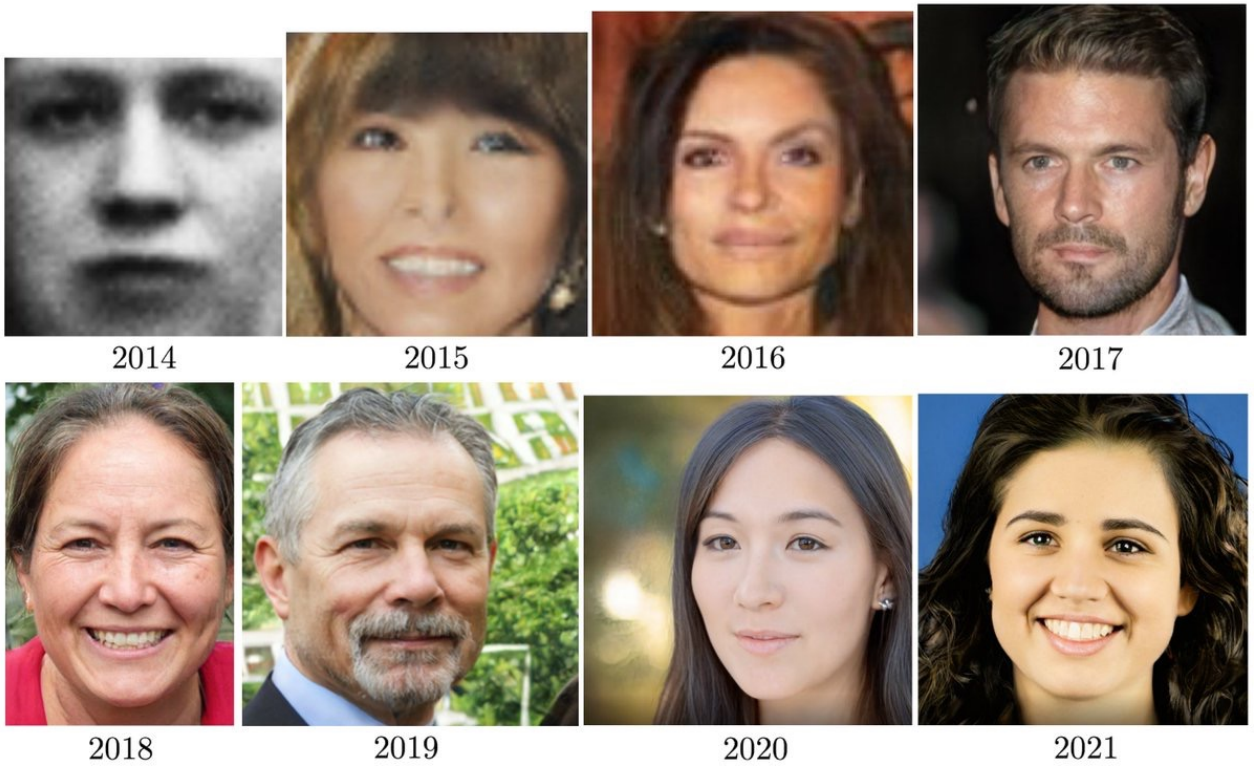
3. Generative Adversarial Networks

M. Ravasi

AI Summer School @ KAUST

GANs early success



Source: https://twitter.com/tamaybes/status/1450873331054383104

GANs early success

Ian Goodfellow

University of Montreal (2014)
DeepMind (now)

One night in 2014, Ian Goodfellow went drinking to celebrate with a fellow doctoral student who had just graduated. At Les 3 Brasseurs (The Three Brewers), a favorite Montreal watering hole, some friends asked for his help with a thorny project they were working on: a computer that could create photos by itself.

But as he pondered the problem over his beer, he hit on an idea. What if you pitted two neural networks against each other? His friends were skeptical, so once he got home, where his girlfriend was already fast asleep, he decided to give it a try. Goodfellow coded into the early hours and then tested his software. It worked the first time.

Source: https://www.technologyreview.com/2018/02/21/145289/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/

GANs early success

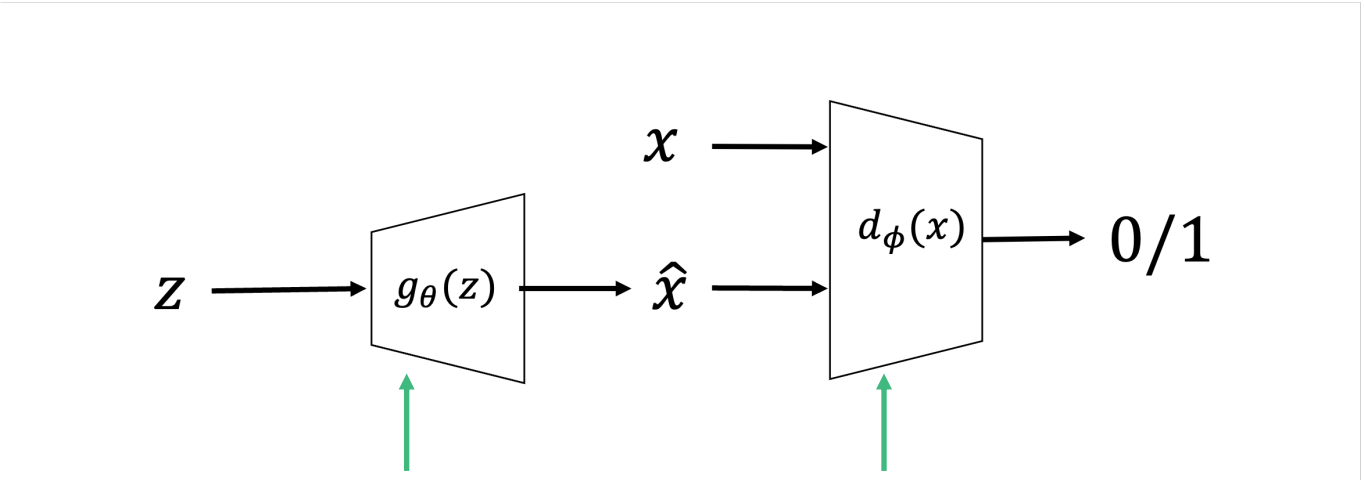
Bedroom dataset

GANs early success

Anime dataset

GANs

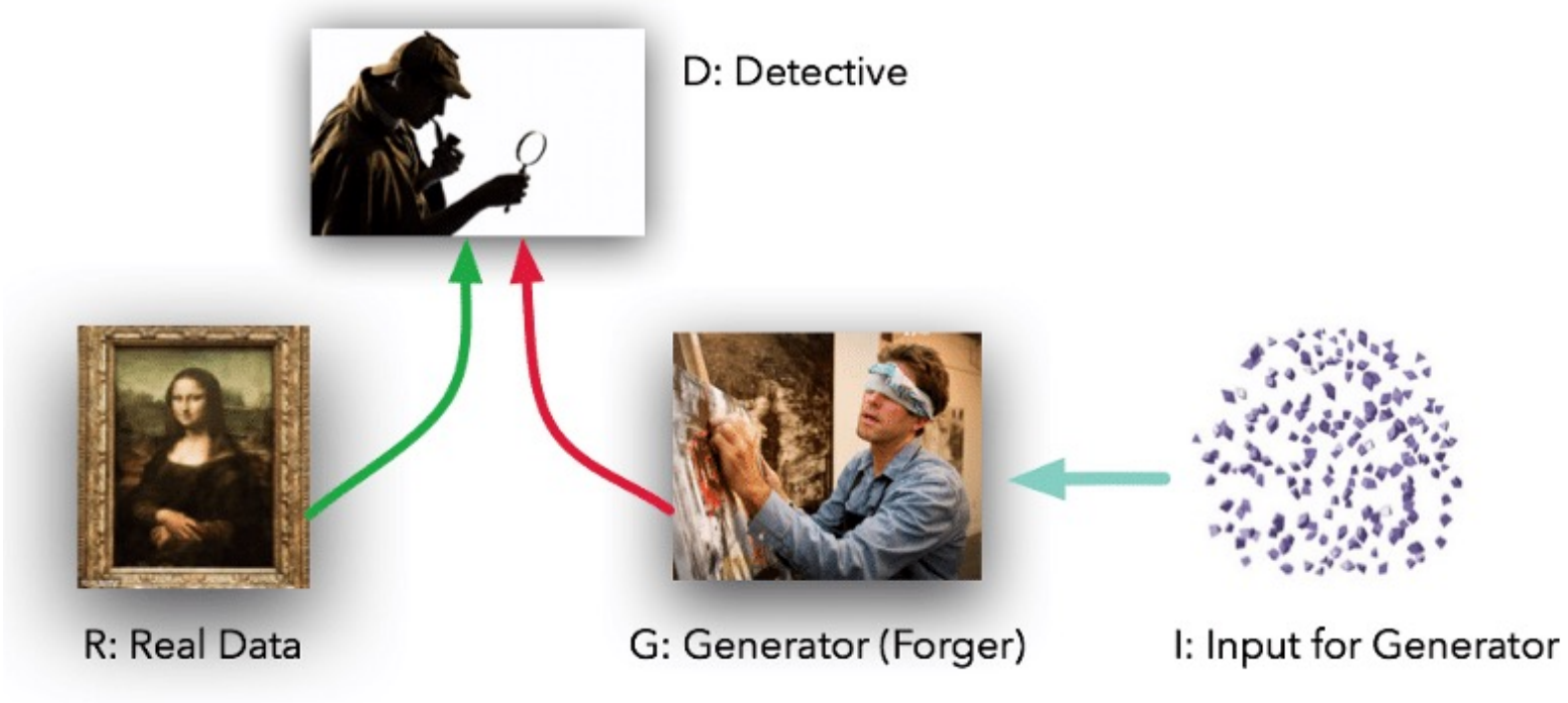
Family of implicit generative models (do not explicitly estimate a density function), which work by having two neural networks competing with each other – ***adversarial training*** *(or 2-player min-max game)*



Generator: create a sample of the distribution of the training data from a noise realization z.

