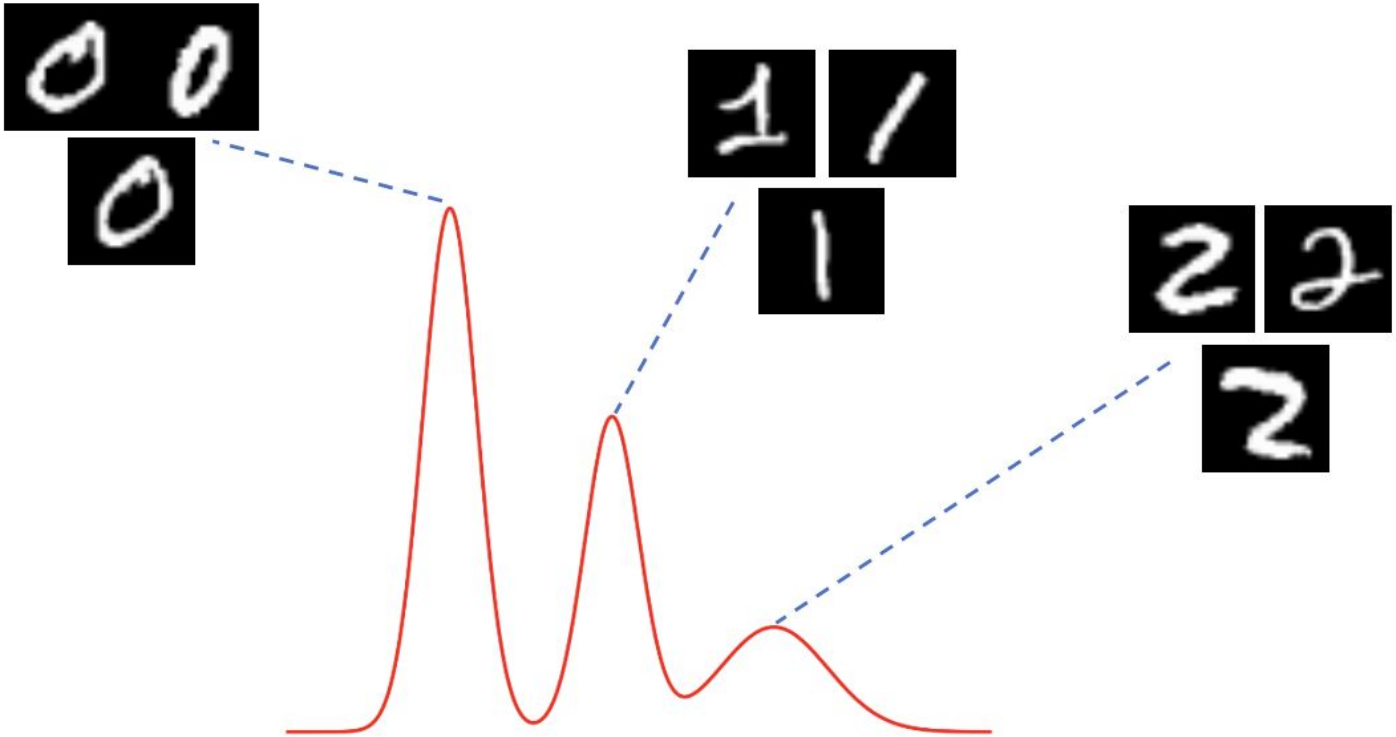
Practices in visual computing 2

Lab3: GANs

Simon Fraser University
Spring 2026

Generative learning



Generative learning

Explicit Models:

- train through directly optimizing p(x)
- Through Maximum likelihood estimation (MLE)
- Example: VAEs

Implicit Models:

- Train to sample from p(x)
- Example: GANs

GAN Conceptual Introduction

Two networks (generator G and discriminator D) compete in a minimax game.

Generator (G): Learns to produce “fake” data that mimic real data.

