EURECOM
Sophia Antipolis

MALCOM

Machine Learning for Communication Systems

Lecture 6
Generative Adversarial Networks (GANs)

Photios Stavrou

(The slides are courtesy of Marios Kountouris)

Spring 2024

Magic of GANs: Image Enhancement

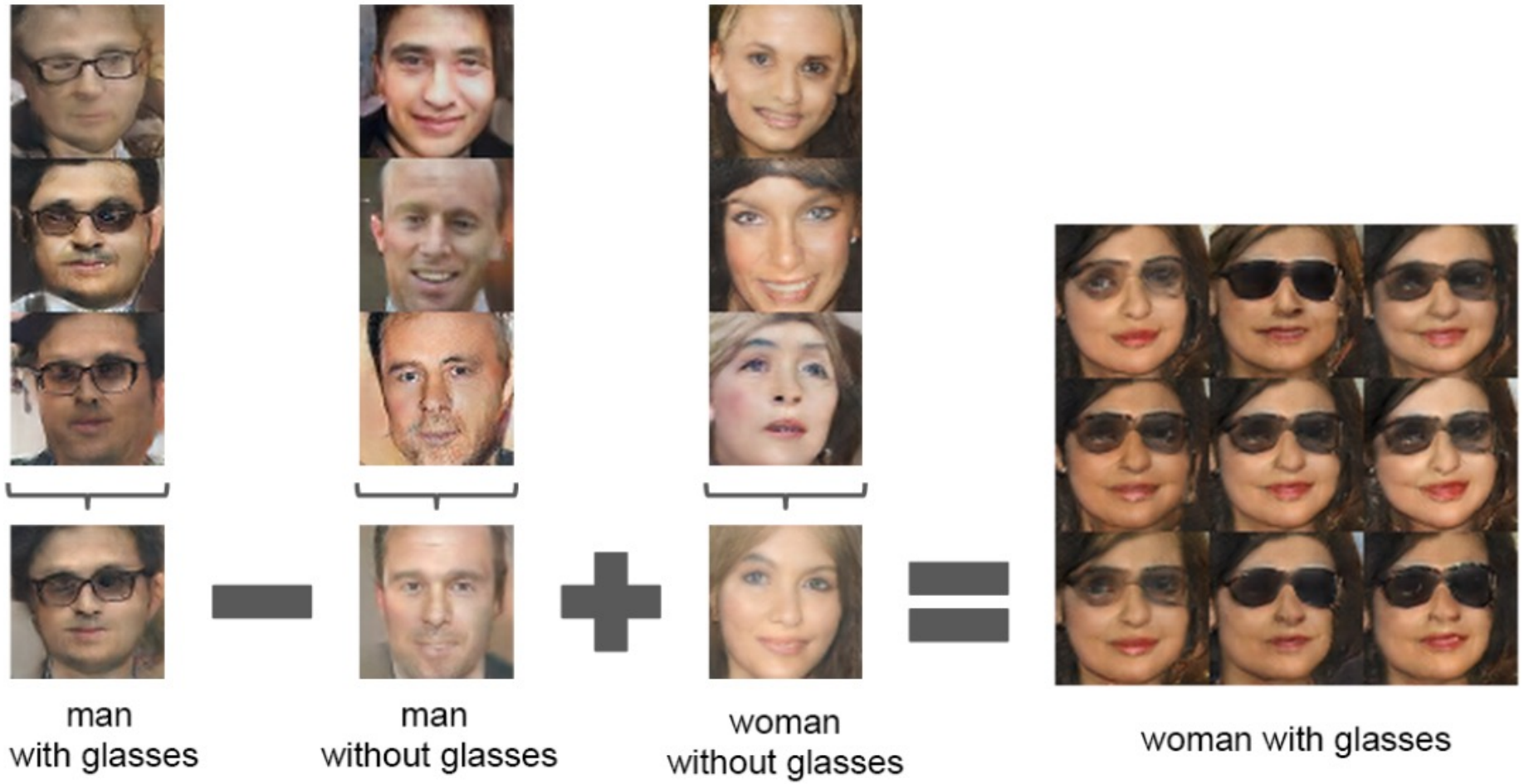
Which one is Computer generated?

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

Magic of GANs: DCGANs in LSUN Dataset

Radford et al.: Unsupervised representation learning with DCGANs, arxiv.org, 2015

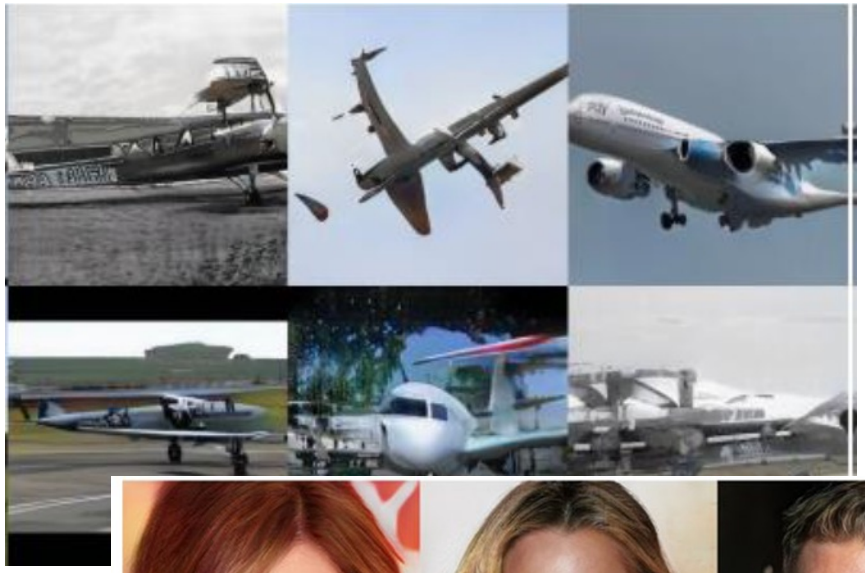
Vector Arithmetic for Visual Concepts

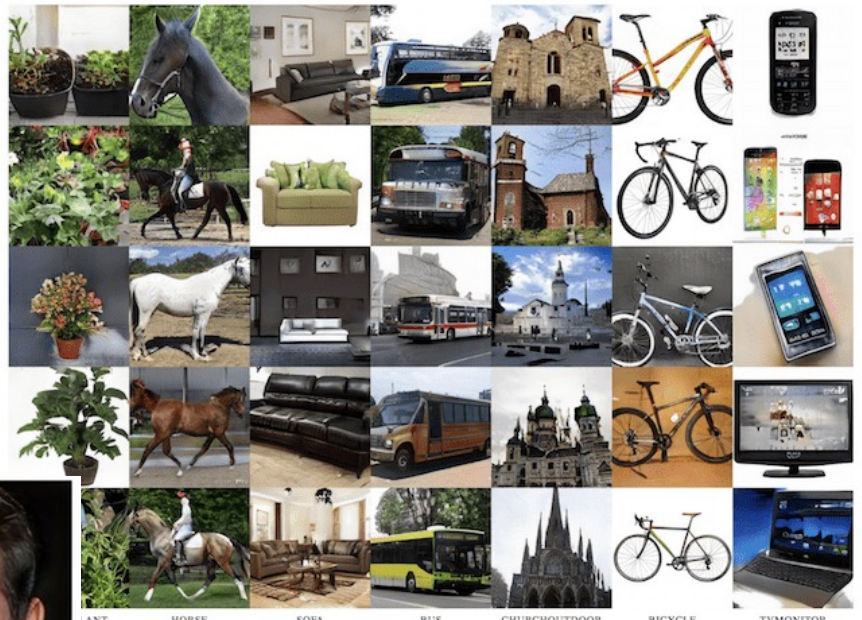


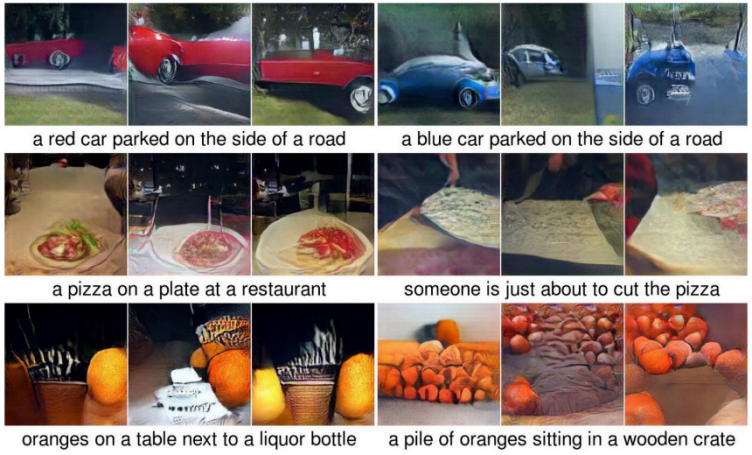
Latent vectors capture interesting patterns!

Radford et al.: Unsupervised representation learning with DCGANs, arxiv.org, 2015

