Deep Learning

Lecture 4: Generative *Adversarial* Models

Amir Atapour-Abarghouei

amir.atapour-abarghouei@durham.ac.uk

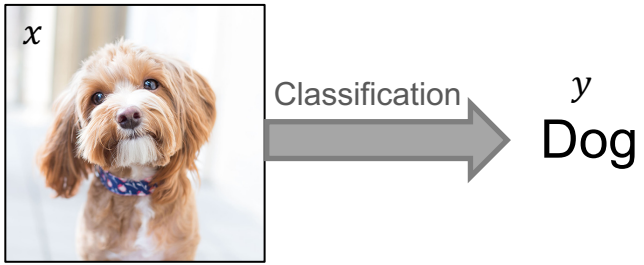
Durham University

Durham University

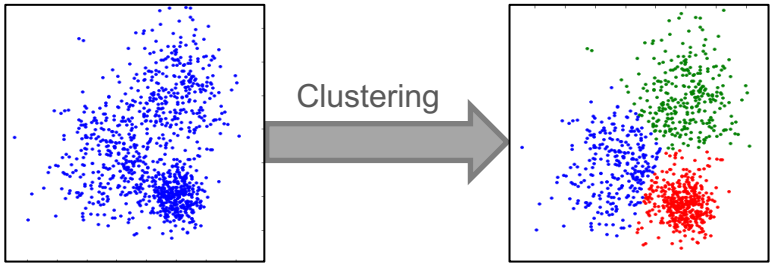
Lecture Overview

1. Generative models
   - Unsupervised training
   - PixelRNN and PixelCNN
2. Generative adversarial networks
   - Definition
   - Common Issues
   - Lipschits Continuity
   - Spectral Normalisation
   - Conditional GANs
   - Other Advanced Variants
3. Popular applications
   - Unpaired Translation
   - Super Resolution
   - Anomaly Detection

Supervised vs Unsupervised Training



- Supervised Learning:
  - Data x – Labels y
  - Learn mapping function $x \rightarrow y$
  - *e.g.* classification, regression, segmentation, object detection, machine translation, sentiment analysis (and everything else we have talked about so far).



- Unsupervised Learning:
  - Data x – **No** Labels
  - Learn underlying structures and patterns in the data.
  - *e.g.* clustering, dimensionality reduction, feature learning, density estimation.
  - **Generative models** are **unsupervised** as the data used has **no labels**.

Generative Models

Definition:

Given a set of data, capture the probability distribution representing this data and generate new data samples from this distribution.

Training Data ~ $P_{\text{data}}(x)$

https://github.com/NVlabs/ffhq-dataset

Generated Sample ~ $P_{\text{model}}(x)$

https://arxiv.org/pdf/1912.04958.pdf

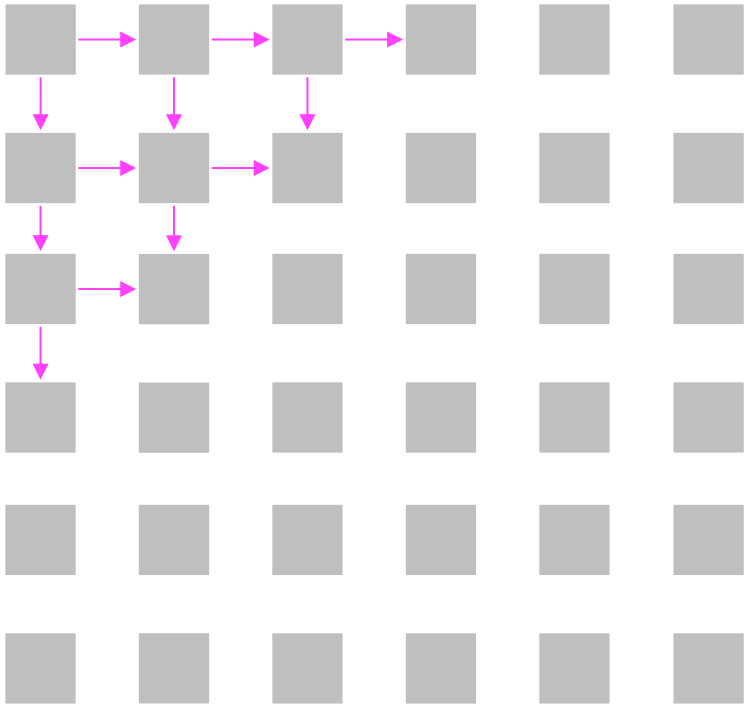
The objective is to learn $P_{\text{model}}(x)$ so it as close as possible to $P_{\text{data}}(x)$.

