# GANs

Generative Adversarial Networks

# GAN Intro

- Generative adversarial networks (GANs) are an exciting recent innovation in ML.
  - Originally proposed in 2014 by Ian Goodfellow et al
  - https://arxiv.org/abs/1406.2661
- GANs are generative models: **they create new data instances** that resemble your training data.
- For example, GANs can create images that look like photographs of human faces, even though the faces don't belong to any real person.
- GANs achieve this level of realism by pairing **two Neural Nets**, a **generator**, which learns to produce the target output, with a **discriminator**, which learns to distinguish true data from the output of the generator.
- The major innovation: the generator tries to fool the discriminator, and the discriminator tries to keep from being fooled.

# GAN Speed of Innovation

Generated >= 2017

# GAN Applications

- GANs' potential for both good and evil is huge, because they can learn to mimic any distribution of data.
- They are used widely in image generation, video generation and voice generation.

# Generative vs Discriminative Models

- For example, given all the words in an email (the data instance), a discriminative algorithm could predict whether the message is spam or not_spam.
  - this case would translate to “the probability that an email is spam given the words it contains.”
  - So discriminative algorithms **map features to labels**.
- Generative models do the opposite. Instead of predicting a label given certain features, they attempt to **predict features given a certain label**.
  - The question a generative algorithm tries to answer is: Assuming this email is spam, how likely are these features? While discriminative models care about the relation between y and x, generative models care about “how you get x.”
- Another way to think about it is to distinguish discriminative from generative like this:
  - Discriminative models learn the boundary between classes
  - Generative models model the distribution of individual classes

# Generative vs. Discriminative Models

- Generative models can generate new data instances.
  - could generate new photos of animals that look like real animals
- Discriminative models discriminate between different kinds of data instances.
  - could tell a dog from a cat
- Given a set of data instances X and a set of labels Y
  - Generative models capture the **joint** probability p(X, Y), or just p(X) if there are no labels.
  - Discriminative models capture the **conditional** probability p(Y | X).

# Modeling Probabilities

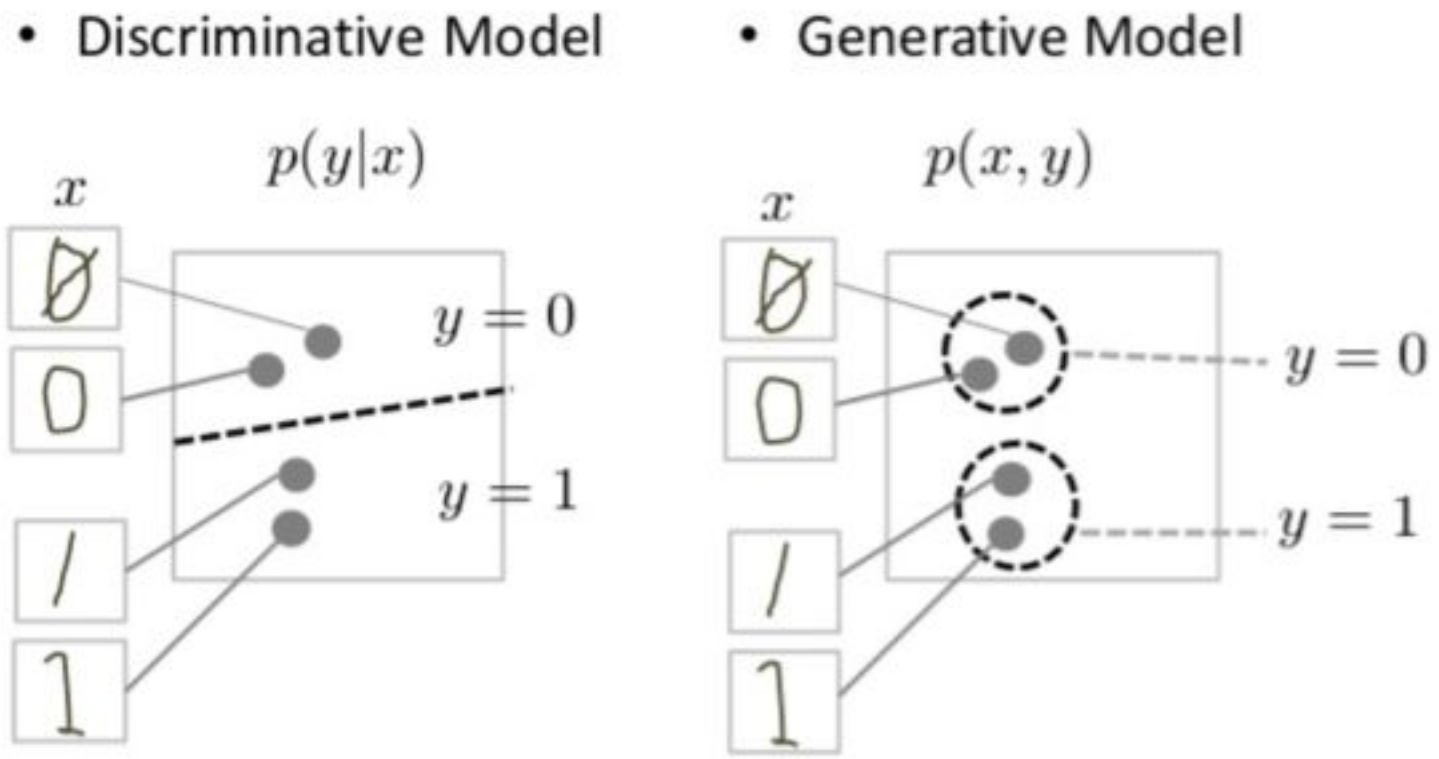
- Neither kind of model has to return a number representing a probability.
- You can model the distribution of data by imitating that distribution.
- For example, a discriminative classifier like a decision tree can label an instance without assigning a probability to that label.
  - Such a classifier would still be a model because the distribution of all predicted labels would model the real distribution of labels in the data.
- Similarly, a generative model can **model a distribution** by producing convincing "fake" data that looks like it's drawn from that distribution.