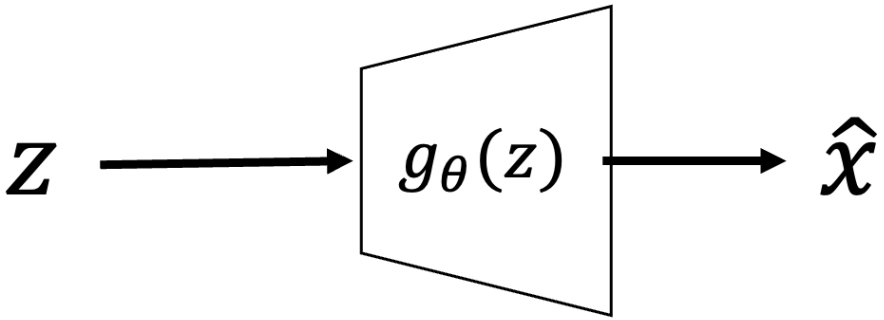
Discriminator: simple classifier tasked to tell if input is true or generated

GANs



Source: https://www.novatec-gmbh.de/en/blog/gans-applied-on-novatec-employees/

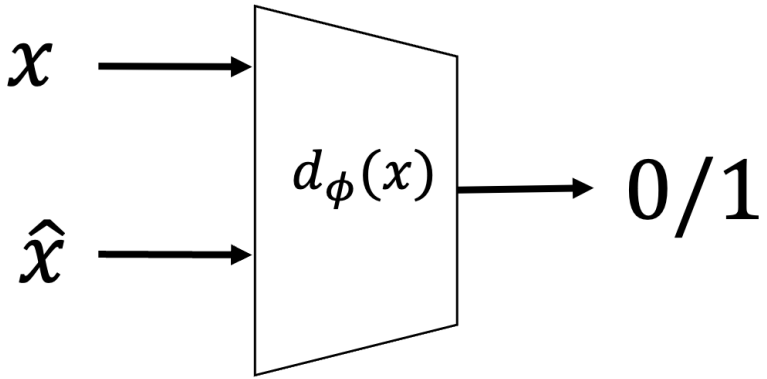
GANs: Generator



- $\boldsymbol{z} \in \mathbb{R}^{n_z}$ (with each element sampled from $\mathcal{N} \sim (0,1)$)
- $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_x}$ created by feeding the vector $\boldsymbol{z}$ into the generator network
- $\boldsymbol{z}$ and $\hat{\boldsymbol{x}}$ can be 1-dimensional, 2-dimensional, or more. Sometimes $\boldsymbol{z}$ can be of the same dimension of $\hat{\boldsymbol{x}}$, but sometimes the network can bring a 1d array into a 2d array (common for images in the early days of GANs).

- **No direct training of generator**, gets an indirect feedback from discriminator.

GANs: Discriminator

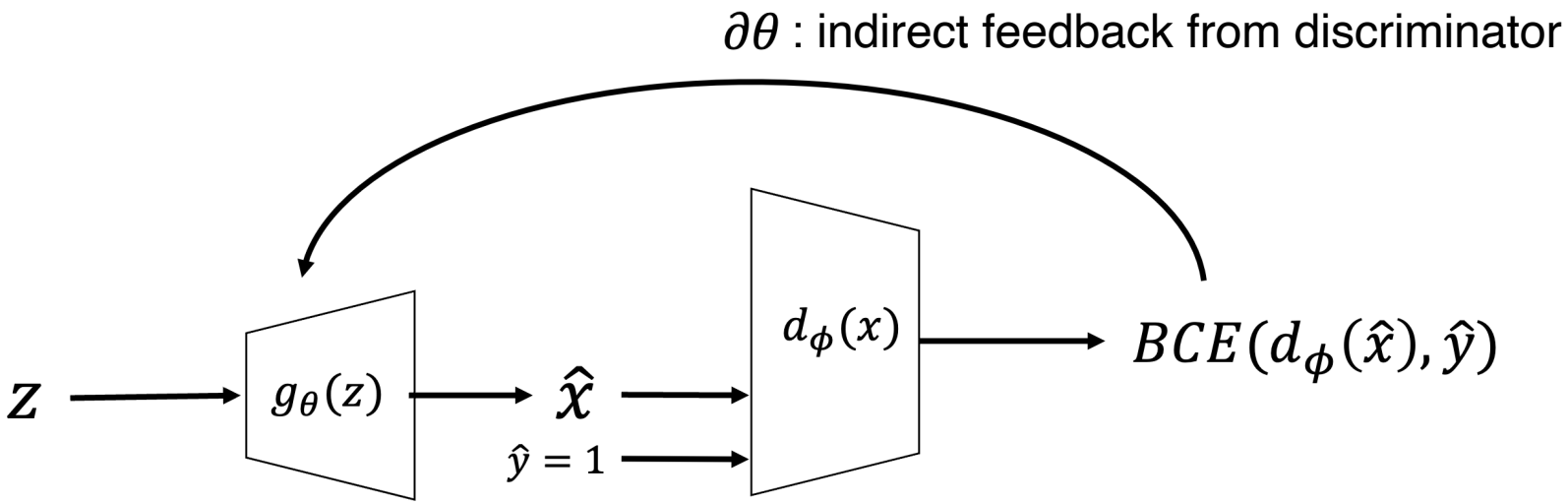


- $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_x}$ created by feeding the vector $\boldsymbol{z}$ into the generator network
- $\boldsymbol{x} \in \mathbb{R}^{n_x}$ sample of the training dataset, from which we want to learn the underlying probability density function $(\mathrm{p}(\boldsymbol{x}))$.

- **Discriminator** is trained as **binary classifier** (i.e., BCE loss)**.** Its goal is to tell apart fake $(\hat{x})$ from true (x) samples.

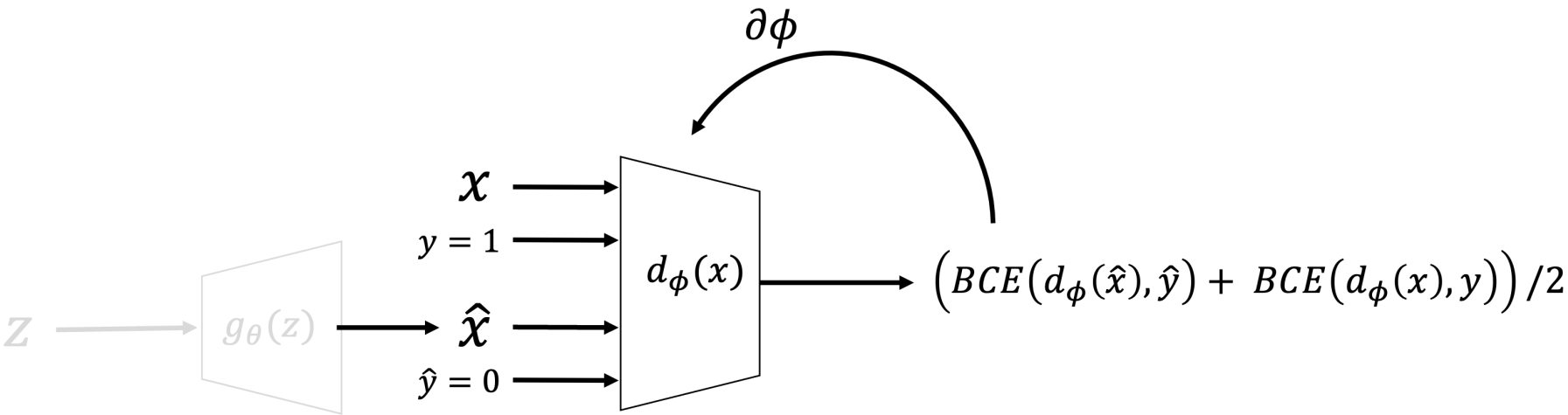
GANs: Training process (intuition)

Training is performed in two phases: **generator phase**



GANs: Training process (intuition)

