Generative Adversarial Networks

11785 Deep Learning

Fall 2022

Shreyas Piplani, Aditya Singh, Talha Faiz

Topics for the week

- Transformers
- GNNs
- VAEs
- GANs
- Connecting the dots

Agenda

- Recap of GANs
- GAN Training
- Issues with GAN Training
- Remedies for GAN Training Issues
- GAN Architectures and Recent Progress

Recap

Given a distribution of inputs X and labels Y.

Discriminative models

- Discriminative models learn conditional distribution P(Y | X)
- Learns decision boundary between classes.
- Limited scope. Can only be used for classification tasks.
- E.g. Logistic regression, SVM etc.

Generative models

- Generative models learn the joint distribution P(Y, X)
- Learns actual probability distribution of data P(X)
- Can do both generative and discriminative tasks.
- E.g. Naïve Bayes, Gaussian Mixture Model etc.
- Harder problem, requires a deeper understanding of the distribution than discriminative models.

The problem

- From a large collection of images of faces, can a network learn to *generate* new portrait
  - Can we generate new samples from the distribution of "face" images

Recap

Given a dataset how can we generate more points like it?

Recap

Given a dataset how can we generate more points like it?

1. Approximately model P(x) and sample from it - VAE (explicit models)

Recap

Given a dataset how can we generate more points like it?

1. Approximately model P(x) and sample from it - VAE (explicit models)

2. Model an approximate sampling function of x, but not P(x) itself - GANs (implicit models)

Recap

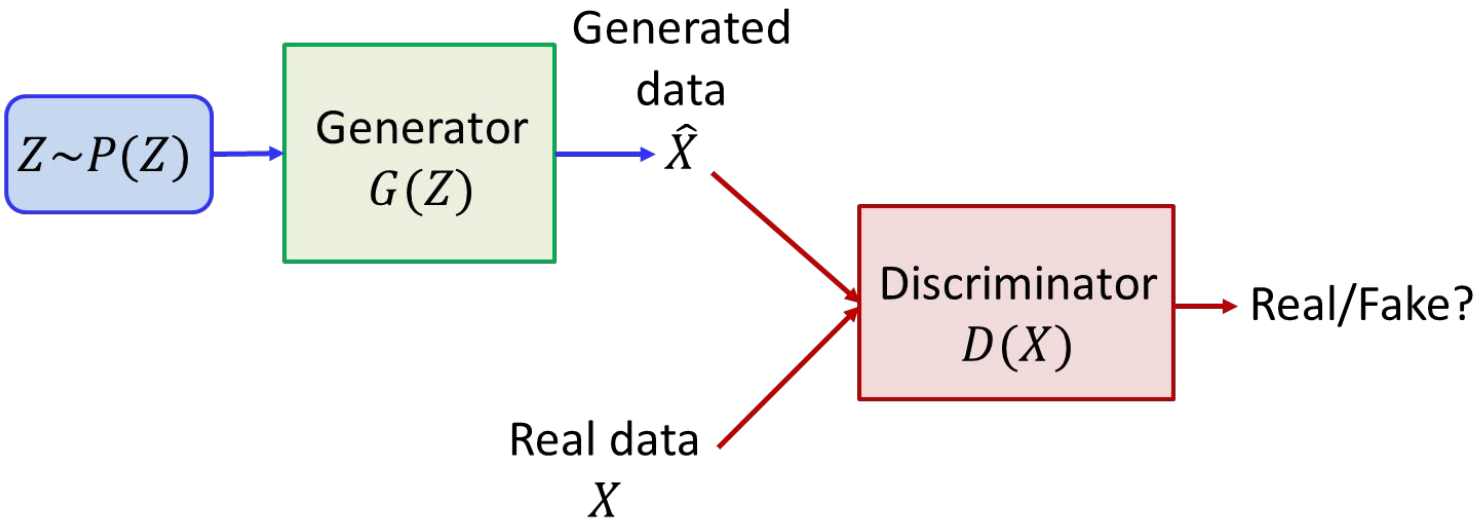
Given a dataset how can we generate more points like it?

1. Approximately model P(x) and sample from it - VAE (explicit models)

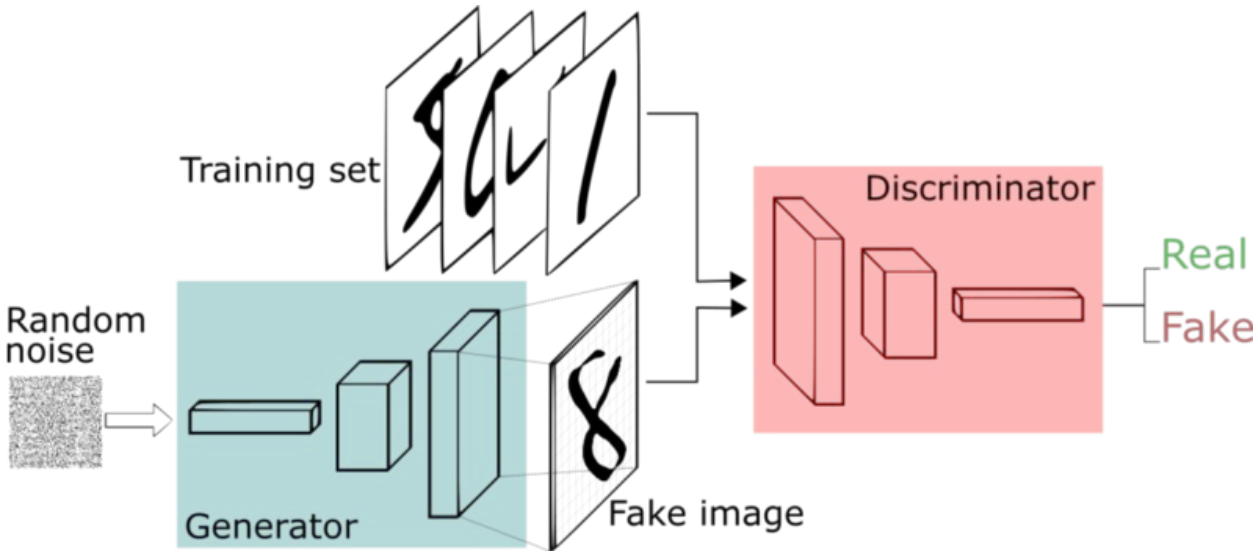
2. Model an approximate sampling function of x, but not P(x) itself - GANs (implicit models)

3. Learn how changing a datapoint x changes its likelihood of being observed P(x) [∇P(x)] - Diffusion Models

GANs

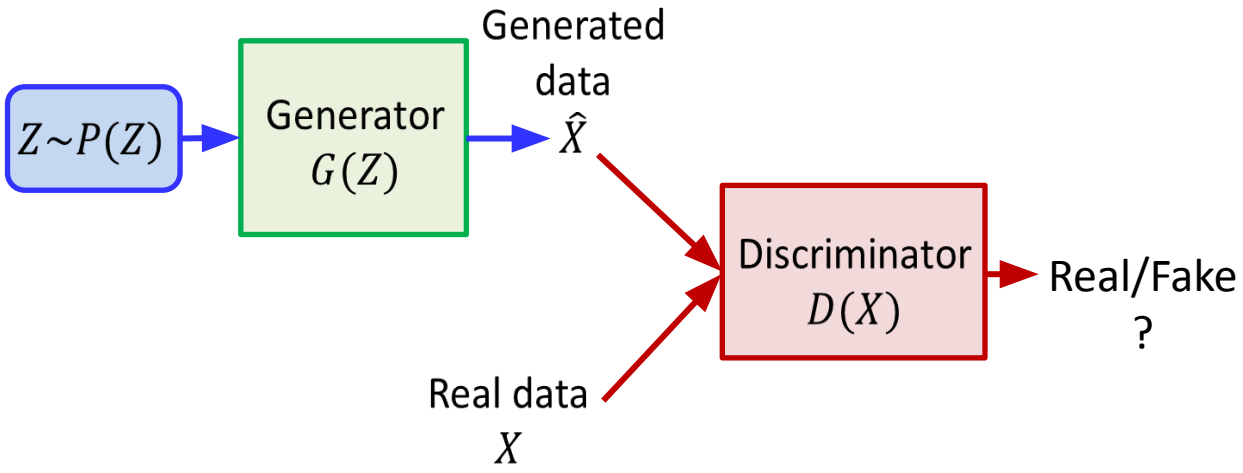

$Z \sim P(Z)$
Generator
$G(Z)$
Generated
data
$\hat{X}$
Discriminator
$D(X)$
Real/Fake?
Real data
X

GANs


Training set
Discriminator
Real
Fake
Random noise
Generator
Fake image

GANs

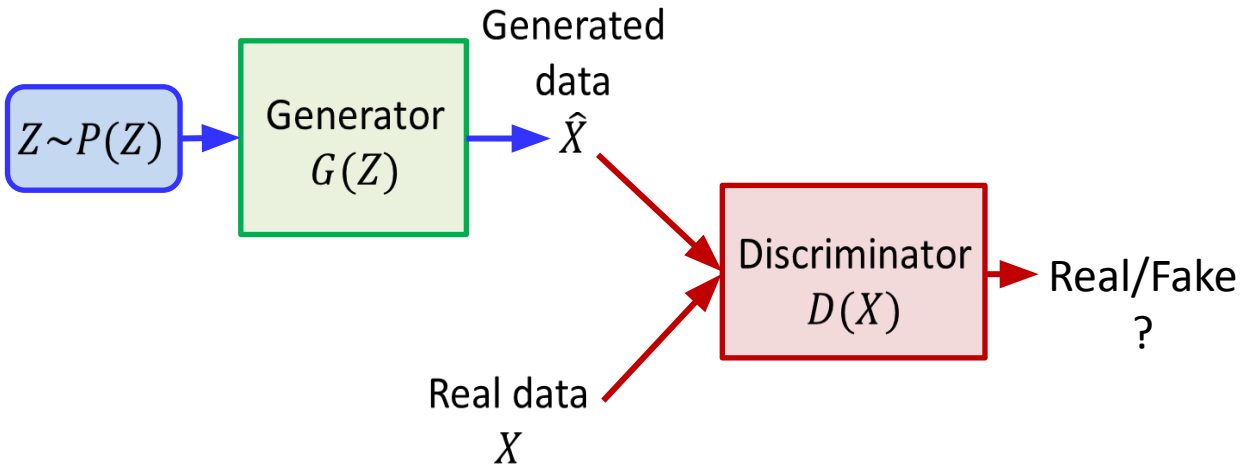
GAN Training - Notation



Description	Notation
Training samples	x
Latent noise vector	z
Discriminator	$D(x;\theta_d)$
Generator	$G(z;\theta_g)$
Probability distribution of real data	P_{data}
Probability distribution of fake data	P_g
Probability distribution of latent noise vector	P_z
Generator output - generated *fake* data	$G(z)$
Discriminator output for *fake* data	$D(G(z))$
Discriminator output for *real* data	$D(x)$

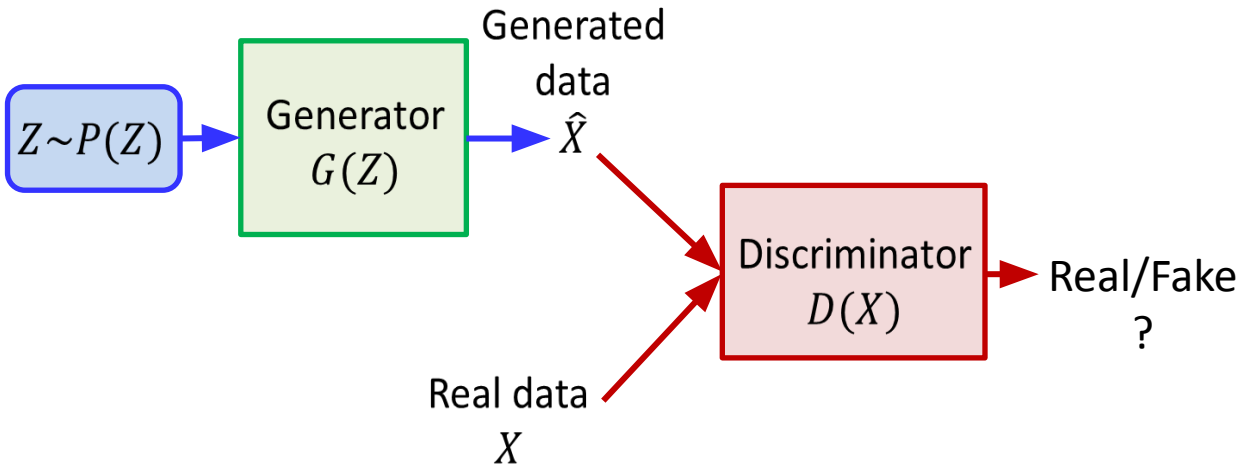
→ N(0,1) or U(-1,1)