Magic of GANs – Synthetic Photos



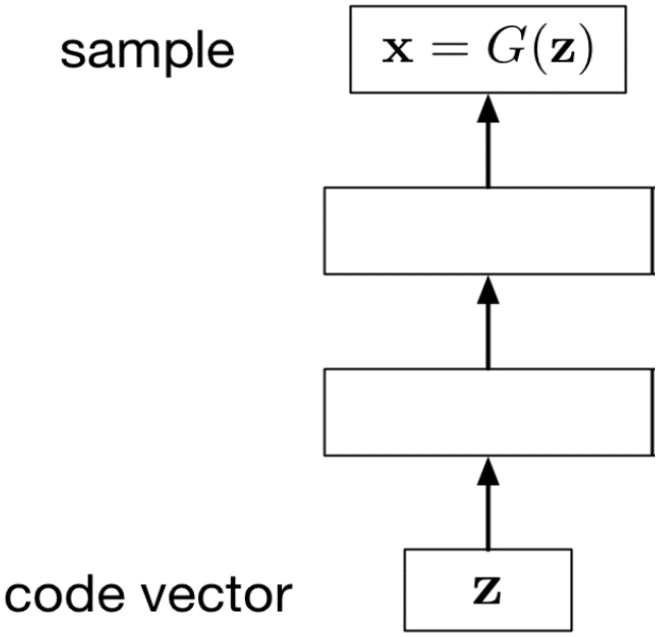




EURECOM
Sophia Antipolis

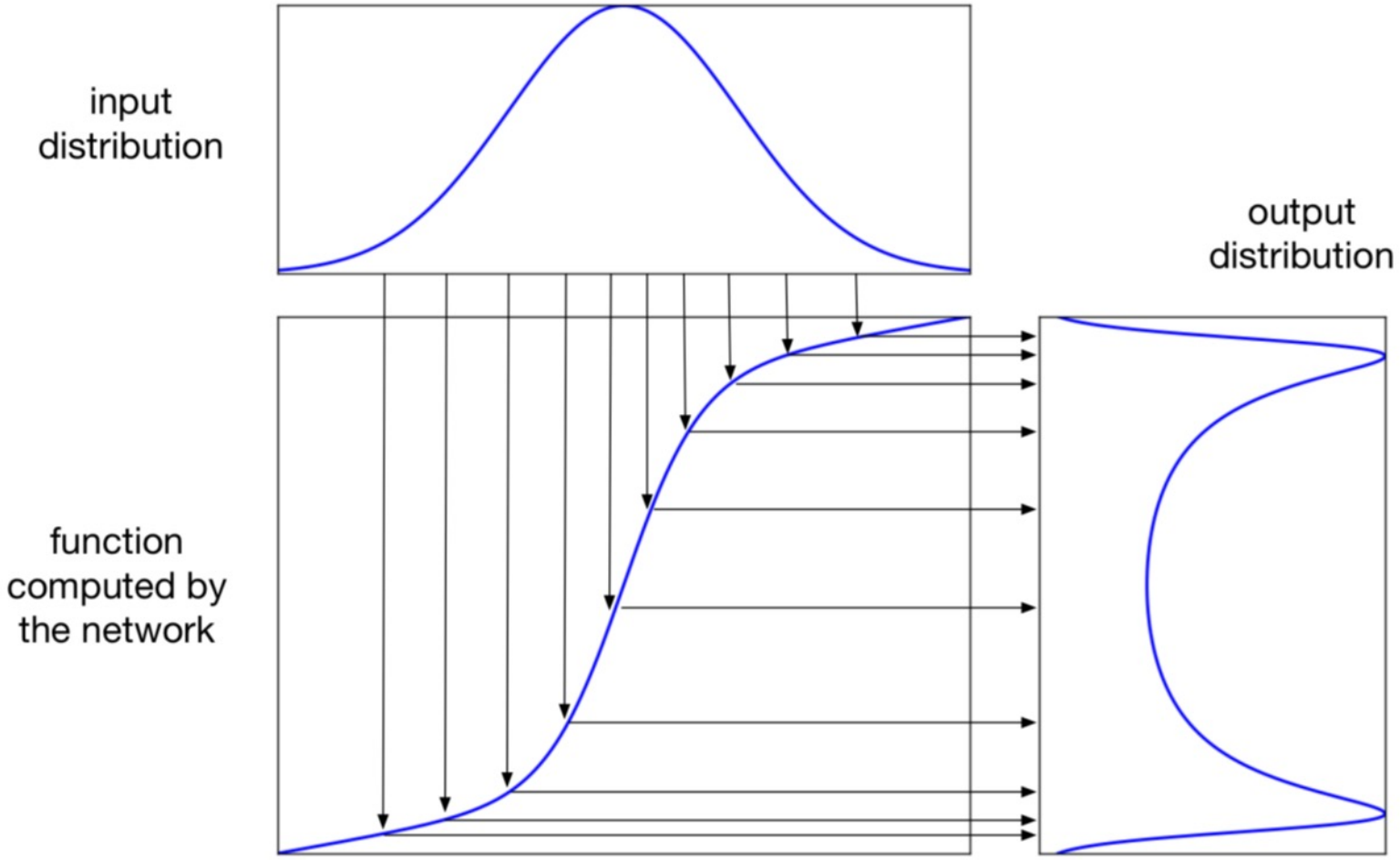
Implicit Generative Models

- Implicit generative models implicitly define a probability distribution
- Start by sampling the code vector z from a fixed, simple distribution (e.g. spherical Gaussian)
- The generator network computes a differentiable function G mapping $\mathbf{z}$ to an $\mathbf{x}$ in data space

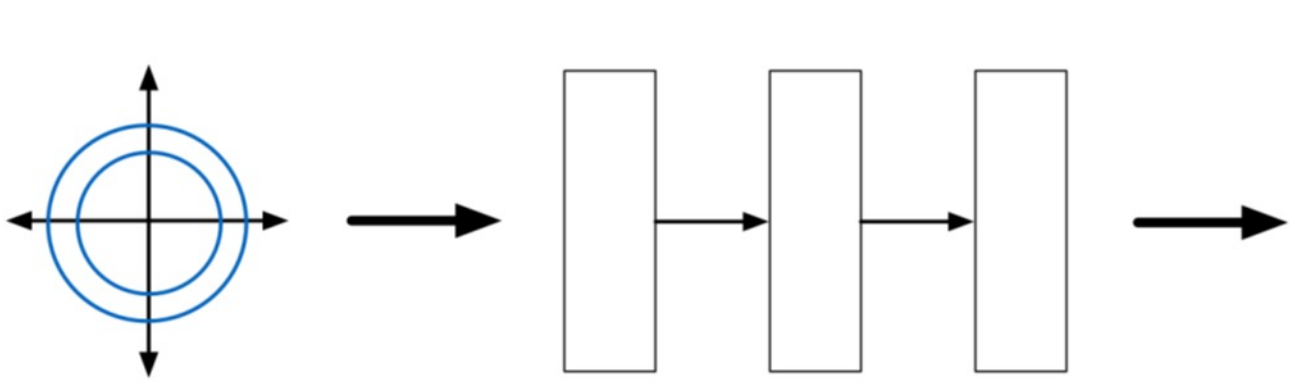


Implicit Generative Models (Cont'd)

- A 1-dimensional example:



Implicit Generative Models (Cont'd)



Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

This is fed to a (deterministic) generator network.

The network outputs an image.

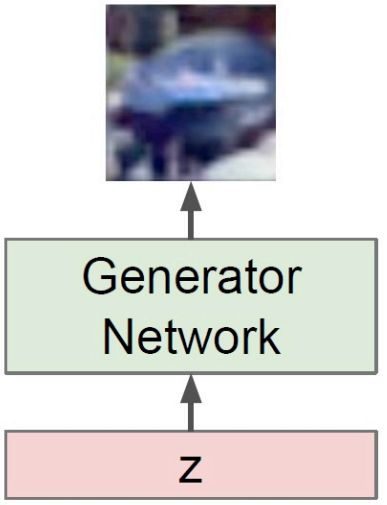
- This sort of architecture may sound absurd, but amazingly, it works!!

What if we just want to sample?

- **Idea:** don't explicitly model density – just sample to generate new instances

- **Problem:** Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- **Solution:** Sample from a simple distribution (e.g. random noise).
  Learn transformation to data distribution.

Output: Sample from training distribution



Input: Random noise

Implicit Models

- Given samples from data distribution p_{data}: $x_1, x_2, \ldots, x_n$
- Given a sampler $q_\phi(z) = \mathrm{DNN}(z; \phi)$ where $z \sim p(z)$
- $\mathrm{x} = q_\phi(z)$ induces a density function p_{model}
- Do not have an explicit form for p_{data} or p_{model}: can only draw samples
- Make p_{model} as close to p_{data} as possible by learning an appropriate ϕ

Deviate from KL beyond explicit models

- Need some measure of how far apart p_{model} and p_{data} are
- With density models, we used KL divergence which gave us the objective $\mathbb{E}_{x \sim p_{data}}[\log p_\theta(x)]$ where we explicitly modeled p_{model} as $p_\theta(x)$
- Not having an explicit $p_\theta(x)$ requires to use distance measures behaving differently from KL
- Example: Maximum mean discrepancy, Jensen-Shannon divergence, Wasserstein

Generative Adversarial Networks – Basic Idea

Compete

- Implicit generative models advantage:
  - if you have some criterion for evaluating the quality of samples

    … then you can compute its gradient with respect to the network parameters, and update the network's parameters to make the sample a little better

Basic idea

Generator

- tries to produce realistic-looking samples
- *creates fake data from random input*

Discriminator

- tries to figure out whether a sample came from the training set or the generator network
- *distinguish real data from fake data*

GANs

Generator

Faking Data

- To create good fake data, the generator must understand what real data looks like
- *Attempts to generate samples that are likely under the true data distribution*
- *Implicitly* learns to *model the true distribution*

Latent Code

- Since the sample is determined by the random noise input, the probability distribution is conditioned on this input
- The random noise is *interpreted by the model as a latent code*, i.e., a point on the manifold

Problem Setup

Generator

- Trained to get better and better at fooling the discriminator (making fake data look real)

Discriminator

- Trained to get better and better at distinguishing real data from fake data

GANs: High Level Explanation

Problem Formulation

Generator $G(z, \theta^{(G)})$

- A differentiable function, G here having parameters $\theta^{(G)}$) mapping from the latent space, $\mathbb{R}^L$ to the data space, $\mathbb{R}^M$

Discriminator $D(z, \theta^{(D)})$

- A differentiable function, D here having parameters $\theta^{(D)}$) mapping from the data space, $\mathbb{R}^M$ to a scalar between 0 and 1 representing the probability that the data is real

Simplifying notation

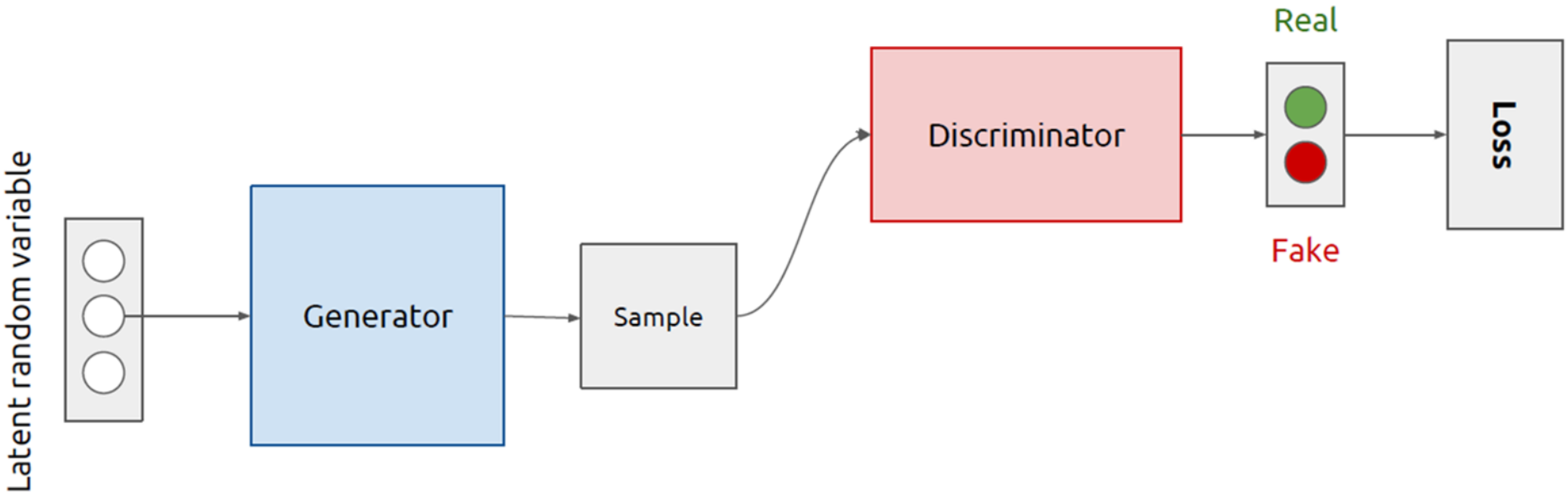
Generator $G(z)$

- Typically G is a Neural Network, but it doesn't have to be
- Note z can go into any layer of the network, not just the first

Discriminator $D(x), D(G(z))$

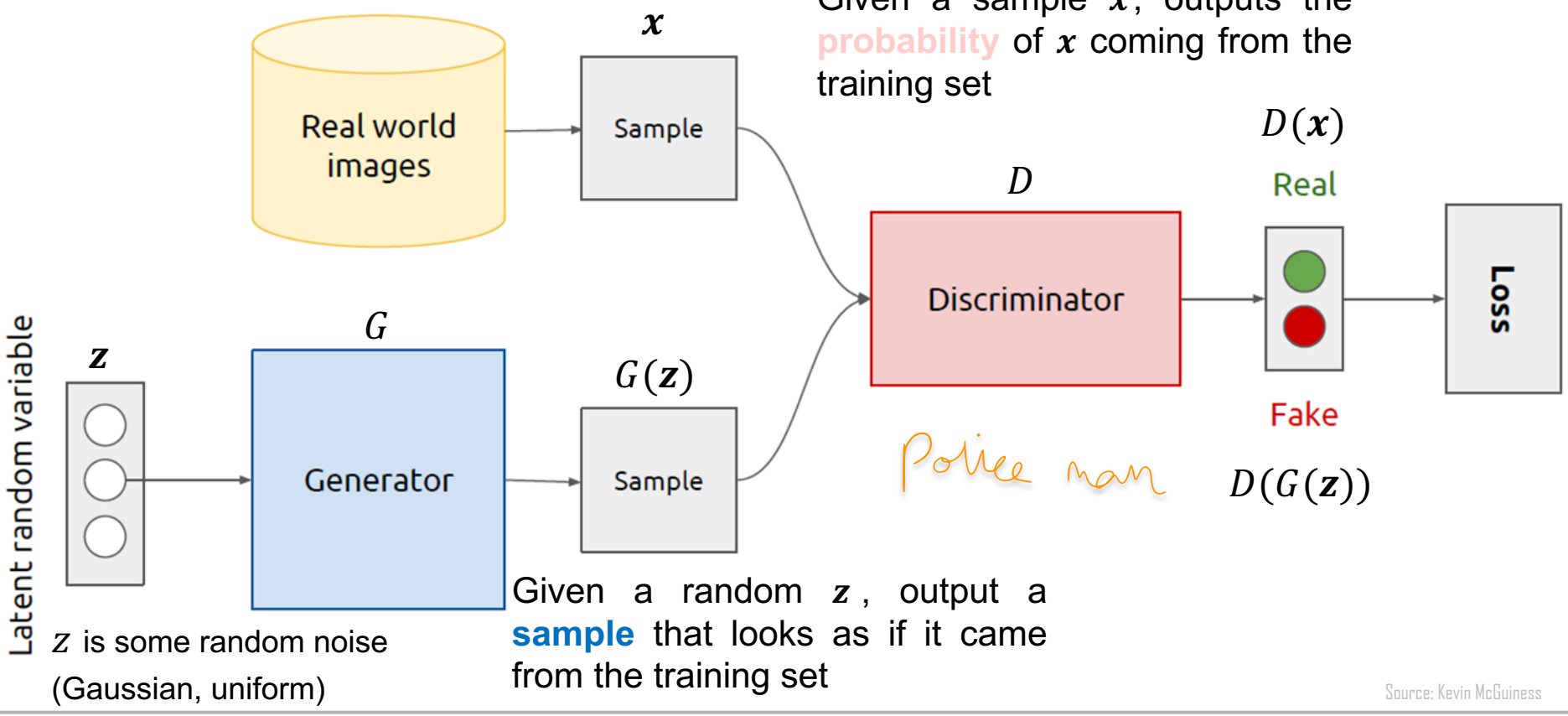
- Typically D is a Neural Network, but it doesn't have to be
- Note that the discriminator can also take the output of the generator as input.

