Generative Adversarial Networks (GANs)

Advanced Research Topics – 7PAM2016

Dr Vanessa Graber (based on slides by Dr William Alston)

Learning outcomes

After this lectures and the next tutorial, you will:

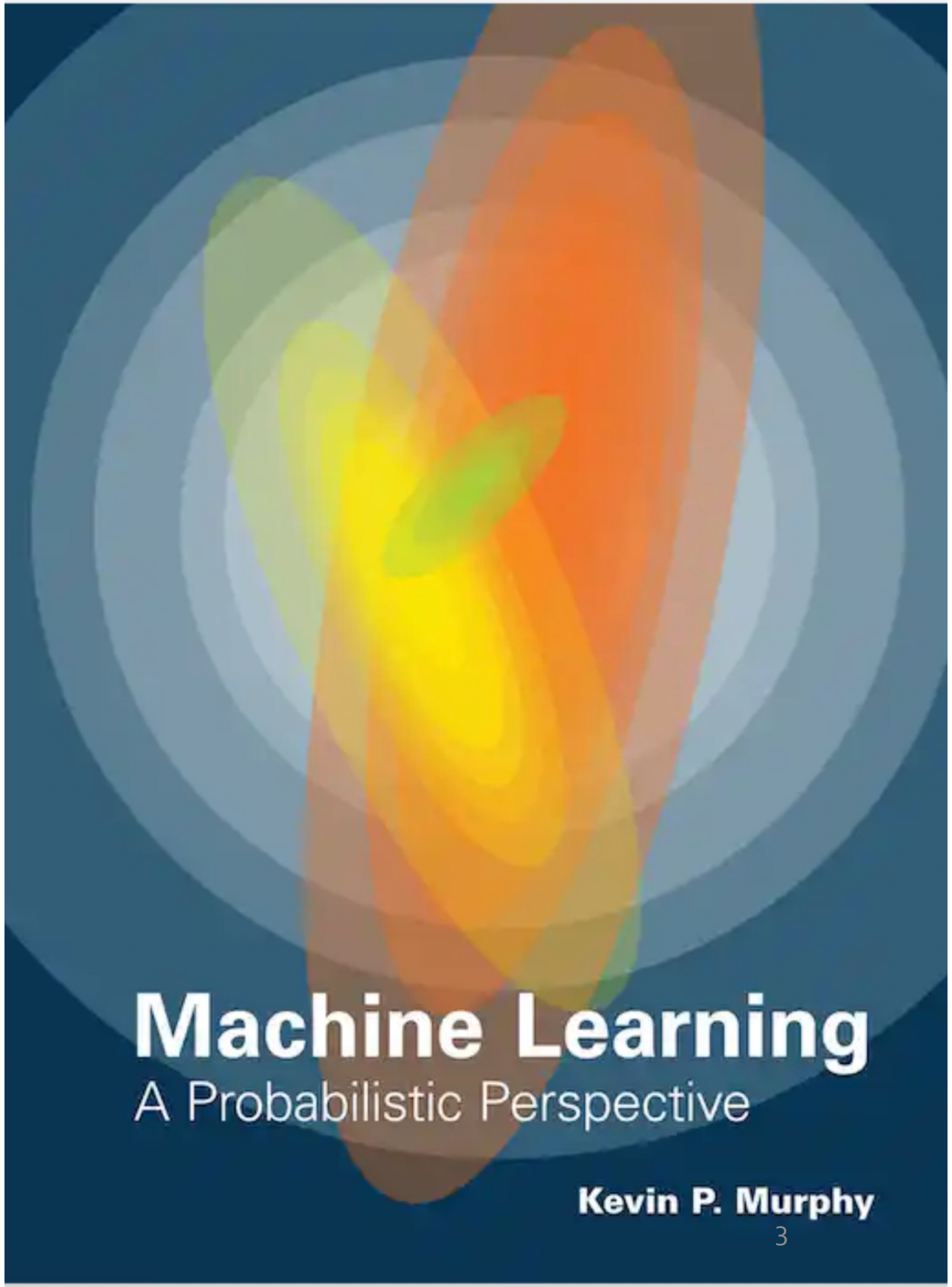
- Understand context for GANs, specifically supervised vs. unsupervised learning and discriminative vs. generative modelling.
- Understand the GAN architecture for automatically training a generative model by treating the unsupervised problem as supervised and using both a generator and a discriminator.
- Have seen research common examples of GAN architectures.
- Be able to implement a simple GAN in Python.