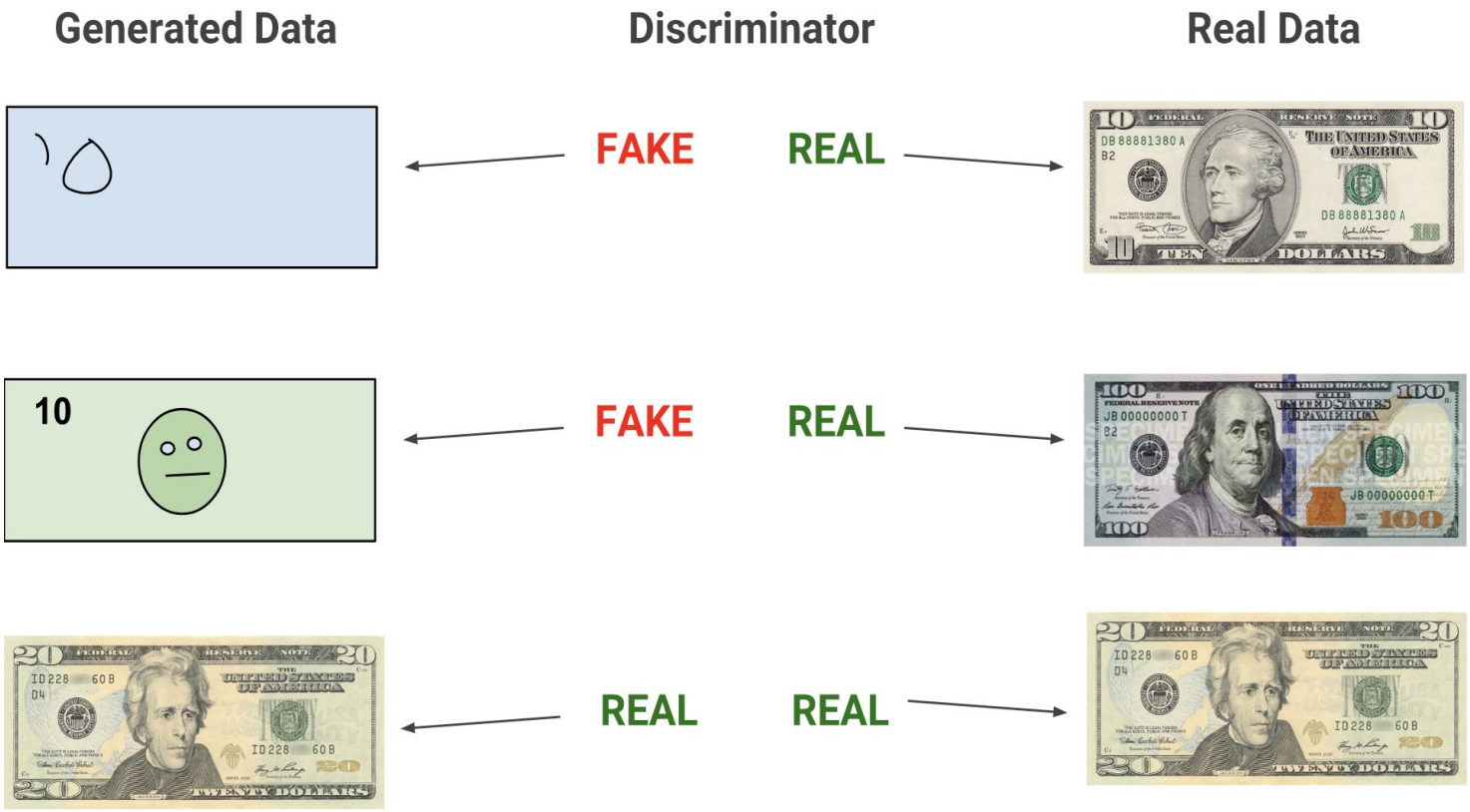
Discriminator (D): Learns to distinguish between real and fake data.

As training progresses: G gets better at fooling D.

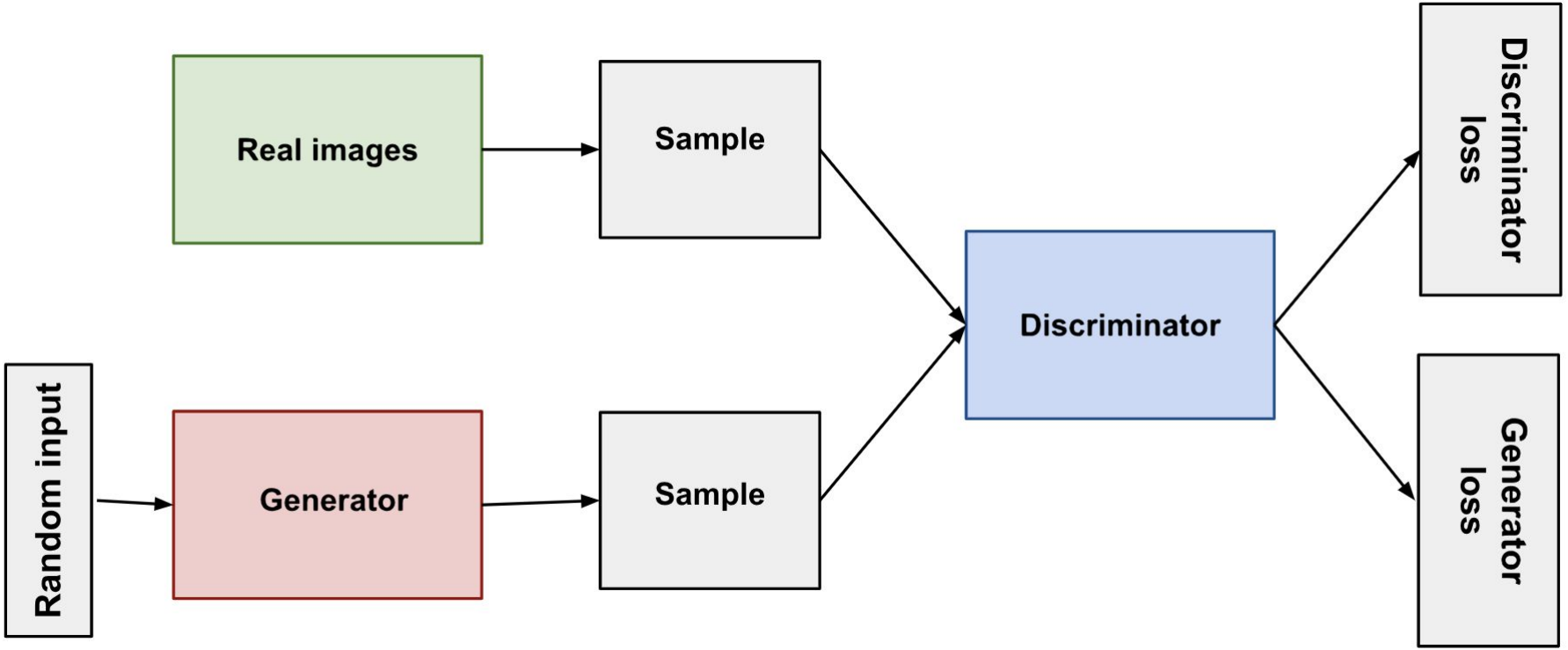
D gets better at identifying fakes.