GANs – Two Players (sub-models)



Source: Kevin McGuiness

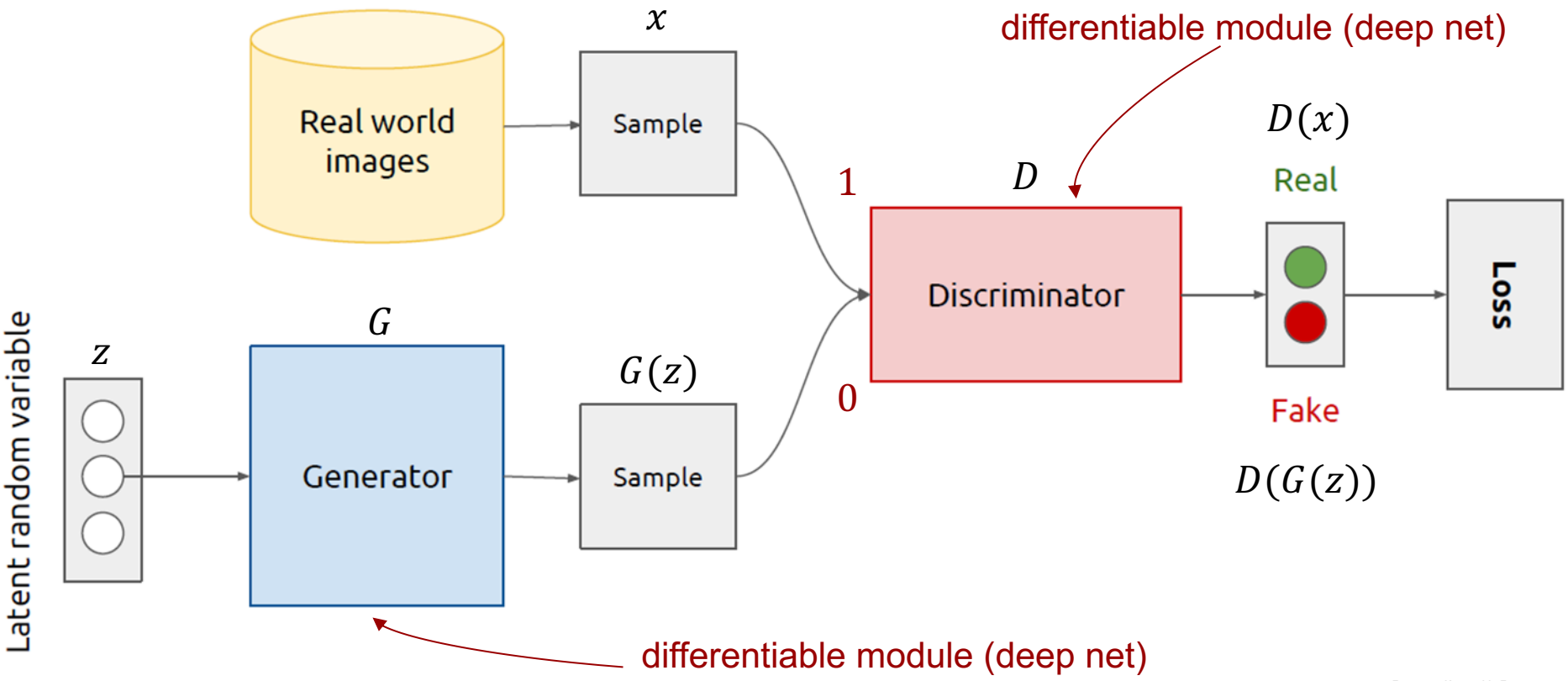
GANs Architecture



Source: Kevin McGuiness

Training GANs: An Alternating Procedure

Alternate between training the discriminator and generator

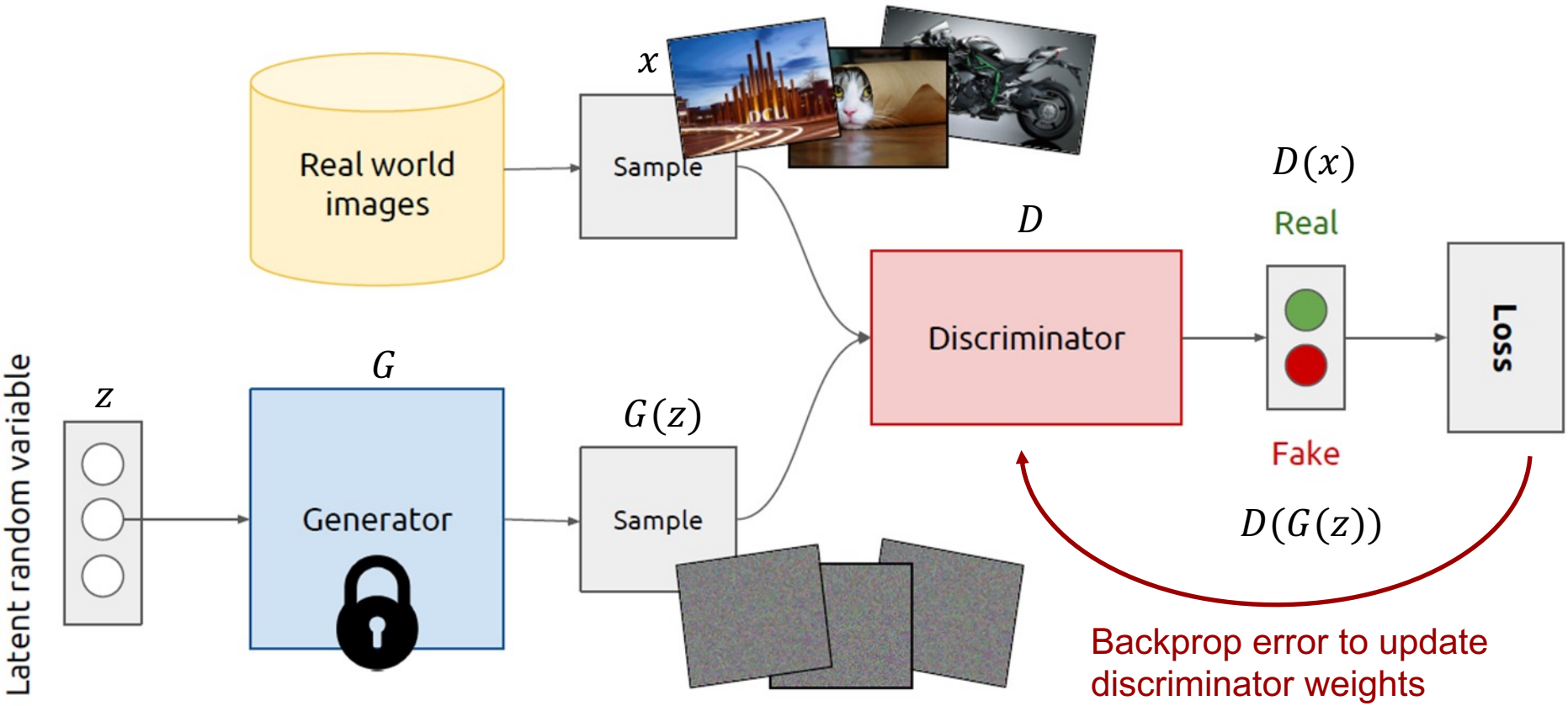


Source: Kevin McGuiness

EURECOM Sophia Antipolis

Training Discriminator

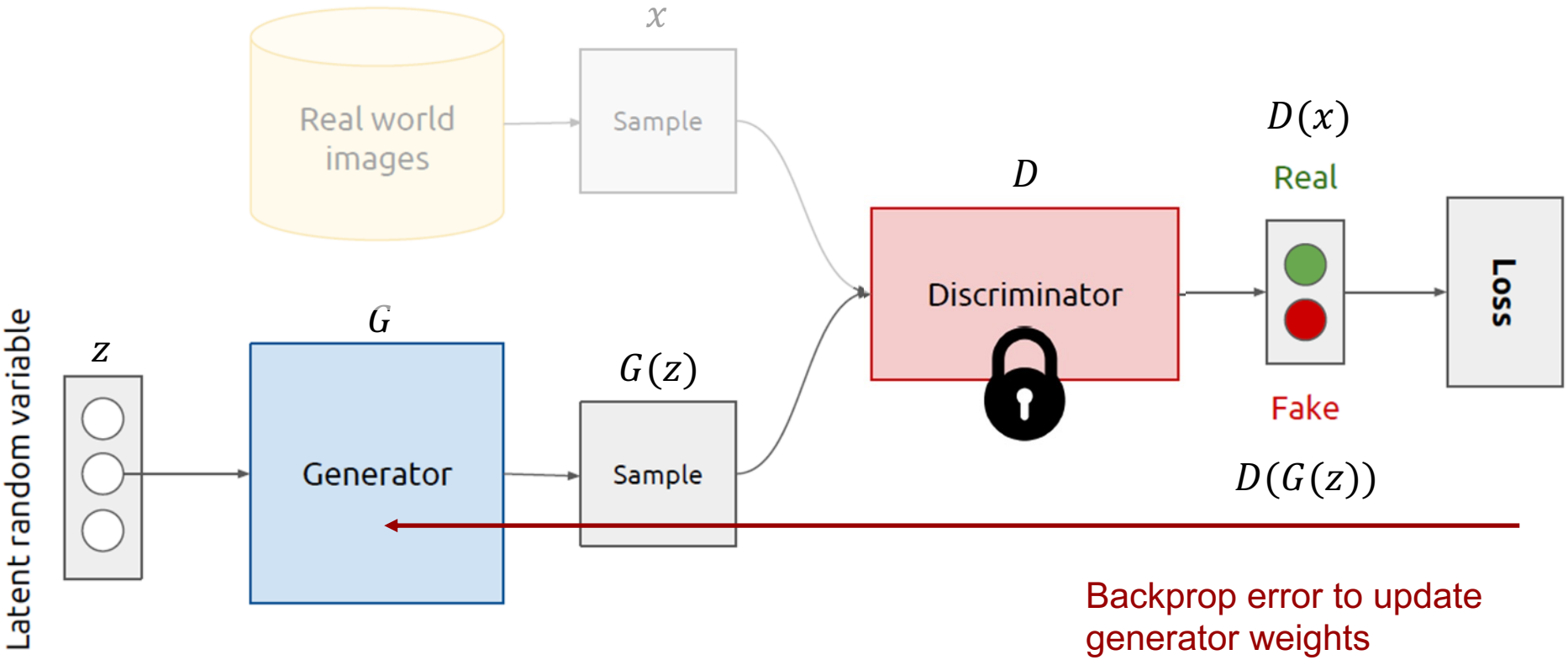
1. Fix generator weights, draw samples from both real world and generated images
2. Train discriminator to distinguish between real world and generated images



Source: Kevin McGuiness

Training Generator

1. Fix discriminator weights
2. Sample from generator
3. Backprop error through discriminator to update generator weights



Source: Kevin McGuiness

EURECOM Sophia Antipolis

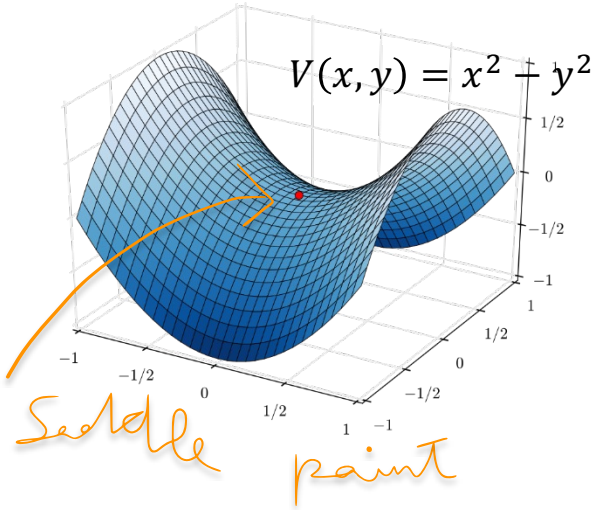
GANs – Game Theoretic Perspective

- The generator and discriminator can be thought of as two adversaries with opposing goals
- Generator (Discriminator) controls only its parameters
- Each seeks to maximize its own success and minimize the success of the other
- Game-theoretic approach: minimax game between *G* and *D*
- Zero-sum game: one player's gain is equivalent to another's loss
  - the net change in benefit is zero.
  - When somebody wins in the game, another person loses the same amount ⇒ the winnings minus the losses equal zero.
- In two-player zero-sum games, the *minimax* solution is the same as the Nash equilibrium.
- Nash equilibrium is a local optimum in a game

Zero-sum Game and Minimax Theorem

J. von Neumann's Minimax Theorem

- Let $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$ be compact convex sets. If $V: X \times Y \to \mathbb{R}$ is a continuous function that is concave-convex, i.e.,
  - $V(\cdot, y): X \to \mathbb{R}$ is concave for fixed y, and
  - $V(x, \cdot): Y \to \mathbb{R}$ is convex for fixed x.