Reading

Probabilistic Machine Learning
Advanced Topics – Chapter 26

Kevin P. Murphy 2022

https://herts.instructure.com/courses/112318/files/7674016/download?download_frd=1



Further reading

Some key references in the field

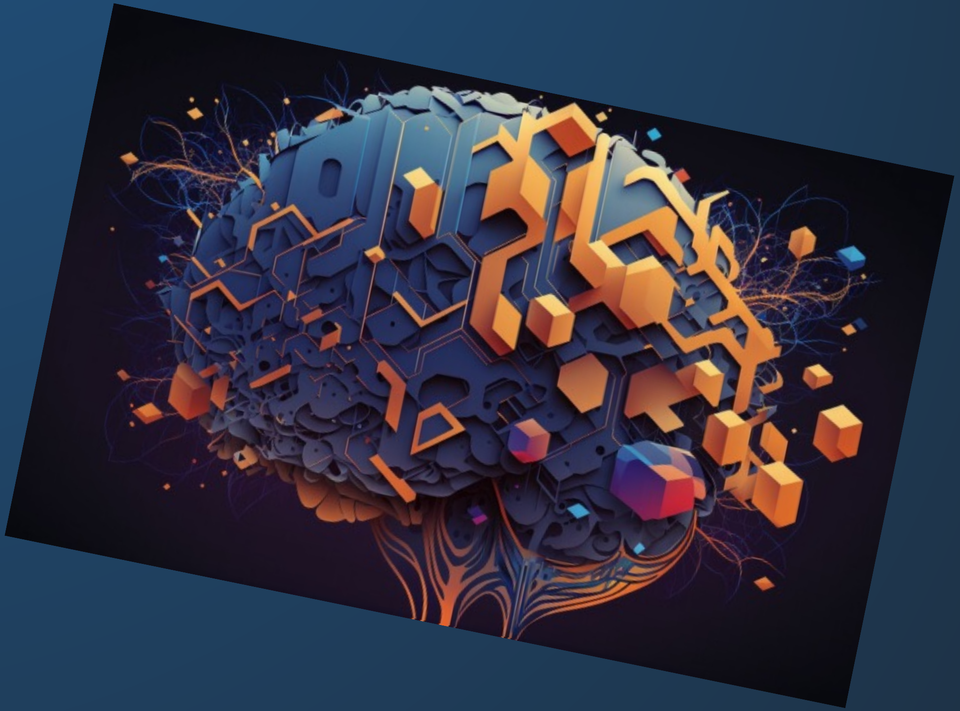
- Goodfellow et al. Generative Adversarial Nets. NIPS (2014)
- Denton, et al. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. NIPS (2015)
- Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. ICLR (2016)
- Miyato et al. Spectral Normalization for Generative Adversarial Networks. ICLR (2018)
- Brock et al. Large Scale GAN Training for High Fidelity Natural Image Synthesis. ICLR (2019)
- Karras et al. A Style-Based Generator Architecture for Generative Adversarial Networks. CVPR (2019)

Introduction and key concepts

How do GANs work?

