COMP 432 Machine Learning

Generative Adversarial Networks (GANs)

Computer Science & Software Engineering
Concordia University, Fall 2021

Generative Adversarial Nets (GANs)

Ian J. Goodfellow[*], **Jean Pouget-Abadie**[†], **Mehdi Mirza**, **Bing Xu**, **David Warde-Farley**, **Sherjil Ozair**[‡], **Aaron Courville**, **Yoshua Bengio**[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal

>10,000 citations since 2014

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Generative Adversarial Networks

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\tilde{\mathbf{x}} = G(\mathbf{z}, \mathbf{w})$$

could be uniform

deterministic generator (decoder) function

but then how is **w** trained?

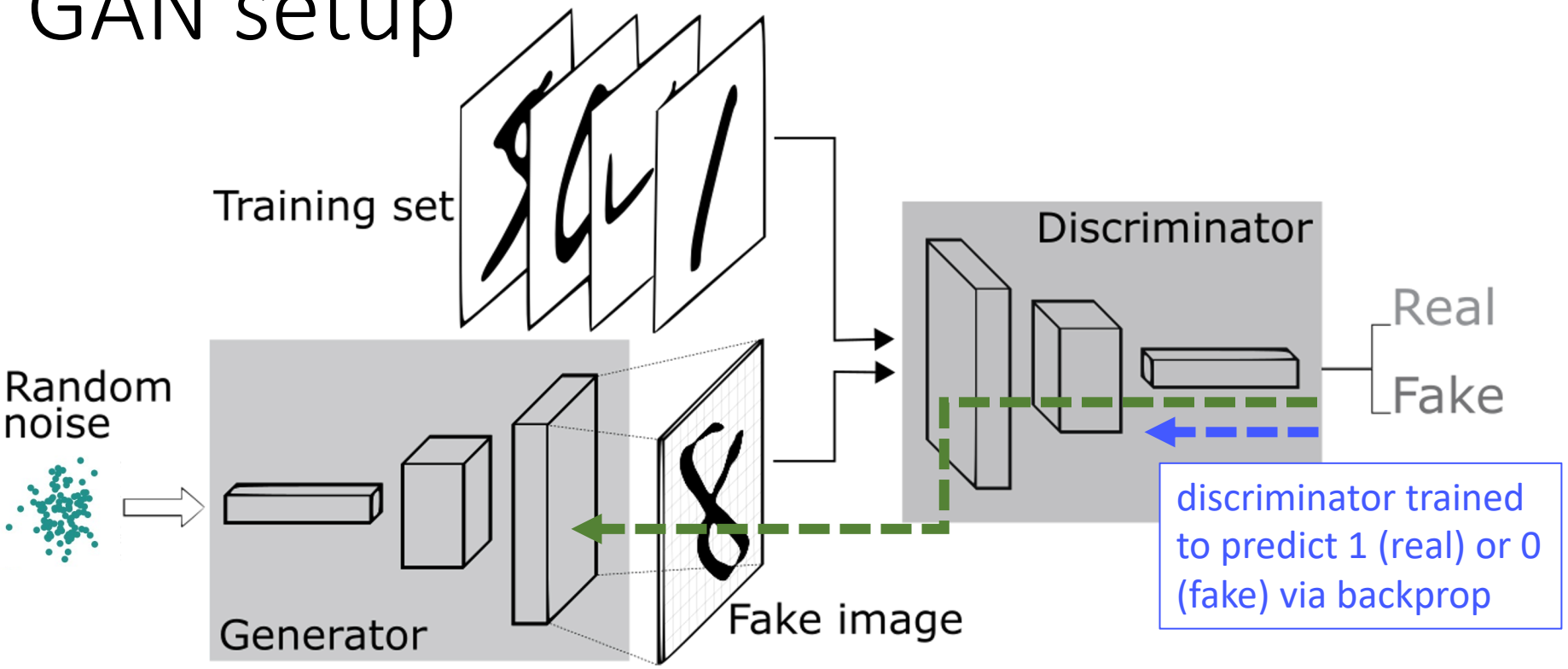
- GANs generate samples the same way a VAE generates samples ...
- Difference is the *training* scheme
  - No encoder, no reconstruction loss
- **Main idea:** if an expert can't discriminate between generated data $\tilde{\mathbf{x}}$ ("fake" data) and training data ("real" data), then the generator G is a good fit!

