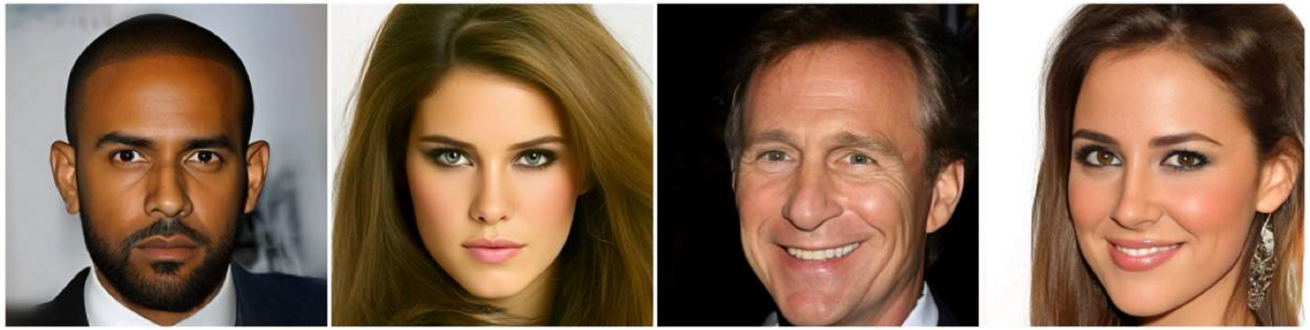


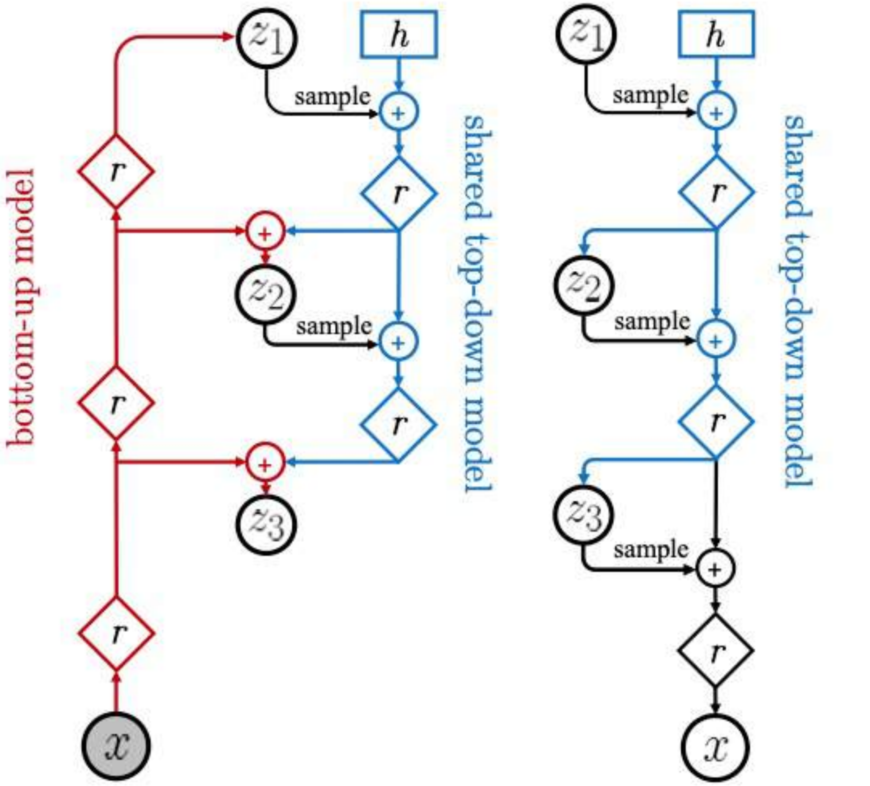
Karras et al. A Style-Based Generator Architecture for Generative Adversarial Networks. CVPR 2019

NVAE – improved VAE image generation

State-of-the-art VAE for image generation (2020):

- Hierarchical LVM with multi-scale architecture
- Using residual networks for the r blocks
- BatchNorm in usage
- Improved q distribution design to control the $KL[q(z|x)||p(z)]$ term