Training is performed in two phases: **discriminator phase**

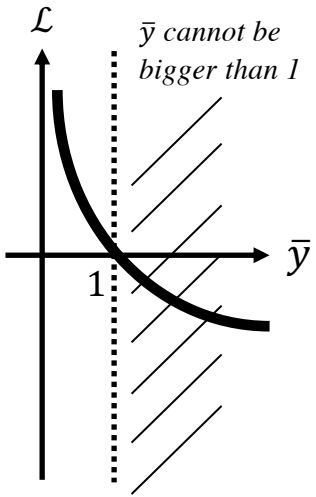


BCE refresher

Pred. True

$$BCE(\bar{y}, y) = -(ylog(\bar{y}) + (1-y)log(1-\bar{y}))$$

$y = 1$ $\quad \mathcal{L}(\bar{y}, y = 1) = -log\bar{y}$



- $\bar{y} \to 0, \mathcal{L} \to \infty$
(need to improve θ)

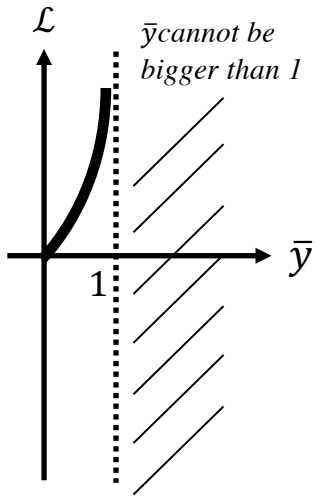
- $\bar{y} \to 1, \mathcal{L} \to 0$
(mimimum reached, no need to improve θ)

BCE refresher

Pred. True

$$BCE(\bar{y}, y) = -(ylog(\bar{y}) + (1-y)log(1-\bar{y}))$$

$y = 0 \quad \mathcal{L}(\bar{y}, y = 0) = -\log(1-\bar{y})$



- $\bar{y} \to 1, \mathcal{L} \to \infty$ *(need to improve θ)*

- $\bar{y} \to 0, \mathcal{L} \to 0$ *(mimimum reached, no need to improve θ)*

GANs: Training process (math)

The generator is trying to minimize the following loss function:

$$\mathcal{L}(\hat{\mathbf{x}}, \hat{\mathbf{y}} = 1) = BCE(d_\phi(g_\theta(\mathbf{z})), \hat{\mathbf{y}} = 1) = -log(d_\phi(g_\theta(\mathbf{z})))$$

Abuse of notation, $\mathcal{L}$(input, label)

GANs: Training process (math)

The generator is trying to minimize the following loss function:

$$\mathcal{L}(\hat{\mathbf{x}}, \hat{\mathbf{y}} = 1) = BCE(d_\phi(g_\theta(\mathbf{z})), \hat{\mathbf{y}} = 1) = -log(d_\phi(g_\theta(\mathbf{z})))$$

The discriminator is trying to minimize the following sum of loss functions:

$$\mathcal{L}(\hat{\mathbf{x}}, \hat{\mathbf{y}} = 0) = BCE(d_\phi(g_\theta(\mathbf{z})), \hat{\mathbf{y}} = 0) = -log(1 - d_\phi(g_\theta(\mathbf{z})))$$

$$\mathcal{L}(\mathbf{x}, \mathbf{y} = 1) = BCE(d_\phi(\mathbf{x}), \mathbf{y} = 1) = -log(d_\phi(\mathbf{x}))$$

GANs: Training process (math)

The generator is trying to minimize the following loss function:

$$\mathcal{L}(\hat{\mathbf{x}}, \hat{\mathbf{y}} = 1) = BCE(d_\phi(g_\theta(\mathbf{z})), \hat{\mathbf{y}} = 1) = -log(d_\phi(g_\theta(\mathbf{z})))$$

Min when discriminator is fooled ($d_\phi(g_\theta(\mathbf{z})) = 1$)

The discriminator is trying to minimize the following sum of loss functions:

$$\mathcal{L}(\hat{\mathbf{x}}, \hat{\mathbf{y}} = 0) = BCE(d_\phi(g_\theta(\mathbf{z})), \hat{\mathbf{y}} = 0) = -log(1 - d_\phi(g_\theta(\mathbf{z})))$$

$$\mathcal{L}(\mathbf{x}, \mathbf{y} = 1) = BCE(d_\phi(\mathbf{x}), \mathbf{y} = 1) = -log(d_\phi(\mathbf{x}))$$

Min when discriminator is good ($d_\phi(g_\theta(\mathbf{z})) = 0, d_\phi(\mathbf{x}) = 1$)

GANs: Min-max game

A more formal way to write the GAN loss function is via a min-max game:

$$arg \min_{g_\theta} \max_{d_\phi} E_{\mathbf{x}\sim p_x}[log(d_\phi(\mathbf{x}))] + E_{\mathbf{z}\sim p_z}[log(1 - d_\phi(g_\theta(\mathbf{z})))]$$

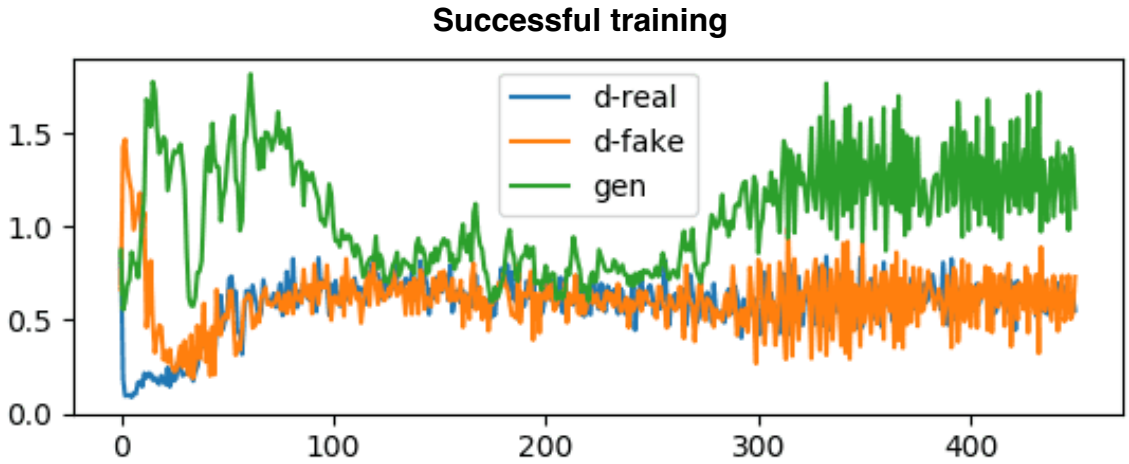
Act on Discriminator output for true data

Act on Discriminator output for fake data

Generator is affected only by second term of loss (wants to maximize it), Discriminator by both (wants to minimize them).

GANs: Training process

This is the real pain point of GANs: **hard to train!**



"A stable GAN will have a discriminator loss around 0.5, typically between 0.5 and maybe as high as 0.7 or 0.8. The generator loss is typically higher and may hover around 1.0, 1.5, 2.0, or even higher."

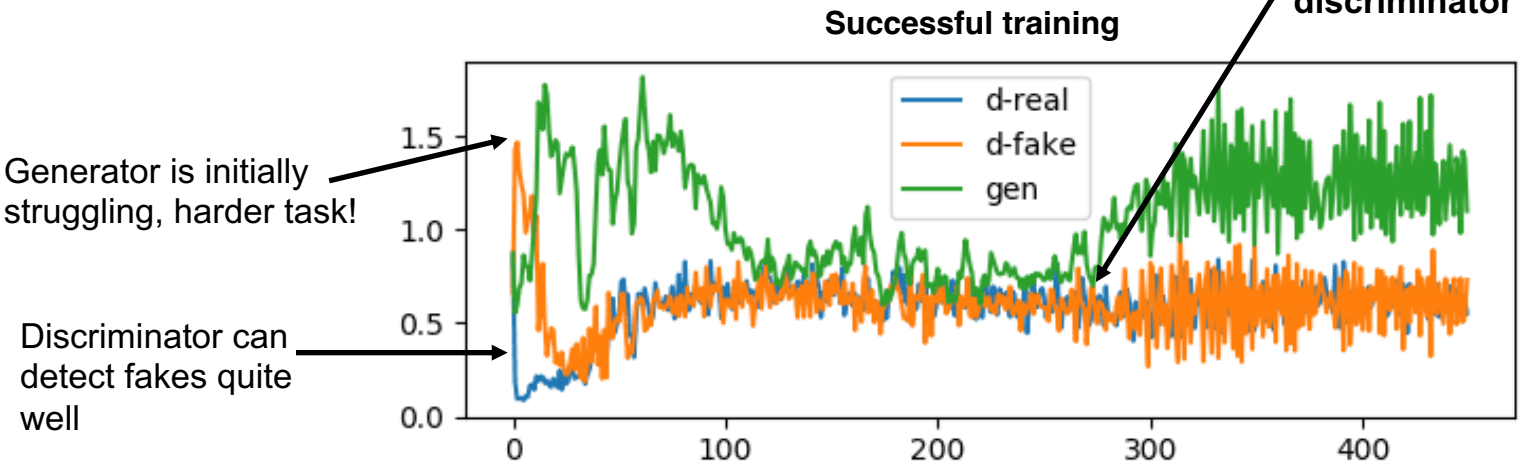
Source: https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

GANs: Training process

This is the real pain point of GANs: **hard to train!**

Generator catches up, and discriminator losses are roughly equal at 0.5.