Goal: The generator produces samples indistinguishable from real ones, and the discriminator fails to tell them apart.

Training GANs



GAN Architecture Diagram



Training GANs

Jointly Train Generator and Discriminator through a minmax loss

- Alternate between:
  - Gradient ascent for Discriminator

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

  - Gradient descent for Generator

$$\min[\mathbb{E}_{z\sim p_z} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)]$$

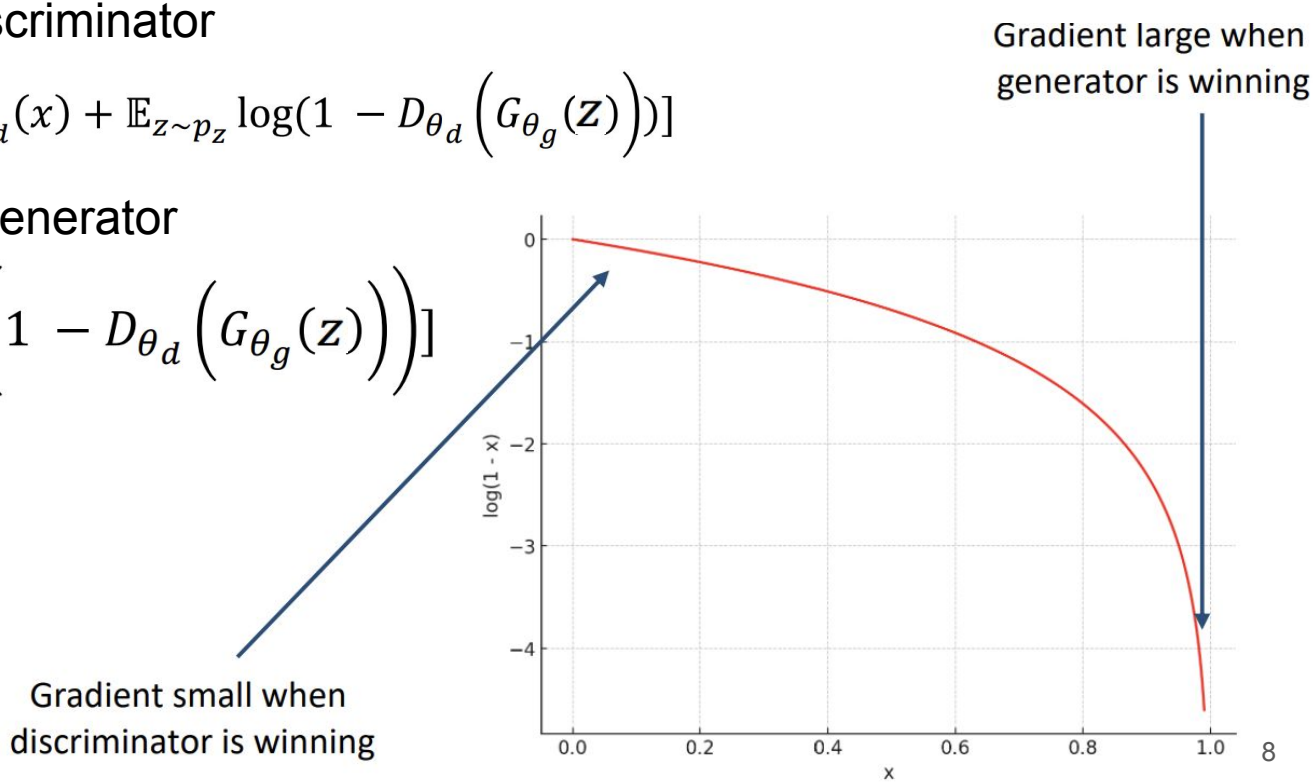
Training GANs

- Gradient ascent for Discriminator

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

- Gradient descent for Generator

$$\min[\mathbb{E}_{z\sim p_z} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)]$$



Training GANs

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:

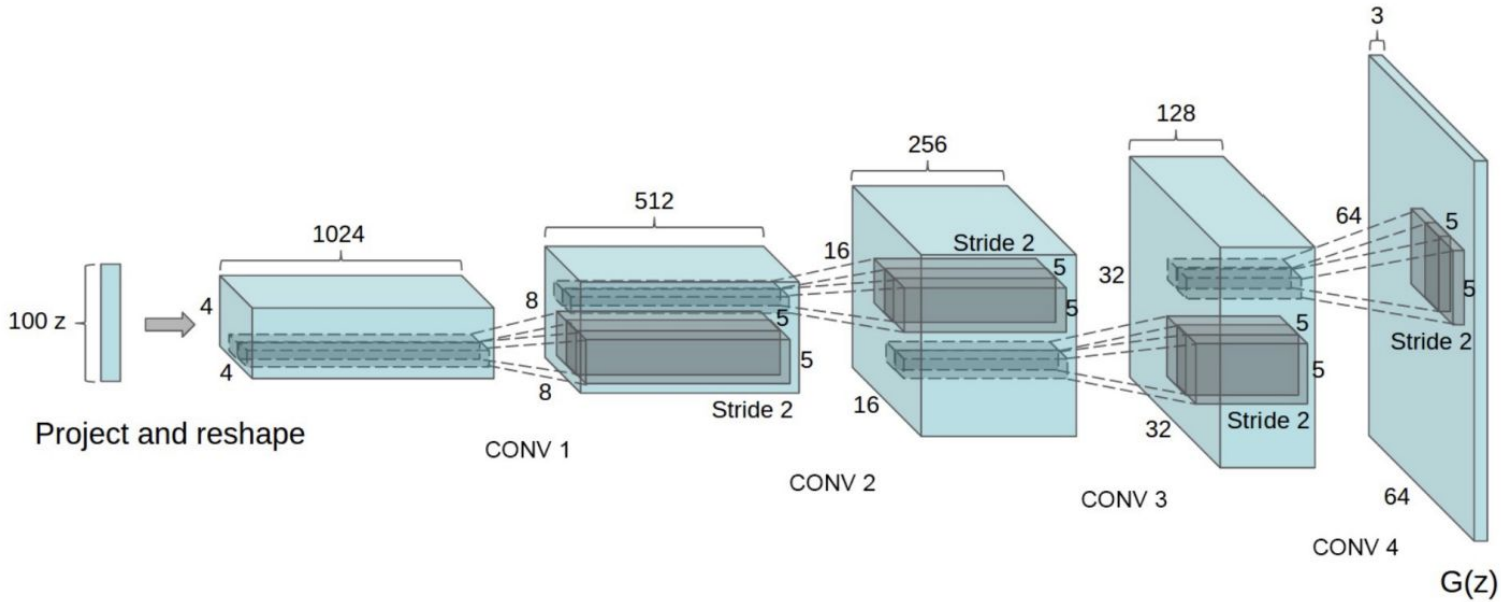
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