The GAN formulation



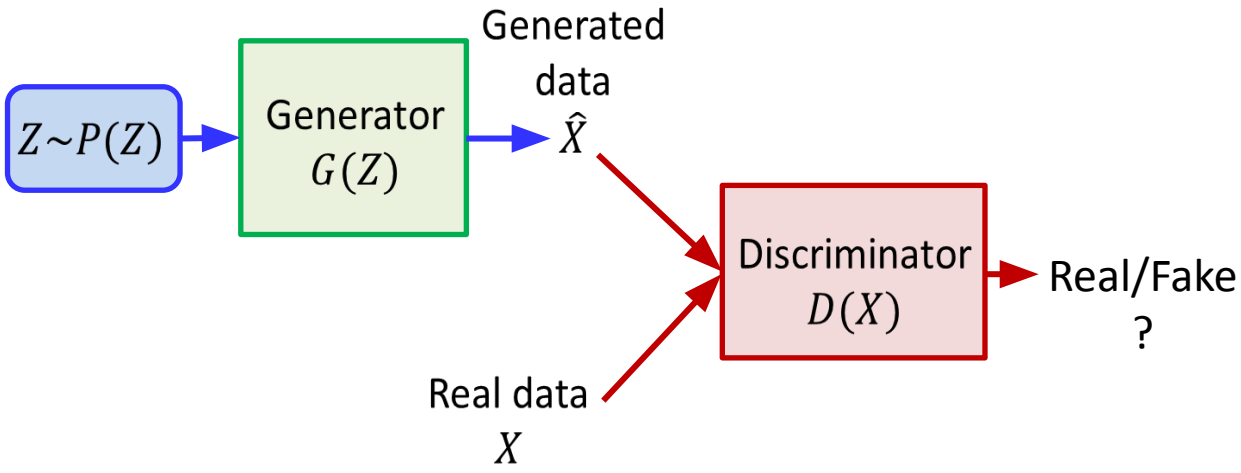
$$\min_G \max_D \; E_{x \sim P_X} \log D(x) + E_{x \sim P_G} \log(1 - D(x)) \quad (1)$$

The GAN formulation



$$\min_G \max_D E_{\boldsymbol{x}\sim \boldsymbol{P}_X} \log D(x) + E_{\boldsymbol{z}\sim \boldsymbol{P}_Z} \log(1 - D(G(z))$$

The GAN formulation



$$\min_G \max_D \boxed{E_{\boldsymbol{x}\sim \boldsymbol{P}_X} \log D(x)} + \boxed{E_{\boldsymbol{z}\sim \boldsymbol{P}_Z} \log(1 - D(G(z))}$$

True **Generated**

GAN Training

Description	Notation
Training samples	x
Latent noise vector	z
Discriminator	$D(x;\theta_d)$
Generator	$G(z;\theta_g)$
Probability distribution of real data	P_{data}
Probability distribution of fake data	P_g
Probability distribution of latent noise vector	P_z
Generator output - generated *fake* data	$G(z)$
Discriminator output for *fake* data	$D(G(z))$
Discriminator output for *real* data	$D(x)$

Discriminator objective:

- maximize output for *real* data $D(x)$
- minimize output for *fake* data $D(G(x))$

OR

- maximize output for *real* data $D(x)$
- maximize output for $1 - D(G(z))$

Generator objective:

- minimize output for *fake* data $1 - D(G(z))$

Since log is a monotonically increasing function:

Discriminator objective:

- maximize output for *real* data $\log(D(x))$
- maximize output for $\log(1 - D(G(z)))$

Generator objective:

- minimize output for *fake* data $\log(1 - D(G(z)))$

GAN Training

Description	Notation
Training samples	x
Latent noise vector	z
Discriminator	$D(x;\theta_d)$
Generator	$G(z;\theta_g)$
Probability distribution of real data	P_{data}
Probability distribution of fake data	P_g
Probability distribution of latent noise vector	P_z
Generator output - generated *fake* data	$G(z)$
Discriminator output for *fake* data	$D(G(z))$
Discriminator output for *real* data	$D(x)$

For one data point:

Discriminator objective:

- $\max_D(\log(D(x)) + \log(1 - D(G(z))))$

Generator objective:

- $\min_G(\log(1 - D(G(z))))$

For the entire distribution:

Discriminator objective:

- $\max_D E_{x\sim P_{data}(x)}[\log D(x)] + E_{z\sim P_z(z)}[1 - \log D(G(z))]$

Generator objective:

- $\min_G E_{z\sim P_z(z)}[1 - \log D(G(z))]$

Which stays same after adding a constant,

- $\min_G E_{x\sim P_{data}(x)}[\log D(x)] + E_{z\sim P_z(z)}[1 - \log D(G(z))]$

GAN Training

Description	Notation
Training samples	x
Latent noise vector	z
Discriminator	$D(x;\theta_d)$
Generator	$G(z;\theta_g)$
Probability distribution of real data	P_{data}
Probability distribution of fake data	P_g
Probability distribution of latent noise vector	P_z
Generator output - generated *fake* data	$G(z)$
Discriminator output for *fake* data	$D(G(z))$
Discriminator output for *real* data	$D(x)$

GAN objective:

- $\min_G \max_D E_{x\sim P_{data}(x)}[\log D(x)] + E_{z\sim P_z(z)}[1 - \log D(G(z))]$
- $\min_G \max_D P_{data}(x)(\log D(x)) + P_g(x)(1 - \log D(x))$

Jointly optimizing min-max is complicated, so we first find the current best D by taking the derivative of GAN objective:

- $-\frac{P_{data}(x)}{D(x)} + \frac{P_g(x)}{1-D(x)} = 0$
- $D(x) = \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$

Substituting this value in the GAN objective for Generator loss:

- $E_{x\sim P_{data}(x)}[\log \frac{P_{data}(x)}{\frac{1}{2}(P_{data}(x)+P_g(x))}] + E_{z\sim P_z(z)}[\log \frac{P_{data}(x)}{\frac{1}{2}(P_{data}(x)+P_g(x))}] - 2.\log 2$

Kullback-Leibler(KL) and Jensen-Shannon(JS) divergences are given by:

- $KL(P_1||P_2) = E_{x\sim P_1(x)}[\log \frac{P_1}{P_2}]$
- $JSD(P_1||P_2) = \frac{1}{2}KL(P_1||\frac{P_1+P_2}{2}) + \frac{1}{2}KL(P_2||\frac{P_1+P_2}{2})$

This makes the Generator loss:

- $2.JSD(P_{data}||P_g) - 2.log2$

GAN Training Routine (Ian Goodfellow et al.)

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.

Poll 1

@1717

Once we have the optimal discriminator D*, which options are true at global minimum of the resultant Generator loss L (Hint: JSD)

- Achieved iff $P_X=P_G$
- Value of L at global min. is -log4
- At global min. D(x)=½

Poll 1

@

Once we have the optimal discriminator D*, which options are true at global minimum of the resultant Generator loss L (Hint: JSD)

- Achieved iff $P_X=P_G$
- Value of L at global min. is -log4
- At global min. D(x)=½

GAN Training

Once we have the optimal Discriminator D^* the generator objective L(G) is-

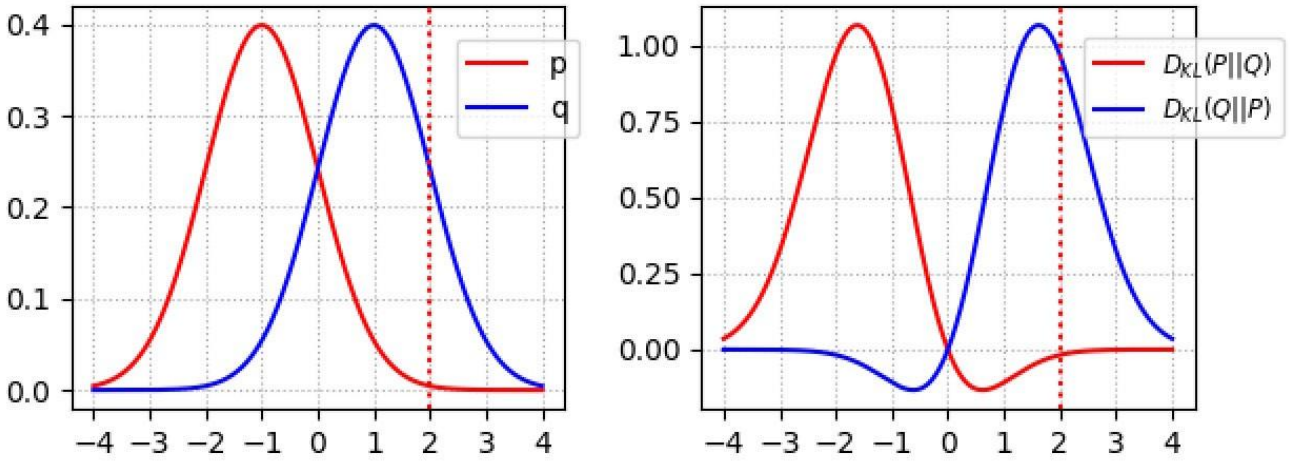
$$L = 2D_{JSD}\left(P_X(x), P_G(x)\right) - \log 4$$

- Theorem 1: Global minimum of L achieved if and only if $P_X = P_G$ and at that point L = -log4 and D(x)=1/2

- Theorem 2: If G and D have enough capacity and we let D reach the optimum given G and P_g to improve Eqn(1) then P_G converges to P_{Data}

(proofs in appendix)

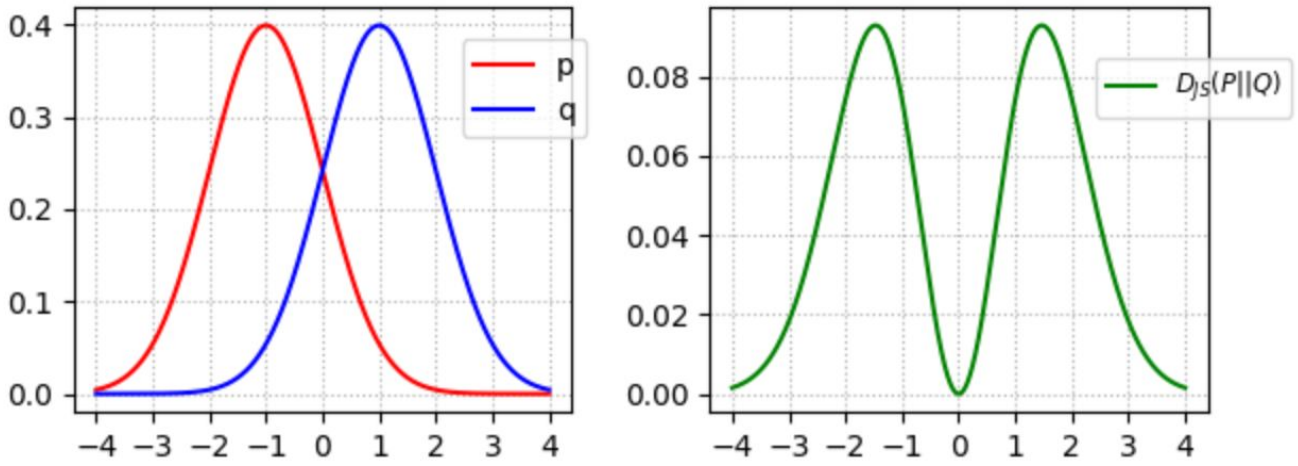
Qualitative Effects of *JS-Divergence*



- The KL-divergence DL(p, q) penalizes the generator if it misses some modes of images: the penalty is high where p(x) > 0 but q(x) → 0. Nevertheless, it is acceptable that some images do not look real. The penalty is low when p(x) → 0 but q(x) > 0.
- Poorer quality but more diverse samples

https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Qualitative Effects of *JS-Divergence*