Yann LeCun

- **Key insight:** if deep nets are good classifiers, why not train one to become our "expert"!

"GANs are the most interesting idea in the last 10 years of machine learning"

GAN setup



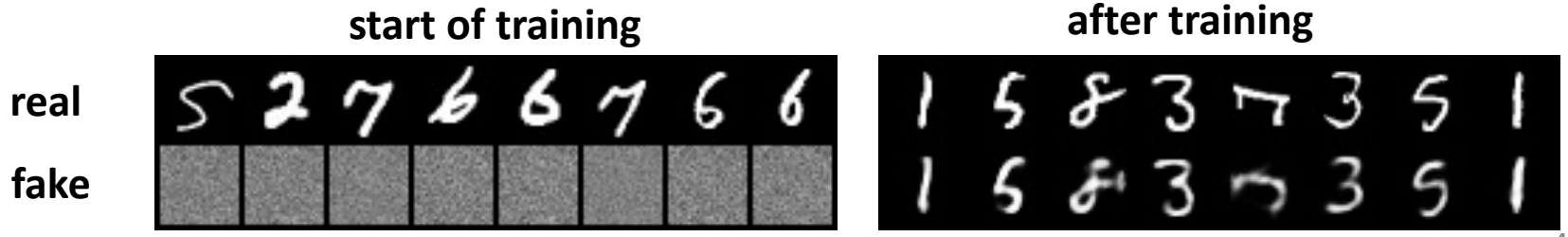


Image credit: Thalles Silva

GAN training objective

- Suppose given training examples $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
- Let $G(\mathbf{z}, \mathbf{w})$ be the *generator network* and $D(\mathbf{x}, \boldsymbol{\omega})$ be the *discriminator network,* and let $\mathcal{N}(\mathbf{0}, \mathbf{I})$ be noise distribution (density over latent codes $\mathbf{z}$).
- GAN training objective is then:

$$\min_{\mathbf{w}} \max_{\boldsymbol{\omega}} \; \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \left[\ln D(\mathbf{x}, \boldsymbol{\omega})\right] \; + \; \mathbb{E}_{\mathbf{z}\sim\mathcal{N}} \left[\ln \left(1 - D(G(\mathbf{z}, \mathbf{w}), \boldsymbol{\omega})\right)\right]$$

Adjust $\boldsymbol{\omega}$ so that *D* predicts 1 for $\mathbf{x}$ ("real data").

Adjust $\boldsymbol{\omega}$ so that *D* predicts 0 for $\tilde{\mathbf{x}} = G(\mathbf{z}, \mathbf{w})$ ("fake data").

Adjust $\mathbf{w}$ so that *G* generates an $\tilde{\mathbf{x}}$ that *D* would predict as 1, *i.e.* make the fake data seem more realistic to *D.*