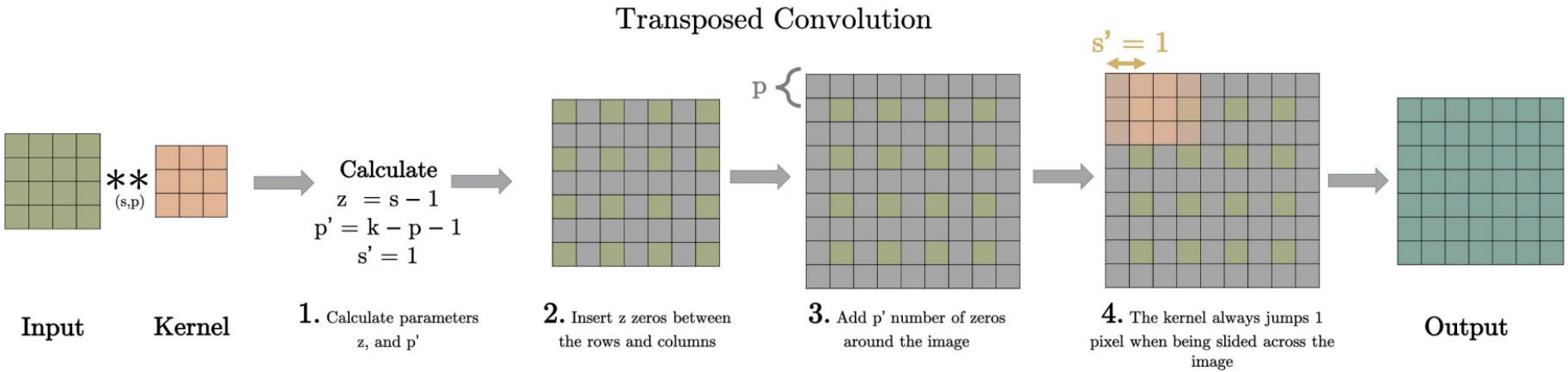
GANs: Convolutional architecture

Generator is a upsampling network with fractionally-strided Convolutions.



GANs: Convolutional architecture

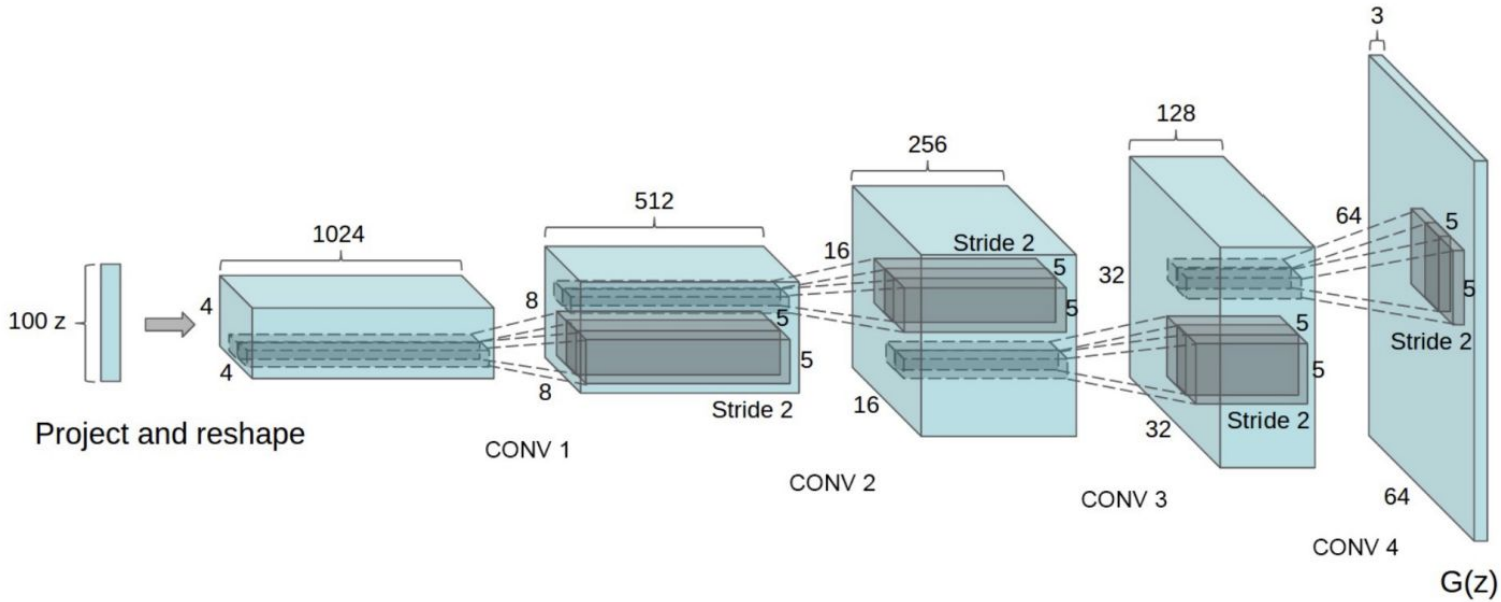
Generator is a upsampling network with fractionally-strided Convolutions.