PixelRNN [van den Oord et al., 2016]

- Explicit tractable density function.
- We generate pixels one by one starting from a corner.
- Dependency on previous pixels is modelled via an RNN.
- During inference, for generation, we initialise the value of the first pixel and the image is generated.

Good results

but RNNs are sequential and thus **inefficient**!



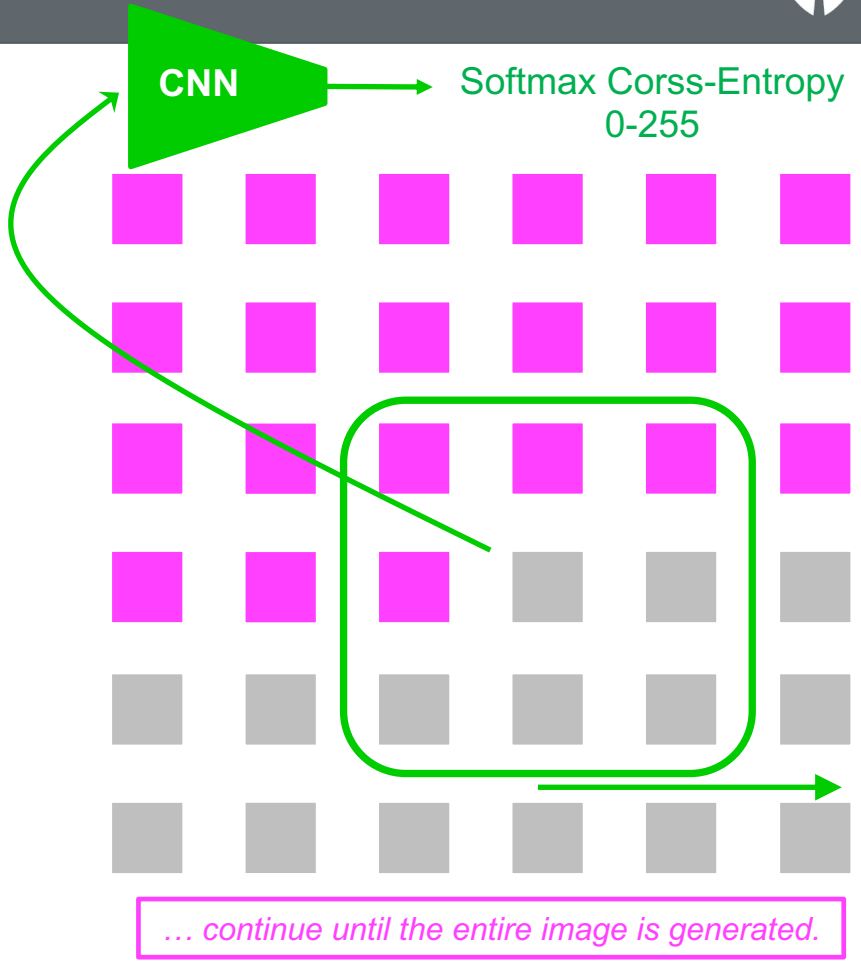
… continue until the entire image is generated.

PixelCNN [van den Oord et al., 2016]

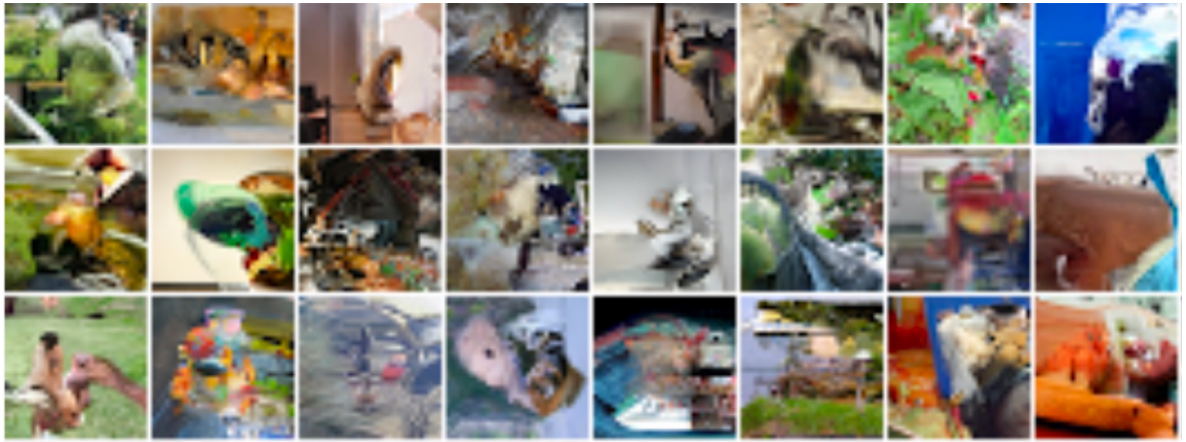
- Explicit tractable density function.
- We generate pixels one by one starting from a corner.
- Dependency on **pixel**s is modelled via a CNN over context patch.