At convergence, the discriminator is discarded!

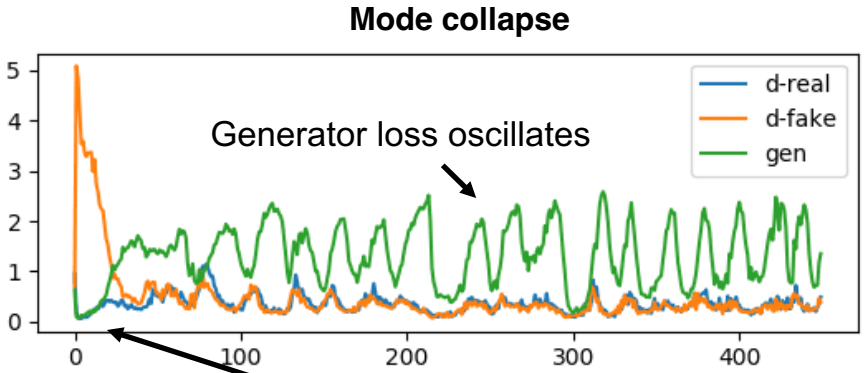


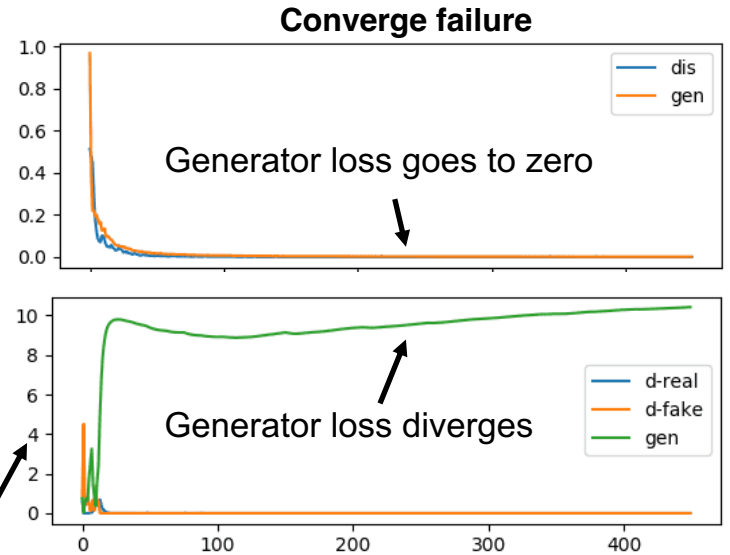
"A stable GAN will have a discriminator loss around 0.5, typically between 0.5 and maybe as high as 0.7 or 0.8. The generator loss is typically higher and may hover around 1.0, 1.5, 2.0, or even higher."

Source: https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

GANs: Training process

This is the real pain point of GANs: **hard to train!**



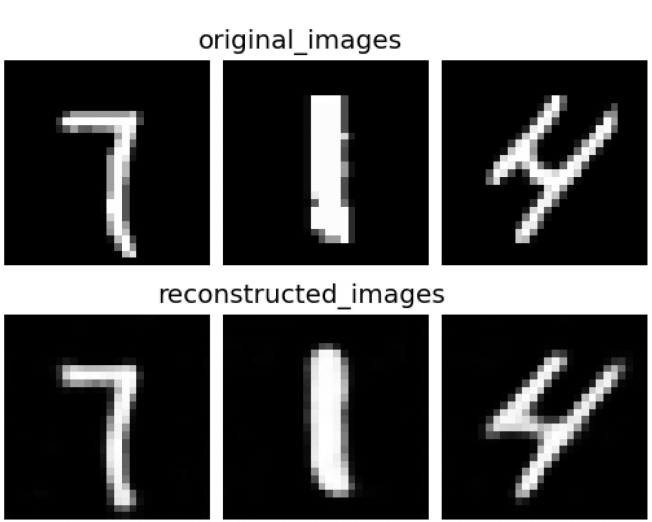


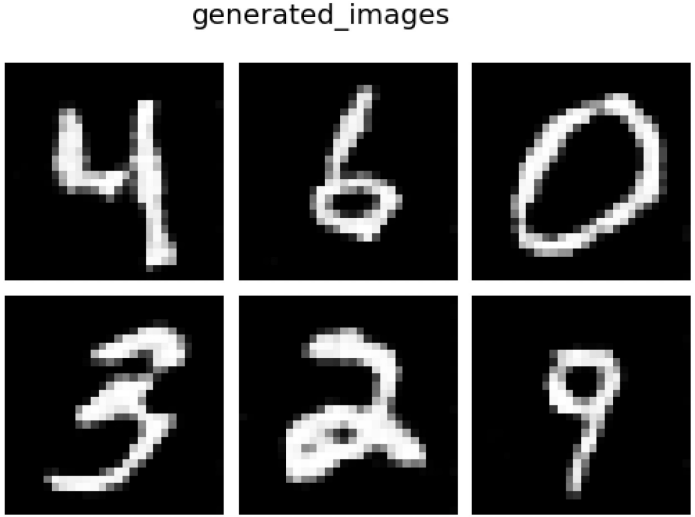
Discriminator loss goes quickly to 0 and remains there

Source: https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

GANs as generative models

Let's now sample z and then have the generator create an image from it?





*Latent space is **continuous***

GANs in practice

Algorithm

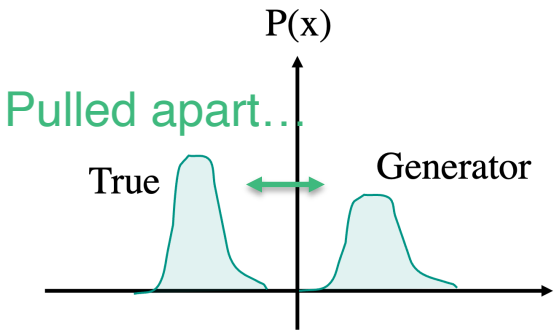
1. Select a batch of training samples and feed them through the discriminator
2. Create a batch of noise samples and feed them through the generator and discriminator
3. Compute the discriminator part of the loss function + backprop.
4. Repeat 1-2-3 N_{disc} times.
5. Create a batch of noise samples and feed them through the generator and discriminator and compute the generator part of the loss function + backprop.

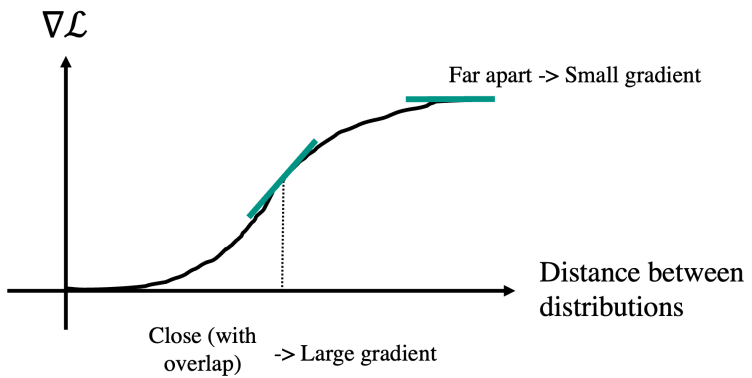
GANs issues: Convergence failure (vanishing gradients)

Usually happening when the discriminator learns too quickly:

$$d_\phi(\mathbf{x}) = 1 \quad \forall \mathbf{x} \qquad\qquad d_\phi(g_\theta(\mathbf{z})) = 0 \quad \forall \mathbf{z}$$

Loss function goes to zero, gradient becomes too small.