- Then:

$$\max_{x \in X} \min_{y \in Y} V(x, y) = \min_{y \in Y} \max_{x \in X} V(x, y)$$

Minimax theorem is equivalent to:

- For every two-player zero-sum game with finitely many strategies, there exists a value V^* and a mixed strategy for each player, such that
  a) Given player 2's strategy, the best payoff possible for player 1 is V^*, and
  b) Given player 1's strategy, the best payoff possible for player 2 is $-V^*$

- Equivalently, Player 1's strategy guarantees them a payoff of V^* regardless of Player 2's strategy, and similarly Player 2 can guarantee themselves a payoff of $-V^*$

Source: https://www.theoremoftheday.org/OR/VonNeumann/TotDVonNeumann.pdf

The Minimax Game in GANs

- Loss function (or Value function):

$$\min_G \max_D V(G,D)$$

Value

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

- What is the max value $V(G,D)$ can take? 0
- What is the min value $V(G,D)$ can take? $-\infty$

- D wants to maximize the objective function (why?)

$$\max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

$$[D(x) = 1] \text{ AND } [D(G(z)) = 0] \Rightarrow V(G,D) = 0$$

training set — generated sample

Minimax Game in GANs (cont'd)

- Loss function (or Value function)

$$\min_G \max_D V(G,D)$$

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

- What is the max value $V(G,D)$ can take? 0
- What is the min value $V(G,D)$ can take? $-\infty$

- G wants to minimize the objective function (why?)

$$\min_G \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1-D(G(z)))]$$

$$D(G(z)) = 1 \Rightarrow V(G,D) = -\infty$$

↓

generated sample

Discriminator Loss Function

- Let D denote the discriminator's predicted probability of being data
- Discriminator's cost function: cross-entropy loss for task of classifying real vs. fake images

$$J_D = \mathbb{E}_{x\sim p_{\text{data}}}[-\log D(x)] + \mathbb{E}_{z\sim p(z)}[-\log(1 - D(G(z)))]$$

Real data (positive classification) Fake data (negative classification)

$$\left(\text{ or } J_D = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}[-\log D(x)] + \frac{1}{2}\mathbb{E}_{z\sim p(z)}\left[-\log\left(1 - D\left(G(z)\right)\right)\right] \right)$$

- Discriminator is trying to maximize its reward (or minimize its loss)
- Binary cross entropy (almost)
- Differs from *cross entropy* only in what we take the expectation over
- Supervised loss on data (because it infers if a sample has generated fake or real data) with no labels

Generator Loss Function

- Generator is trying to minimize Discriminator's reward (or maximize its loss)
- One possible cost function for the Generator: the opposite of the Discriminator's

Seed →

$$J_G = -J_D$$
$$= \text{const} + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

- Generator minimizes the log-probability of the discriminator being correct
- Resembles Jensen-Shannon divergence (*this will become evident later*)

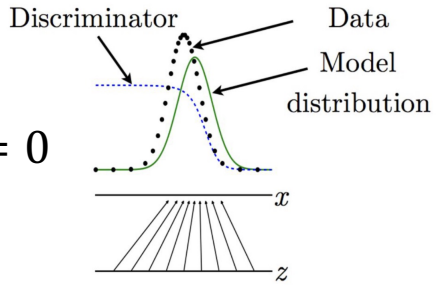
- Value function describing a zero sum game:

$$\min_G \max_D V(G, D) = \min_G \max_D -J_D$$

- Nash equilibrium (saddle point of discriminator's loss) of this game is achieved at:

$$p_{\text{data}}(x) = p_{\text{g}}(x)$$
$$D(x) = \frac{1}{2}, \forall x$$

$$\frac{\partial J_D}{\partial D(x)} = 0$$

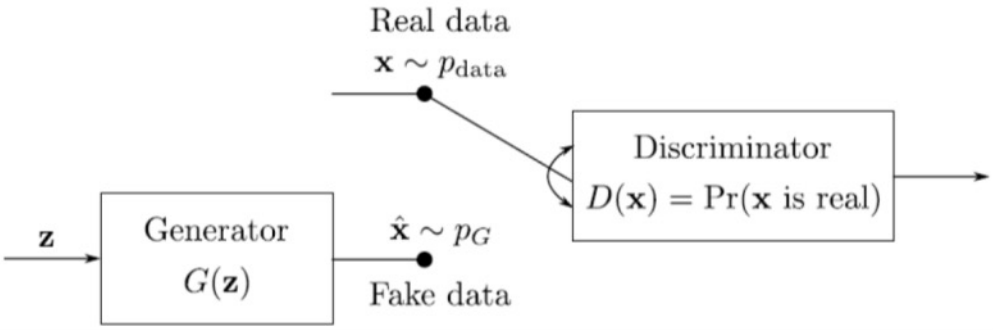


EURECOM
Sophia Antipolis

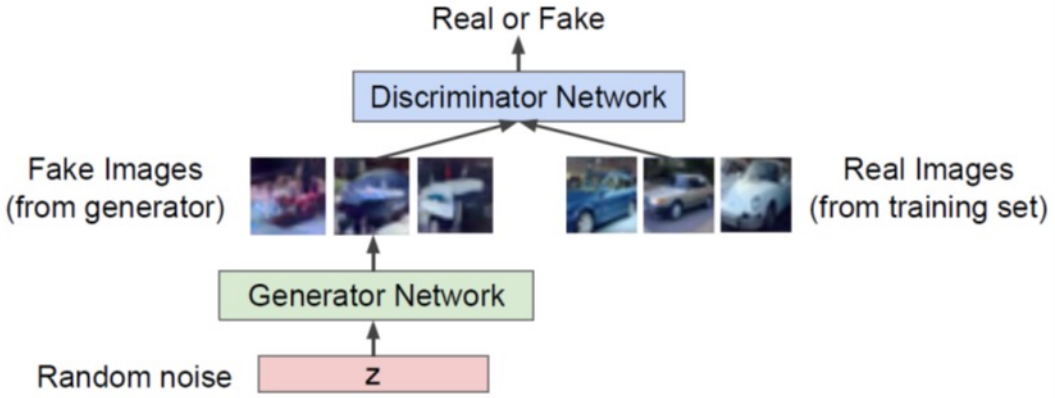
GANs – Recap

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

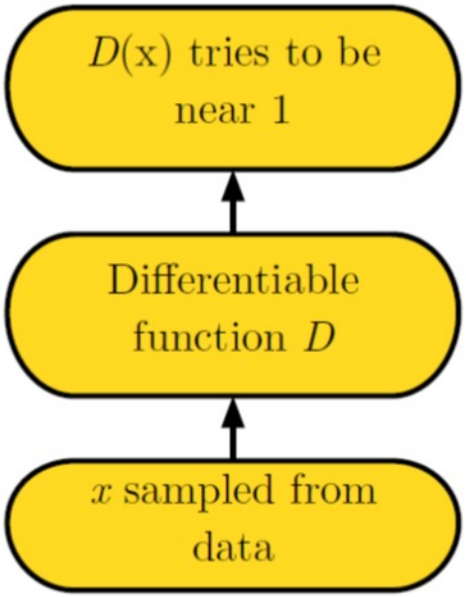
- Generator network $G(z)$:
  - Tries to maximize the *log-probability* of its samples being classified as "fake" by the discriminator (D)

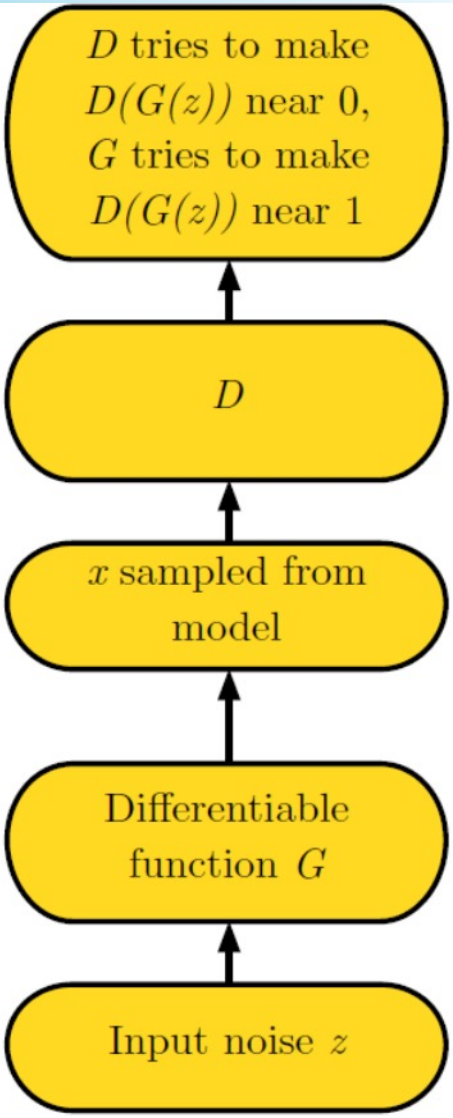


- Discriminator network $D(x)$:
  - Maximize the *log-likelihood* for the binary classification problem
    - Data: real (1)
    - Generated: fake (0)



Adversarial Nets Framework





I. Goodfellow NIPS2016

GANs – Parameterized Optimization Model

Generator/Discriminator are DNNs trained jointly

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log\left(1 - D_{\theta_d}(G_{\theta_g}(z))\right) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data $G(z)$

- Generator (θ_g) wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)
- $G = G_{\theta_g}: \mathbb{R}^N \mapsto \mathbb{R}^n$ is a neural network
- $\hat{x} = G_{\theta_g}(\boldsymbol{z}) \sim p_g$, for some prior distribution on the noise $\boldsymbol{z} \sim p_{\boldsymbol{z}}$
- Typical choice $p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}, \sigma^2 \boldsymbol{I})$

- Discriminator (θ_d) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- $D = D_{\theta_d}: \mathbb{R}^n \mapsto [0,1]$ is another neural network
- D_{θ_d} is a binary classifier with sigmoid output activation
- $D_{\theta_d} = \mathbb{P}(\boldsymbol{x} \text{ is generated by } p_{\text{data}})$

Vanilla GAN Training Algorithm

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.

Optional… Some find $k = 1$ more stable, others use $k > 1$, no best rule.

Recent work (e.g. Wasserstein GAN) alleviates this problem, better stability!