# Generative Models Are Hard(er)

- Generative models tackle a more difficult task than analogous discriminative models. Generative models have to model more.
- A generative model for images might capture **correlations** like "things that look like boats are probably going to appear near things that look like water" and "eyes are unlikely to appear on foreheads." These are very complicated distributions.
- In contrast, a discriminative model might learn the difference between "sailboat" or "not sailboat" by just looking for a few tell-tale patterns. It could ignore many of the correlations that the generative model must get right.

# Generative Models Are Hard(er)

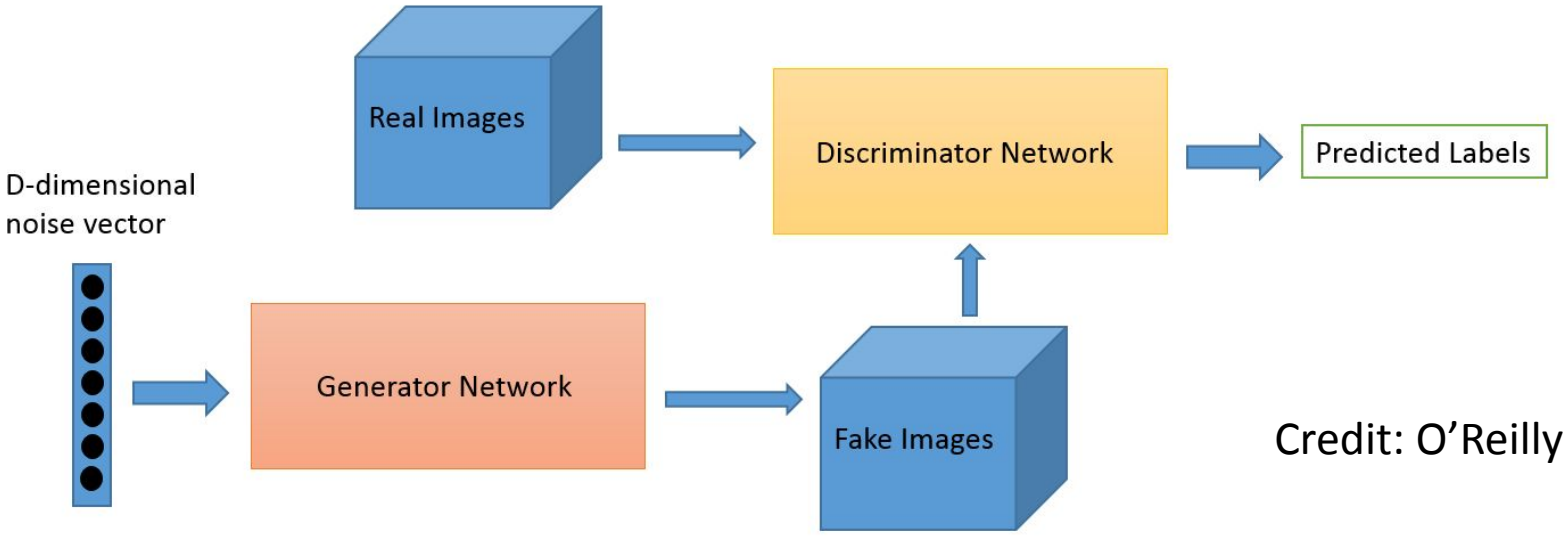
- Discriminative models try to draw **boundaries in the data space**, while generative models try to model **how data is placed throughout the space**.
- For example, the following diagram shows models of handwritten digits:



# Handwritten Digits Example

- In the previous example, the discriminative model tries to tell the difference between handwritten 0's and 1's by drawing a line in the data space.
  - **If it gets the line right, it can distinguish 0's from 1's without ever having to model exactly where the instances are placed in the data space on either side of the line.**
- In contrast, the generative model tries to produce convincing 1's and 0's by generating digits that fall close to their real counterparts in the data space.
  - **It has to model the distribution throughout the data space.**
- **GANs are a type of generative model**. They offer an effective way to train such rich models to resemble a real distribution. To understand how they work we'll need to understand the basic structure of a GAN.