$$o = (i - 1) \times s + k - 2p$$

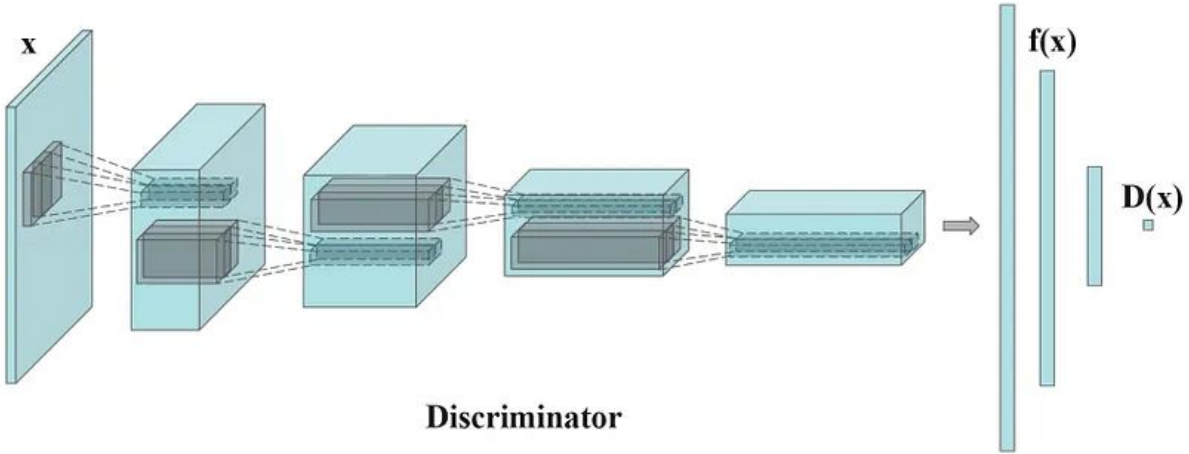
GANs: Convolutional architecture

Generator is a upsampling network with fractionally-strided Convolutions.



GANs: Convolutional architecture

- Generator is a upsampling network with fractionally-strided Convolutions.
- Discriminator is a CNN.



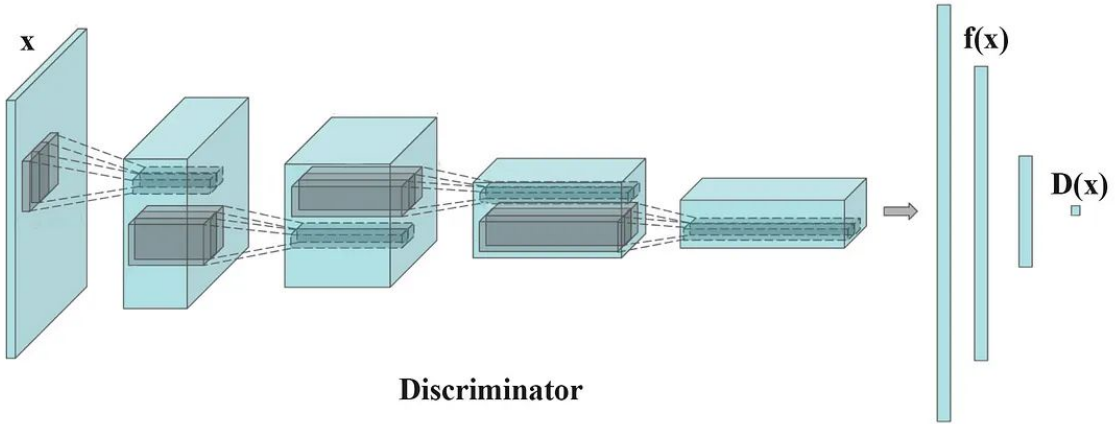
Tips and Tricks

Training GANs is difficult

- Stability between Generator and Discriminator
  - Vanishing Gradients - Exploding Gradients
- Mode Collapse
  - The generator repeatedly produces similar or identical samples.

Feature Matching

- Feature matching changes the generator's goal from fooling the discriminator to matching real data statistics.
- The generator minimizes the L2 distance between feature means of real and generated images. $||\mathbb{E}_{x \sim p_{\text{data}}} \mathbf{f}(x) - \mathbb{E}_{z \sim p_z(z)} \mathbf{f}(G(z))||_2^2$
- Features are taken from an intermediate discriminator layer.
- This reduces greediness, mode collapse, and training instability.