- JS-divergence is symmetrical. Unlike KL-divergence, it will penalize poor images badly, but can allow less diversity

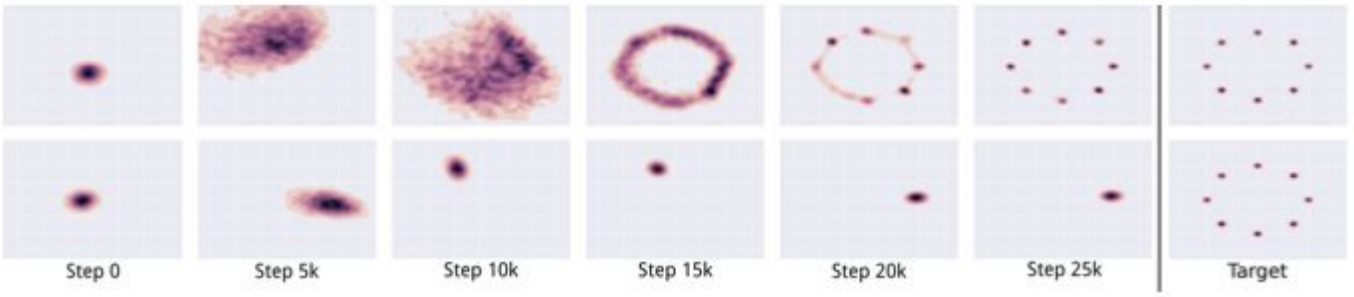
https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

GAN Training Issues

- Mode Collapse
- Vanishing Gradient
- Convergence and Oscillation

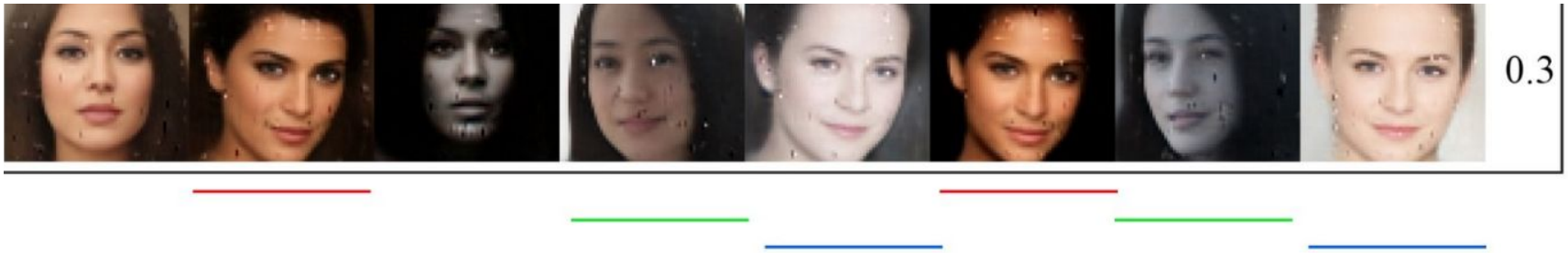
GANs Training Issues - Mode Collapse

- If a generator produces an especially plausible output, the generator may learn to produce only that output
- The **Generator gets stuck** at a point where it only produces a limited variety of samples or one sample repeatedly during or after training
- Each iteration of Generator over-optimizes for a particular Discriminator, and the Discriminator never manages to learn its way out of the trap



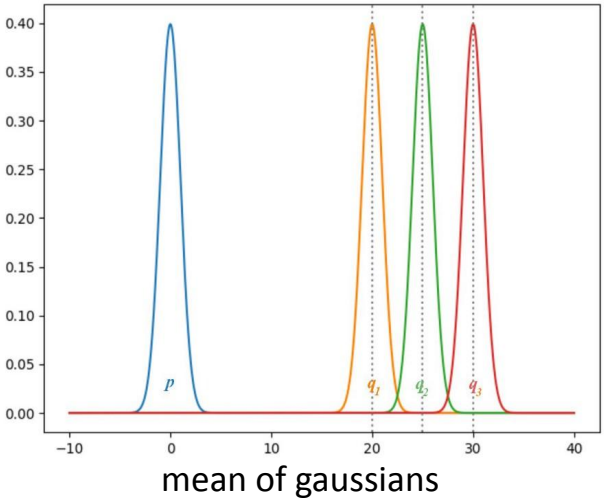
https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

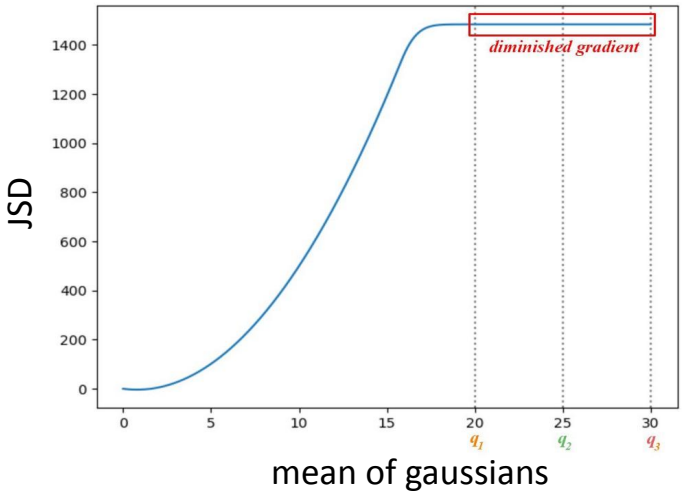
GANs Training Issues - Mode Collapse



https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

GANs Training Issues - Vanishing Gradients





- If the Discriminator is too good, then the Generator training can fail due to vanishing gradients. An optimal Discriminator doesn't provide enough information for the Generator to make progress
- Below image shows how gradients will vanish if the distribution of generated images (*p*) is too different than the distribution of real images (*q1, q2, q3*)

https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

GANs Training Issues - Convergence and Oscillation

- GAN training is based on a zero-sum, non-cooperative, minmax game. In short, if one wins the other loses

- In game theory, the GAN model converges when the discriminator and the generator reach a Nash equilibrium. This is the optimal point for the GAN objective

- Simplified example:

  Consider two player A and B which control the value of *x* and *y* respectively.

  Player A wants to maximize the value *xy* while B wants to minimize it

GANs Training Issues - Convergence and Oscillation

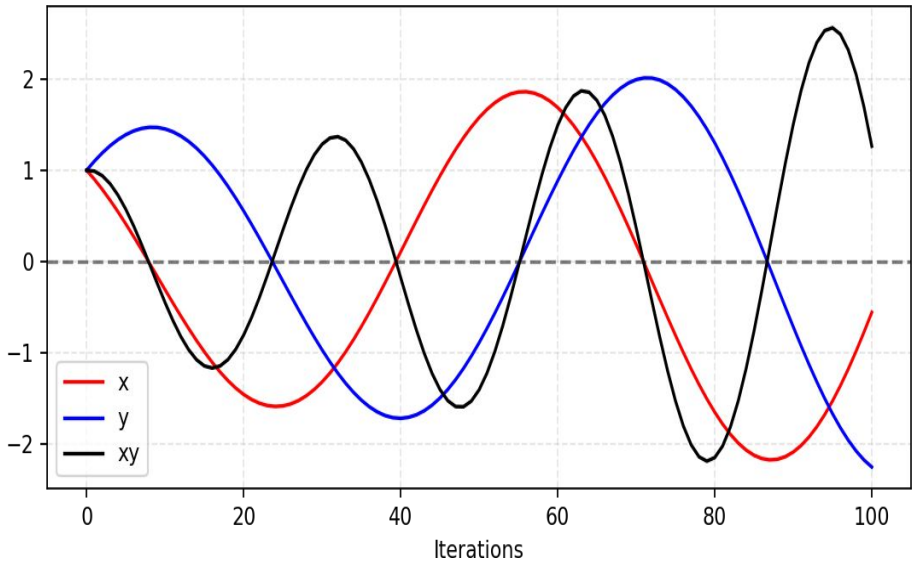
- We update the parameter *x* and *y* based on the gradient of the functions -

  $f(x) = \max_x xy$, $f(y) = \max_y -xy$

  $\partial f/\partial x = y$ and $\partial f/\partial y = -x$

  $x \rightarrow x - \alpha \cdot y$ and $y \rightarrow y + \alpha \cdot x$ (α is learning rate)

GANs Training Issues - Convergence and Oscillation



- $\partial f/\partial x = y$ and $\partial f/\partial y = -x$

  $x \longrightarrow x - \alpha \cdot y$ and $y \longrightarrow y + \alpha \cdot x$ (α is learning rate)

- The Nash equilibrium is **x = y = 0**. This is the only state where the action of your opponent does not matter. It is the only state that any opponents' actions will not change the game outcome

https://medium.com/deep-math-machine-learning-ai/ch-14-general-adversarial-networks-gans-with-math-1318faf46b43

Remedies

- Feature Matching
- Mini-batch Discrimination
- Historical Averaging
- One-sides label smoothing
- Virtual Batch Normalisation

Feature Matching

$$\|E_X D(X) - E_Z D(G(Z))\|_2^2$$

$$\|E_X f(X) - E_Z f(G(Z))\|_2^2$$

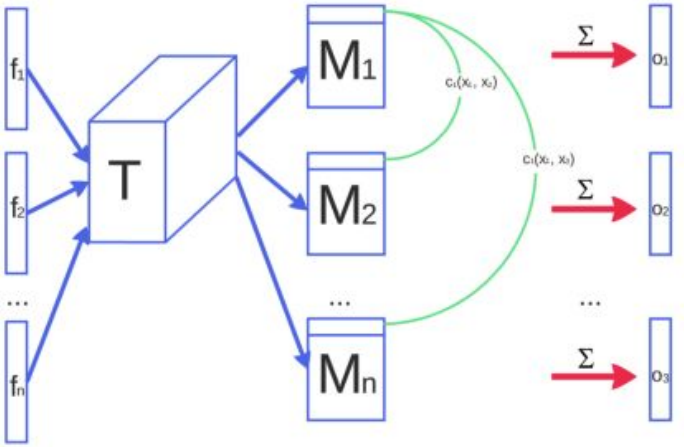
Statistics of generated images should match statistics of real images.
Discriminator **produces multidimensional output, a "statistic" of data**.
Generator trained to minimize L2 between real and generated data.
Discriminator trained to maximize L2 between real and generated data.
Goal: matching features in real images

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

Minibatch Discrimination

Discriminator can look at multiple inputs at once and decide if those inputs come from the real or generated distribution.