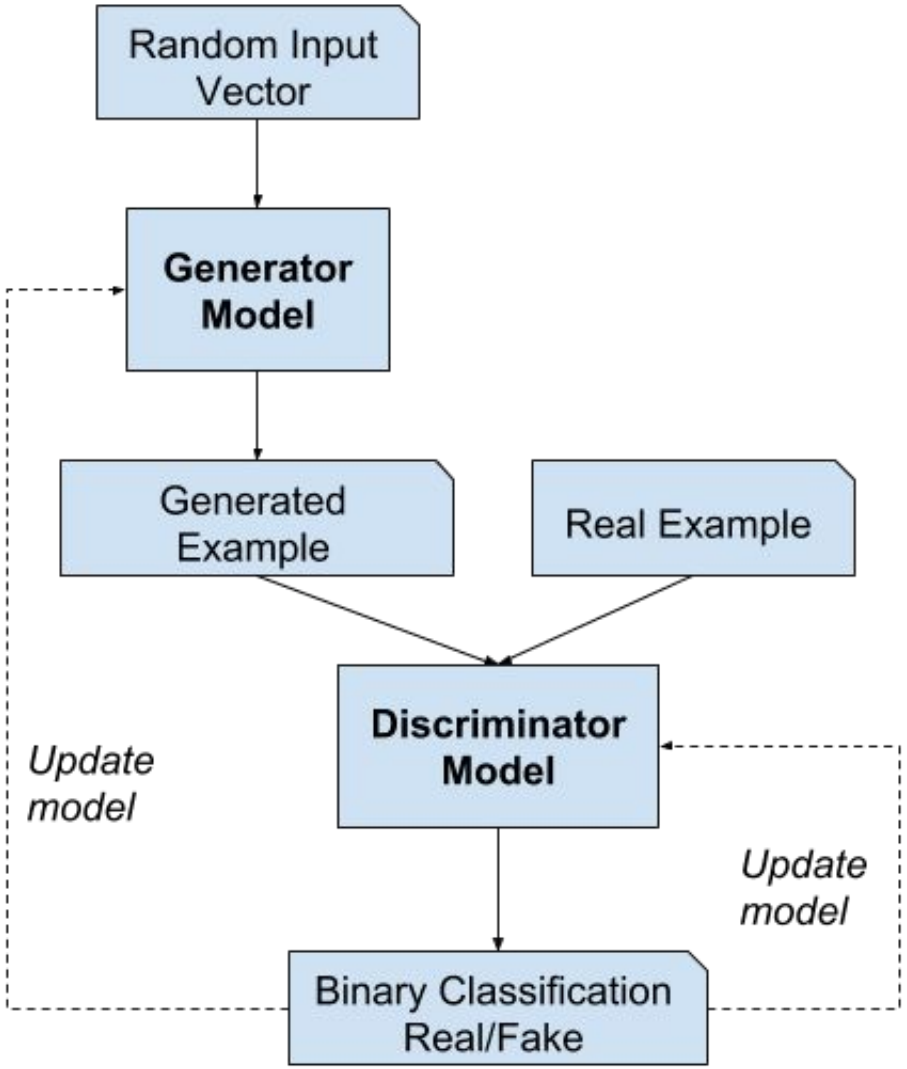
# GAN Structure



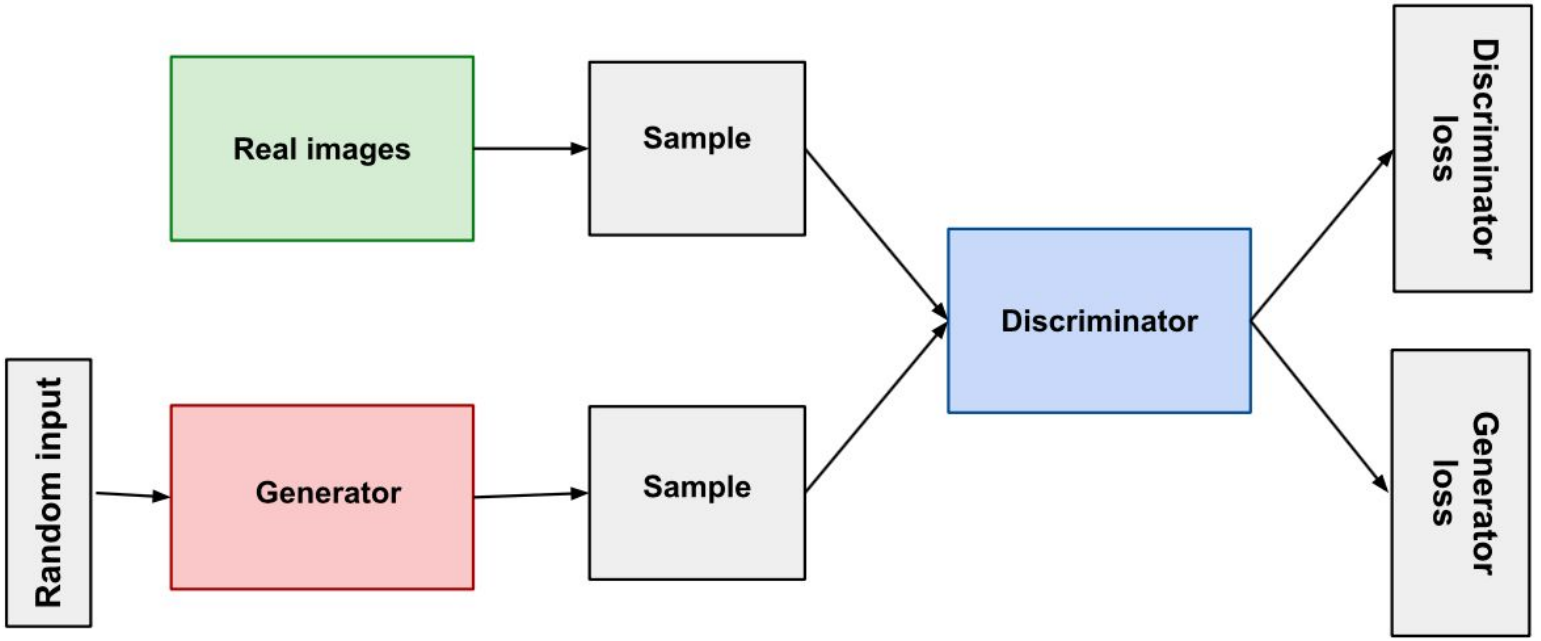

Credit: O'Reilly

- Here are the steps a GAN takes:
  - The generator takes in random numbers and returns an image.
  - This generated image is fed into the discriminator alongside a stream of images taken from the actual, ground-truth dataset.
  - The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.
- So you have a double feedback loop:
  - The discriminator is in a feedback loop with the ground truth of the images, which we know.
  - The generator is in a feedback loop with the discriminator.