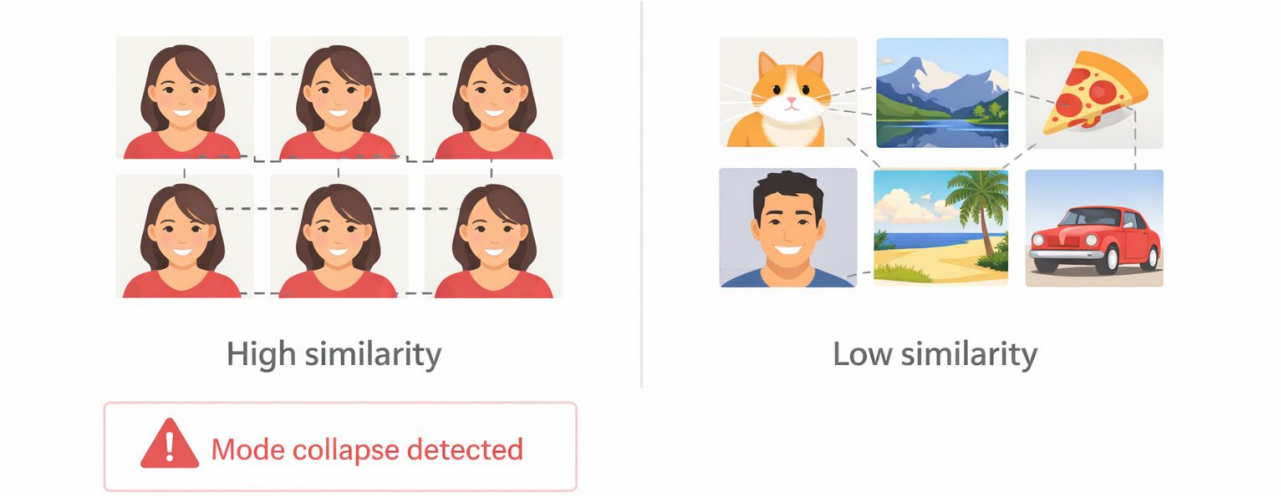
Minibatch Discrimination

Mode collapse occurs when generated images become too similar.

Minibatch discrimination lets the discriminator compare images within a batch.

Each image is evaluated based on its similarity to other images in the same batch.

This similarity is added as an extra signal in the discriminator.



Minibatch Discrimination

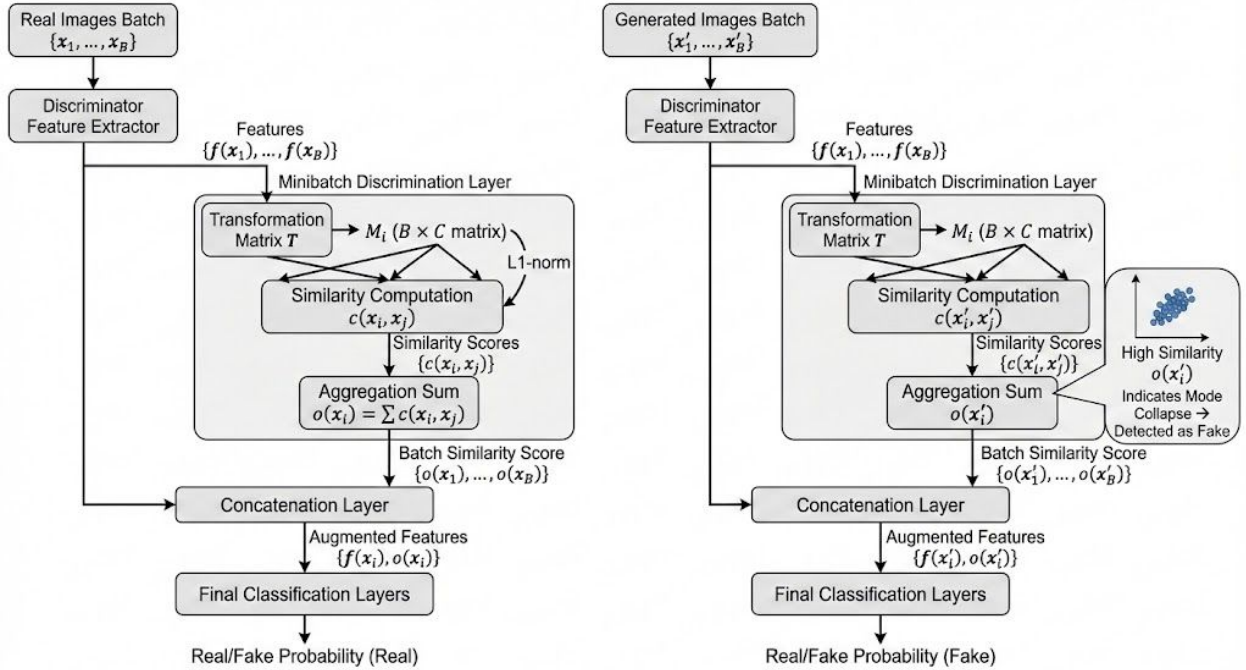
If mode collapse happens, generated images show high mutual similarity.

The discriminator uses this signal to detect fake samples.

The generator is penalized when diversity decreases.

This encourages more diverse and stable generation.