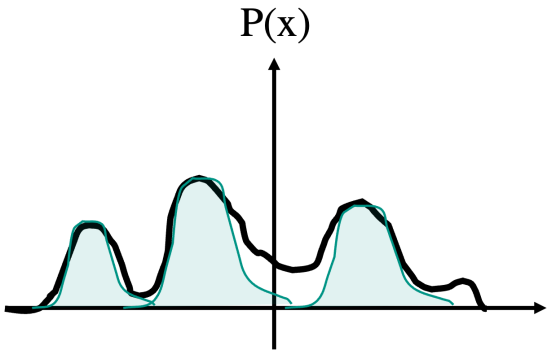




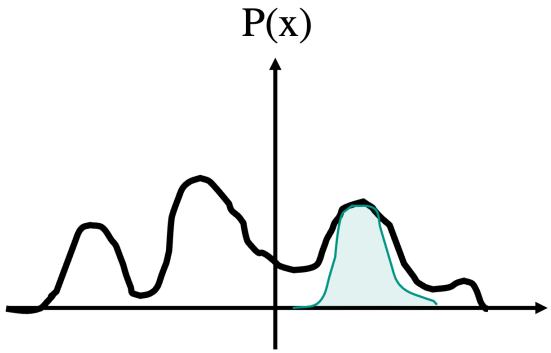
GANs issues: Mode collapse

Generator's goal: trick the discriminator into thinking that fake outputs are real

If generator produces one sample which is realistic and the discriminator finds it hard to distinguish between them -> **generator is not encouraged to explore more…**



Successful GAN training



'Collapsed' GAN training

GANs: Alternative implementations

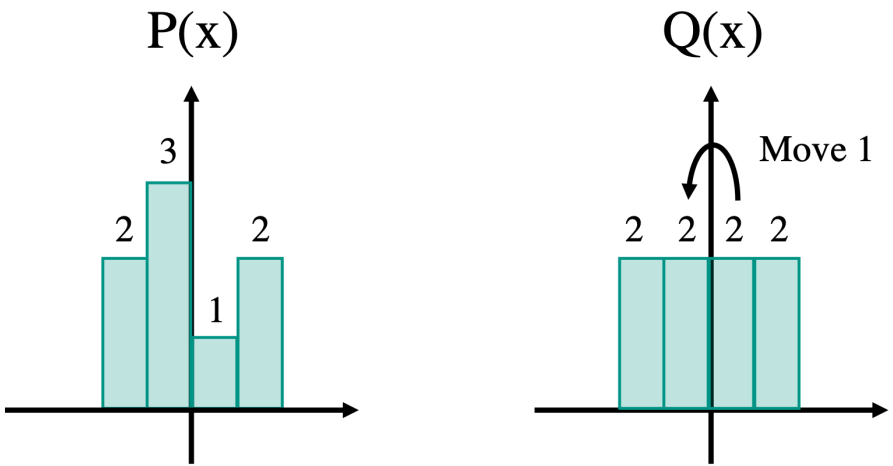
Overall consensus: **BCE loss is the main cause for unstable training**

Possible solutions:

- *Gradient clipping/penalty:* cheap way to avoid exploding gradients.
- *Spectral normalization*: more sophisticated way to ensure stable training, forcing discriminator to be 1-Lipschitz continuous (expensive!)
- *Pro-GAN (Progressive growing GANs):* progressively train from smaller, lower resolution version of training data to higher resolution.
- *Wasserstein GANs*: change BCE loss into Wasserstein loss
- …

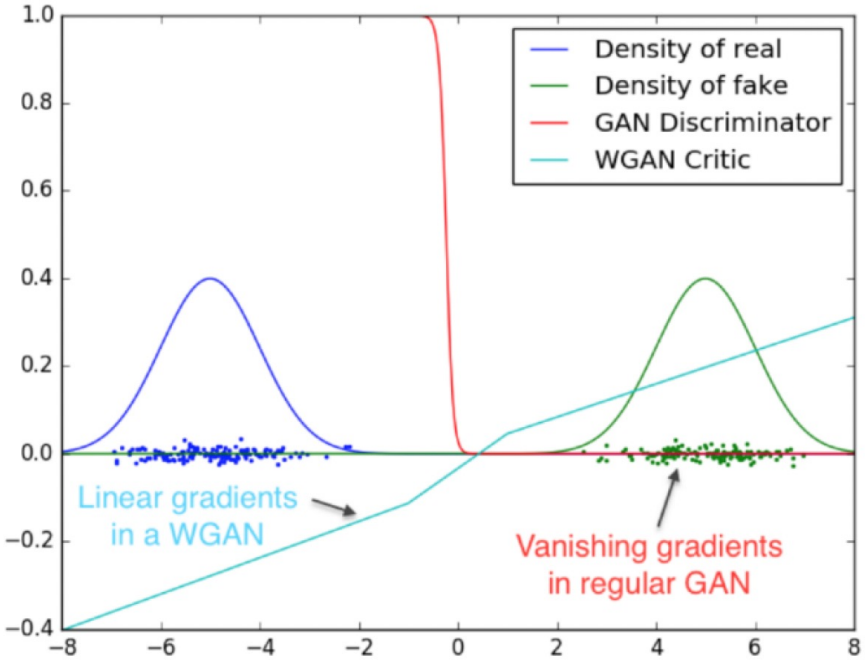
Wasserstein GAN

Wasserstein distance (or Earth's mover distance):



Distance function between two probability distributions that computes the ***amount of earth (or soil) that needs to be moved from one probability to match another****.*

Wasserstein GAN



Source: Arjovsky et al. (2017)

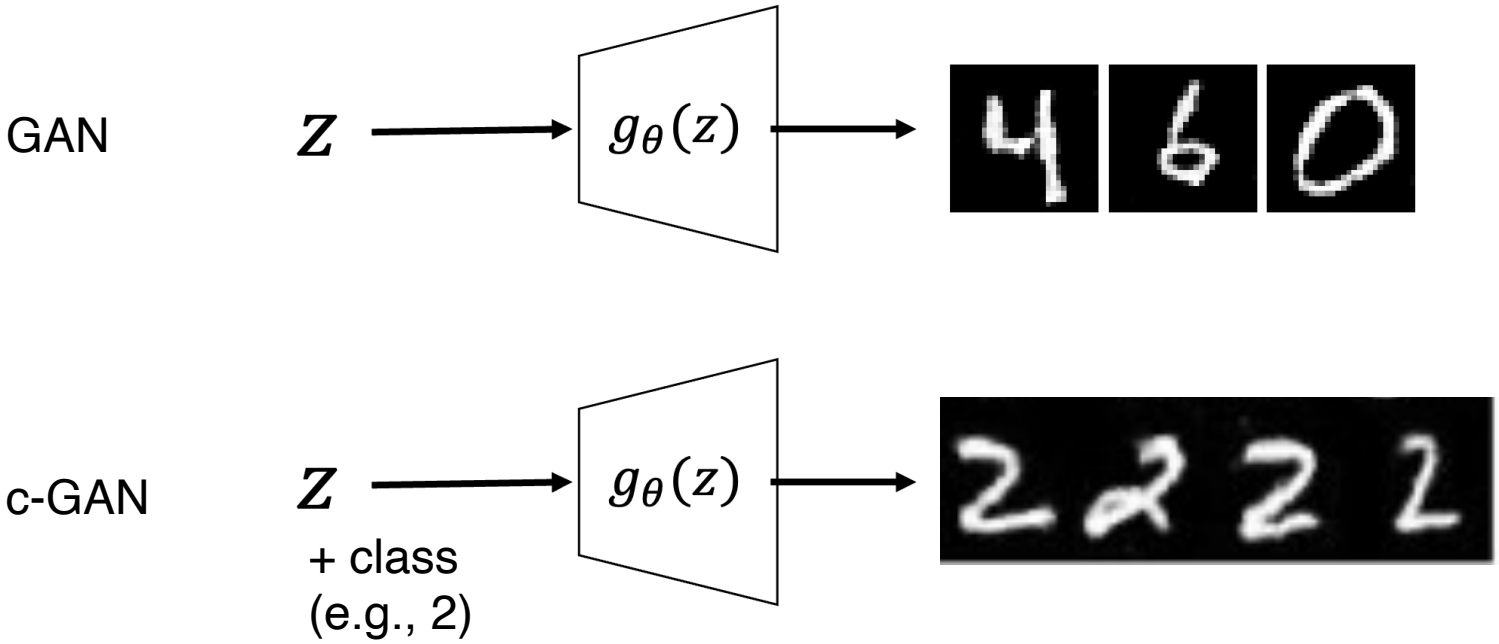
WGAN Loss:

$$arg \min_{g_\theta} \max_{d_\phi} E_{\mathbf{x}\sim p_x}[d_\phi(\mathbf{x})] - E_{\mathbf{z}\sim p_z}[d_\phi(g_\theta(\mathbf{z}))]$$

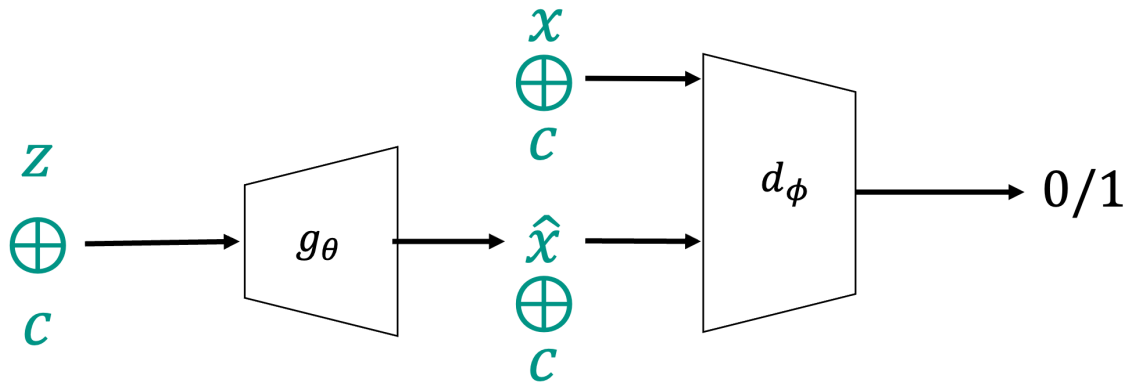
Discriminator is now called Critic (no classifier!)