Types of GANs and their applications

Summary

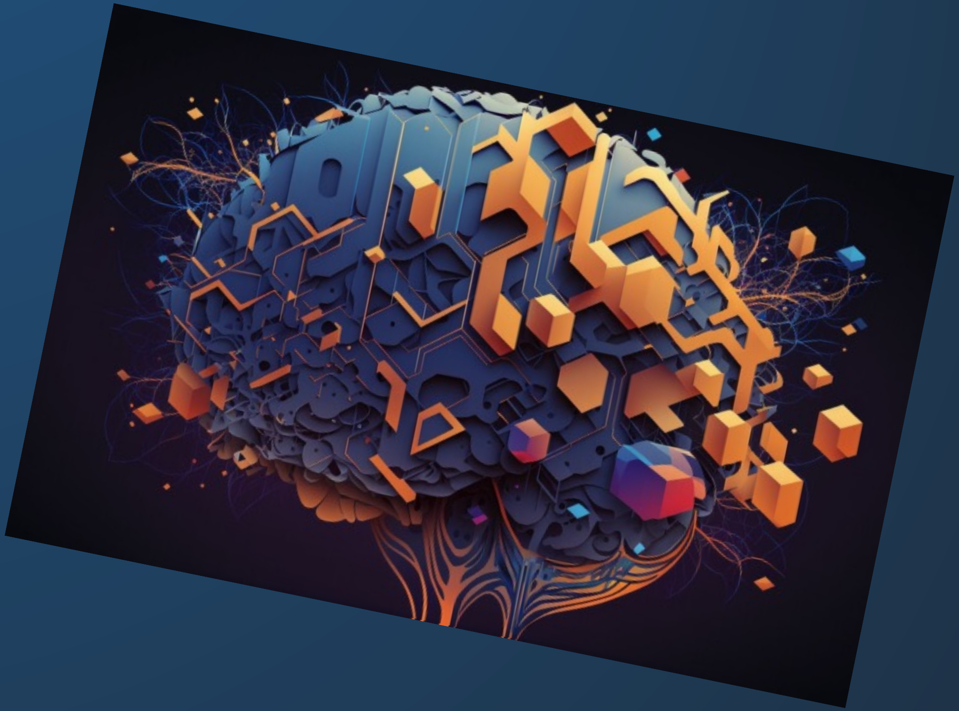


Introduction and key concepts

How do GANs work?