CNN is parallelisable and thus **efficient**!

During inference, generation is sequential and still very slow...!



PixelRNN / PixelCNN [van den Oord et al., 2016]



PixelRNN

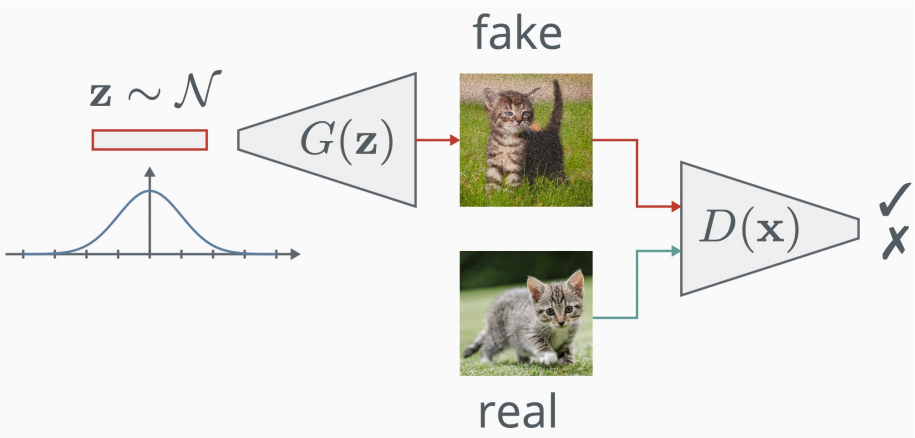
PixelCNN

Generative Adversarial Networks

Definition:

A generative adversarial network (GAN) is a non-coorporative zero-sum game where two networks compete against each other [Goodfellow et al., 2014].

One network $G(z)$ generates new samples, whereas D estimates the probability the sample was from the training data rather than G:



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

Generative Adversarial Networks

Generator

1. I see!!

Fake!

2. Hmm!!

Fake!

3. Aha!!

Fake?