I. Goodfellow et al. 2014

GANs: Bayes-Optimal Discriminator (fixed G)

- What's the optimal discriminator $D(x)$ for any generated $p_g(x)$ and true $p_{data}(x)$ distributions?

$$
\begin{aligned}
V(G,D) &= \mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1-D(G(z)))\right] \\
&= \int_x p_{\text{data}}(x)\log D(x)dx + \int_z p(z)\log(1-D(G(z)))dz \\
&= \int_x p_{\text{data}}(x)\log D(x)dx + \int_x p_g(x)\log(1-D(x))dx \\
&= \int_x \left[p_{\text{data}}(x)\log D(x) + p_g(x)\log(1-D(x))\right]dx
\end{aligned}
$$

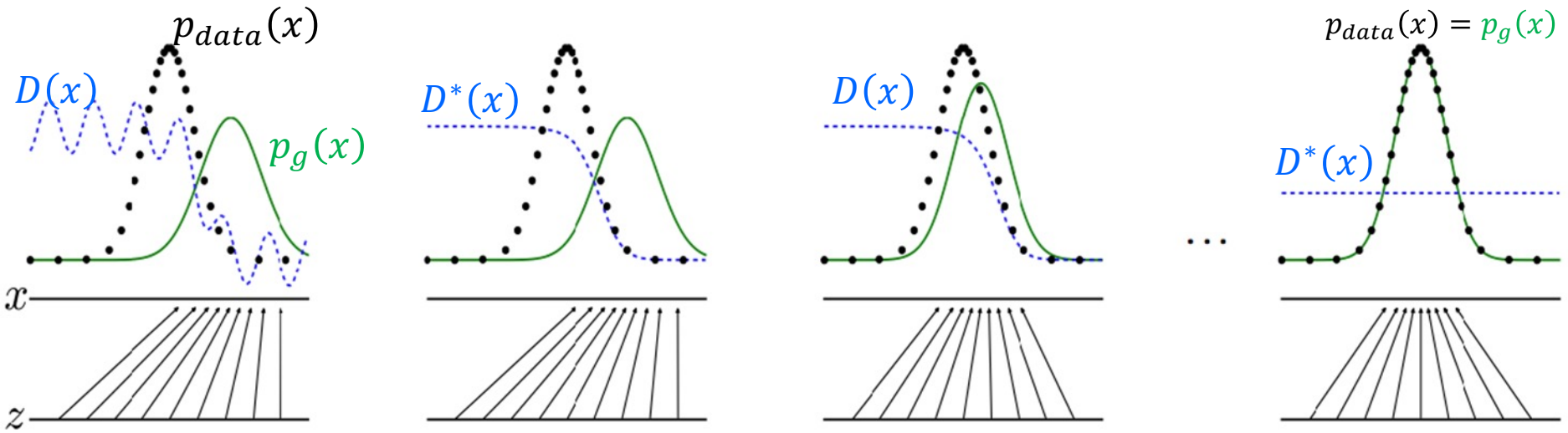
$$
\nabla_y \left[a\log y + b\log(1-y)\right] = 0 \implies y^* = \frac{a}{a+b} \quad \forall \quad [a,b]\in\mathbb{R}^2\backslash[0,0]
$$

$$
\frac{\partial V(G,D)}{\partial D(x)} = 0 \implies D^*(x) = \frac{p_{\text{data}}(x)}{\left(p_{\text{data}}(x)+p_g(x)\right)}
$$

We assume that $p_{data}(x)$ and $p_g(x)$ are non-zero everywhere

Training Process and Optimal Discriminator

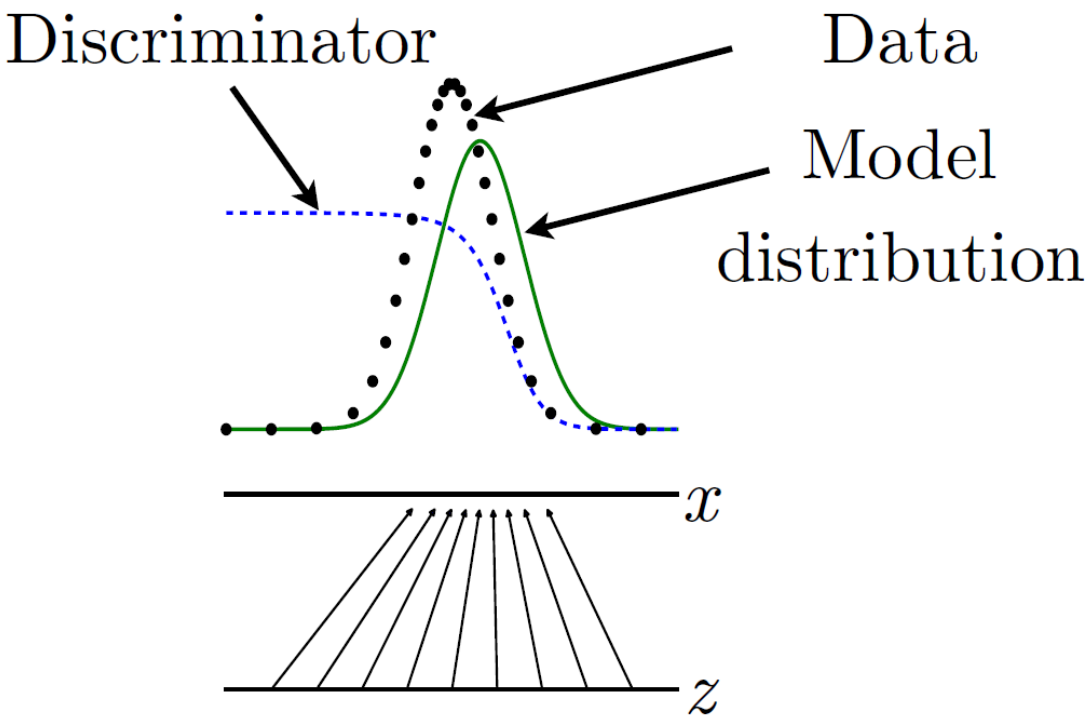
- Iterate the two steps until convergence (which may not happen!)
  - Updating the Discriminator should make it better at discriminating between real images and generated ones (Discriminator improves)
  - Updating the Generator makes it better at fooling the current Discriminator (Generator improves)
- Eventually (hope) the Generator gets so good that it is impossible for the Discriminator to tell the difference between real and generated images (discriminator accuracy = 0.5)
- Best strategy for the Discriminator: learn the ratio of the probabilities of x under the data distribution and the generator distribution: $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)} = p_{data}(data|x)$



GANs: Bayes-Optimal Discriminator

- Estimating this ratio using supervised learning is the key approximation mechanism used by GANs

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$



Mapping from z to x (**black arrows**) is *non-uniform* so that $p_{model}(x)$ (**green** curve) is greater in places where z values are brought together more densely.

Discriminator (**dashed blue** line) estimates the ratio between the data density (**black dots**) and the sum of the data and model densities.

Wherever the output of the Discriminator is large, the model density is too low
Wherever the output of the Discriminator is small, the model density is too high.

Generator can learn to produce a better model density by following the Discriminator uphill; each $G(z)$ value should move slightly in the direction that increases $D(G(z))$.

Optimal Generator (under fixed optimal D*)

Generator Objective under Bayes-Optimal Discriminator $D^*(x)$

$$
\begin{aligned}
V(G, D^*) &= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log D^*(x) \right] + \mathbb{E}_{x \sim p_g} \left[\log (1 - D^*(x)) \right] \\
&= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \right] \\
&= -\log(4) + \underbrace{KL\left(p_{\text{data}} \middle\| \left(\frac{p_{\text{data}} + p_g}{2} \right) \right) + KL\left(p_g \middle\| \left(\frac{p_{\text{data}} + p_g}{2} \right) \right)}_{\text{(Jensen-Shannon Divergence (JSD) of } p_{\text{data}} \text{ and } p_g) \geq 0}
\end{aligned}
$$

$$
V(G^*, D^*) = -\log(4) \text{ when } p_g = p_{\text{data}}
$$

- Global optimum for this game is achieved *if and only if* $p_{data}(x) = p_g(x)$
- We are, in theory, minimizing the **Jensen-Shannon divergence** between the Generator distribution and the true data distribution!

Gaps between Theory and Practice

- For models that have enough capacity if we use $J_G = -J_D$, and if D is set to its global optimum given G at every iteration and G improves the criterion at every iteration, then *alternating optimization* will get us to the global optimum

- In practice:
  - D,G may not have enough capacity
  - We do not get to find the global optimum for D at each iteration

- Theory assumes we can reach a global optimum
  - we have functions which are ***non convex*** in the parameters

- Theory assumes that $p_{data}(x)$ and $p_g(x)$ are non zero everywhere.
  - This may not hold especially if we have data lying on a manifold.
  - Even when it holds the ratio can be numerically unstable

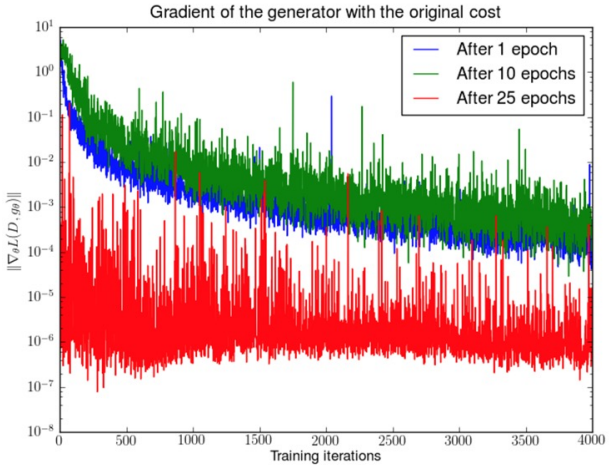
- Theory assumes that the optimal discriminator is unique.
  - In practice other discriminators can do nearly as good a job: i.e. the discriminator can overfit the data

Vanishing Gradient Problem with Generator

$$\min_G \max_D V(G,D)$$

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

$$\nabla_{\theta_G} V(G,D) = \nabla \mathbb{E}_{z\sim p(z)}[\log(1-D(G(z)))]$$



- If D is confident, gradient goes to 0, i.e., $D(G(z)) \to 0$
- There is no gradient signal to help the generator improve

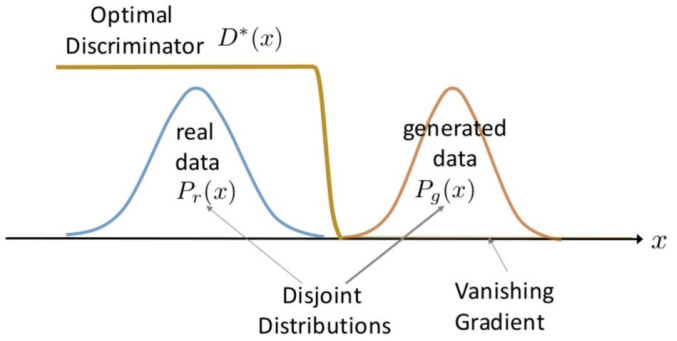
Solution (heuristic)

$$J_G = -\mathbb{E}_{z\sim p(z)}[\log(D(G(z)))]$$

- Generator maximizes the log probability of the discriminator's mistake

$$D\big(G(z)\big) \to 0 \Rightarrow -\mathbb{E}_{z\sim p(z)}[\log(D(G(z)))] \to \infty$$

- Gradient gets bigger when D gets better
- Shown to give the same stationary point (under some assumptions) as $J_G = -J_D$