Conditional GANs: classes

Idea: can we know upfront (before sampling) what we will produce...



Conditional GANs: classes

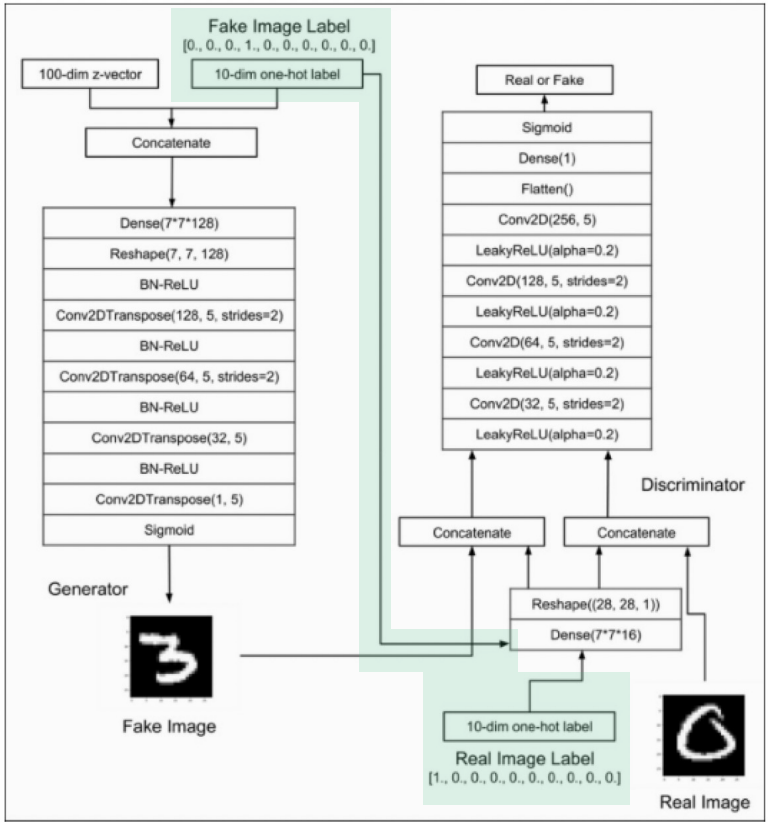


- Concatenate a label c representing a class to random noise ($\boldsymbol{z} \oplus c$), or its one-hot encoded version ($\boldsymbol{z} \oplus \boldsymbol{c}$). Inform generator both at train and generation time.

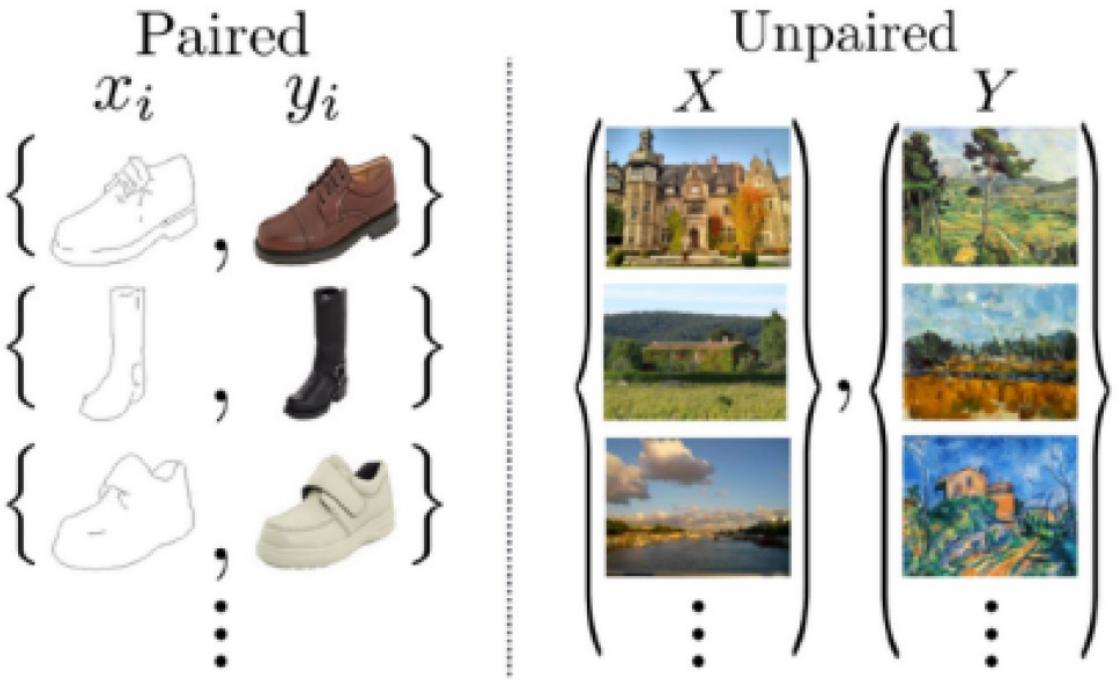
- Concatenate the same label to $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ before feeding them to the discriminator.

- When working with Nd-arrays (e.g. images), concatenation is replaced by channel concatenation (c or $\boldsymbol{c}$ is replicated over the entire 'image').