# GAM Structure



# GAN Structure



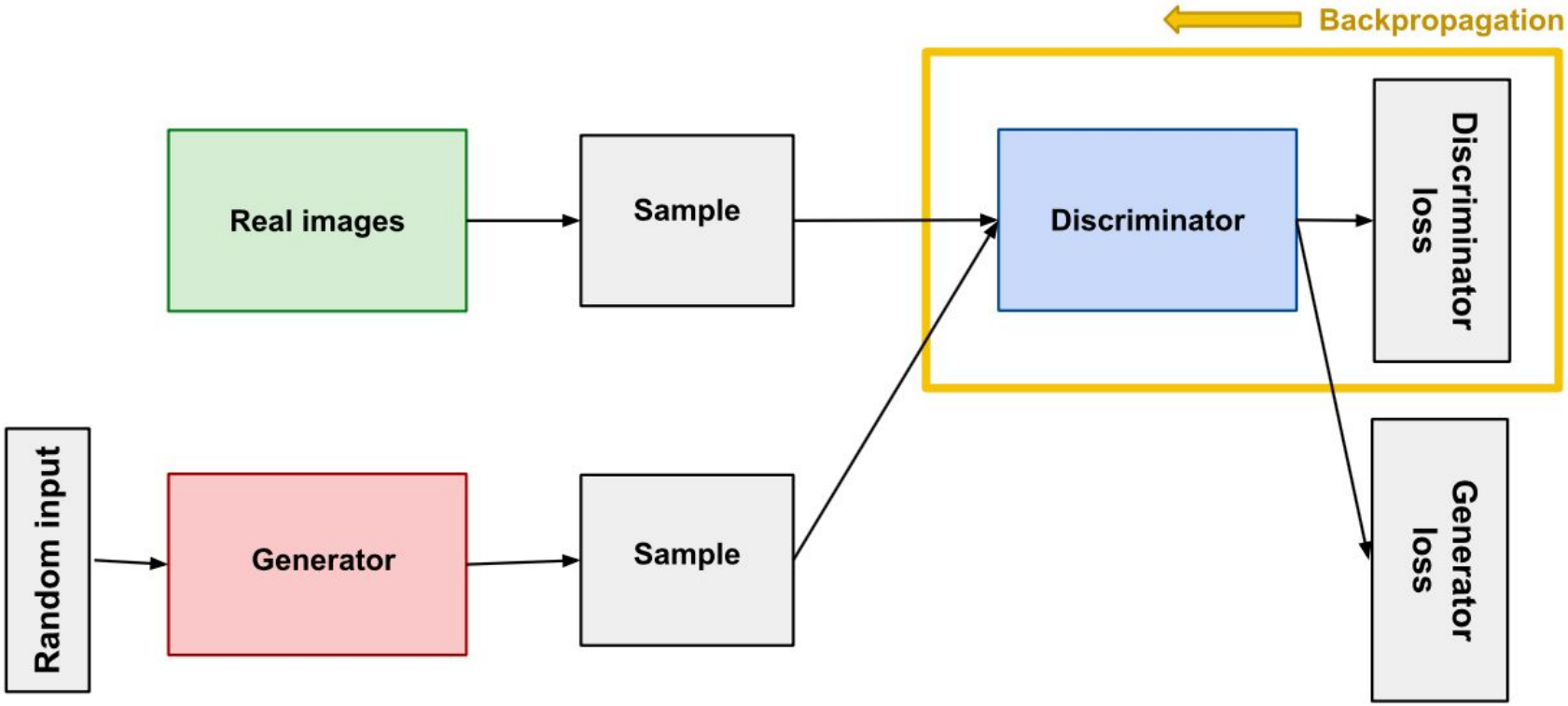

- The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.
- Both are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake:


As training progresses, the generator gets closer to producing output that can fool the discriminator:


Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

# Discriminator




- The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator.
- It could use any network architecture appropriate to the type of data it's classifying.