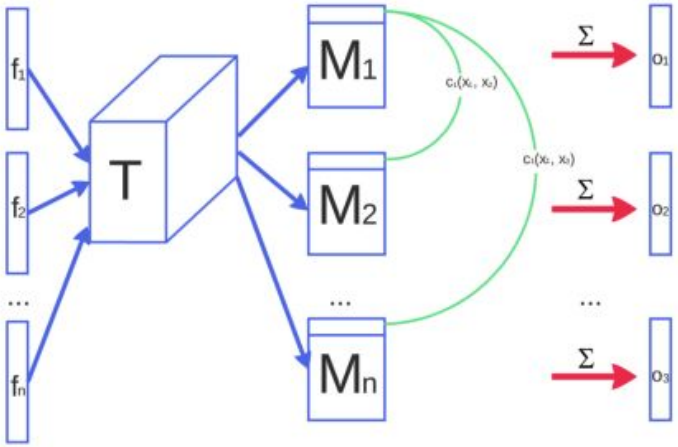
- GANs frequently collapse to a single point
- Discriminator needs to differentiate between two distributions
- Easier task if looking at multiple samples



Append the **similarity between the image and other images in the same batch** in **one of the dense layers in the discriminator** to classify whether this image is real or generated.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

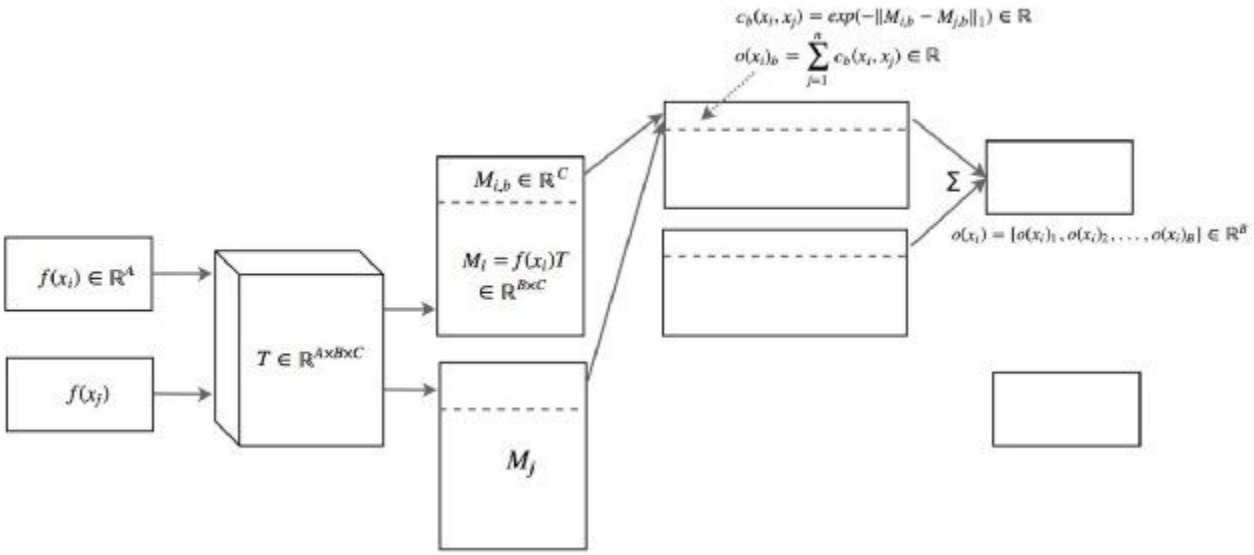
Minibatch Discrimination



$f(x_i) \in R^A$: vector of features for input x_i

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

Minibatch Discrimination



Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

Historical averaging

Dampen oscillations by encouraging updates to converge to a mean.

- GANs frequently create a cycle or experience oscillations
- **Add a term** to reduce oscillations that encourage the current parameters to be near a **moving average of the parameters**

$$\left\| \theta - \frac{1}{t} \sum_{i}^{t} \theta_i \right\|_2^2$$

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

One-sided Label Smoothing

Don’t over-penalize generated images

- Label smoothing is a common and easy technique that improves performance across many domains - reduces overconfidence

  - Sigmoid tries to saturate to 0 or 1 but can never quite reach that goal

  - Provide targets that are epsilon or 1- epsilon so the sigmoid doesn’t saturate

  - Sigmoid : 0->0.1 and 1->0.9

- Experimentally, **smooth the real targets** but **do not smooth the generated targets** when training the discriminator.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

One-sided Label Smoothing

Replacing positive targets (True) with $\boldsymbol{\alpha}$
and negative targets (Generator) with $\boldsymbol{\beta}$

The optimal discriminator becomes -

$$D(\boldsymbol{x}) = \frac{\alpha p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

One-sided Label Smoothing

Replacing positive targets (True) with $\boldsymbol{\alpha}$
and negative targets (Generator) with $\boldsymbol{\beta}$

The optimal discriminator becomes -

$$D(\boldsymbol{x}) = \frac{\alpha p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

One-sided Label Smoothing

Replacing positive targets (True) with $\boldsymbol{\alpha}$

The optimal discriminator becomes -

$$D(\boldsymbol{x}) = \frac{\alpha p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

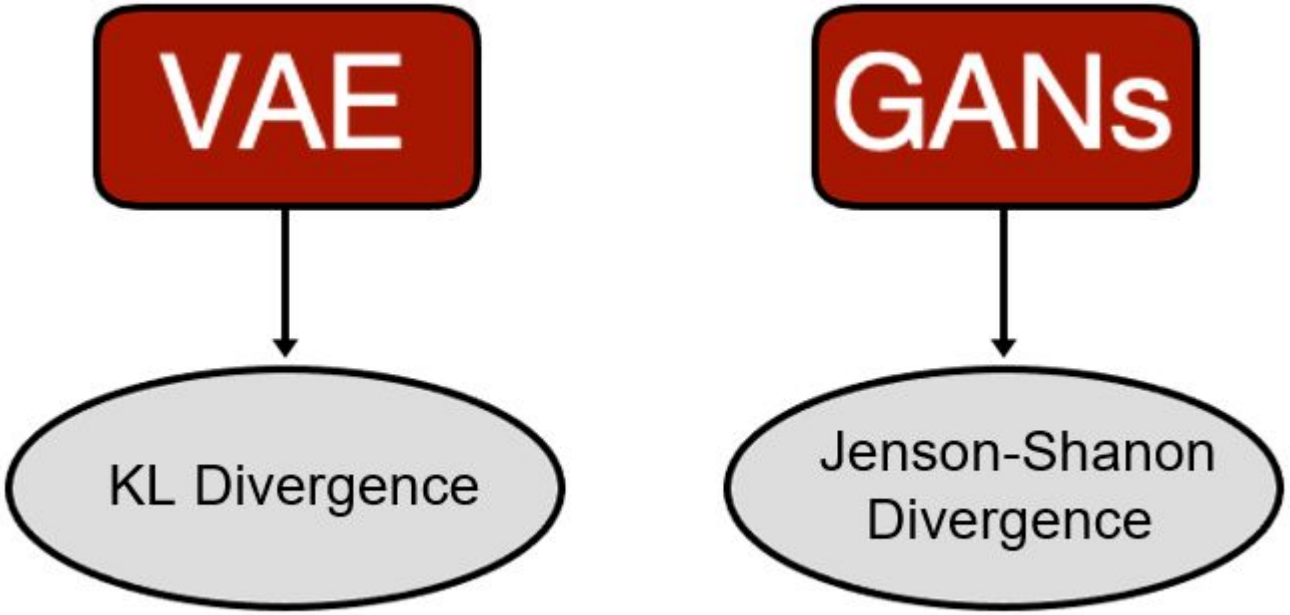
Virtual Batch Normalization

Use batch normalization to accelerate convergence

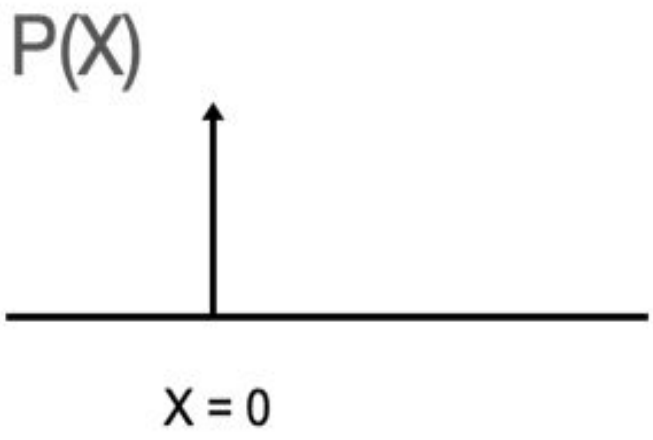
- Batch normalization accelerates convergence
- However, hard to apply in adversarial setting
- Collect statistics on **fixed batch** of real data and use to normalize other data.

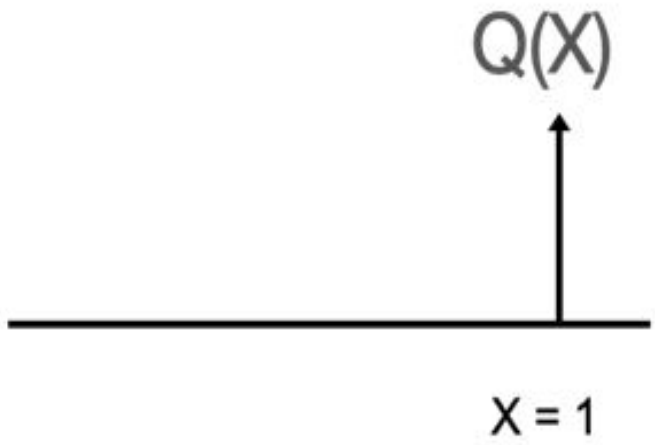
Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, Improved techniques for training gans, CoRR abs/1606.03498 (2016).

Recap



Special case





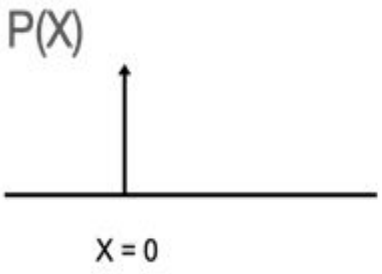
KL Divergence

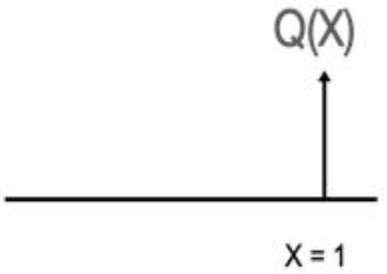
Let θ be the distance between the two peaks of the distribution.

If θ ≠ 0, $$KL(P\|Q) = 1log(\frac{1}{0}) = \infty$$

If θ = 0, $$KL(P\|Q) = 1log(\frac{1}{1}) = 0$$

Not differentiable w.r.t θ





Jenson-Shannon Divergence

$$m(X) = \frac{P_D + P_G}{2}$$

$$JS(P_D \| P_G) = \frac{1}{2} KL(P_D \| m) + \frac{1}{2} KL(P_G | m)$$

If θ ≠ 0, $JSD(P\|Q) = 0.5 * (1log(\frac{1}{0.5}) + 1log(\frac{1}{0.5})) = log2$

If θ = 0, $JSD(P\|Q) = 0.5 * (1log(\frac{1}{1}) + 1log(\frac{1}{1})) = 0$

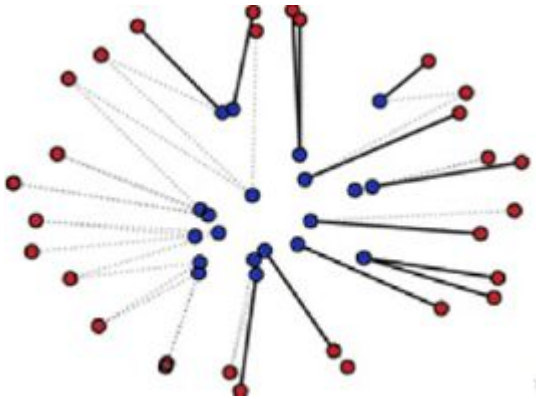
Constant to θ

Wasserstein distance

Also called Kantorovich–Rubinstein metric.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right]$$

- A distance function defined between probability distributions on a given metric space.
- Minimum cost of turning one pile of dirt into another pile of dirt, when both distributions are treated as pile of dirt.
- The total **(Σ mass) × mean** distance required to transform one distribution to another i.e the amount of dirt that needs to be moved times the mean distance it has to be moved.



Red points, Blue points represent two different distributions.