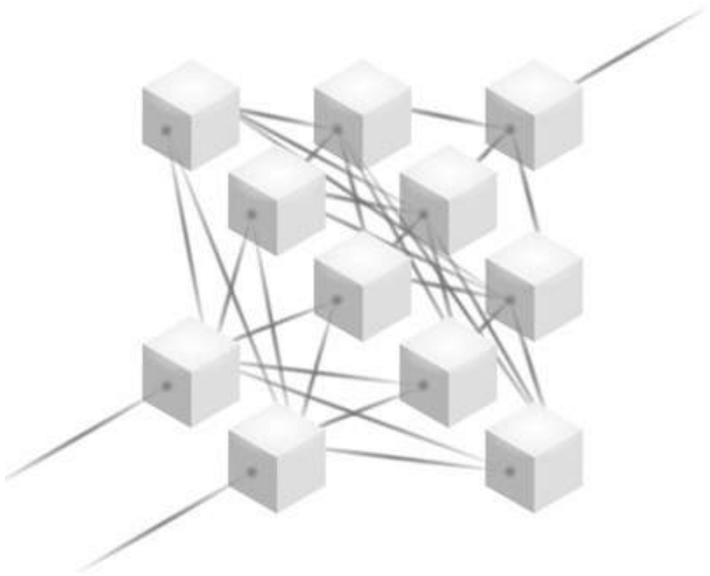


(a) Bidirectional Encoder (b) Generative Model

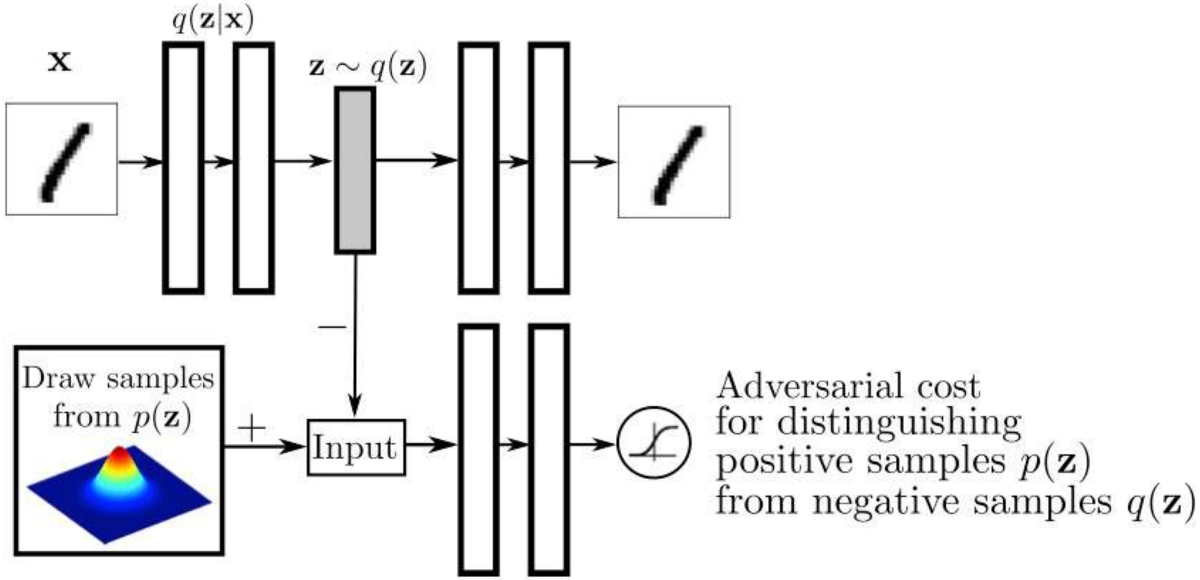
Vahdat and Kautz. NVAE: A Deep Hierarchical Variational Autoencoder. NeurIPS 2020

Progress in architecture design

- GAN progression:
  - DCGAN – fully convolutional neural networks
  - LAPGAN & Progressive GAN – multi-scale architectures
  - StyleGAN – disentangling sources of randomness
- VAE progression:
  - Hierarchical LVMs
  - Tuning the KL regulariser
  - Deep learning tricks applied
  - Incorporate design ideas from GAN networks