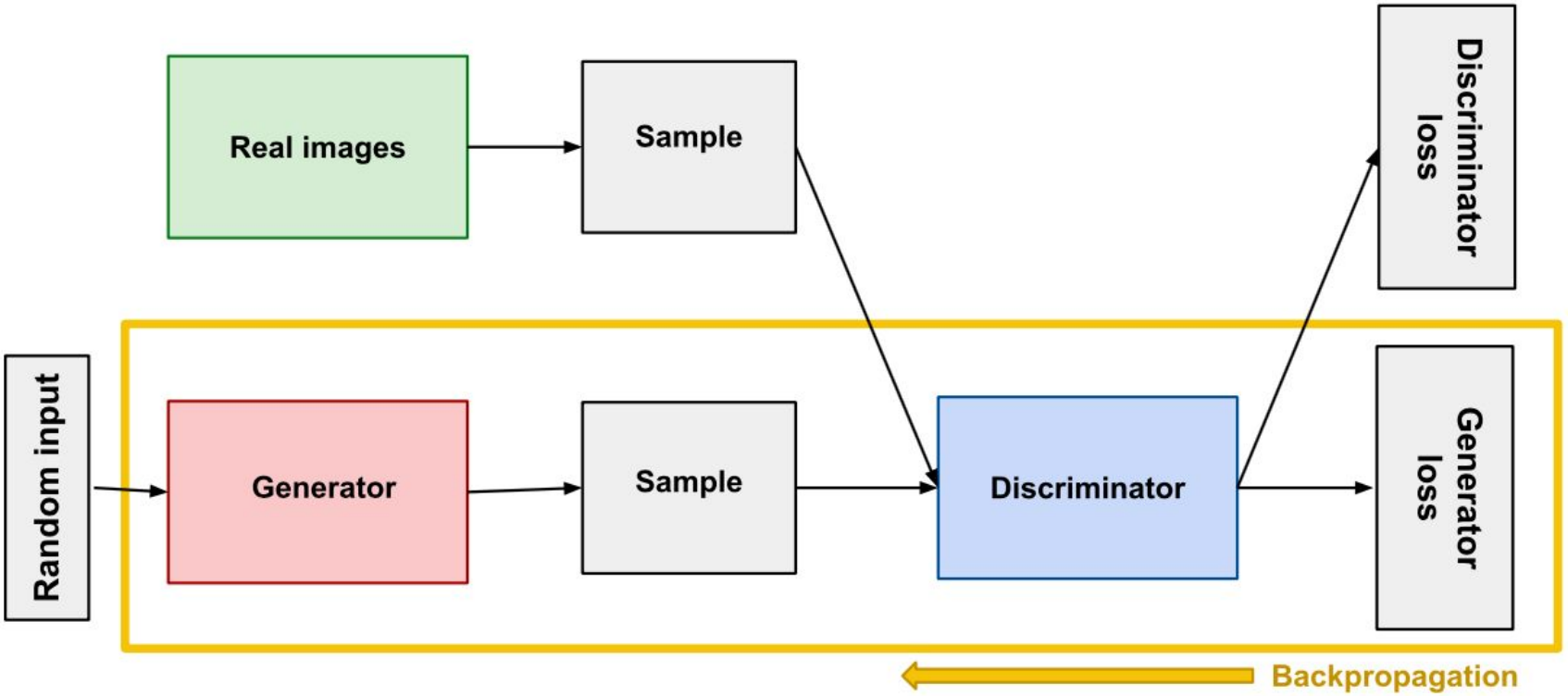
# Discriminator Training Data

- The discriminator's training data comes from two sources:
  - Real data instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
  - Fake data instances created by the generator. The discriminator uses these instances as negative examples during training.
- **During discriminator training the generator does not train.** Its weights remain constant while it produces examples for the discriminator to train on.

# Training the Discriminator

- The discriminator connects to two loss functions.
- During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. We use the generator loss during generator training.
- During discriminator training:
  - The discriminator classifies both real data and fake data from the generator.
  - The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
  - The discriminator updates its weights through **backpropagation** from the discriminator loss through the discriminator network.

# Generator




- The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real.
- Generator training requires tighter integration between the generator and the discriminator than discriminator training requires.

# Training the Generator

The portion of the GAN that trains the generator includes:

- random input
- generator network, which transforms the random input into a data instance
- discriminator network, which classifies the generated data
- discriminator output
- generator loss, which penalizes the generator for failing to fool the discriminator

# Random Input

- Neural networks need some form of input. Normally we input data that we want to do something with, like an instance that we want to classify or make a prediction about. But what do we use as input for a network that outputs entirely new data instances?
- In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, we can get the GAN to produce a wide variety of data, sampling from different places in the target distribution.
- Experiments suggest that the distribution of the noise doesn't matter much, so we can choose something that's easy to sample from, like a uniform distribution. For convenience the space from which the noise is sampled is usually of smaller dimension than the dimensionality of the output space.
- Note: Some GANs use non-random input to shape the output. See GAN Variations.

# Using the Discriminator to Train the Generator

- To train a neural net, we alter the net's weights to reduce the error or loss of its output.
- In our GAN, however, the generator is not directly connected to the loss that we're trying to affect. The generator feeds into the discriminator net, and the discriminator produces the output we're trying to affect. **The generator loss penalizes the generator for producing a sample that the discriminator network classifies as fake.**