Maximize log likelihood of *real* data under *D.*

Maximize log likelihood of *fake* data under *D.*

GAN training dynamics

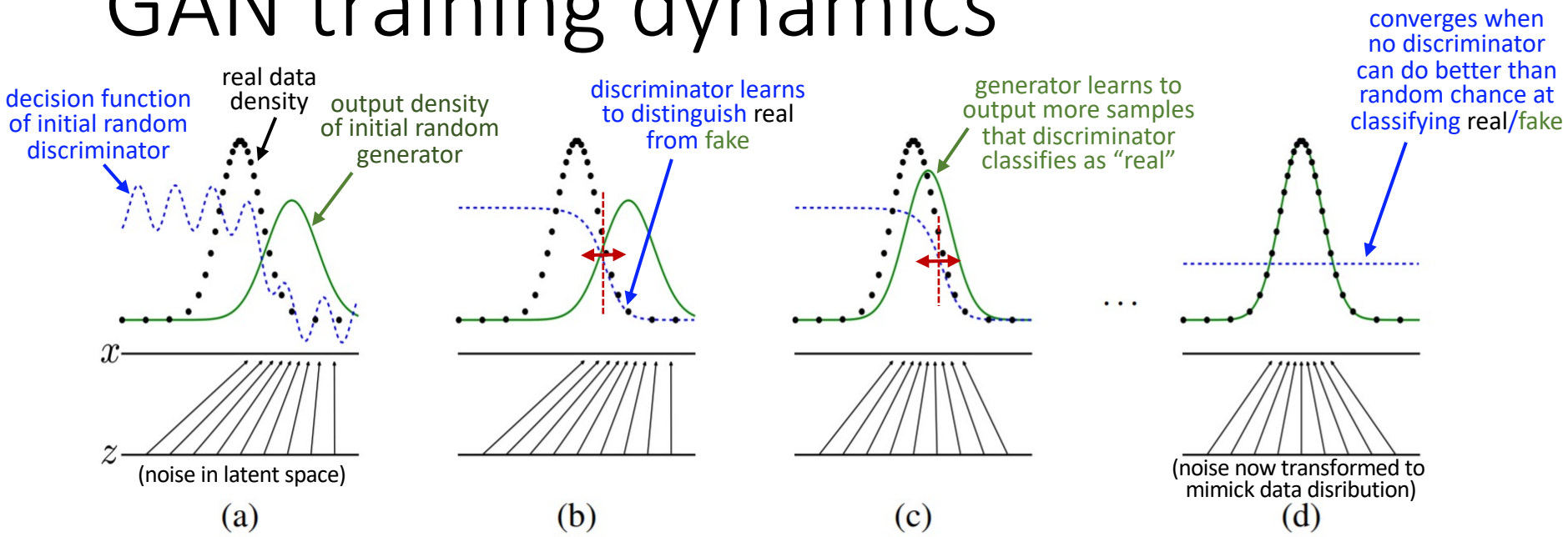
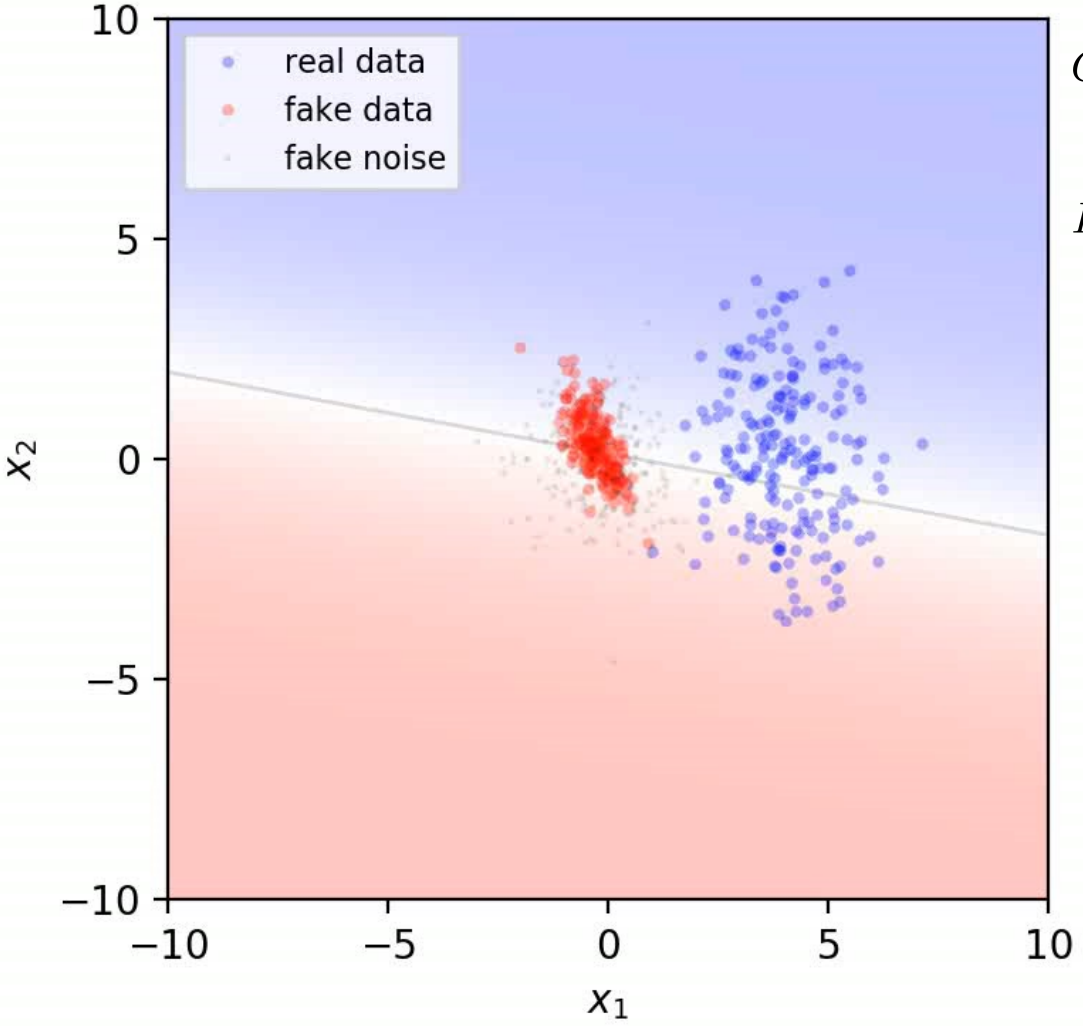