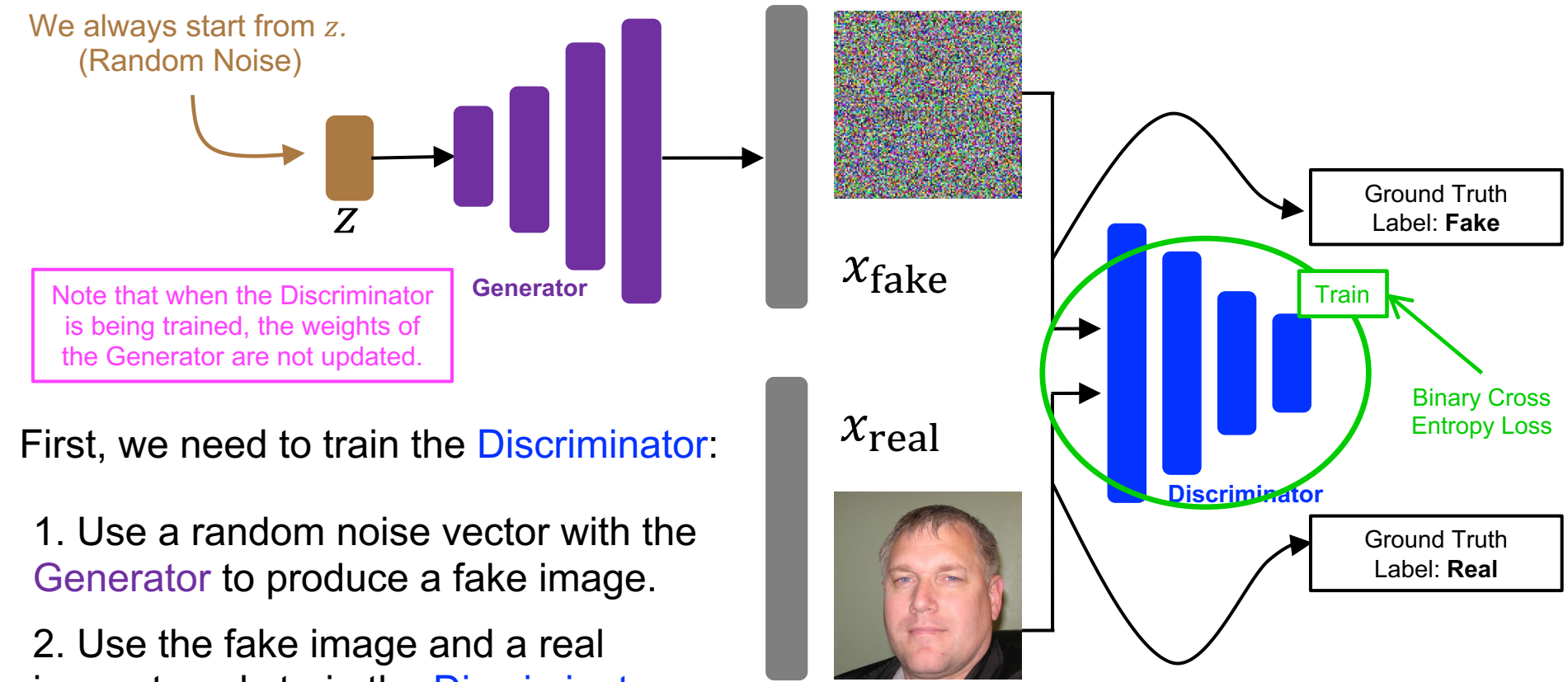
4. **Success!**

Real?

Discriminator

Generative Adversarial Networks

Training

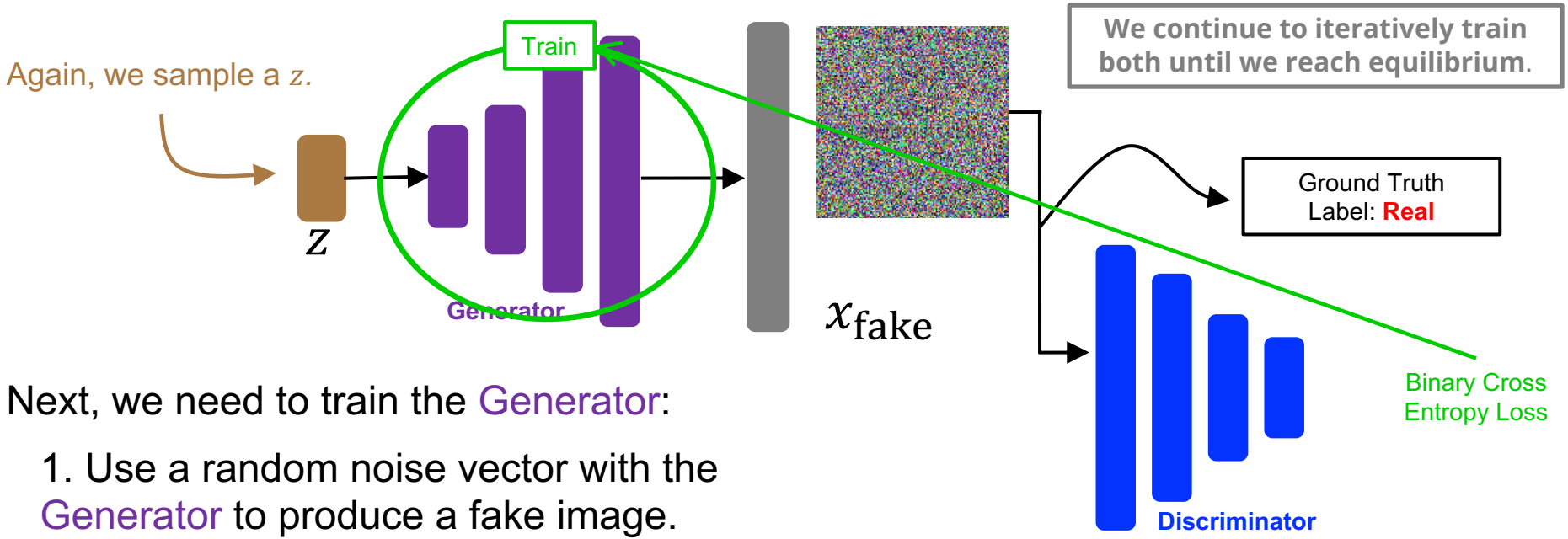


First, we need to train the Discriminator:

1. Use a random noise vector with the Generator to produce a fake image.
2. Use the fake image and a real image to only train the Discriminator.

Generative Adversarial Networks

Training



Next, we need to train the Generator:

1. Use a random noise vector with the Generator to produce a fake image.

2. Pass the fake image through the Discriminator, but with a Real label.

3. We then use the gradients from the Discriminator to train the Generator.

Note that we are trying to ***fool*** the Discriminator, which is why the Real label is used for the fake image.

Generative Adversarial Networks

Objective

- The objective function:
  - Generator and Discriminator are trained jointly in minimax game:

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

Discriminator output for real data x

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

- Discriminator (θ_d) wants to **maximise** the objective so $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake).
- Generator (θ_g) wants to **minimise** the objective so $D(G(z))$ is close to 1 (the discriminator will think the fake output is real).

Let’s Look at Some Code!

Simple GAN Example

Code on GitHub : https://github.com/atapour/dl-pytorch/blob/main/Simple_GAN_Example/Simple_GAN_Example.ipynb

Code on Colab: https://colab.research.google.com/github/atapour/dl-pytorch/blob/main/Simple_GAN_Example/Simple_GAN_Example.ipynb

Issues:

- **Non-Convergence:** Model is unstable so parameters oscillate and never converge.
- **Diminishing Gradients:** The Discriminator reaches its optimal state too soon and its small gradients cannot train the Generator.
- **Hyperparameter Sensitivity:** The model is too sensitive to any changes in hyperparameters (*e.g.,* learning rate).
- **Mode Collapse:** The Generator collapses and produces a limited variety of samples.

Definition: Lipschitz function

A function f is Lipschitz continuous if it is bounded by how fast it can change. Specifically if there exists a positive real constant k where:

$$|f(x) - f(y)| \leq k|x - y|,$$

for all y sufficiently near x. For example, any function with a bounded first derivative is a Lipschitz function.

The slope of any secant line to f is between $-k$ and k

Lower Lipschitz constant makes the function smooth and easy to optimise.

If the inputs x and y are close/similar, we want model outputs, $f(x)$ and $f(y)$ to also be similar.

Generative Adversarial Networks

- Wasserstein GAN [Arjovsky et al., 2017]
  - Limiting the gradients of the discriminator will make it a smoother function.
  - Done by clipping the gradients.

- Improved Wasserstein GAN [Gulrajani et al., 2017]
  - Add a penalty term to penalise the gradients of the discriminator directly.
  - Requires second order differentiation, which PyTorch supports, but it is slow.

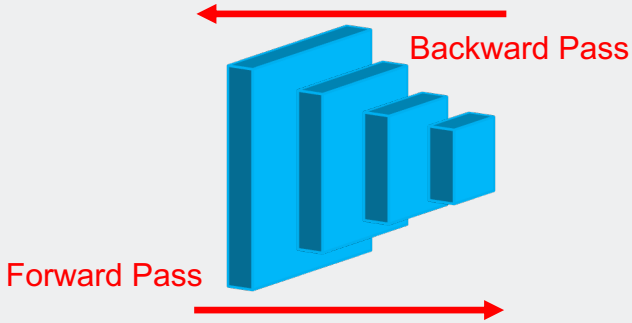
Generative Adversarial Networks

- There is another solution using Normalisation!

First, what is normalisation? – BatchNorm [Ioffe and Szegedy, 2015]

- All layers are updated in the backward pass.
- Every layer assumes other layers are fixed.

The weights of a layer can be updated based on the expectation that the prior layer produces values with a given distribution. However, this distribution is likely to change once that layer is updated.