- This extra chunk of network must be included in backpropagation. Backpropagation adjusts each weight in the right direction by calculating the weight's impact on the output — how the output would change if you changed the weight. But the impact of a generator weight depends on the impact of the discriminator weights it feeds into. **So backpropagation starts at the output and flows back through the discriminator into the generator.**
- At the same time, we don't want the discriminator to change during generator training. Trying to hit a moving target would make a hard problem even harder for the generator.

# Using the Discriminator to Train the Generator

- So we train the generator with the following procedure:
  1. Sample random noise.
  2. Produce generator output from sampled random noise.
  3. Get discriminator "Real" or "Fake" classification for generator output.
  4. Calculate loss from discriminator classification.
  5. Backpropagate through both the discriminator and generator to obtain gradients.
  6. Use gradients to change only the generator weights.
- This is one iteration of generator training.

# GAN Training

- Because a GAN contains two separately trained networks, its training algorithm must address two complications:
  - GANs must juggle two different kinds of training (generator and discriminator).
  - GAN convergence is hard to identify.

# Alternating Training

- The generator and the discriminator have different training processes. So how do we train the GAN as a whole?
  - We keep the generator constant during the discriminator training phase.
  - Similarly, we keep the discriminator constant during the generator training phase.
- GAN training proceeds in alternating periods:
  - The discriminator trains for one or more epochs.
  - The generator trains for one or more epochs.
  - Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

# (Fleeting) Convergence

- As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake.
- If the generator succeeds perfectly, then the discriminator has a **50% accuracy**. In effect, the discriminator flips a coin to make its prediction.
- This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, **then the generator starts to train on junk feedback**, and its own quality may collapse.
- **For a GAN, convergence is often a fleeting, rather than stable, state.**

# One Loss Function or Two?

- A GAN can have two loss functions: one for generator training and one for discriminator training.
- How can two loss functions work together to reflect a distance measure between probability distributions?
- In the loss schemes we'll look at here, the generator and discriminator losses derive from a single measure of distance between probability distributions.
- In both of these schemes, however, the generator can only affect one term in the distance measure: the term that reflects the distribution of the fake data.
- So during generator training we drop the other term, which reflects the distribution of the real data.
- The generator and discriminator losses look different in the end, even though they derive from a single formula.