Combining VAEs & GANs



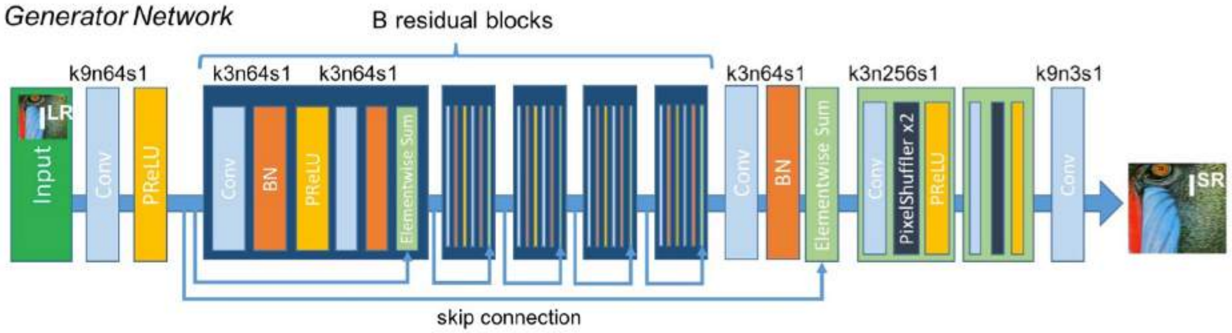
Adversarial auto-encoders:

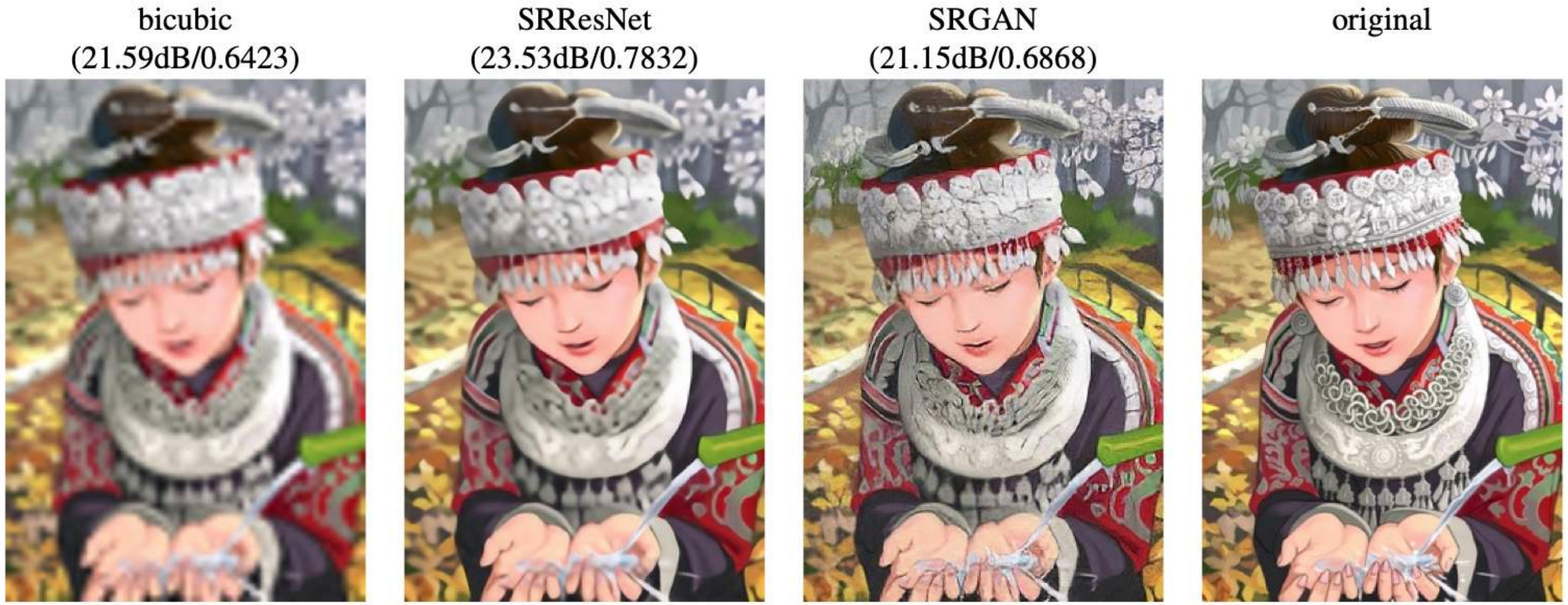
- Reconstruction loss in x space (similar to VAEs)
- Adversarial loss in z space (similar to GANs)