Batch Normalisation: Layer outputs are rescaled, so they have a mean of *zero* and a standard deviation of *one:*

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Weight Norm
- Layer Norm
- Instance Norm
- Group Norm
- Switchable Norm

Etc.

Problem: Internal Covariate Shift
(change in the distribution of input during training)

Generative Adversarial Networks

- Spectral Normalisation [Miyato et al., 2018]

Definition: spectral normalisation

The matrix (spectral) norm defines how much a matrix can stretch a vector x:

$$\| A \| = \max_{x \neq 0} \frac{\| Ax \|}{x}$$

Spectral norm normalises the weights for each layer using the spectral norm $\sigma(W)$ such that the Lipschitz constant for every layer and the whole network is 1.

$$\widehat{W}_{\text{SN}} = W / \sigma(W)$$

$$\sigma\left(\widehat{W}_{\text{SN}}(W)\right) = 1$$

$$\| f \|_{\text{Lip}} = 1$$

Pseudocode: 1-Lipschitz discriminator

```
class Discriminator(nn.Module):
  def __init__(self, f=64):
    super().__init__()
    self.discriminate = nn.Sequential(
      spectral_norm(Conv2d(1, f, 3, 1, 1)),
      nn.LeakyReLU(0.1, inplace=True),
      nn.MaxPool2d(kernel_size=(2,2)),
      spectral_norm(Conv2d(f, f*2, 3, 1, 1)),
      nn.LeakyReLU(0.1, inplace=True),
      nn.MaxPool2d(kernel_size=(2,2)),
      spectral_norm(Conv2d(f*2, f*4, 3, 1, 1)),
      nn.LeakyReLU(0.1, inplace=True),
      nn.MaxPool2d(kernel_size=(2,2)),
      spectral_norm(Conv2d(f*4, f*8, 3, 1, 1)),
      nn.LeakyReLU(0.1, inplace=True),
      nn.MaxPool2d(kernel_size=(2,2)),
      spectral_norm(Conv2d(f*8, 1, 3, 1, 1)),
      nn.Sigmoid()
    )
```

Generative Adversarial Networks

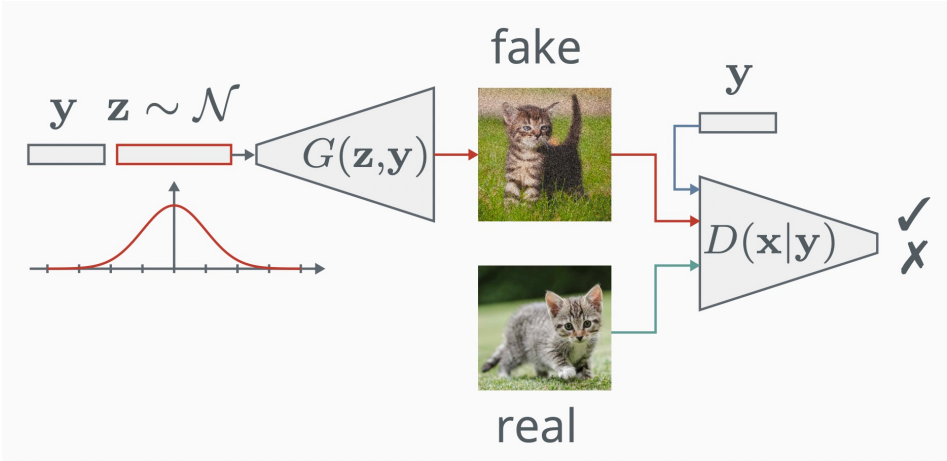
- **DCGAN:** The entire architecture is fully convolutional.
- **Alternative Objective:** Wasserstein GAN - LSGAN
- **Two Timescale Update:** The Generator is updated more slowly than the Discriminator (SAGAN).
- **Stacking GANs:** Multiple GANs are placed consecutively, and each GAN solves an easier version of the problem (FashionGAN).
- **Progressive Growing GAN:** Higher quality and larger outputs by incrementally increasing the size of the model during training.

Mean squared loss

Generative Adversarial Networks

Definition: Conditional GANs

GANs can be conditioned, for instance with labels y, if available, [Mirza et al., 2014] by feeding the label information into both the generator and the discriminator:



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

Let’s Look at Some Code!