# Loss Functions

- **GANs try to replicate a probability distribution**. They should therefore use loss functions that reflect the distance between the distribution of the data generated by the GAN and the distribution of the real data.
- How do you capture the difference between two distributions in GAN loss functions? This question is an area of active research, and many approaches have been proposed.
- We'll address two common GAN loss functions here, both of which are implemented in TF-GAN:
  - **minimax** loss: The loss function used in the paper that introduced GANs: https://arxiv.org/abs/1406.2661
  - **Wasserstein** loss: The default loss function for TF-GAN Estimators. First described in a 2017 paper.
- TF-GAN implements many other loss functions as well.

# Minimax Loss

- In the paper that introduced GANs, the **generator** tries to **minimize** the following function while the **discriminator** tries to **maximize** it:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

- In this function:
  - D(x) is the discriminator's estimate of the probability that real data instance x is real.
  - Ex is the expected value over all real data instances.
  - G(z) is the generator's output when given noise z.
  - D(G(z)) is the discriminator's estimate of the probability that a fake instance is real.
  - Ez is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z)).
  - The formula derives from the cross-entropy between the real and generated distributions.
- The generator can't directly affect the log(D(x)) term in the function, so, for the generator, minimizing the loss is equivalent to **minimizing log(1 - D(G(z)))**.

# Modified Minimax Loss

- The original GAN paper notes that the minimax loss function can cause the GAN to get stuck in the early stages of GAN training when the discriminator's job is very easy.
- The paper therefore suggests modifying the generator loss so that the generator tries to **maximize log D(G(z))**.

# Wasserstein Loss

- This loss function depends on a modification of the GAN scheme (called "Wasserstein GAN" or "WGAN") in which the discriminator does not actually classify instances. For each instance it outputs a number.
- This number does not have to be less than one or greater than 0, so we can't use 0.5 as a threshold to decide whether an instance is real or fake. Discriminator training just tries to make the output bigger for real instances than for fake instances.
- Because it can't really discriminate between real and fake, the WGAN discriminator is actually called a "critic" instead of a "discriminator". This distinction has theoretical importance, but for practical purposes we can treat it as an acknowledgement that the inputs to the loss functions don't have to be probabilities.
- The theoretical justification for the Wasserstein GAN (or WGAN) requires that the weights throughout the GAN be clipped so that they remain within a constrained range.
- Wasserstein GANs are less vulnerable to getting stuck than minimax-based GANs, and avoid problems with vanishing gradients.
- Paper: https://arxiv.org/abs/1701.07875

# Wasserstein Loss

- Critic Loss: D(x) - D(G(z))
  - The discriminator tries to maximize this function. In other words, it tries to maximize the difference between its output on real instances and its output on fake instances.
- Generator Loss: D(G(z))
  - The generator tries to maximize this function. In other words, it tries to maximize the discriminator's output for its fake instances.

# Wasserstein Loss

In these functions:

- D(x) is the critic's output for a real instance.
- G(z) is the generator's output when given noise z.
- D(G(z)) is the critic's output for a fake instance.
- The output of critic D does not have to be between 1 and 0.
- The formulas derive from the distance between the real and generated distributions.

# GAN Common Problems

- **Vanishing Gradients** - Research has suggested that if your discriminator is too good, then generator training can fail due to vanishing gradients. In effect, an optimal discriminator doesn't provide enough information for the generator to make progress.
  - Attempts to Remedy
    - Wasserstein loss
    - Modified minimax loss

# GAN Common Problems

- **Mode Collapse**
  - Usually you want your GAN to produce a wide variety of outputs. You want, for example, a different face for every random input to your face generator.
  - However, if a generator produces an especially plausible output, the generator may learn to produce only that output. In fact, the generator is always trying to find the one output that seems most plausible to the discriminator.
  - If the generator starts producing the same output (or a small set of outputs) over and over again, the discriminator's best strategy is to learn to always reject that output. But if the next generation of discriminator gets stuck in a local minimum and doesn't find the best strategy, then it's too easy for the next generator iteration to find the most plausible output for the current discriminator.
  - Each iteration of generator over-optimizes for a particular discriminator, and the discriminator never manages to learn its way out of the trap. As a result the generators rotate through a small set of output types. This form of GAN failure is called **mode collapse.**
  - Attempts to Remedy
    - Wasserstein loss
    - Unrolled GANs https://arxiv.org/pdf/1611.02163.pdf