Makhzani et al. Adversarial Autoencoders. arXiv:1511.05644

Applications of Generative Models

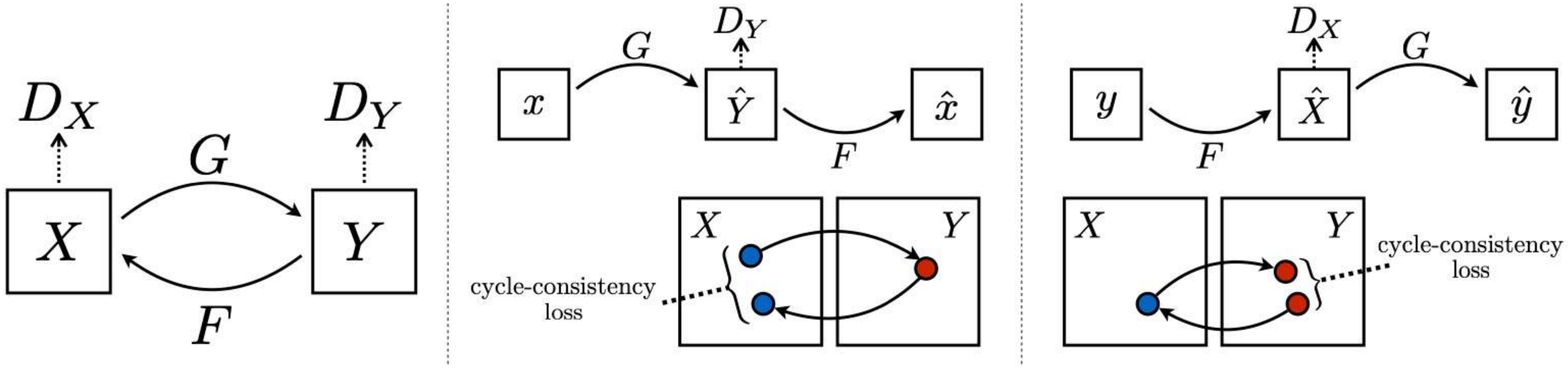
Super Resolution





Ledig et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. CVPR 2017

Image-to-Image Translation



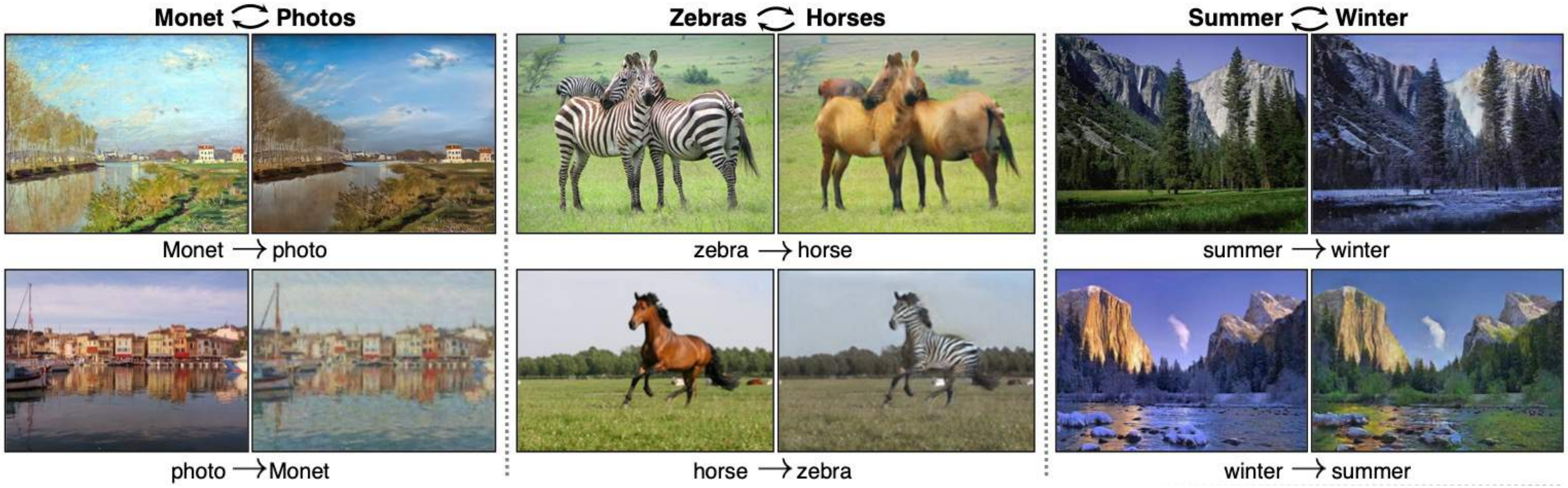
Translation between images in two domains X and Y:

- GAN training applied to generators $y = G(x)$ and $x = F(y)$
- Consistency loss applied in both direction: enforcing

$$x \approx F(G(x)), \qquad y \approx G(F(y))$$

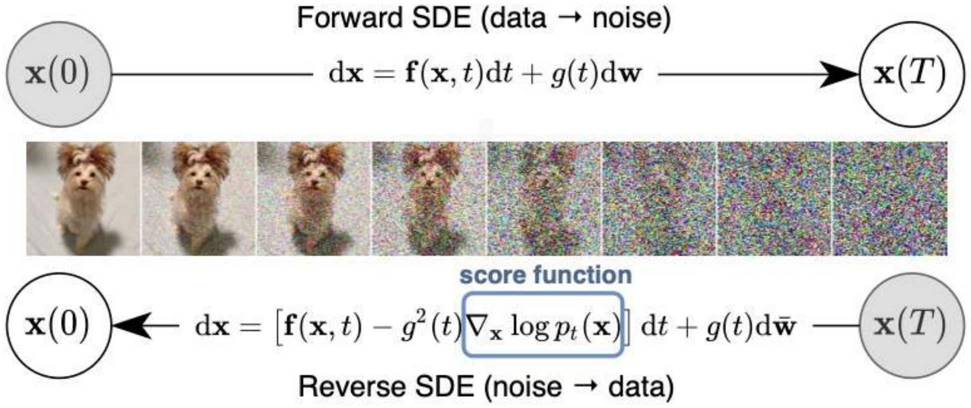
Zhu et al. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017

Image-to-Image Translation

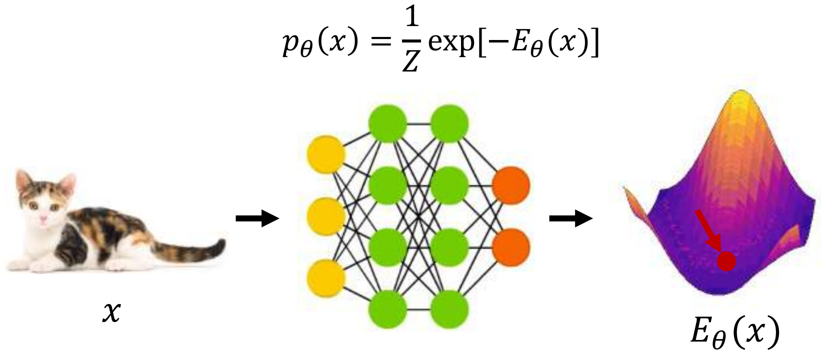


Zhu et al. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017

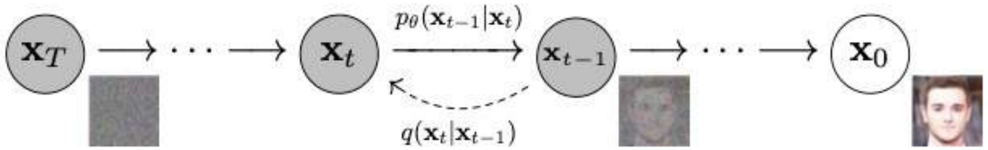
Other types of generative models



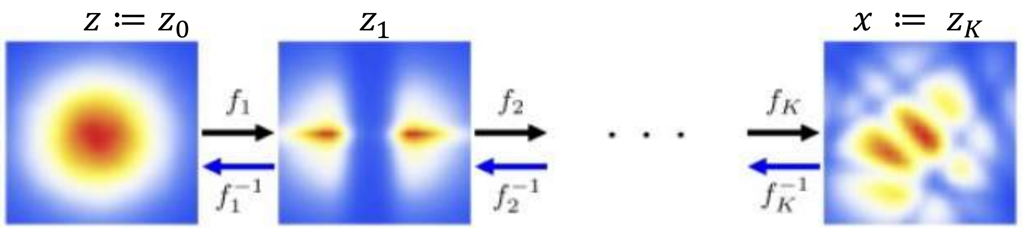
Score-based generative models



Energy-based models



Denoising diffusion probabilistic models
(as a hierarchical latent variable model)