Conditional GAN Example

Code on GitHub : https://github.com/atapour/dl-pytorch/blob/main/Conditional_GAN_Example/Conditional_GAN_Example.ipynb

Code on Colab: https://colab.research.google.com/github/atapour/dl-pytorch/blob/main/Conditional_GAN_Example/Conditional_GAN_Example.ipynb

Generative Adversarial Networks

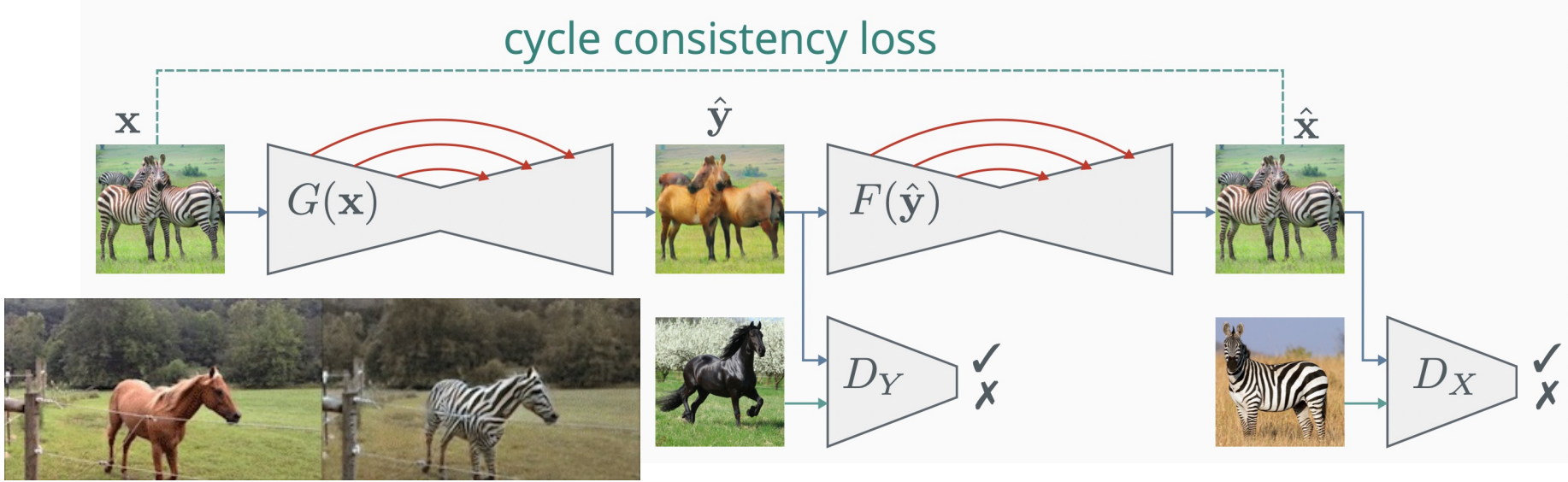
- Auxiliary Classifier GAN (ACGAN)
- InfoGAN
- Least Square GAN (LSGAN)
- Adversarial Autoencoder
- Boundary Equilibrium GAN (BEGAN)
- Energy-Based GAN
- StyleGAN
- etc. etc. etc. ...

Popular Applications

Unpaired Translation (**CycleGAN**)

Definition: CycleGAN [Zhu et al., 2017]

An adversarial architecture for unpaired image translation. It has twin generators with skip connections and two discriminators, which translate between the domains, alongside a cycle consistency loss (L1 distance) to ensure the mapping recovers the original image.