# GAN Common Problems

- **Failure to Converge** - GANs frequently fail to converge
  - Attempts to Remedy via regularization
    - Adding noise to discriminator inputs https://arxiv.org/pdf/1701.04862.pdf
    - Penalizing discriminator weights https://arxiv.org/pdf/1705.09367.pdf

# GAN Variants

- Progressive GANs
- Conditional GANs
- Image-to-Image Translation
- CycleGAN
- Text-to-Image Synthesis
- Super-resolution
- Face Inpainting
- Text-to-Speech

# Other Generative Models*

- Gaussian mixture model (and other types of mixture model)
- Hidden Markov model
- Probabilistic context-free grammar
- Bayesian network (e.g. Naive bayes, Autoregressive model)
- Averaged one-dependence estimators
- Latent Dirichlet allocation
- Boltzmann machine (e.g. Restricted Boltzmann machine, Deep belief network)
- Variational autoencoder
- Flow-based generative model
- Energy based model

*Wikipedia

# GANs for Finance

Cornell University

arXiv.org > q-fin > arXiv:1907.06673

Quantitative Finance > Mathematical Finance

*[Submitted on 15 Jul 2019 (v1), last revised 21 Dec 2019 (this version, v2)]*

## Quant GANs: Deep Generation of Financial Time Series

Magnus Wiese, Robert Knobloch, Ralf Korn, Peter Kretschmer

Cornell University

arXiv.org > q-fin > arXiv:2106.06364

Quantitative Finance > Computational Finance

*[Submitted on 11 Jun 2021 (v1), last revised 6 Jul 2021 (this version, v2)]*

## Generative Adversarial Networks in finance: an overview

Florian Eckerli, Joerg Osterrieder

Google Scholar

generative adversarial networks for finance

Articles

About 18,600 results (**0.06** sec)

Any time
Since 2021
Since 2020
Since 2017
Custom range...

Sort by relevance
Sort by date

Any type
include patents
include citations

Review articles

Create alert

[HTML] Modeling **financial** time-series with **generative adversarial networks**
S Takahashi, Y Chen, K Tanaka-Ishii - Physica A: Statistical Mechanics and …, 2019 - Elsevier
**Financial** time-series modeling is a challenging problem as it retains various complex statistical properties and the mechanism behind the process is unrevealed to a large extent …
☆ Cite Cited by 41 Related articles All 7 versions

Using **generative adversarial networks** to synthesize artificial **financial** datasets
D Efimov, D Xu, L Kong, A Nefedov… - arXiv preprint arXiv …, 2020 - arxiv.org
**Generative Adversarial Networks** (GANs) became very popular for generation of realistically looking images. In this paper, we propose to use GANs to synthesize artificial **financial** data …
☆ Cite Cited by 7 Related articles All 7 versions »

**Generative adversarial networks** for **financial** trading strategies fine-tuning and combination
A Koshiyama, N Firoozye, P Treleaven - arXiv preprint arXiv:1901.01751, 2019 - arxiv.org
Systematic trading strategies are algorithmic procedures that allocate assets aiming to optimize a certain performance criterion. To obtain an edge in a highly competitive …
☆ Cite Cited by 25 Related articles All 6 versions »

Corrgan: Sampling realistic **financial** correlation matrices using **generative adversarial networks**
G Marti - ICASSP 2020-2020 IEEE International Conference on …, 2020 - ieeexplore.ieee.org
We propose a novel approach for sampling realistic **financial** correlation matrices. This approach is based on **generative adversarial networks**. Experiments demonstrate that …
☆ Cite Cited by 17 Related articles All 7 versions

Enriching **financial** datasets with **generative adversarial networks**
F De Meer Pardo - 2019 - repository.tudelft.nl
The scarcity of historical **financial** data has been a huge hindrance for the development algorithmic trading models ever since the first models were devised. Most **financial** models …
☆ Cite Cited by 7 Related articles »

# Deepfakes

- **Deep** Learning + **Fake** images & videos = **Deepfakes**
  - **https://www.kdnuggets.com/2018/03/exploring-deepfakes.html/2**
- Utilizes tech like
  - Generative Adversarial Networks
  - Autoencoders –
    - Unsupervised learning technique
    - A neural network architecture capable of discovering structure within data in order to develop a compressed representation of the input.
    - https://www.jeremyjordan.me/autoencoders/
- Powerful technology with incredible potential for good and bad use cases
  - https://www.youtube.com/watch?v=6yw9YszJEmA
  - https://www.cbsnews.com/news/deepfake-artificial-intelligence-60-minutes-2021-10-10/
- Good
  - Education/training
  - Your future financial advisor (deepfake connected to financial chatbot)?
- Beware
  - Blackmail
  - **Fake data**
  - Fake news
  - Financial fraud
    - Social engineering
  - Sockpuppets - an alternative online identity or user account used for purposes of deception.