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Image source: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

G linear, *D* logistic regression



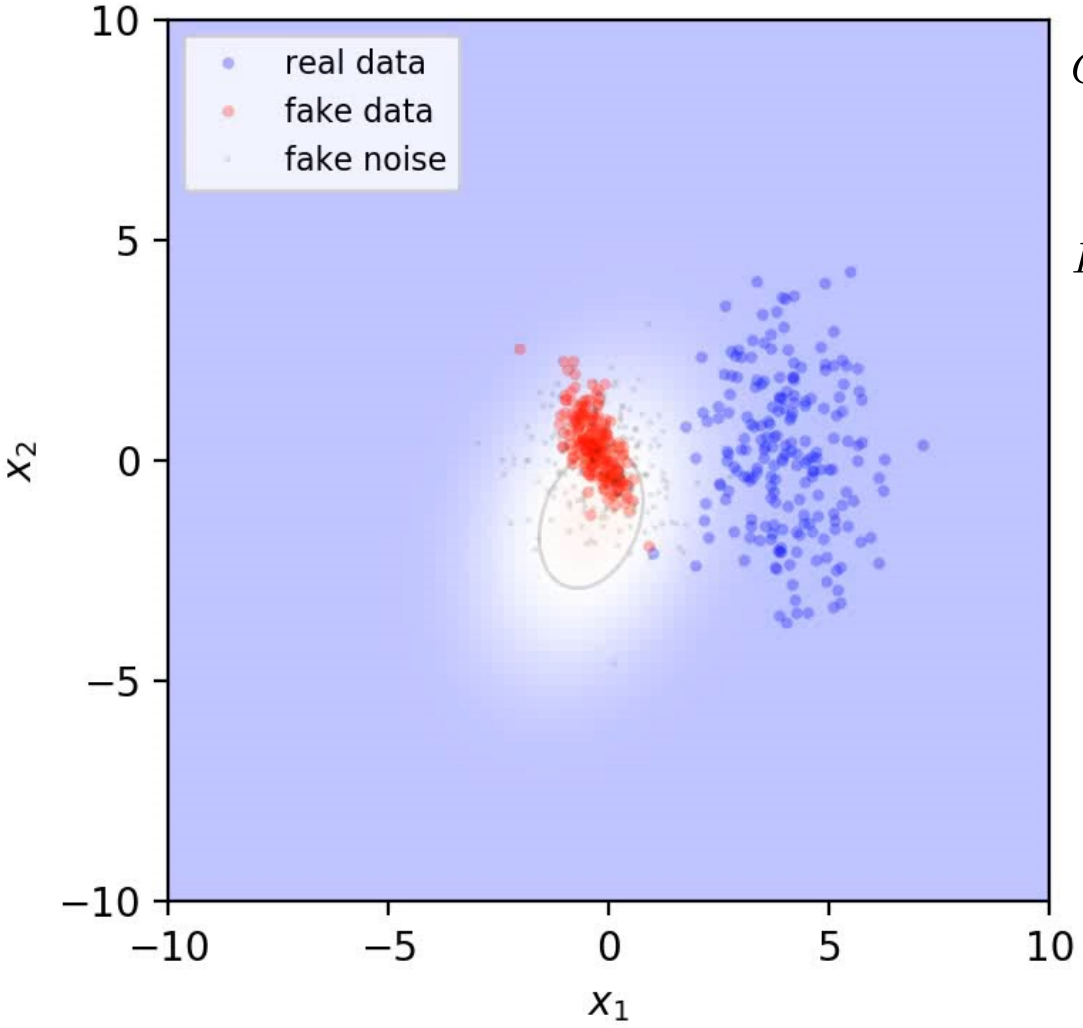
$$G(\mathbf{z}, \mathbf{w}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}$$