(discrete-time) normalising flow

Song et al. Score-based Generative Modeling Through Stochastic Differential Equations. ICLR 2021
Ho et al. Denoising Diffusion Probabilistic Models. NeurIPS 2020
Du and Mordatch. Implicit Generation and Generalization with Energy Based Models. NeurIPS 2019
Rezende and Mohamed. Variational Inference with Normalizing Flows. ICML 2015

Model design in practice

- “Algorithms/paradigms” vs “network architecture”
  - VAE/GAN/flow/EBM/SGM as “algorithms/modelling paradigms”
  - MLP/CNN/Transformer as “network architecture”

Model design in practice

- On choosing a modelling paradigm (e.g. VAE or GAN or EBM…)
  - Depending on the specific application
    - GAN is often preferred for better visual quality (VAEs & others are catching up)
    - VAE, flow, etc. preferred for applications that need good likelihood estimates
      - E.g. neural data compression
  - Practical solutions are often a mix of many paradigms

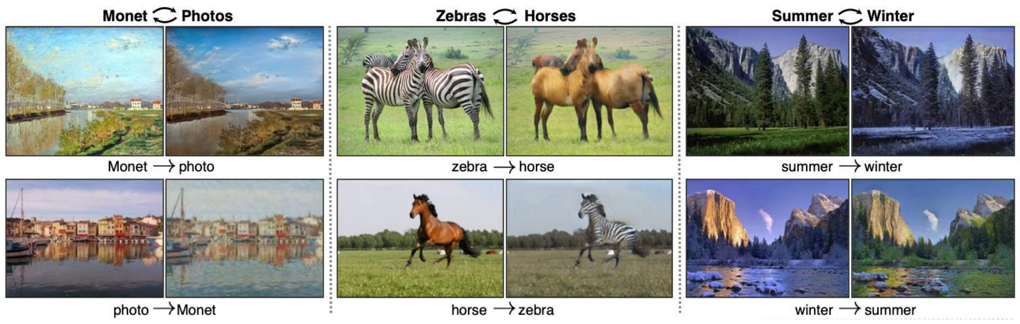
Latest generation results from diffusion models

learned compression

JPEG2000 compression

Dhariwal and Nichol. Diffusion Models Beat GANs on Image Synthesis. NeurIPS 2021
Mentzer et al. Conditional Probability Models for Deep Image Compression. CVPR 2018

Applications of Generative Models

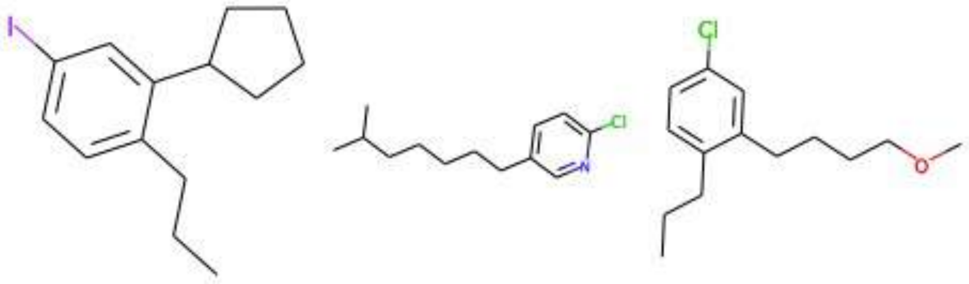


computer vision

speech synthesis

Human: Hey, how are you doing?
AI: I'm good! What would you like to chat about?
Human: Have you done any coding with Python?
AI: Yep! I've done a lot of Python coding as it is my favorite language.
Human: Nice, it's my favorite too! What have you done with it?
AI: Oh, a lot. I have done everything from creating Python modules to creating micro-websites with it.
Human: What web framework do you use?
AI: I used to use Django, but it was too difficult for me. I have found Flask is a better option as it's lighter and simpler.
Human: Wow, I agree 100%. Have you used WebSocket at all?
AI: Yes, I actually have written a chatbot using it!
Human:

natural language generation



molecule structure generation

Deep Fake

Concerns emerge as generative models improve:

- Generated visuals are getting very photo-realistic
- Has been used in fraud and scam videos
- Regulations and detection techniques needed

Channel 4's deep fake queen speech, Dec 25th, 2020

Disclaimer: I do NOT endorse this deep fake broadcast.