Wasserstein distance

Let θ be the distance between the two peaks of the distribution.

$$W(P,Q) = |\theta|$$

Differentialble w.r.t θ

WGAN

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right]$$

Kantorovich-Rubinstein duality

$$\min_G \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r} \left[D(\boldsymbol{x}) \right] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g} \left[D(\tilde{\boldsymbol{x}}) \right]$$

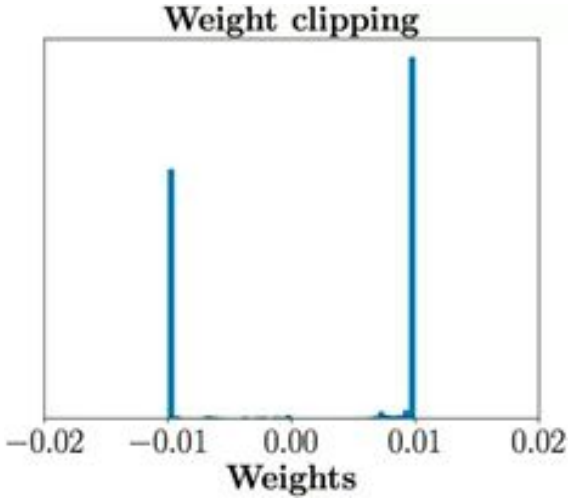
'D' should be a 1-Lipschitz function: $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$

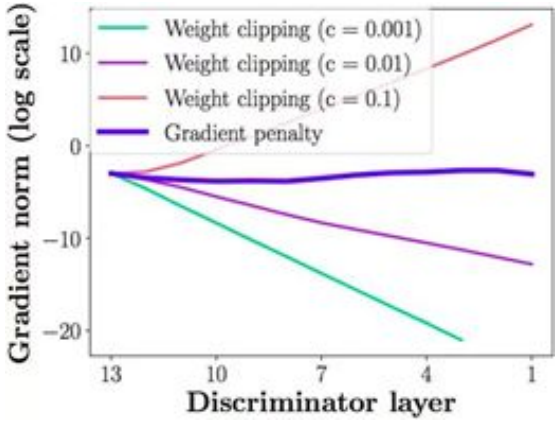
- A function is K-Lipschitz if its gradients are at most K everywhere.
- W(Pr, Pg) is continuous everywhere, and differentiable almost everywhere

WGAN

Weight Clipping:

- Restricts weights between [-c, c]





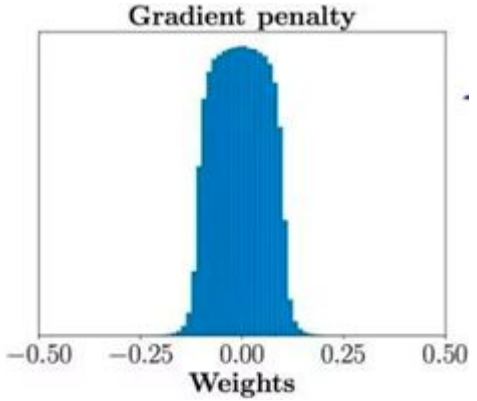
WGAN

Gradient penalty:

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}}.$$

Gradient penalty introduces a softer constraint on gradients

- more stable training
- requires very little hyper-parameter tuning



GAN architectures

There are many variations of GANs for modeling different tasks. This is not meant to be exhaustive but a sample of the probabilities.

- GAN
- Conditional GAN
- LapGAN
- BiGAN
- Recurrent Adversarial Network
- Categorical GAN
- InfoGAN
- AAE
- CycleGAN

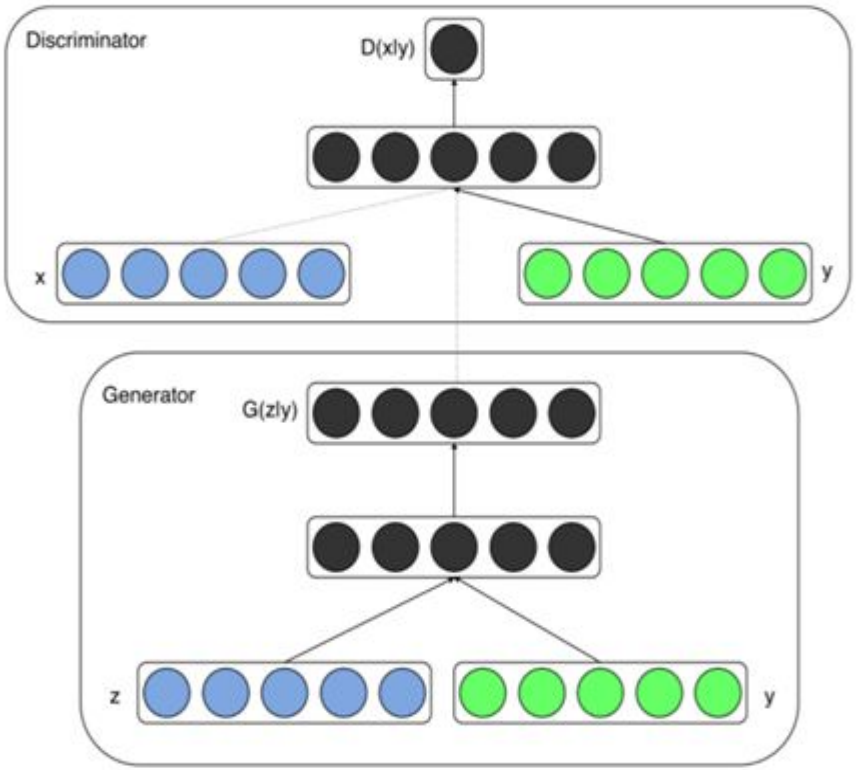
Conditional GAN

- A conditional GAN models *P(X | Y).* For example, generate samples of MNIST conditioned on the digit you are generating. The model is constructed by adding the labels Y as an input to both generator and discriminator

$$\min_G \max_D V(D,G) = \mathbb{E}_X \log D(X,Y) + \mathbb{E}_Z \log D(G(Z,Y),Y)$$

Conditional GAN

Architecture:



Conditional GAN

Results for the MNIST experiment:

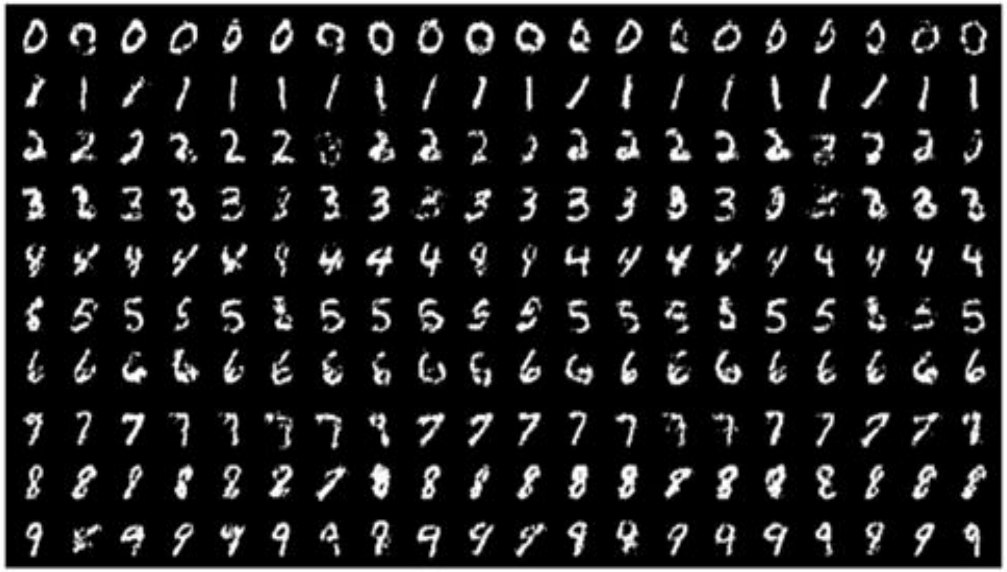


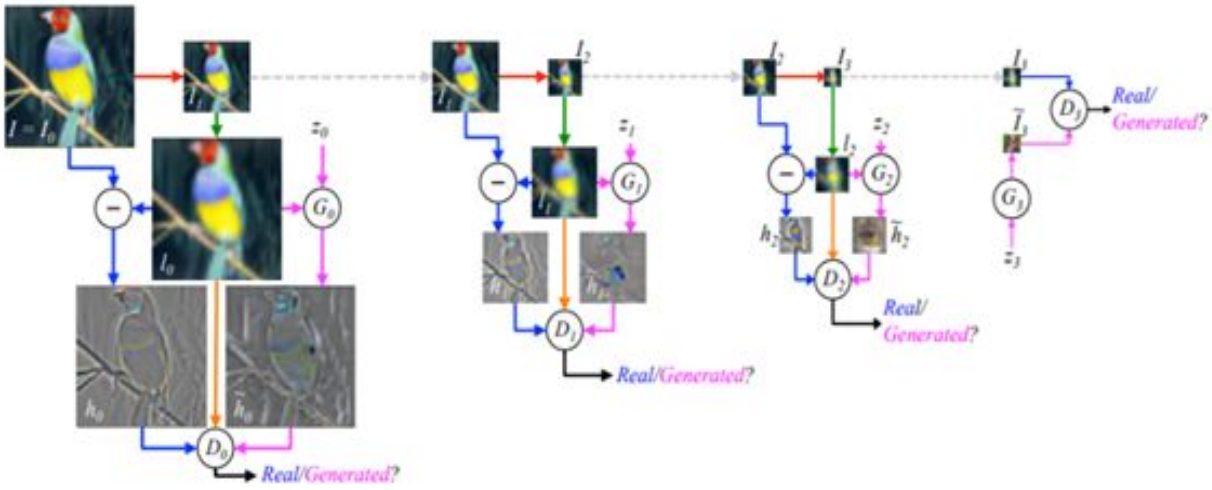
Figure 2: Generated MNIST digits, each row conditioned on one label

LapGAN

A Laplacian GAN is constructed of a chain of conditional GANs, to generate progressively larger images. A GAN generates small, blurry images. A conditional GAN generates larger images conditioned on the smaller image, repeated until you reach the desired size.

LapGAN

Architecture:



POLL

@1718

POLL

Poll 2: The main difference between a Vanilla GAN and Wasserstein GAN is that in Wasserstein GAN, the discriminator is restricted to have a bounded Lipshcitz norm

a. True
b. False

Original paper (GAN, 2014)

Output of original GAN paper, 2014 [GPM+14]

a) b) c) d)

GANs with time

- Better quality
- High Resolution

https://twitter.com/goodfellow_ian/status/1084973596236144640?lang=en

StarGAN(2018)

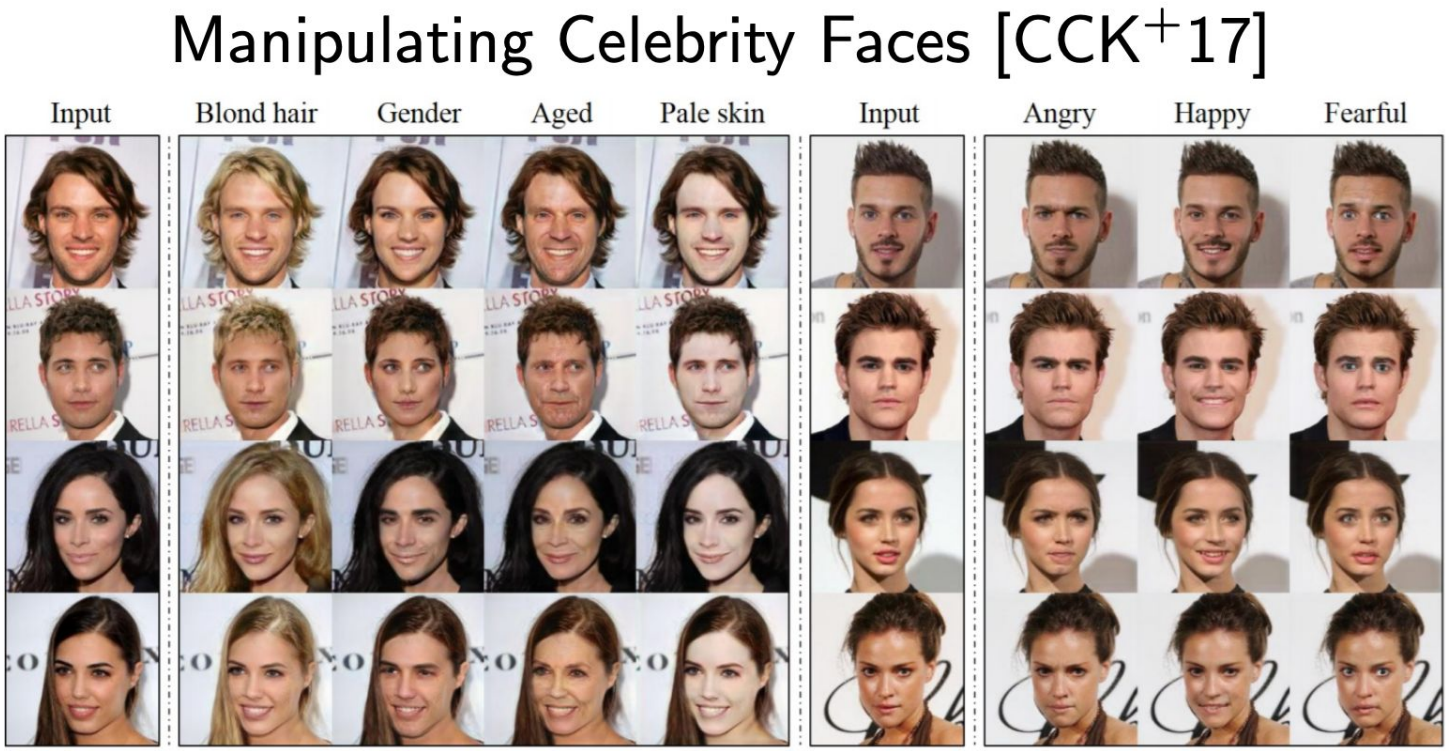


Figure 1. Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

Progressive growing of GANs (2018)

Figure 5: 1024×1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

High fidelity natural images (2019)

BiGAN (2016)

- Introduced by Jeff Donahue et.al. in “Adversarial Feature Learning”
- A BiGAN, or Bidirectional GAN, is a type of generative adversarial network where the generator not only maps latent samples to generated data, but also has an inverse mapping from data to the latent representation. The motivation is to make a type of GAN that can learn rich representations for us in applications like unsupervised learning.