Conditional GANs: classes



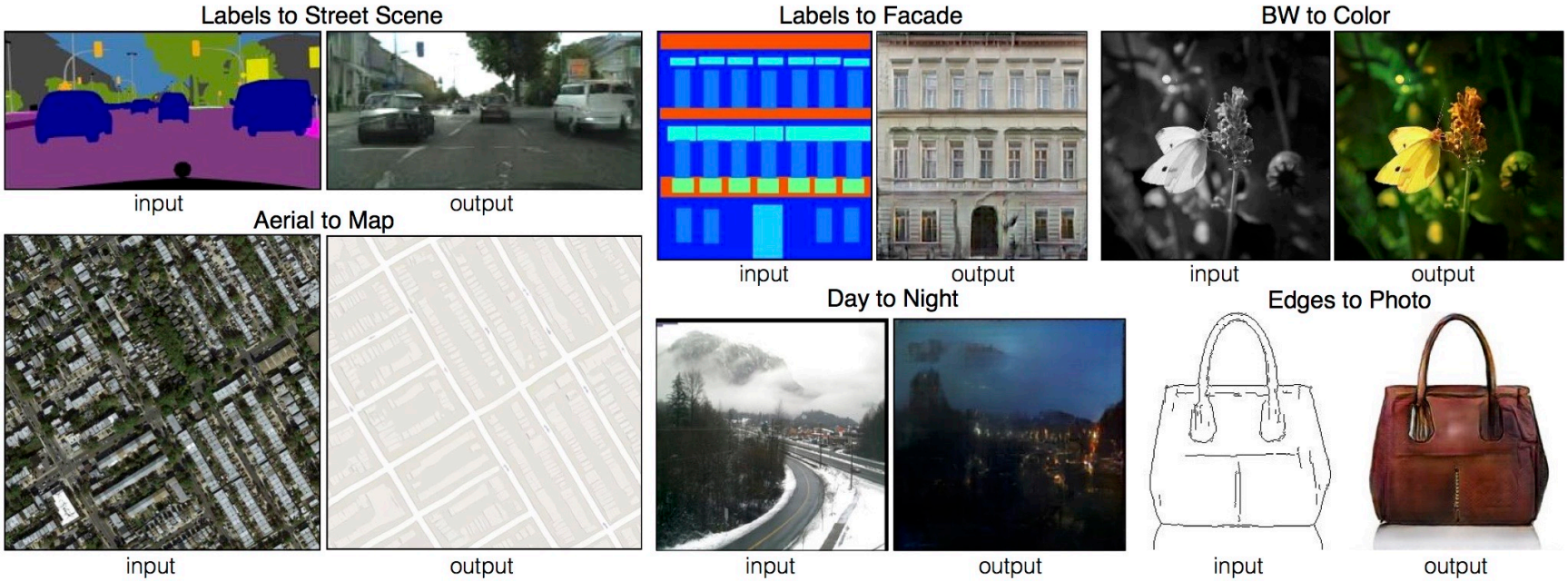
Conditional GANs: domain translation



Pairs of input-output are available

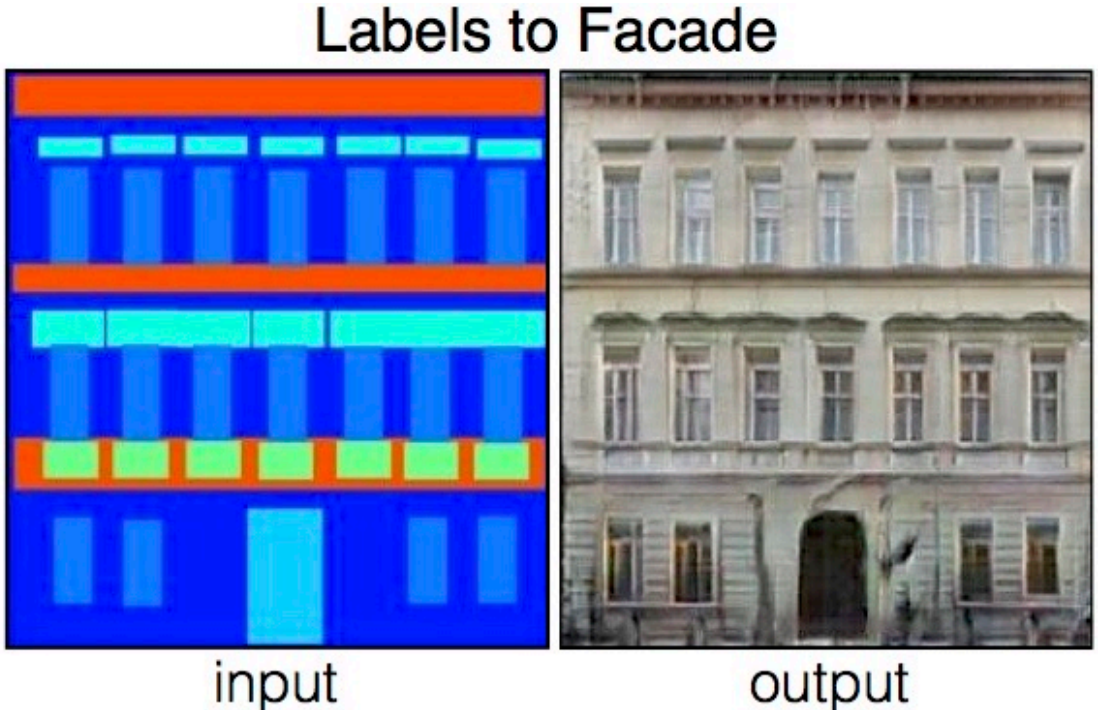
Only a set of samples from each domain

Pix2Pix : paired domain translation



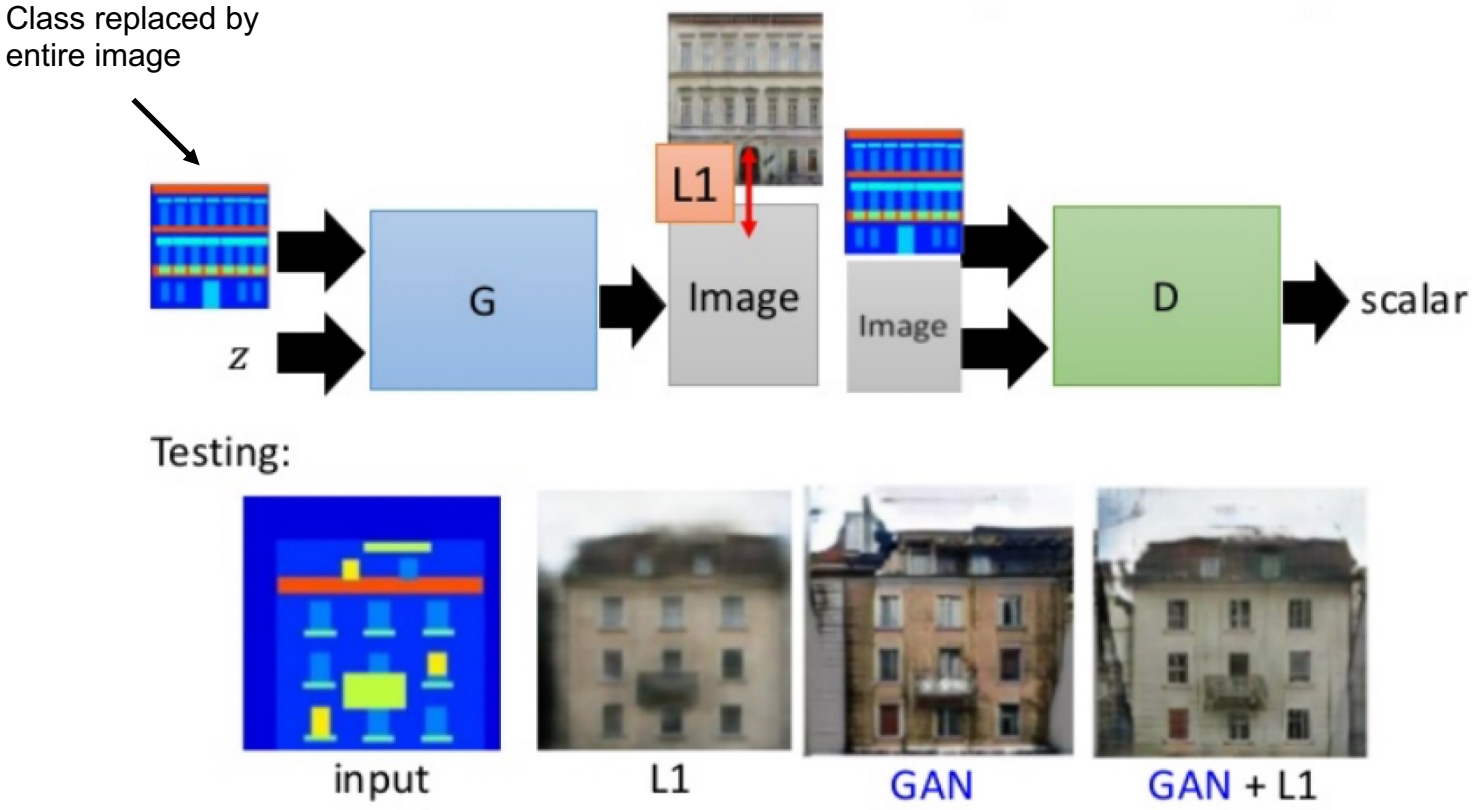
Source: Isola et al. (2017)

Pix2Pix : paired domain translation

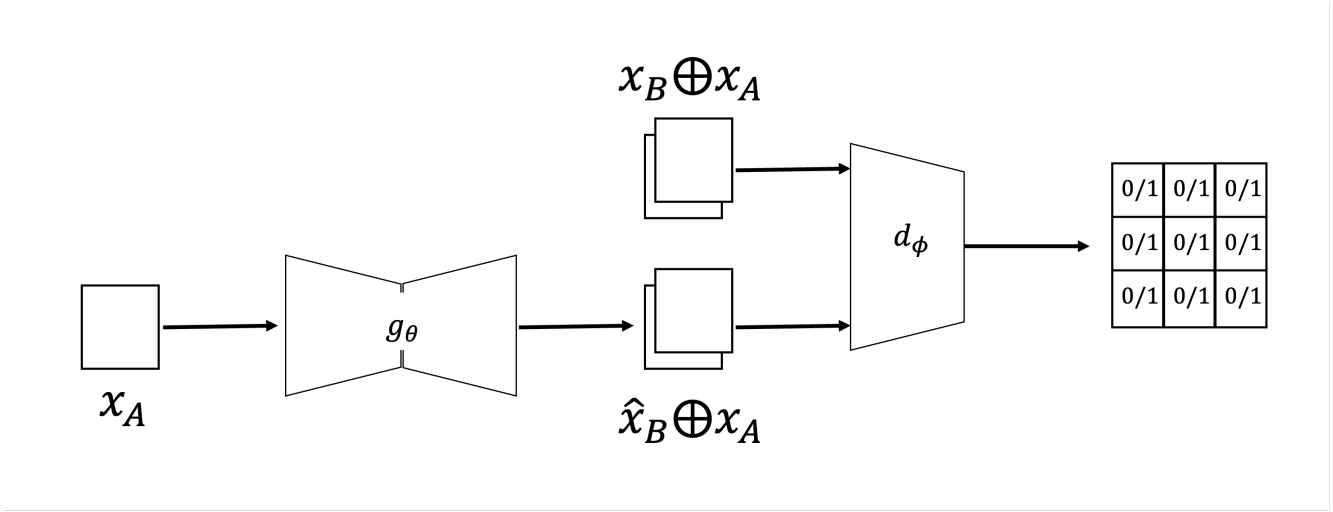


*This image-to-image (or domain translation) task can be solved with any NN architecture (UNet usually best) and any loss → but **c-GAN loss seems the best!***