Minibatch Discrimination



Minibatch Discrimination

$$
\begin{aligned}
f(x_i) &\in \mathbb{R}^A \\
T &\in \mathbb{R}^{A \times B \times C} \\
M_i &= f(x_i)T \in \mathbb{R}^{B \times C} \\
c_b(x_i, x_j) &= exp(-\|M_{i,b} - M_{j,b}\|_1) \in \mathbb{R} \\
o(x_i)_b &= \sum_{j=1}^{n} c_b(x_i, x_j) \in \mathbb{R} \\
o(x_i) &= [o(x_i)_1, o(x_i)_2, \ldots, o(x_i)_B] \in \mathbb{R}^B
\end{aligned}
$$

Historical Averaging

Model parameters are averaged over recent training steps.

The current model is compared to this historical average.

An L2 penalty is added for deviating from the average.

Large parameter jumps increase the training loss.

Oscillations around the GAN equilibrium are reduced.

Training converges more smoothly and stably.

$$\|\theta - \frac{1}{t}\sum_{i=1}^{t}\theta[i]\|_2$$

Using labels (CGAN)

Many datasets provide class labels for samples

GAN training is difficult without additional guidance

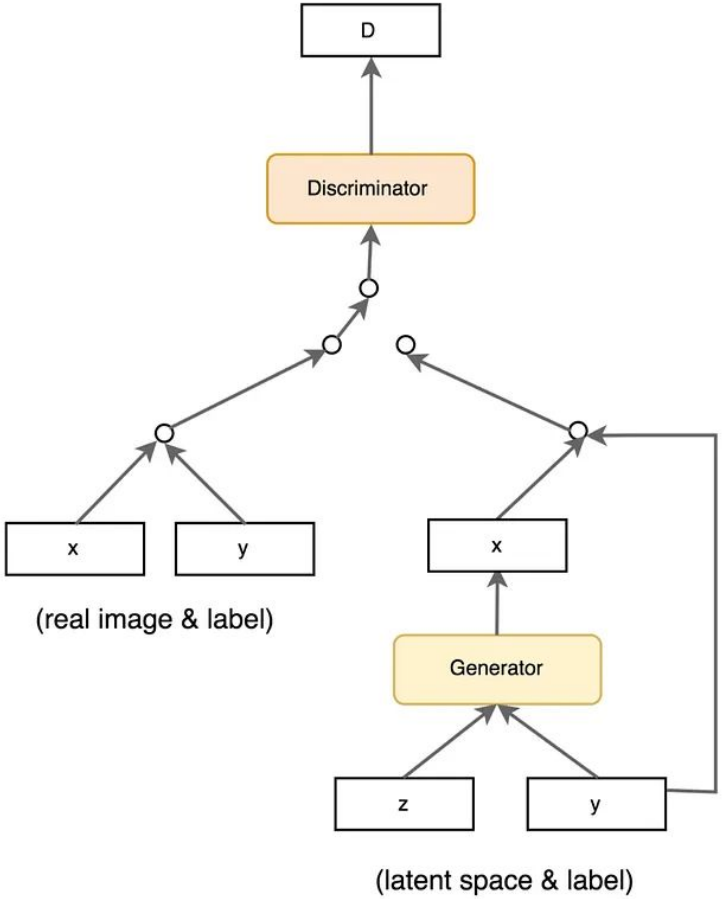
Labels are added as conditioning information

The generator receives noise z and a label

The discriminator checks realism and label consistency

Conditioning improves stability and sample quality

Using labels (CGAN)



Cost Functions

GAN (original)

Loss:

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

$$L_G^{\text{GAN}} = \mathbb{E}_{z \sim p_z}[\log D(G(z))]$$

What problem it has?

- Vanishing gradients
- Training stalls when generator is bad

Cost Functions

LSGAN

Loss:

$$L_D^{\text{LSGAN}} = \mathbb{E}_{x \sim p_{\text{data}}}[(D(x) - 1)^2] + \mathbb{E}_{z \sim p_z}[D(G(z))^2]$$

$$L_G^{\text{LSGAN}} = \mathbb{E}_{z \sim p_z}[(D(G(z)) - 1)^2]$$

What the discriminator is estimating

- No probability
- Learns a regression function
- Penalizes distance from target labels

Cost Functions

WGAN

Loss:

$$L_D^{\text{WGAN}} = \mathbb{E}_{x \sim p_{\text{data}}}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))]$$

$$L_G^{\text{WGAN}} = \mathbb{E}_{z \sim p_z}[D(G(z))]$$

$$W_D \leftarrow \text{clip\_by\_value}(W_D, -0.01, 0.01)$$

What problem it fixes

- No vanishing gradients
- Loss correlates with sample quality

Cost Functions

DRAGAN

Loss:

$$L_D^{\mathrm{DRAGAN}} = L_D^{\mathrm{GAN}} + \lambda \, \mathbb{E}_{\hat{x}} \left[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1 \right)^2 \right]$$

$$L_G^{\mathrm{DRAGAN}} = L_G^{\mathrm{GAN}}$$

What it enforces

- Smooth discriminator near real data
- Not globally

What problem it fixes

- Sharp gradients near real samples
- Early training instability

Cost Functions

CGAN

Loss:

$$L_D^{\mathrm{CGAN}} = \mathbb{E}_{x \sim p_{\mathrm{data}}}[\log D(x, c)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z), c))]$$

$$L_G^{\mathrm{CGAN}} = \mathbb{E}_{z \sim p_z}[\log D(G(z), c)]$$

What the discriminator is estimating

- Learns conditional distributions p(real | x,c)
- Generator is forced to match p(x | c)

What problem it fixes

- Reduces mode mixing
- Simplifies learning by conditioning

Cost Functions

BEGAN

Loss:

$$L_D^{\text{BEGAN}} = D_{AE}(x) - k_t D_{AE}(G(z))$$

$$L_G^{\text{BEGAN}} = D_{AE}(G(z))$$

$$k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$$

- Discriminator is an autoencoder
- Output is reconstruction error
- Real images → low error
- Fake images → high error
- No real/fake classifier