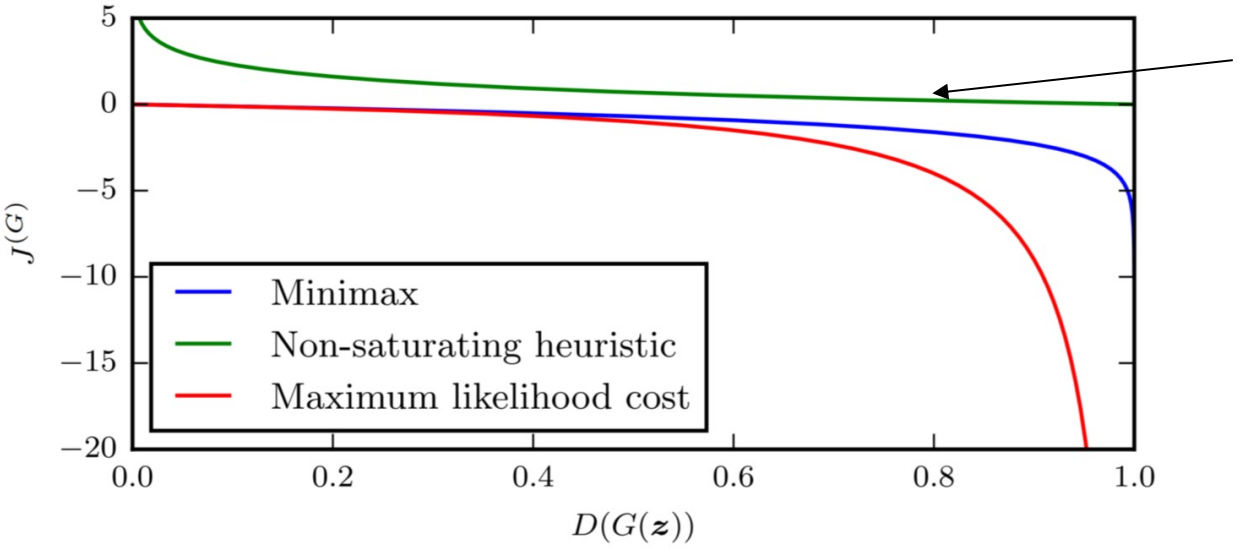


EURECOM Sophia Antipolis

Comparing Various Generator Loss Functions

- Generators cost is a function $D(G(z))$
  - Minimax: $J_G = -J_D$
  - Non-saturating heuristic: $J_G = -\mathbb{E}_{z\sim p(z)}[\log(D(G(z)))]$
  - Maximum Likelihood cost: $J_G = -\mathbb{E}_{z\sim p(z)}\exp(\sigma^{-1}(D(G(z)))$

ML Estimation: can be shown equivalent to minimizing KL divergence between $p_{data}(x)$ and $p_g(x)$ (under certain assumptions)



able to learn even if the gradient signal is low

- Other options in the loss
  - Energy-based GAN
  - f-GAN
  - Wasserstein GAN

Training GANs: Two Player Game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

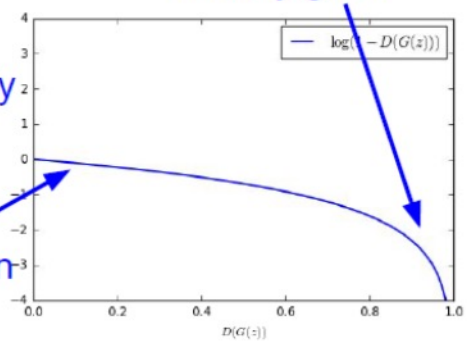
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



I. Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Training GANs: Two Player Game (Cont'd)

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

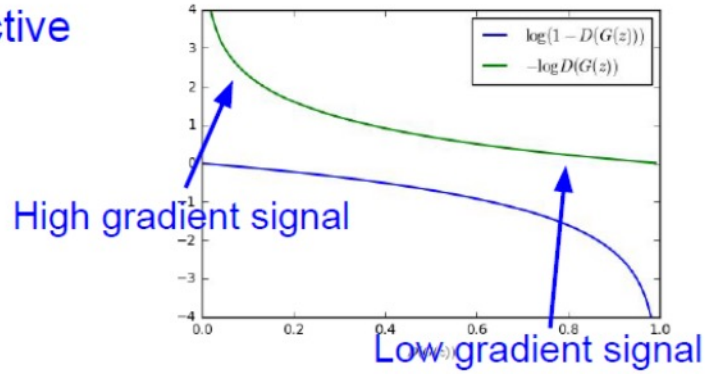
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

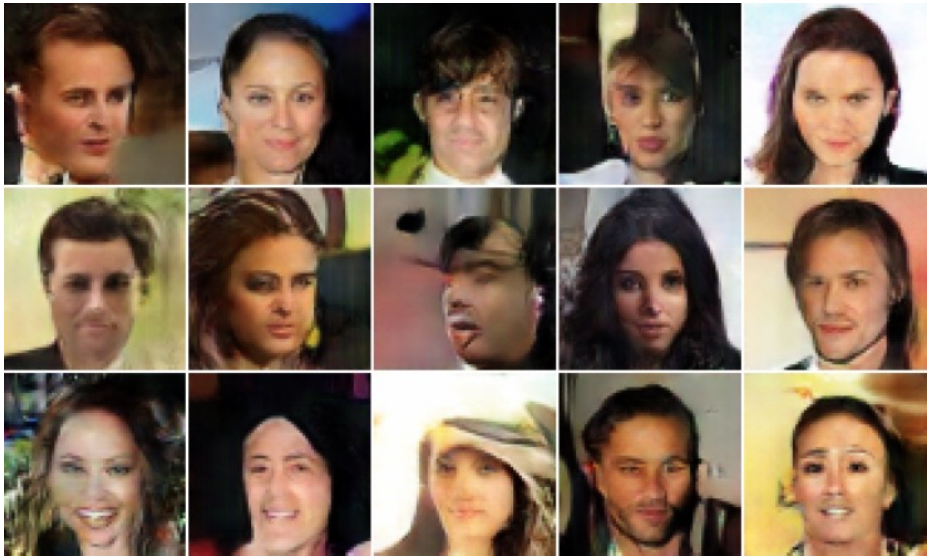


- Equilibrium is a saddle point of the discriminator loss
- Resembles Jensen-Shannon divergence
- Generator minimizes the log-probability of the discriminator being correct

I. Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

GAN samples look sharp

Real samples

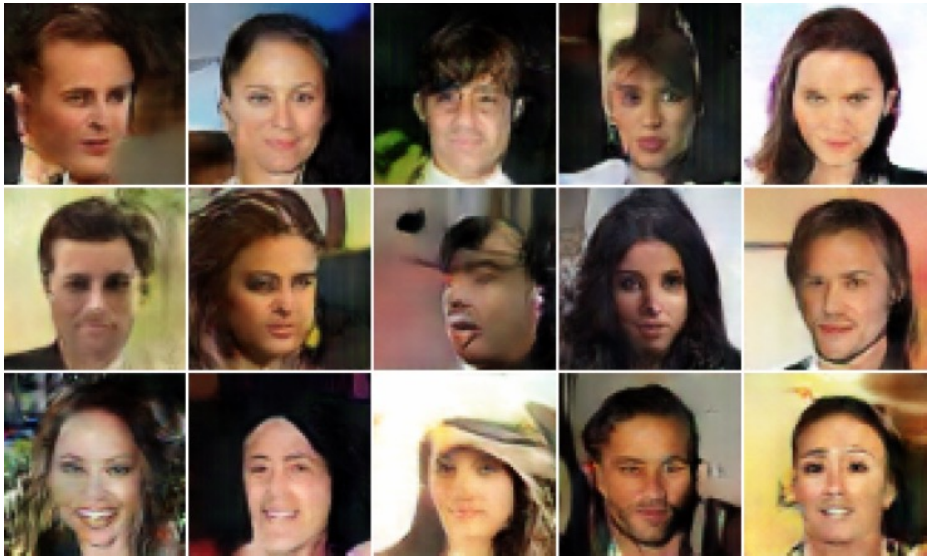


Fake samples

Berthelot et al.: BEGAN: Boundary Equilibrium GANs, arxiv.org, 2017

GAN samples look sharp

~~Real samples~~
Boundary Equilibrium GAN (BEGAN)



~~Fake samples~~
Energy based GAN

Berthelot et al.: BEGAN: Boundary Equilibrium GANs, arxiv.org, 2017

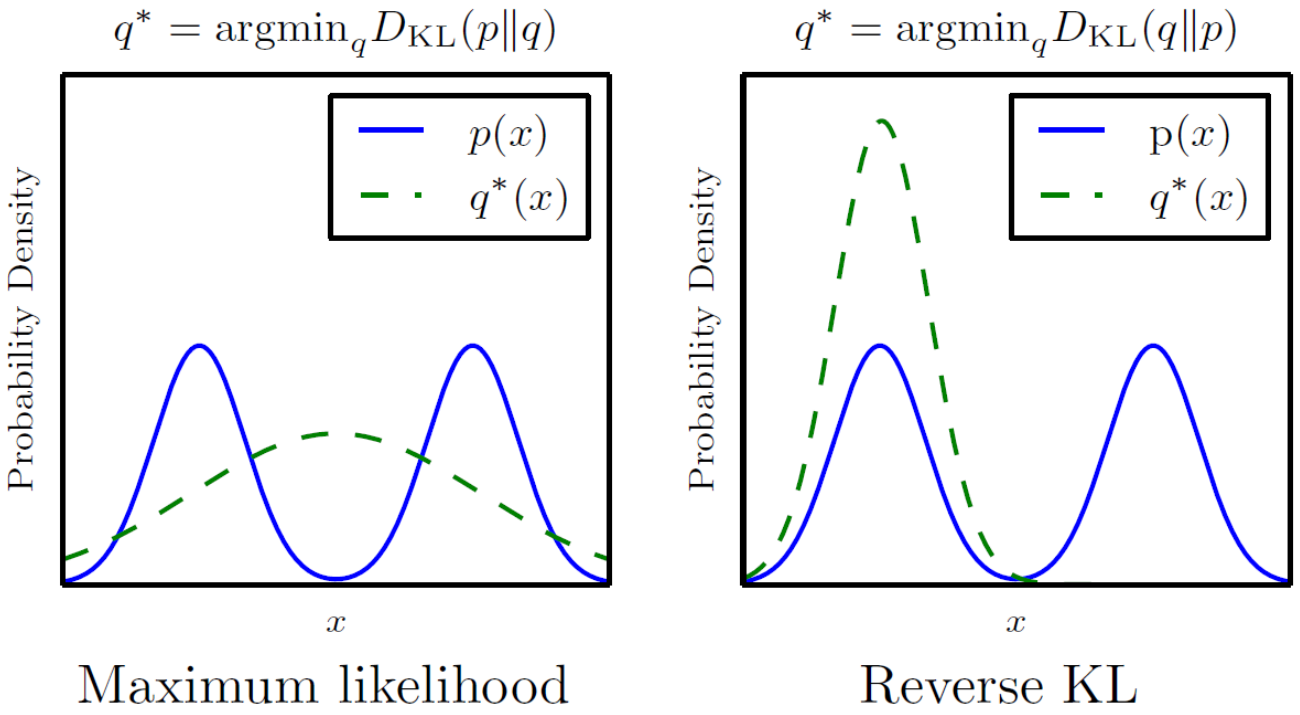
Why GANs work better than VAEs?

- GANs can be trained with ML and still generate sharp samples

KL divergence not symmetric minimizing
$D_{KL}\ (p_{data}\ ||\ p_{model})$

is different

from minimizing
$D_{KL}\ (p_{model}\ ||\ p_{data})$

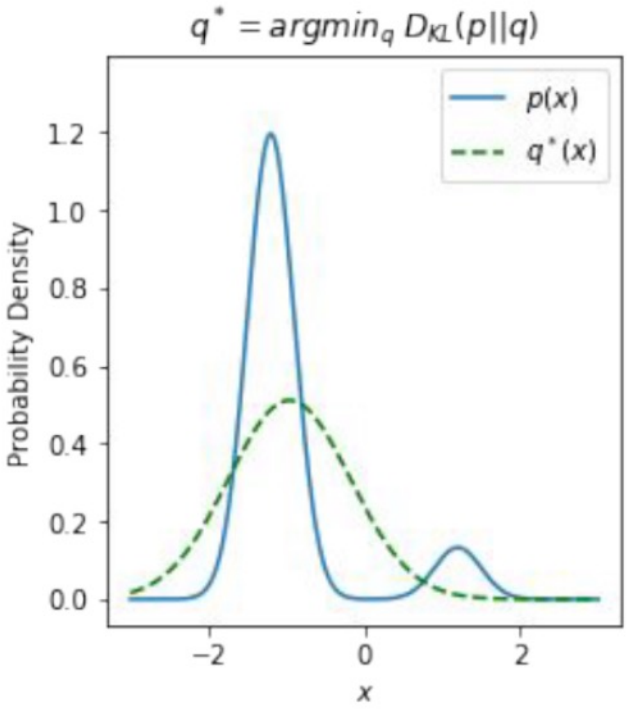


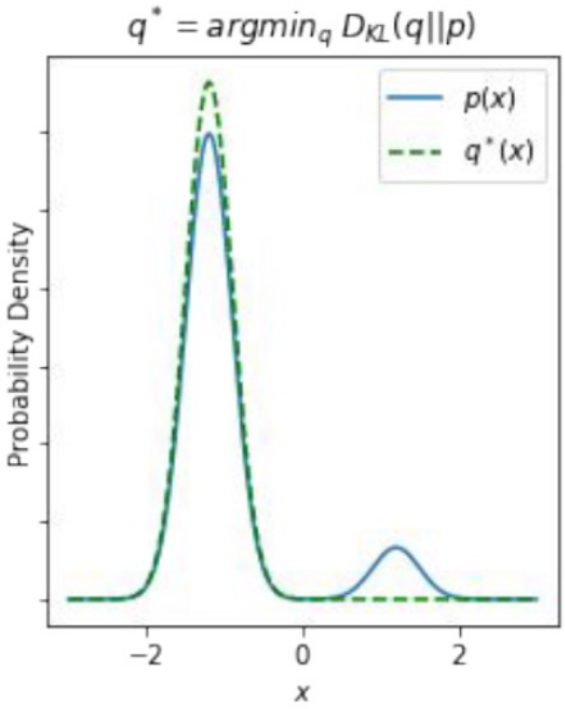
We use a mixture of two Gaussians for p (data distribution), and a single Gaussian for q (model family).
ML: model chooses to average out the two modes (places high probability on both of them)
Reverse KL: model chooses to capture only one of the two modes (both are local minima)