Pix2Pix : paired domain translation



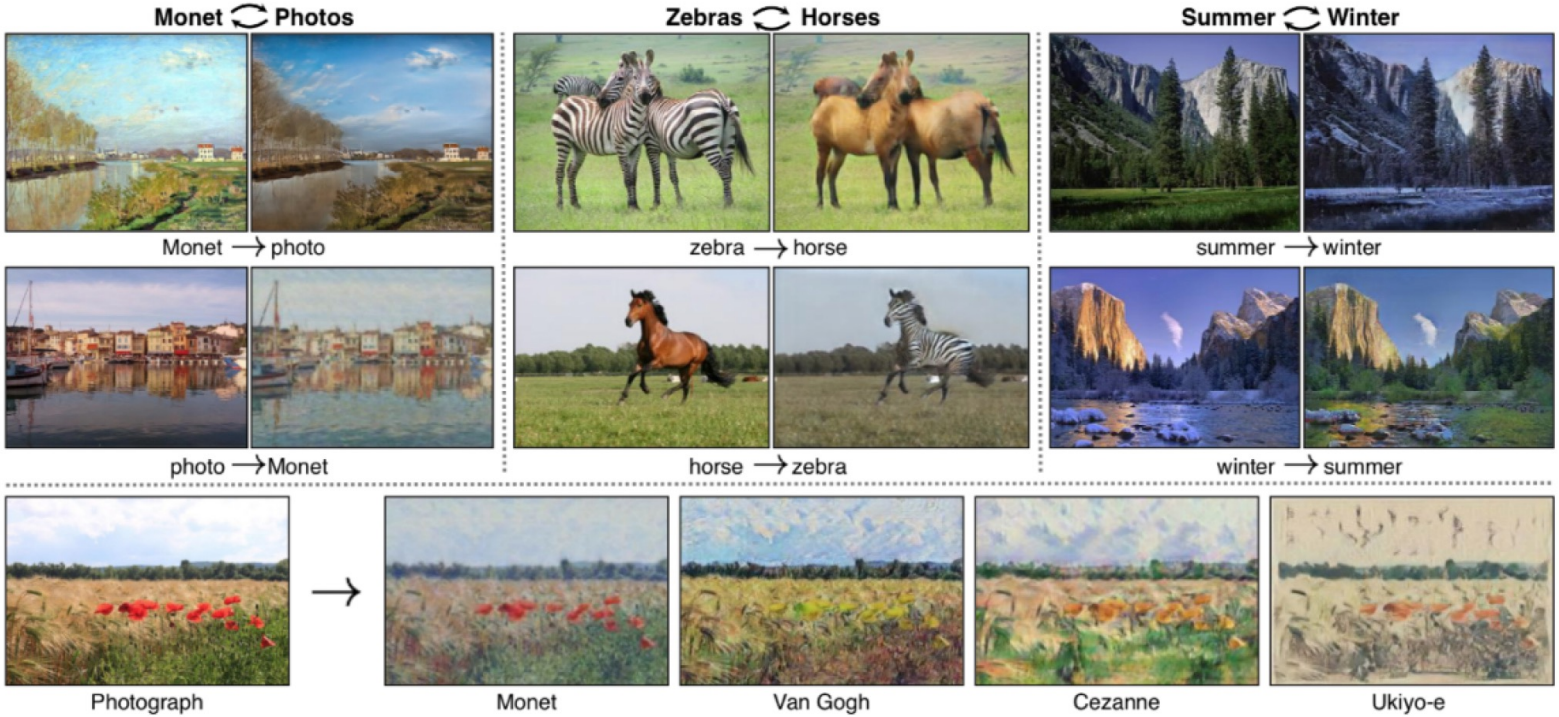
Pix2Pix : paired domain translation



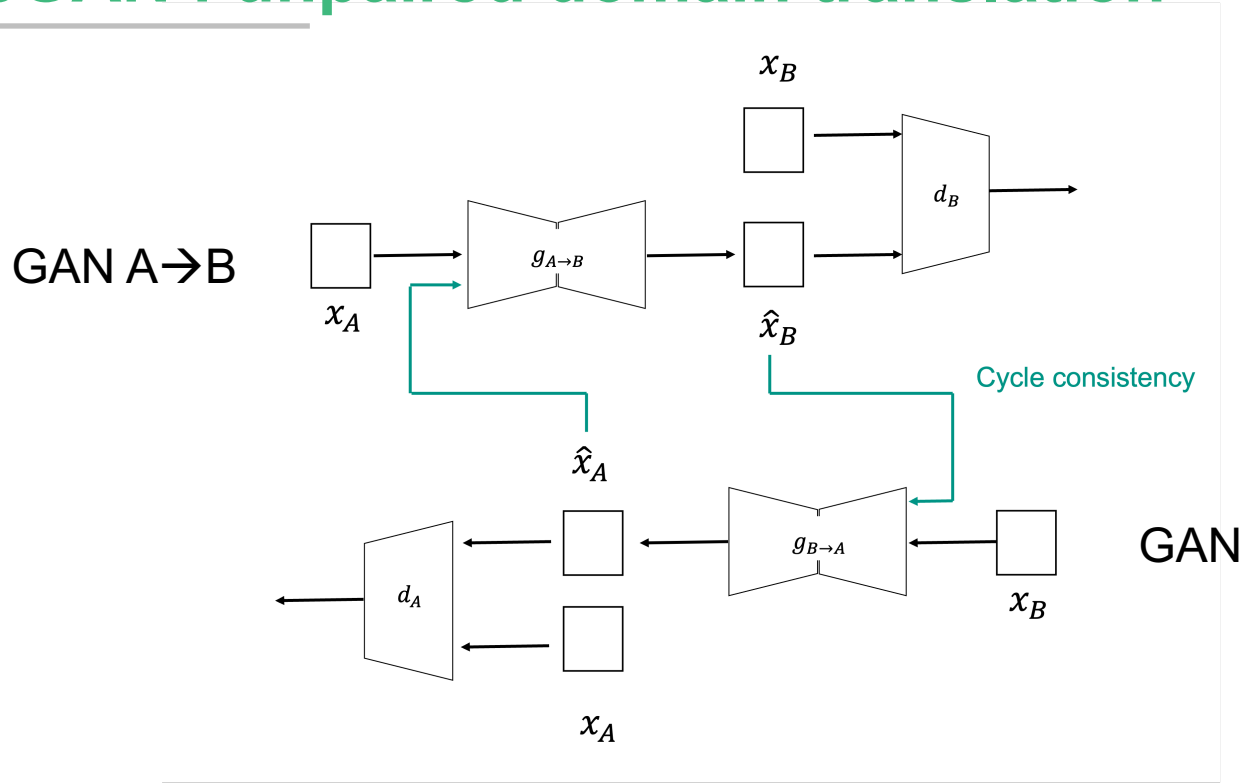
In practice, few additional changes:

- **No need for noise in input**, just the input from domain A.
- **UNet** architecture used for Generator
- **PatchGAN** discriminator: not single binary classification, but on multiple patches

CycleGAN : unpaired domain translation



CycleGAN : unpaired domain translation



$$\mathcal{L} = \sum_{i \in (A \to B, B \to A)} (\mathcal{L}_{adv,i} + \lambda_C(\mathcal{L}_{cycle,i})) + \lambda_I(\mathcal{L}_{identity,A} + \mathcal{L}_{identity,B})$$

CycleGAN : unpaired domain translation

Adversarial training: Train 2 GANs in parallel (2 discriminators and 2 generators): A → B and B → A

$$\mathcal{L}_{adv,g_{A\to B}} = E_{\mathbf{x}_A\sim p_{x,A}}[(d_B(g_{A\to B}(\mathbf{x}_A))-1)^2],$$

MSE loss

$$\mathcal{L}_{adv,g_{B\to A}} = E_{\mathbf{x}_B\sim p_{x,B}}[(d_A(g_{B\to A}(\mathbf{x}_B))-1)^2],$$

$$\mathcal{L}_{adv,d_A} = E_{\mathbf{x}_A\sim p_{x,A}}[(d_A(\mathbf{x}_A)-1)^2] + E_{\mathbf{x}_B\sim p_{x,B}}[d_B(g_{B\to A}(\mathbf{x}_B))^2],$$

$$\mathcal{L}_{adv,d_B} = E_{\mathbf{x}_B\sim p_{x,B}}[(d_B(\mathbf{x}_B)-1)^2] + E_{\mathbf{x}_A\sim p_{x,A}}[d_B(g_{A\to B}(\mathbf{x}_A))^2],$$

CycleGAN : unpaired domain translation

Cycle-consistency loss: prediction from $G_{A \to B}$ is fed to $G_{B \to A}$ and output is compared to input

$$\mathcal{L}_{cycle, A \to B} = MSE(\mathbf{x}_A, g_{B \to A}(g_{A \to B}(\mathbf{x}_A))$$
$$\mathcal{L}_{cycle, B \to A} = MSE(\mathbf{x}_B, g_{A \to B}(g_{B \to A}(\mathbf{x}_B))$$

Identity loss: Sample from A fed to $G_{B \to A}$ and check output is same as input

$$\mathcal{L}_{identity, A} = MSE(g_{B \to A}(\mathbf{x}_A), \mathbf{x}_A)$$
$$\mathcal{L}_{identity, B} = MSE(g_{A \to B}(\mathbf{x}_B), \mathbf{x}_B)$$