BiGAN Architecture

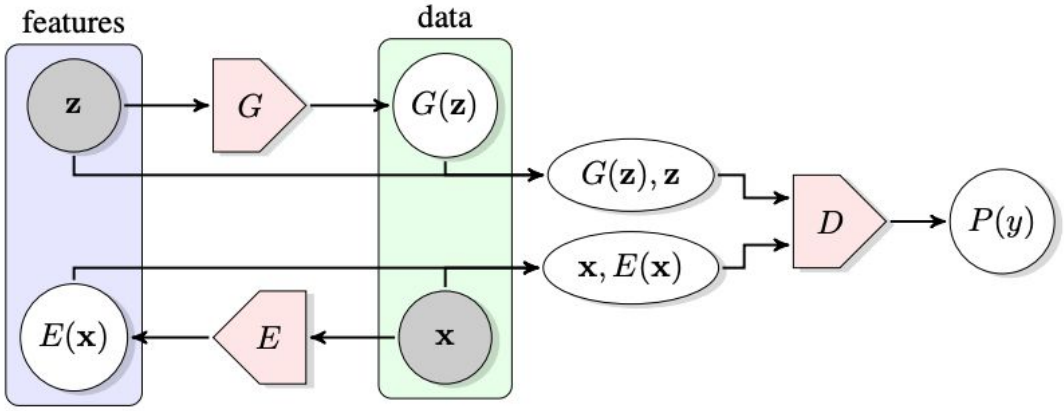
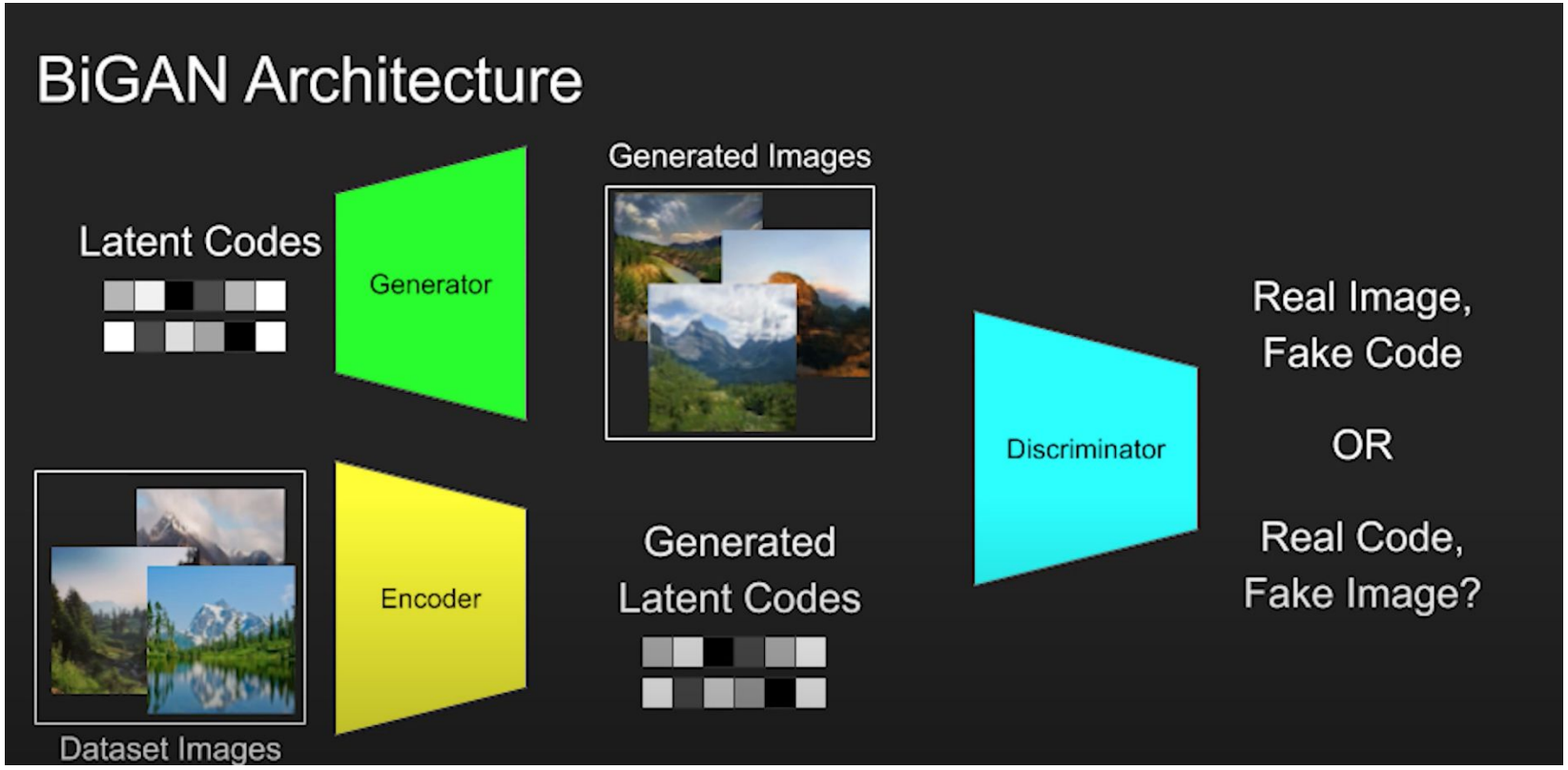


Figure 1: The structure of Bidirectional Generative Adversarial Networks (BiGAN).

- In addition to the generator G from the standard GAN framework, BiGAN includes an encoder E which maps data x to latent representations z.
- The BiGAN discriminator D discriminates not only in data space (x versus G(z)), but jointly in data and latent space (tuples (x, E(x)) versus (G(z), z)), where the latent component is either an encoder output E(x) or a generator input z

BiGAN Architecture



BiGAN Training Objective

The BiGAN training objective is defined as a minimax objective

$$\min_{G,E} \max_{D} V(D, E, G) \tag{2}$$

where

$$V(D, E, G) := \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{X}}} \Big[\underbrace{\mathbb{E}_{\mathbf{z} \sim p_E(\cdot|\mathbf{x})} \left[\log D(\mathbf{x}, \mathbf{z}) \right]}_{\log D(\mathbf{x}, E(\mathbf{x}))} \Big] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{Z}}} \Big[\underbrace{\mathbb{E}_{\mathbf{x} \sim p_G(\cdot|\mathbf{z})} \left[\log \left(1 - D(\mathbf{x}, \mathbf{z}) \right) \right]}_{\log(1 - D(G(\mathbf{z}), \mathbf{z}))} \Big]. \tag{3}$$

- Minimax objective optimized using alternating gradient based optimization as Goodfellow et al.
- Benchmarked using classification accuracy (%)

BiGAN Qualitative Results

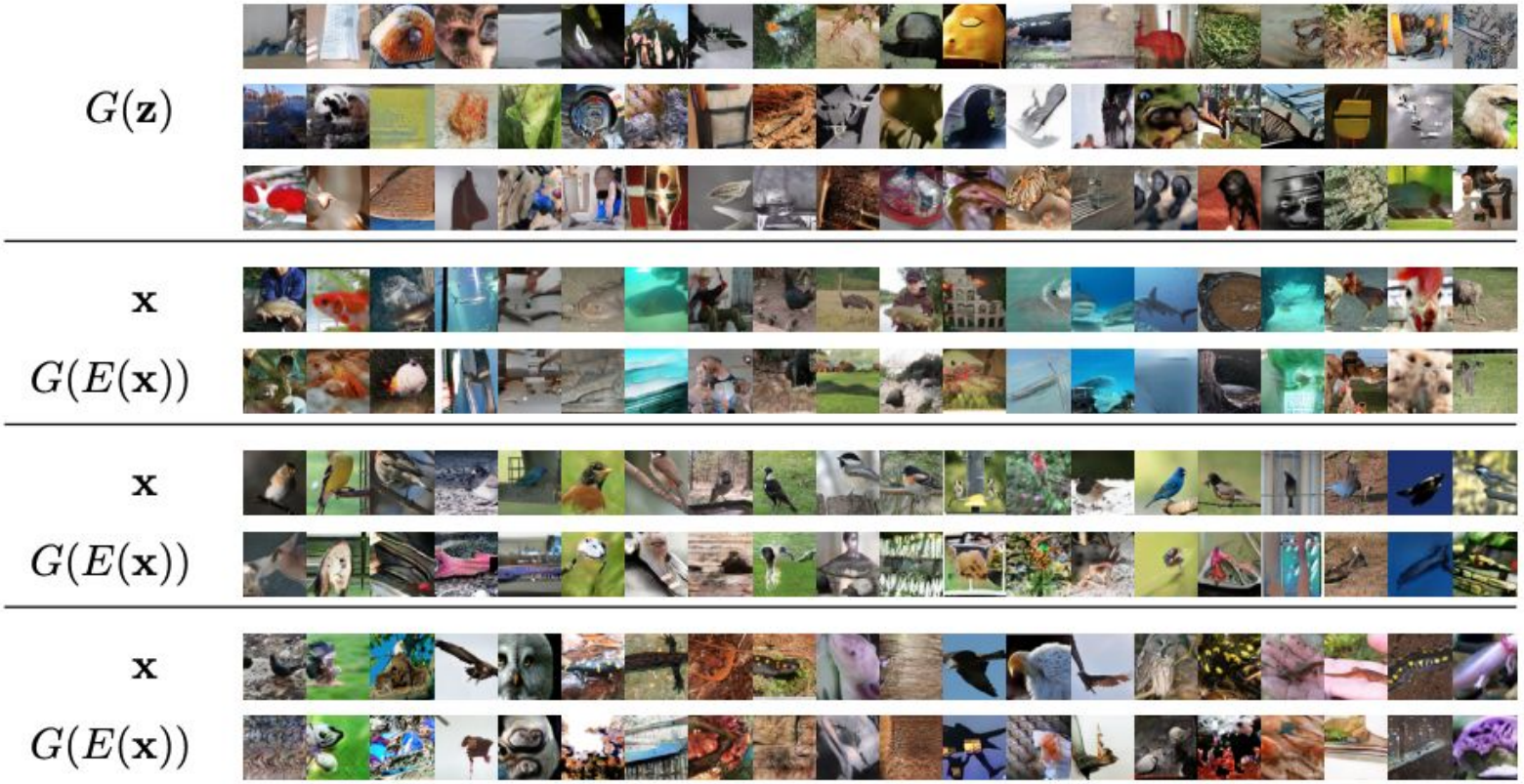


Figure 4: Qualitative results for ImageNet BiGAN training, including generator samples $G(\mathbf{z})$, real data $\mathbf{x}$, and corresponding reconstructions $G(E(\mathbf{x}))$.