$$D(\mathbf{x}, \boldsymbol{\omega}) = \sigma\left(\begin{bmatrix} \omega_1 & \omega_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \omega_3 \end{bmatrix} \right)$$

Sort of works, but once fake data $\tilde{\mathbf{x}}$ (red) becomes centered on real data $\mathbf{x}$ (blue) the discriminator's linear decision boundary is too weak to further distinguish between real and fake data.

This slide is a video

G linear, *D* logistic regression on features



$$G(\mathbf{z}, \mathbf{w}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}$$

$$D(\mathbf{x}, \boldsymbol{\omega}) = \sigma \left(\begin{bmatrix} \omega_1 & \cdots & \omega_5 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_5 \end{bmatrix} + \begin{bmatrix} \omega_6 \end{bmatrix} \right)$$

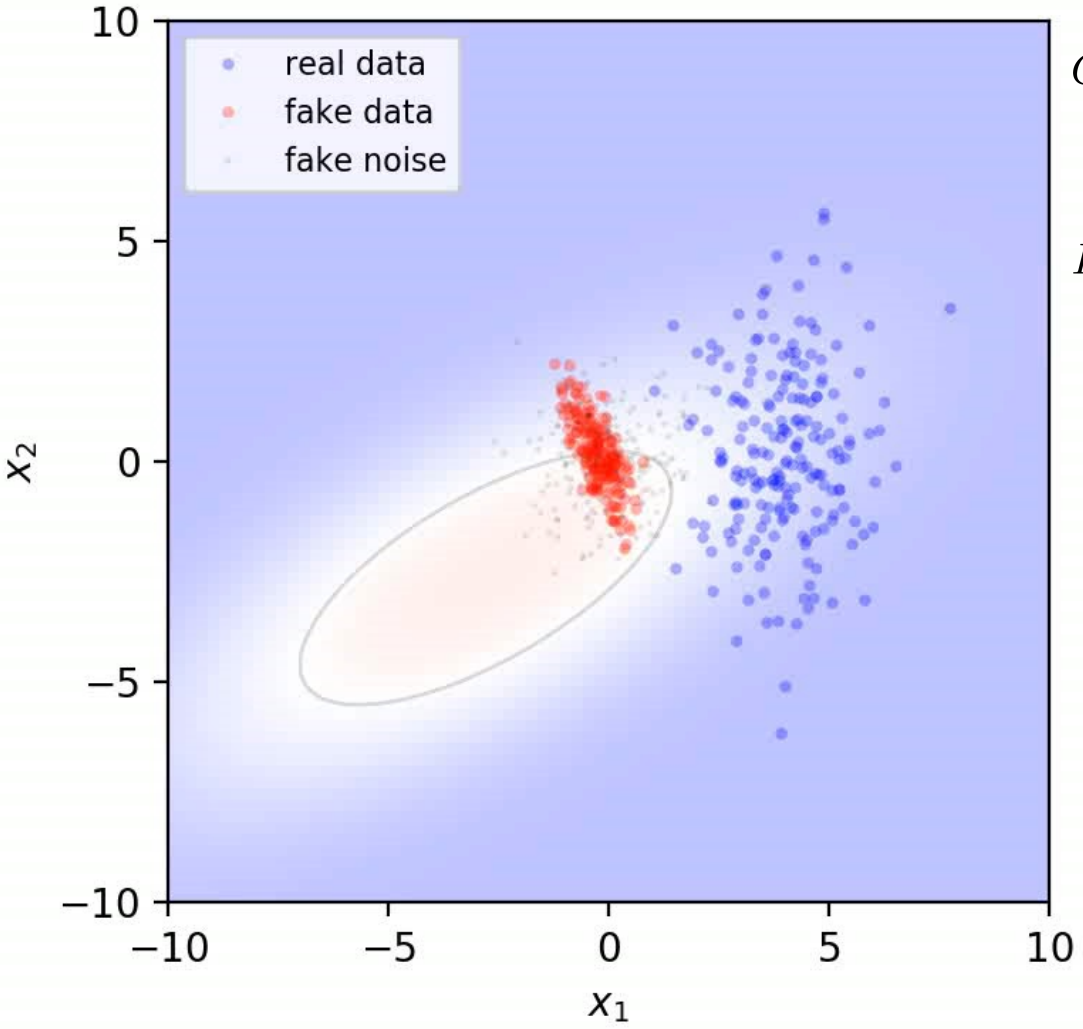
$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_1 x_2 & x_1^2 & x_2^2 \end{bmatrix}^T$$

If we give discriminator quadratic features, it can more accurately distinguish between real and fake data, giving *G* more gradient signal and thereby a tighter fit to data.

This slide is a video

G linear, *D* logistic regression on features

where data is continually re-sampled (like real GAN training!)



$$G(\mathbf{z}, \mathbf{w}) = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}$$

$$D(\mathbf{x}, \boldsymbol{\omega}) = \sigma \left(\begin{bmatrix} \omega_1 & \cdots & \omega_5 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_5 \end{bmatrix} + \begin{bmatrix} \omega_6 \end{bmatrix} \right)$$

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_1 x_2 & x_1^2 & x_2^2 \end{bmatrix}^T$$

Previous videos used same batch of latent code samples $\mathbf{z}$ (grey) and same batch of training data points $\mathbf{x}$ (blue).

Real GAN training uses a new mini-batch of samples for each training step, so the training dynamics are more chaotic, like in this video.

This slide is a video

GAN lab training loop (PyTorch)