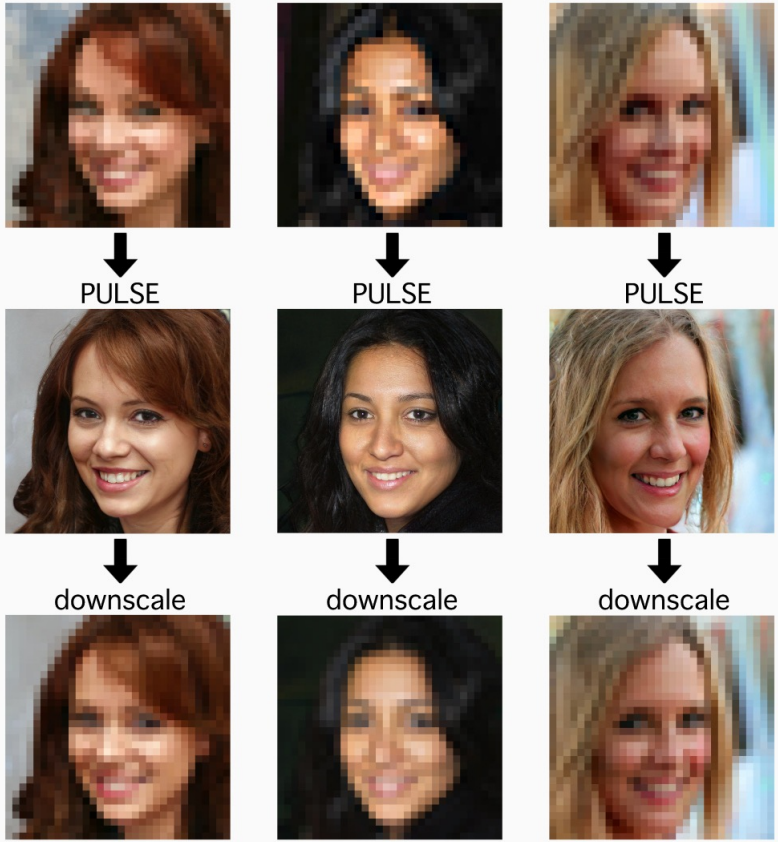


Popular Applications

Super-resolution

Definition: Super-resolution

The objective is to map a single low-resolution input to a distribution of high-resolution outputs. PULSE [Menon et al., 2020] projects points in the search of the latent space of StyleGAN (a large conditional GAN) onto a hypersphere, which ensures probable outputs in the high-dimensional latent space.



Popular Applications

Anomaly Detection

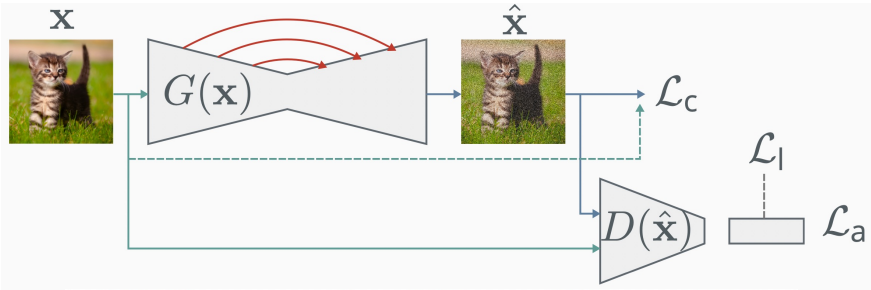
Definition: Anomaly Detection

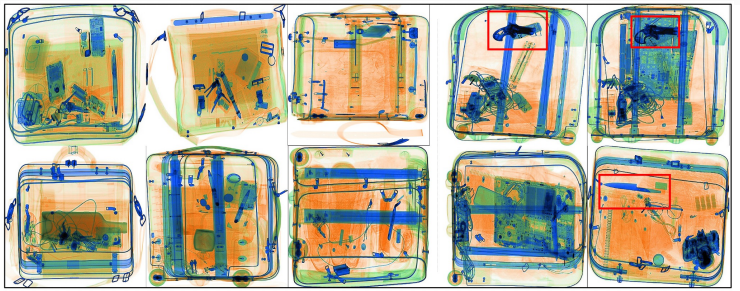
The objective is to find “anomalies” in the data. Unsupervised anomaly detectors [1,2] learn a normal distribution over (healthy) observations. Then, when they observe something not observed in training (unhealthy/dangerous), they fail to reconstruct - detecting it as an anomaly. Region-based anomaly detectors [3] learn a distribution over inpainted (erased) regions.

[1] Akçay, **Atapour-Abarghouei**, and Breckon. “GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training”, ACCV, 2018.

[2] Akçay, **Atapour-Abarghouei**, and Breckon. “Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection”, IJCNN,. 2019.

[3] Nguyen, Feldman, Bethapudi, Jennings, and Willcocks. “Unsupervised Region-based Anomaly Detection in Brain MRI with Adversarial Image Inpainting”, arXiv:2010.01942.





What we learned today!

1. Generative models
   - Unsupervised training
   - PixelRNN and PixelCNN
2. Generative adversarial networks
   - Definition
   - Common Issues
   - Lipschits Continuity
   - Spectral Normalisation
   - Conditional GANs
   - Other Advanced Variants
3. Popular applications
   - Unpaired Translation
   - Super Resolution
   - Anomaly Detection