BiGAN Takeaways

- > unsup. and >= self-sup. feature learning approaches specific to the visual domain
- Domain agnostic
- Needn't suffer from domain shift

Can learn feature representations from unlabeled static images

		Classification (% mAP)			*FRCN* Detection (% mAP)	*FCN* Segmentation (% mIU)
	trained layers	fc8	fc6-8	all	all	all
sup.	ImageNet (Krizhevsky et al., 2012)	**77.0**	**78.8**	**78.3**	**56.8**	**48.0**
self-sup.	Agrawal et al. (2015)	31.2	31.0	54.2	43.9	-
	Pathak et al. (2016)	30.5	34.6	56.5	44.5	**30.0**
	Wang & Gupta (2015)	28.4	**55.6**	63.1	47.4	-
	Doersch et al. (2015)	**44.7**	55.1	**65.3**	**51.1**	-
unsup.	k-means (Krähenbühl et al., 2016)	32.0	39.2	56.6	45.6	32.6
	Discriminator (D)	30.7	40.5	56.4	-	-
	Latent Regressor (LR)	36.9	47.9	57.1	-	-
	Joint LR	37.1	47.9	56.5	-	-
	Autoencoder (ℓ_2)	24.8	16.0	53.8	41.9	-
	BiGAN (ours)	37.5	48.7	58.9	46.2	34.9
	BiGAN, 112×112 E (ours)	**41.7**	**52.5**	**60.3**	**46.9**	**35.2**

Table 3: Classification and Fast R-CNN (Girshick, 2015) detection results for the PASCAL VOC 2007 (Everingham et al., 2014) test set, and FCN (Long et al., 2015) segmentation results on the PASCAL VOC 2012 validation set, under the standard mean average precision (mAP) or mean intersection over union (mIU) metrics for each task. Classification models are trained with various portions of the *AlexNet* (Krizhevsky et al., 2012) model frozen. In the *fc8* column, only the linear classifier (a multinomial logistic regression) is learned – in the case of BiGAN, on top of randomly initialized fully connected (FC) layers *fc6* and *fc7*. In the *fc6-8* column, all three FC layers are trained fully supervised with all convolution layers frozen. Finally, in the *all* column, the entire network is "fine-tuned". BiGAN outperforms other unsupervised (*unsup.*) feature learning approaches, including the GAN-based baselines described in Section 4.1, and despite its generality, is competitive with contemporary self-supervised (*self-sup.*) feature learning approaches specific to the visual domain.

CycleGAN (2017)

- Introduced by Jun-Yan Zhu (currently a CMU professor) et.al. in "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks"
- Trains a pair of Conditional GAN's to perform image-to-image translation
  - GAN A trained to convert from X to Y
  - GAN B trained to convert from Y to X
  - Additional "cycle-consistency" losses

CycleGAN Results

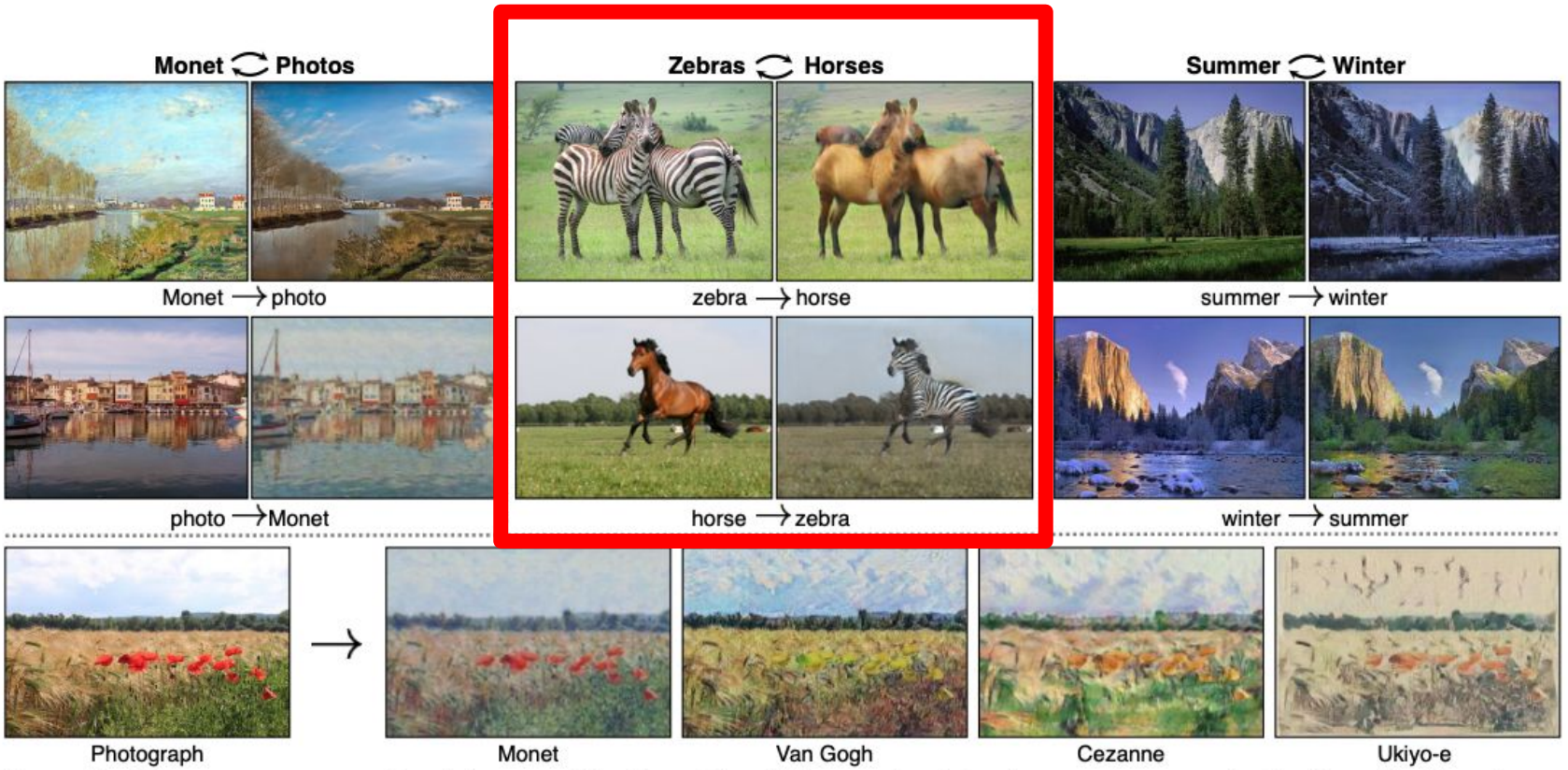
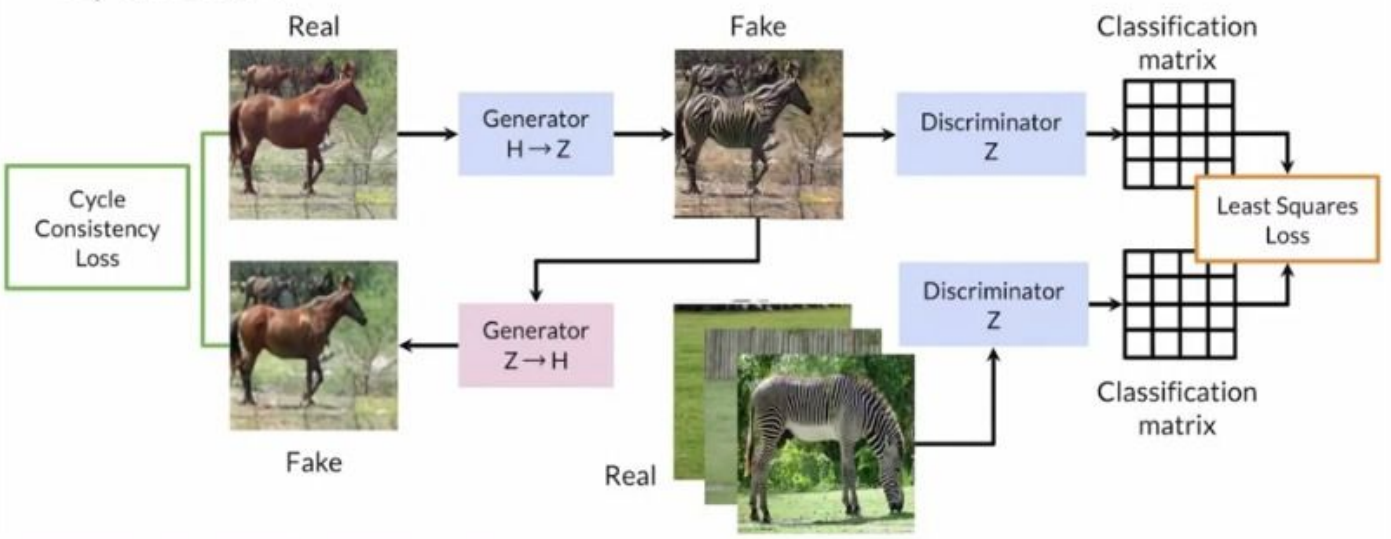


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

CycleGAN Architecture



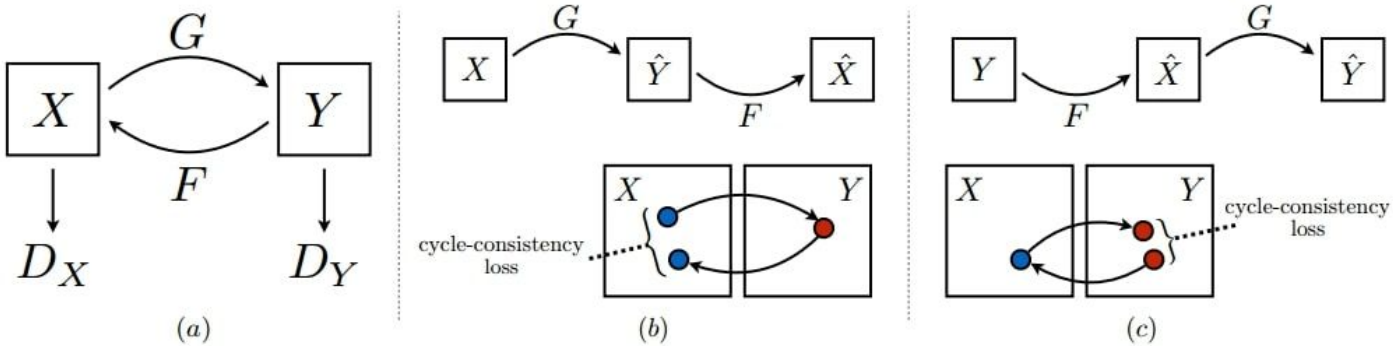
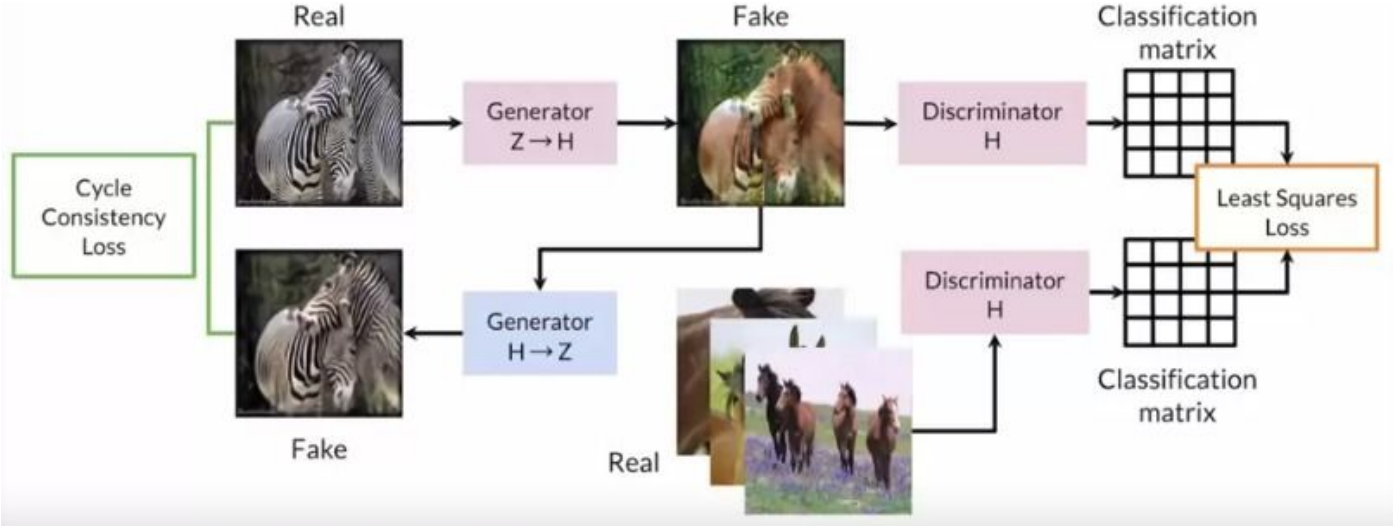