Understand how this code achieves GAN training!

```
# Create vectors to act as binary training targets
y_real = torch.ones((batch_size, 1))   # 1 means "this data point is real"
y_fake = torch.zeros((batch_size, 1))  # 0 means "this data point is fake"

def update_discriminator():
    discriminator.zero_grad()          # Get ready to accumulate gradient for discriminator weights.

    X = sample_real_data(batch_size)   # Sample real data.
    y = discriminator(X)               # Predict real or fake.
    l = loss(y, y_real)                # Discriminator wants to predict "real" for these cases.
    l.backward()                       # Accumulate discriminator gradient.

    X = sample_fake_data(batch_size).detach()  # Sample fake data. (Detach so no generator gradient.)
    y = discriminator(X)                       # Predict real or fake.
    l = loss(y, y_fake)                        # Discriminator Wants to predict "fake" for these.
    l.backward()                               # Accumulate discriminator gradient.

    discriminator_optim.step()         # Use gradient to improve the discriminator!

def update_generator():
    generator.zero_grad()              # Get ready to accumulate gradient for generator weights.

    X = sample_fake_data(batch_size)   # Sample fake data. (Don't detach, so backprop to generator.)
    y = discriminator(X)               # Predict real or fake.
    l = loss(y, y_real)                # Generator wants discriminator to predict "real" for these.
    l.backward()                       # Accumulate generator gradient, backprop through discriminator!

    generator_optim.step()             # Use gradient to improve generator!
```

Training GANs is notoriously hard

Don’t start from scratch! Learn tips from the best!
For example: https://github.com/soumith/ganhacks

How to Train a GAN? Tips and tricks to make GANs work

While research in Generative Adversarial Networks (GANs) continues to improve the fundamental stability of these models, we use a bunch of tricks to train them and make them stable day to day.

Here are a summary of some of the tricks.

Here's a link to the authors of this document

If you find a trick that is particularly useful in practice, please open a Pull Request to add it to the document. If we find it to be reasonable and verified, we will merge it in.

1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

2: A modified loss function

In GAN papers, the loss function to optimize G is `min (log 1-D)`, but in practice folks practically use `max log D`

- because the first formulation has vanishing gradients early on
- Goodfellow et. al (2014)

“Flavours” of GANs

- Crazy number of GAN varieties have been proposed

Table 1: Generator and discriminator loss functions. The main difference whether the discriminator outputs a probability (MM GAN, NS GAN, DRAGAN) or its output is unbounded (WGAN, WGAN GP, LS GAN, BEGAN), whether the gradient penalty is present (WGAN GP, DRAGAN) and where is it evaluated.

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x-\mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$

Lucic et al 2018 https://arxiv.org/pdf/1711.10337.pdf

GAN extensions (there are *many*)

- **Conditional GAN:** Once generator trained, can get some control over the type of samples it generates https://arxiv.org/pdf/1411.1784.pdf
  - Also InfoGAN, CycleGAN, etc
- **Deep Convolutional GAN (DCGAN):** Better generator for spatial/temporal data based on convolutions https://arxiv.org/abs/1511.06434
- **VAE-GAN:** Use GAN-like discriminator with VAE for 'learned' reconstruction loss, replacing squared error https://arxiv.org/pdf/1512.09300.pdf
- **Adversarial Autoencoder:** VAE that uses GAN to impose any desired structure on latent codes https://arxiv.org/pdf/1511.05644.pdf