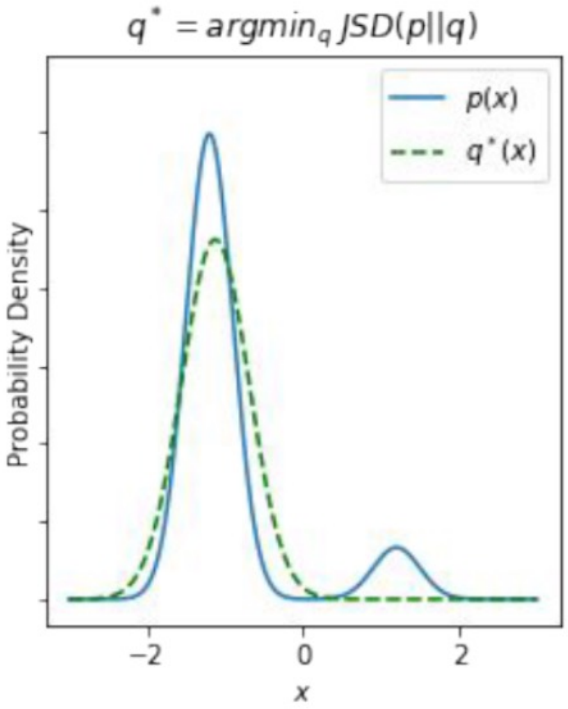
$D_{KL}\ (p_{data}\ ||\ p_{model})$: place high probability everywhere that the data occurs
$D_{KL}\ (p_{model}\ ||\ p_{data})$: prefers to place low probability wherever the data does not occur

GANs often choose to generate from very few modes - Reverse KL prefers to generate from as many modes of the data distribution as the model is able to; it does not prefer fewer modes in general.

KL and JSD







Training Problem: Non Convergence

- Deep Learning models (in general) involve a single player
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - Problem: With non-convexity, we might converge to local optima.

$$\min_{G} L(G)$$

- GANs instead involve two (or more) players
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.

$$\min_{G} \max_{D} V(G, D)$$

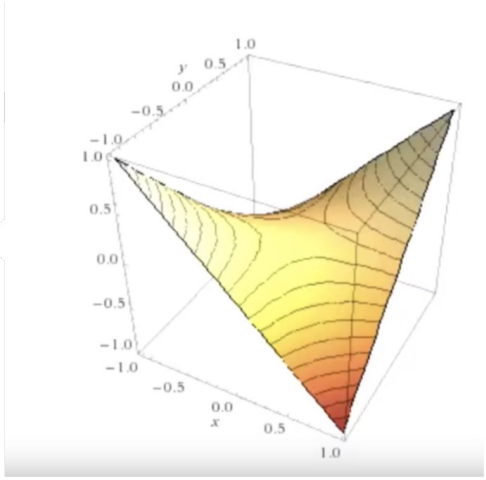
  - SGD was not designed to find the Nash equilibrium of a game.
  - Problem: We might not converge to the Nash equilibrium at all.

An Example of Non Convergence

$$\min_x \max_y V(x,y)$$

with $V(x,y) = xy$

• State 1:	x > 0	y > 0	V > 0	Increase y	Decrease x
• State 2:	x < 0	y > 0	V < 0	Decrease y	Decrease x
• State 3:	x < 0	y < 0	V > 0	Decrease y	Increase x
• State 4 :	x > 0	y < 0	V < 0	Increase y	Increase x
• State 5:	x > 0	y > 0	V > 0 == State 1	Increase y	Decrease x



$$V(x(t), y(t)) = x(t)y(t)$$

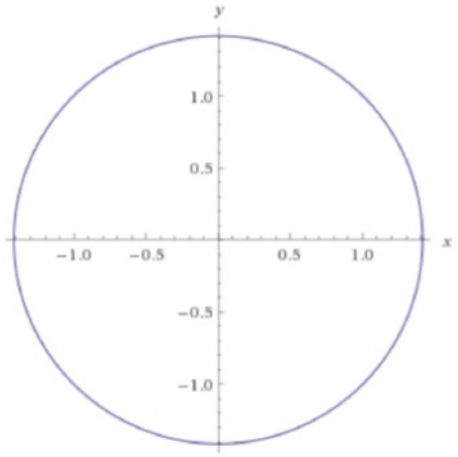
$$\frac{\partial x}{\partial t} = -y(t)$$

$$\frac{\partial y}{\partial t} = x(t)$$

$$\frac{\partial^2 y}{\partial t^2} = \frac{\partial x}{\partial t}. = -y(t)$$

$$x(t) = x(0)cost(t) - y(0)\sin(t)$$

$$y(t) = x(0)cost(t) - y(0)\sin(t)$$

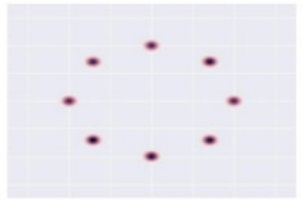


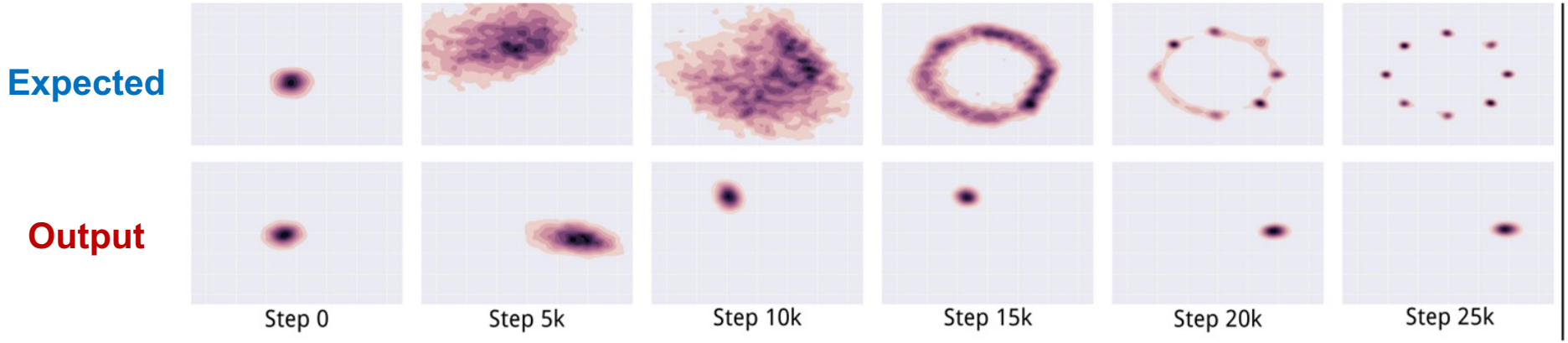
- D & G nullifies each others learning in every iteration
- Differential equation's solution has sinusoidal terms
- Even with a small learning rate, it will not converge

I. Goodfellow, NIPS 2016 Tutorial, arxiv.org, 2017

Training Problem: Mode Collapse (Helvetica)

- Generator fails to output diverse samples
- Generator produces good samples, but a very few of them.
- Thus, Discriminator can't tag them as fake.





Mode Collapse – Some Solutions

Mini-batch GANs - Reward sample complexity

- Let D look at the entire batch instead of single examples
- Lack of diversity, examples will be marked as fake ⇒ G forced to produce diverse samples.
  - Extract features that capture diversity in the mini-batch (e.g., L2 norm of the difference between all pairs from the batch)
  - Feed those features to the discriminator along with the image
  - Feature values will differ black and white diverse and non-diverse batches
  - Thus, D will rely on those features for classification

    ⇒ will force the G to match those feature values with the real data - will generate diverse

Supervision with labels

- Label information of the real data might help

- Unrolled GANs
- WGANs

Training GANs – In a Nutshell

Why GANs are hard to train?

- Generator keeps generating similar images – so nothing to learn
- Maintain tradeoff of generating more accurate vs high coverage samples
- The two learning tasks need to have balance to achieve stability
  - If Discriminator is not sufficiently trained – it can worse generator
  - If Discriminator is over-trained - will produce no gradients

Tricks to Train GAN

- One sided label smoothing
- Historical generated batches
- Feature Matching
- Mini-batch Discrimination
- Virtual Batch Normalization
- Regularizing discriminator gradient in region around real data (DRAGAN)
- …

GANs: Pros and Cons

Pros (Why GANs?)

- Sampling (or generation) is straightforward.
- Training does not involve ML estimation.
- Robust to Overfitting since Generator never sees the training data.
- Empirically, GANs are good at capturing the modes of the distribution
- Can be trained to draw realistic sharp samples! (best possible)

Cons

- Trickier / more unstable to train properly, they do not always converge
- Optimization can oscillate between solutions
- Generator can collapse
- Probability Distribution is Implicit: not straightforward to solve inference queries such as $p(x), p(z|x)$
  - Vanilla GANs are only good for Sampling/Generation.
- Training requires finding the Nash equilibrium of a game – a more difficult problem than optimizing an objective function

GAN Samples

GAN samples

- GANs revolutionized generative modeling by producing crisp, high-resolution images.
- The catch: we don't know how well they're modeling the distribution.
- Can't measure the log-likelihood they assign to held-out data.
- Could they be memorizing training examples? (E.g., maybe they sometimes produce photos of real celebrities?)
- We have no way to tell if they are dropping important modes from the distribution.

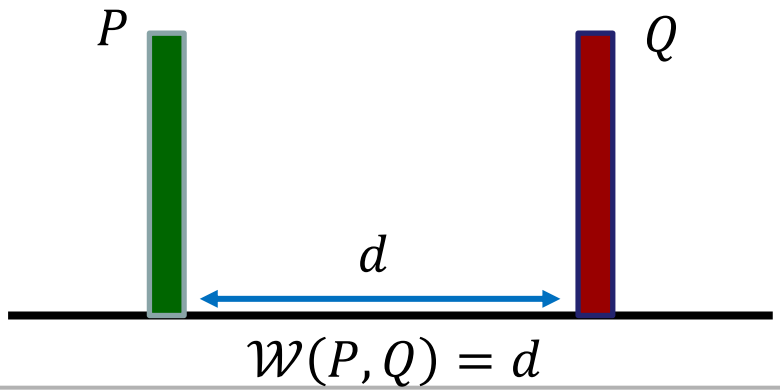
GAN Evaluation

- Another issue with GANs is quantitative comparison
- There is no explicit likelihood to calculate
- Post hoc density estimation can be used, but is inaccurate
- Subjective evaluation by humans currently the best method (Fréchet inception distance)

- Active area of research – various challenges:
  - How to evaluate GANs??
  - Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
  - Finding Nash equilibria in high-dimensional, continuous, nonconvex games

WGAN

- If data lies on a low dimensional manifold of a high dimensional space the model's manifold and the true data manifold can have a negligible intersection in practice
- KL divergence is undefined or infinite
- The loss function and gradients may not be continuous and well defined
- Wasserstein distance is well defined:
  - Minimum transportation cost for making one pile of dirt (pdf/pmf) look like the other
- Earth Mover's Distance (Wasserstein-1 metric):
  - Considering one distribution P as a pile of earth (total amount of earth is 1), and another distribution Q (another pile of earth) as the target
  - EMD is the average distance the earth mover has to move the earth in an optimal plan.