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, we introduce two "cycle consistency losses" that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

CycleGAN Architecture



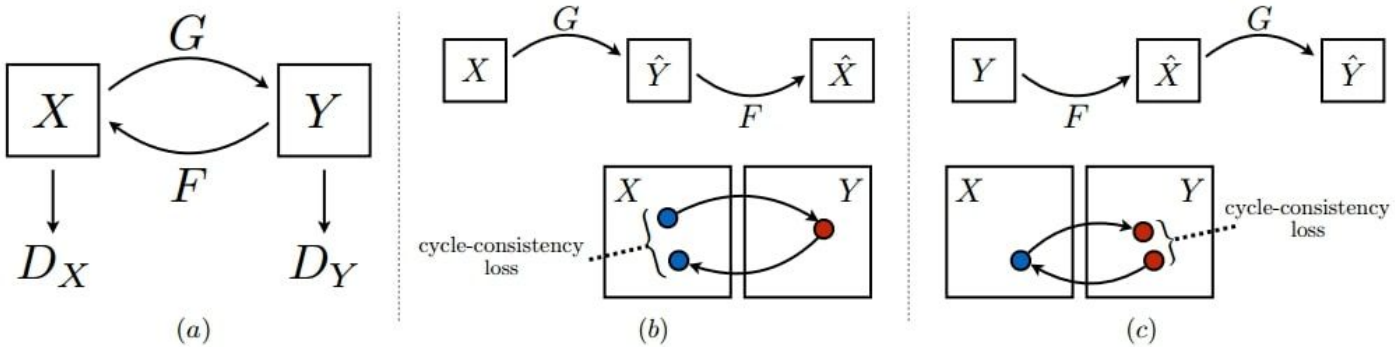


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, we introduce two "cycle consistency losses" that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Training Objective

- Adversarial Loss: $$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \quad (1)$$

- Cycle Consistency Loss: $$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]. \quad (2)$$

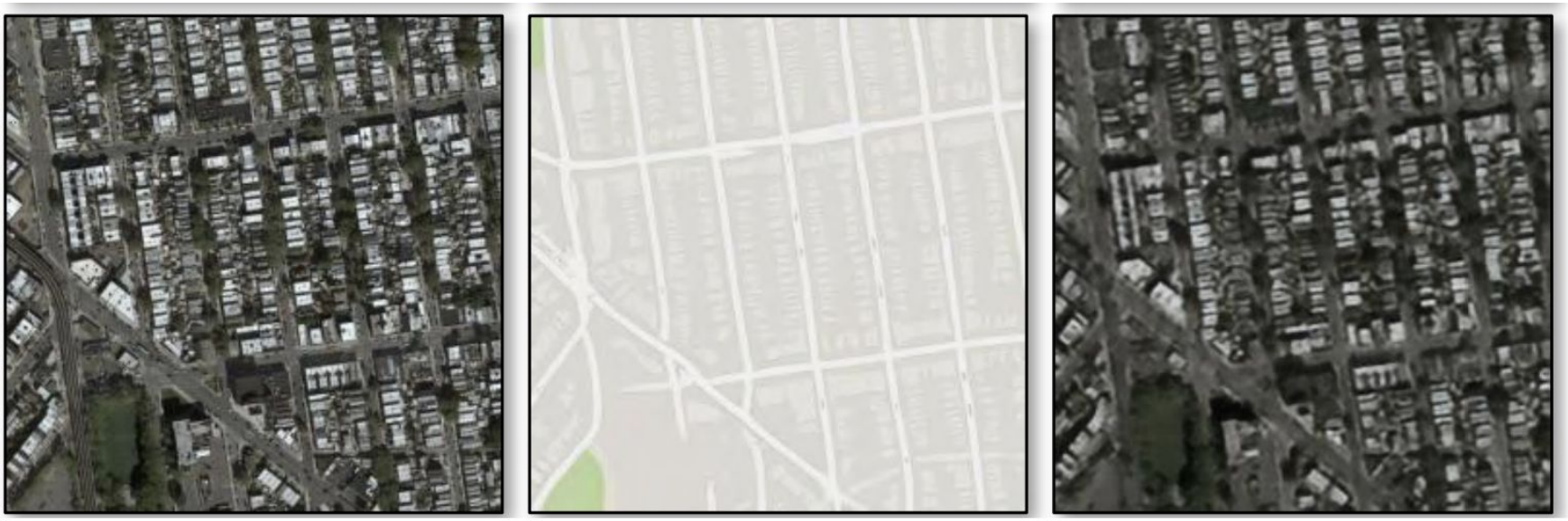
Our full objective is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

We aim to solve:

$$G^*, F^* = \arg \min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y).$$

Application 1 - Google Maps



Convert satellite imagery to vector maps (and vice-versa if needed)

Application 2 - Night to Day

Input Left Image

Input Right Image

Rendered Image Left

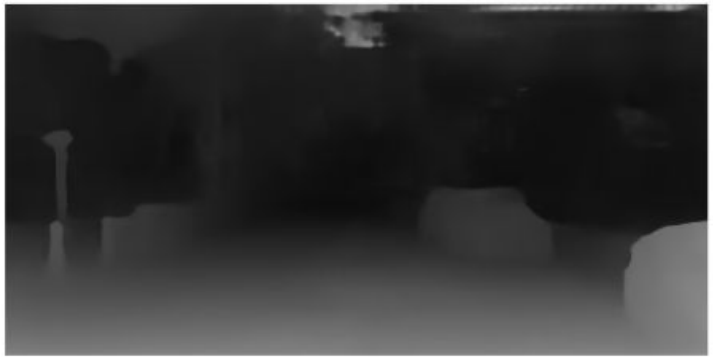
Rendered Image Right

(From Talha Faiz's research group at CMU Robotics Institute. Image courtesy Vanshaj Chowdhary)

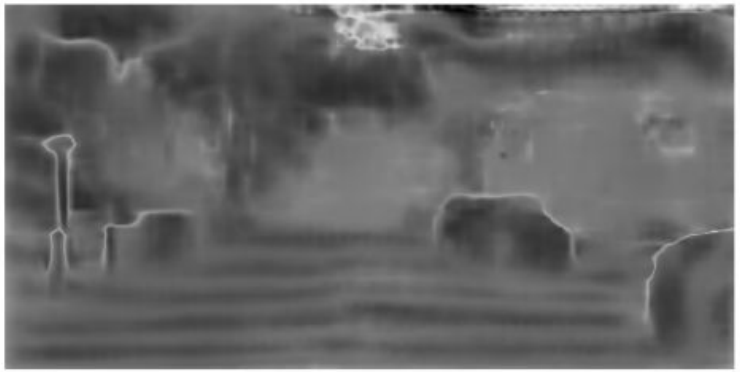
Application 2 - Night to Day

Input Left Image

Input Right Image



Stereo Output



Entropy Output

(From Talha Faiz's research group at CMU Robotics Institute. Image courtesy Vanshaj Chowdhary)

CycleGAN Takeaways

- When paired training data is unavailable
- No easy discriminative method to train classes (like zebras from horses)
- Edge cases: Performs poorly
  - if test image is rather different from the training dataset.
  - on some tasks like Photos <-> Labels
- Succeeded by contrastive unpaired translation (CUT), an unpaired img2img translation model that enables fast and memory-efficient training (Look this up!)

Neural Style Transfer*

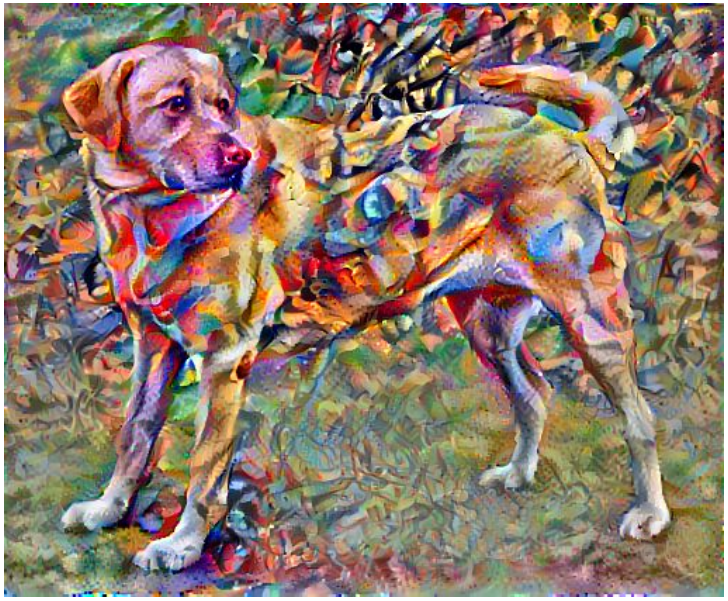
- Neural style transfer is an optimization technique used to take two images—a content image and a style reference image (such as an artwork by a famous painter)—and blend them together so the output image looks like the content image, but “painted” in the style of the style reference image.
- Extended by NVIDIA with StyleGAN-I

Neural Style Transfer Input

Content

Style

Neural Style Transfer Results



Style Transfer Objective Function

- Content Loss: $\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2$

- Style Loss: $E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

- Total Loss: $\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$

Neural Style Transfer vs. CycleGAN

- Neural Style Transfer
  - Need a *content* and *style* image
  - Specific & small number of images
  - More control
  - https://reiinakano.com/arbitrary-image-stylization-tfjs/

- CycleGAN
  - Just need 2 domains of images. No need for specific *content* or *style* images
  - Many similar pictures
  - Specificity of images doesn't really matter


Just Think an Extra
couple of Seconds
before Assuming
Something is Real

Appendix

Math: Theorem 1

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

Proof. For $p_g = p_{\text{data}}$, $D_G^*(\boldsymbol{x}) = \frac{1}{2}$, (consider Eq. 2). Hence, by inspecting Eq. 4 at $D_G^*(\boldsymbol{x}) = \frac{1}{2}$, we find $C(G) = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$. To see that this is the best possible value of $C(G)$, reached only for $p_g = p_{\text{data}}$, observe that

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} [-\log 2] + \mathbb{E}_{\boldsymbol{x} \sim p_g} [-\log 2] = -\log 4$$

and that by subtracting this expression from $C(G) = V(D_G^*, G)$, we obtain:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right) \tag{5}$$

where KL is the Kullback–Leibler divergence. We recognize in the previous expression the Jensen–Shannon divergence between the model's distribution and the data generating process:

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{6}$$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal, we have shown that $C^* = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{\text{data}}$, i.e., the generative model perfectly replicating the data generating process. □

Math: Theorem 2

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Proof. Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if $f(x) = \sup_{\alpha\in\mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in x for every α, then $\partial f_\beta(x) \in \partial f$ if $\beta = \arg\sup_{\alpha\in\mathcal{A}} f_\alpha(x)$. This is equivalent to computing a gradient descent update for p_g at the optimal D given the corresponding G. $\sup_D U(p_g, D)$ is convex in p_g with a unique global optima as proven in Thm 1, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof. □

In practice, adversarial nets represent a limited family of p_g distributions via the function $G(\boldsymbol{z}; \theta_g)$, and we optimize θ_g rather than p_g itself. Using a multilayer perceptron to define G introduces multiple critical points in parameter space. However, the excellent performance of multilayer perceptrons in practice suggests that they are a reasonable model to use despite their lack of theoretical guarantees.