Example:
conditioning the generator on rich label information

https://arxiv.org/pdf/1611.07004.pdf

Image-to-Image Translation with Conditional Adversarial Networks

Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros

Berkeley AI Research (BAIR) Laboratory, UC Berkeley

{isola,junyanz,tinghuiz,efros}@eecs.berkeley.edu

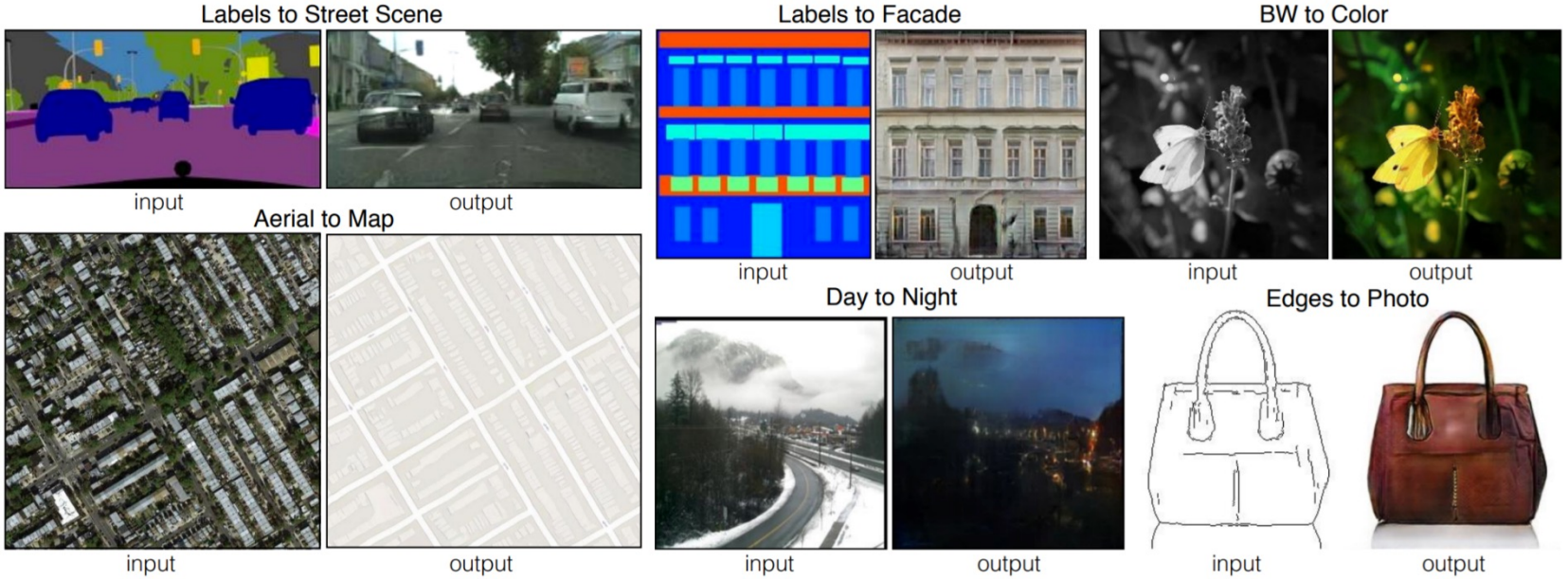


Figure 1: Many problems in image processing, graphics, and vision involve translating an input image into a corresponding output image. These problems are often treated with application-specific algorithms, even though the setting is always the same: map pixels to pixels. Conditional adversarial nets are a general-purpose solution that appears to work well on a wide variety of these problems. Here we show results of the method on several. In each case we use the same architecture and objective, and simply train on different data.

Example: generative “inpainting”

Globally and Locally Consistent Image Completion

SATOSHI IIZUKA, Waseda University
EDGAR SIMO-SERRA, Waseda University
HIROSHI ISHIKAWA, Waseda University

Fig. 1. Image completion results by our approach. The masked area is shown in white. Our approach can generate novel fragments that are not present elsewhere in the image, such as needed for completing faces; this is not possible with patch-based methods. Photographs courtesy of Michael D Beckwith (CC0), Mon Mer (Public Domain), davidgsteadman (Public Domain), and Owen Lucas (Public Domain).

Globally and Locally Consistent Image Completion

Satoshi Iizuka Edgar Simo-Serra Hiroshi Ishikawa

SIGGRAPH 2017

Fig. 4: **Anonymized Images from DeepPrivacy.** Every single face in the images has been generated.

https://arxiv.org/pdf/1909.04538.pdf

Designed to Deceive: Do These People Look Real to You?

The people in this story may look familiar, like ones you've seen on Facebook or Twitter or Tinder. But they don't exist. They were born from a computer.

https://www.nytimes.com/interactive/2020/11/21/science/artificial-intelligence-fake-people-faces.html

Note: A pre-trained version of Nvidia's StyleGAN2 package, implemented in TensorFlow, was used to generate the images for this story. The networks trained on the Flickr-Faces-HQ dataset, which included over 70,000 photographs of people. Improvements to the original StyleGAN architecture by Karras et. al.