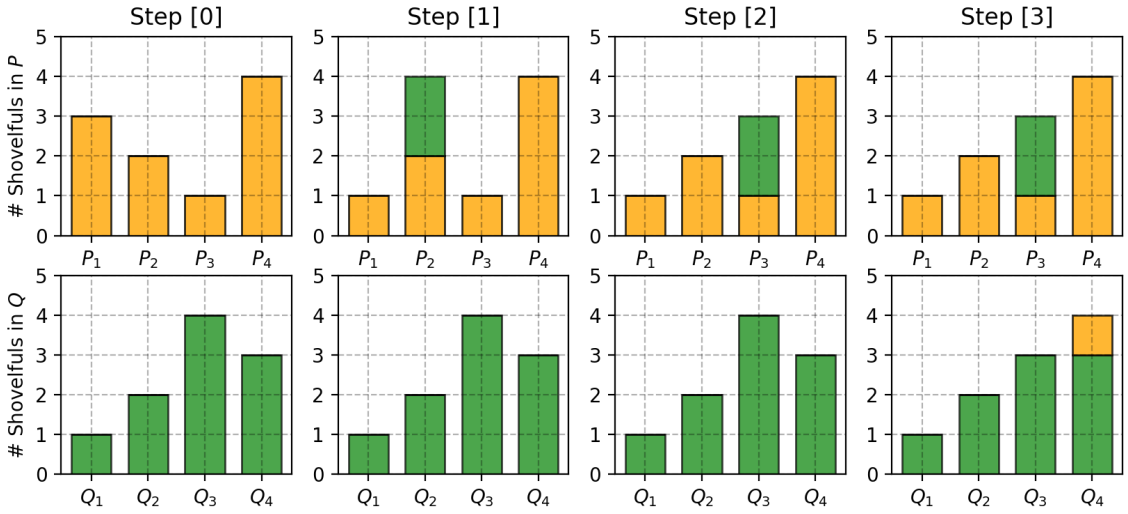


Wasserstein Distance

- Discrete case: suppose we have two distributions P and Q, each has four piles of dirt and both have ten shovelfuls of dirt in total.
- The numbers of shovelfuls in each dirt pile are assigned as follows:

$$P1 = 3, P2 = 2, P3 = 1, P4 = 4$$
$$Q1 = 1, Q2 = 2, Q3 = 4, Q4 = 3$$

- First move 2 shovelfuls from P_1 to P_2 => (P_1, Q_1) match up.
- Then move 2 shovelfuls from P_2 to P_3 => (P_2, Q_2) match up.
- Finally move 1 shovelfuls from Q_3 to Q_4 => (P_3, Q_3) and (P_4, Q_4) match up.



- Label the cost to pay to make P_i and Q_i match as δ_i:

$$\delta_{i+1} = \delta_i + P_i - Q_i$$

$$\delta_0 = 0$$
$$\delta_1 = 0 + 3 - 1 = 2$$
$$\delta_2 = 2 + 2 - 2 = 2$$
$$\delta_3 = 2 + 1 - 4 = -1$$
$$\delta_4 = -1 + 4 - 3 = 0$$

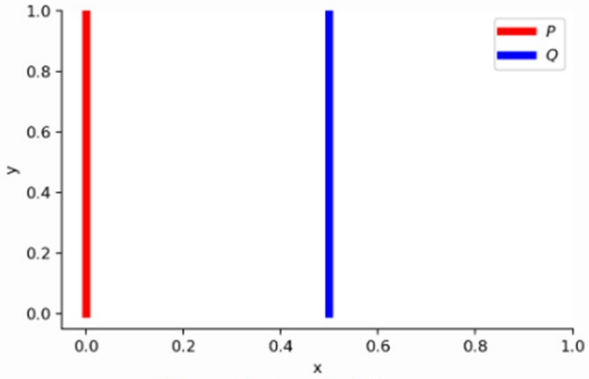
- So the Wasserstein distance is

$$W = \sum |\delta_i| = 5.$$

Wasserstein Distance vs. KL vs. JSD

Suppose we have two probability distributions, P and Q:

$$\forall (x, y) \in P, x = 0 \text{ and } y \sim U(0, 1)$$
$$\forall (x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1)$$



There is no overlap between P and Q when $\theta \neq 0$.

When $\theta \neq 0$:

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

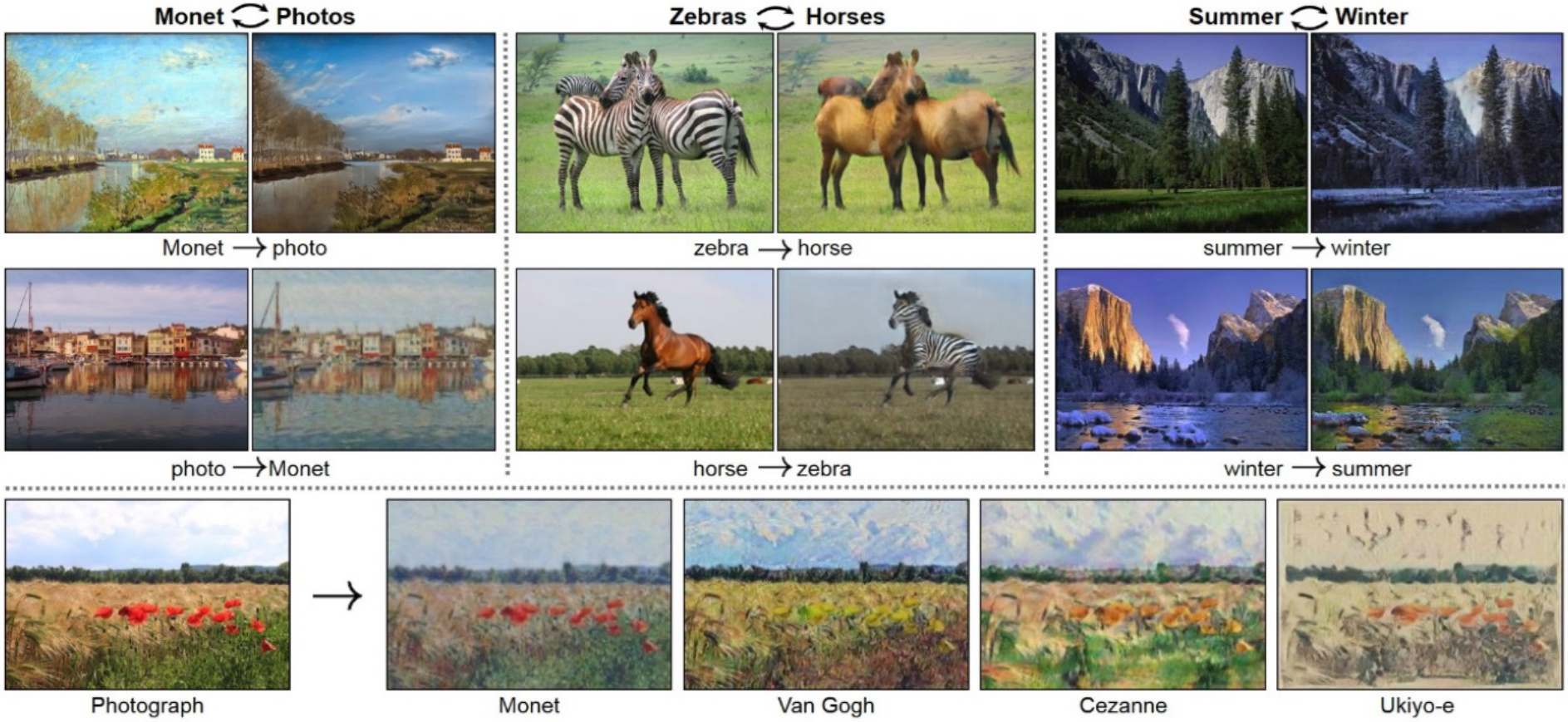
But when $\theta = 0$, two distributions are fully overlapped:

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

D_{KL} gives us inifity when two distributions are disjoint. The value of D_{JS} has sudden jump, not differentiable at $\theta = 0$. Only Wasserstein metric provides a smooth measure, which is super helpful for a stable learning process using gradient descents.

CycleGAN

- Style transfer problem: change the style of an image while preserving the content.



- Data: Two unrelated collections of images, one for each style

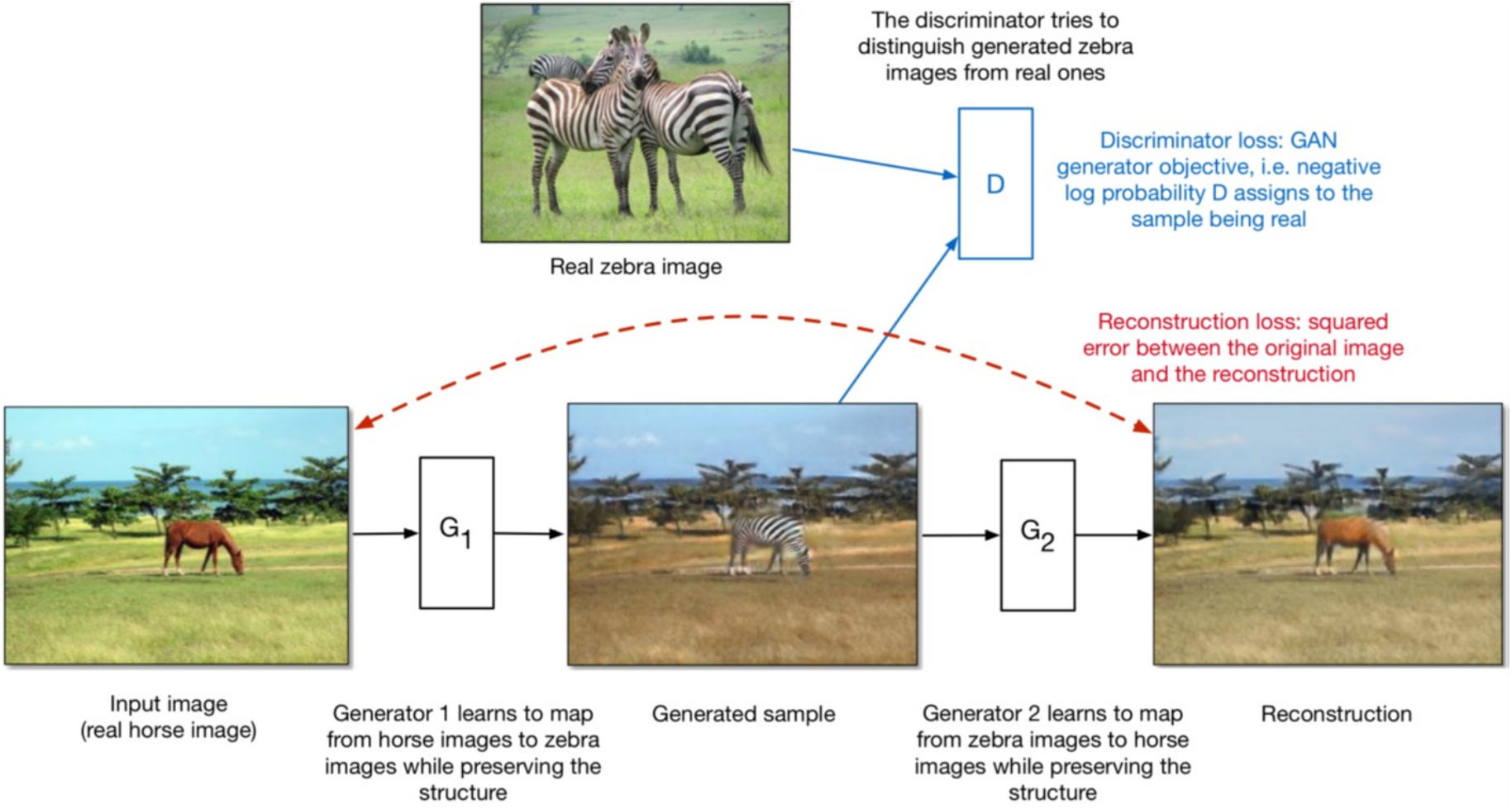
CycleGAN

- If data was paired (same content in both styles), this would be a supervised learning problem. But this is hard to find.

- The CycleGAN architecture learns to do it from unpaired data.

- Train two different generator nets to go from style 1 to style 2, and vice versa.
- Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
- Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.

CycleGAN



Total loss = discriminator loss + reconstruction loss