Cost Functions

BEGAN

Loss:

$$L_D^{\text{BEGAN}} = D_{AE}(x) - k_t D_{AE}(G(z))$$

$$L_G^{\text{BEGAN}} = D_{AE}(G(z))$$

$$k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$$

- Uses a control variable k_t
- Adjusts discriminator strength automatically
- Increases pressure when generator is too strong
- Decreases pressure when generator is too weak
- Keeps generator and discriminator in balance

Cost Functions

BEGAN

Loss:

$$L_D^{\text{BEGAN}} = D_{AE}(x) - k_t D_{AE}(G(z))$$

$$L_G^{\text{BEGAN}} = D_{AE}(G(z))$$

$$k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$$

- Mode collapse produces very similar fake images
- Autoencoder sees the same fake samples repeatedly
- Repeated samples are easy to reconstruct
- Fake reconstruction error drops quickly
- This signals generator collapse

Cost Functions

Name	Value Function
GAN	$L_D^{GAN} = E\left[\log\left(D(x)\right)\right] + E\left[\log\left(1 - D(G(z))\right)\right]$ $L_G^{GAN} = E\left[\log\left(D(G(z))\right)\right]$
LSGAN	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[\left(\lvert\nabla D(\alpha x - (1 - \alpha x_p))\rvert - 1\right)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$

CGAN	$L_D^{CGAN} = E\left[\log\left(D(x, c)\right)\right] + E\left[\log\left(1 - D(G(z), c)\right)\right]$ $L_G^{CGAN} = E\left[\log\left(D(G(z), c)\right)\right]$
infoGAN	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\lvert x)] + E[P(class = c\lvert G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\lvert G(z))]$
EBGAN	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

Cost Functions

	MNIST	FASHION	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6*	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8*
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

Implementation tips

Scale the image pixel value between -1 and 1. Use tanh as the output layer for the generator.

Experiment sampling z with Gaussian distributions.

Batch normalization often stabilizes training.

Use PixelShuffle and transpose convolution for upsampling.

Avoid max pooling for downsampling. Use convolution stride.

Adam optimizer usually works better than other methods.

Add noise to the real and generated images before feeding them into the discriminator.

Orthogonal Regularization

Training stability depends on the condition of weight matrices

Orthogonal matrices do not amplify input changes

This helps prevent exploding or vanishing signals

Orthogonal regularization penalizes deviation from orthogonality

Strict orthogonality is too restrictive in practice

BigGAN uses a relaxed orthogonal regularization

piece-wise multiplication

matrix with all elements equal to 1

$$R_\beta(W) = \beta \| W^\top W \odot (\mathbf{1} - I) \|_{\mathrm{F}}^2$$

Frobenius norm

Extra Tips and Tricks

There are general tips that can help train GANs

From https://github.com/soumith/ganhacks

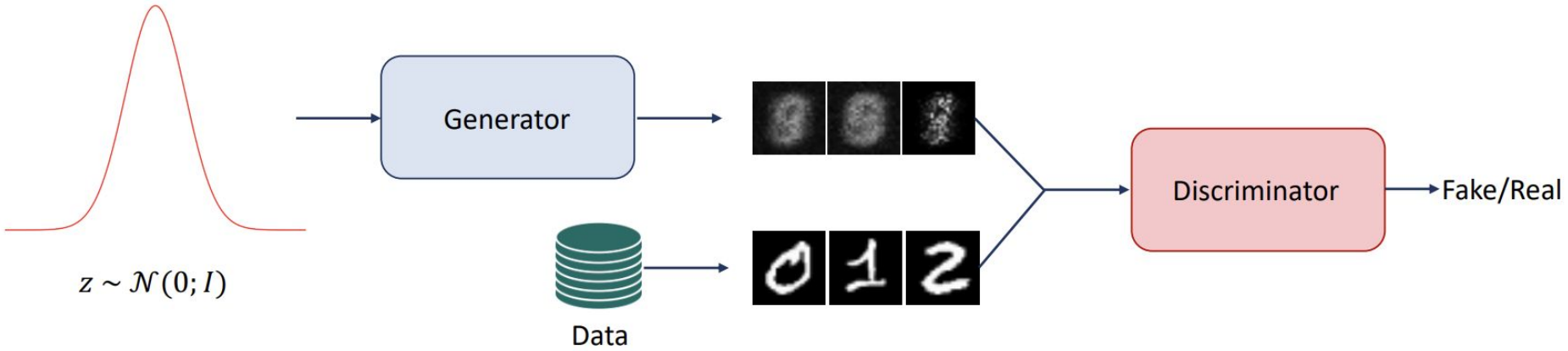
Normalize input in range (-1, 1)

- Use Tanh activation for the last layer of the Generator output

1. Avoid sparse gradients
   - Avoid using ReLU and maxpool
2. Track failures early
   - If Discriminator gradients go to zero: failure mode
   - If there is high variance/spikes in the Discriminator loss: failure mode
   - If loss of Generator steadily decreases it is fooling the Discriminator with garbage
3. Etc.

GAN progress on face generation

GAN MNIST



Reference

https://developers.google.com/machine-learning/gan/gan_structure

https://machinelearningmastery.com/how-to-code-the-generative-adversarial-network-training-algorithm-and-loss-functions/

https://medium.com/data-science/gan-ways-to-improve-gan-performance-acf37f9f59b

Old slides of CMPT743