Types of GANs and their applications

Summary

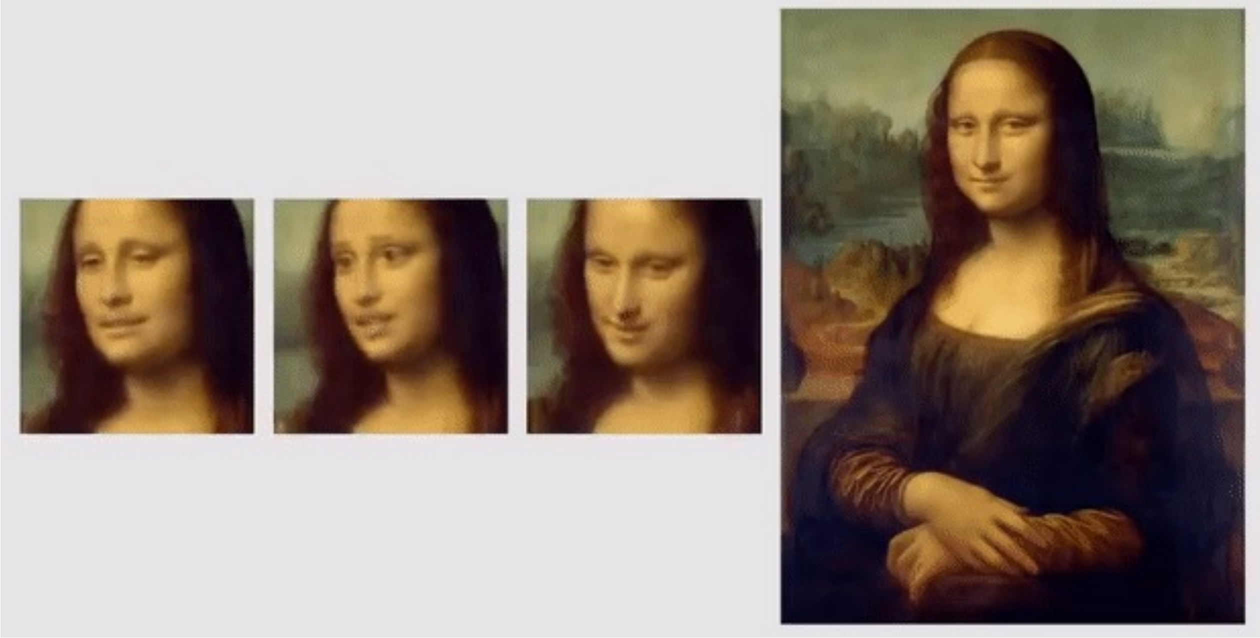


What GANs can do

An example: generate a ”living portrait” (Zakharov et al. 2019)

- A generative adversarial network (GAN) is an unsupervised deep learning model using, e.g., CNNs to generate new samples that are indistinguishable from a set of training data.

- There is no easy way to assess how likely it is that a data point was generated from the model.

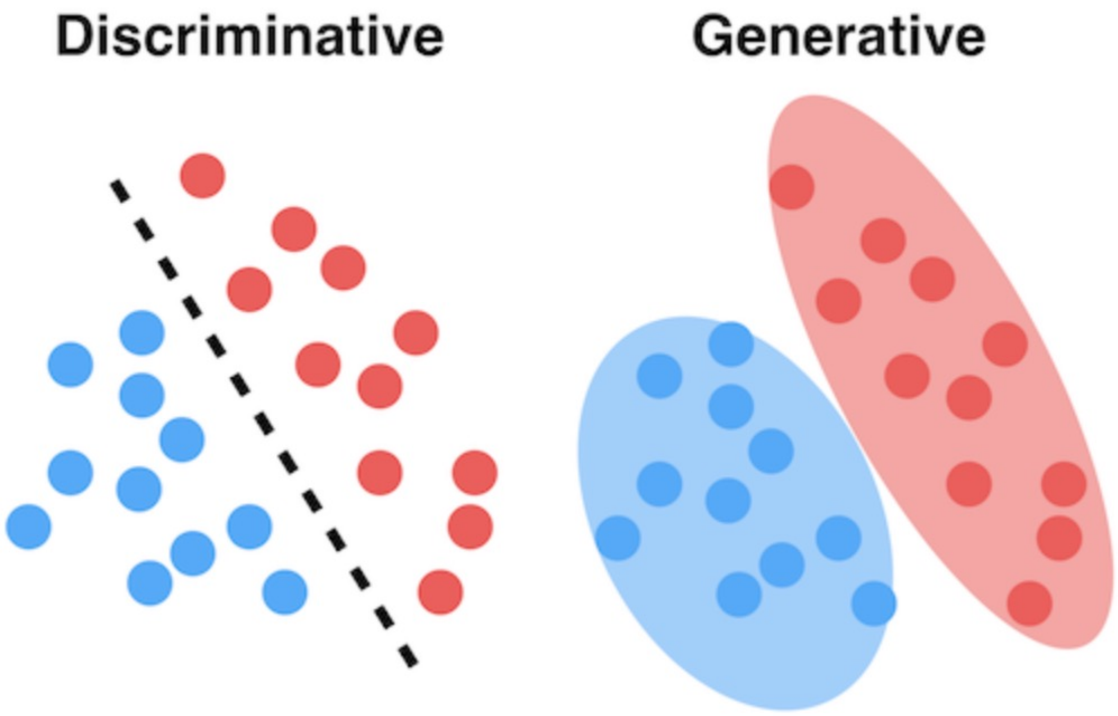


GANs have a lot of applications but also pose serious risks (deepfakes, etc.)!!

Discrimination vs. generation

The concepts

- Most of the neural network applications you have come across so far were likely implemented using **discriminative models**, which aim
  - to draw boundaries in data spaces.
  - to predict the labels of the data.



- GANs, on the other hand, are part of a different class of models known as **generative models**. These aim
  - to learn the true (unknown) underlying distribution from the training data.
  - to understand how the data was generated.

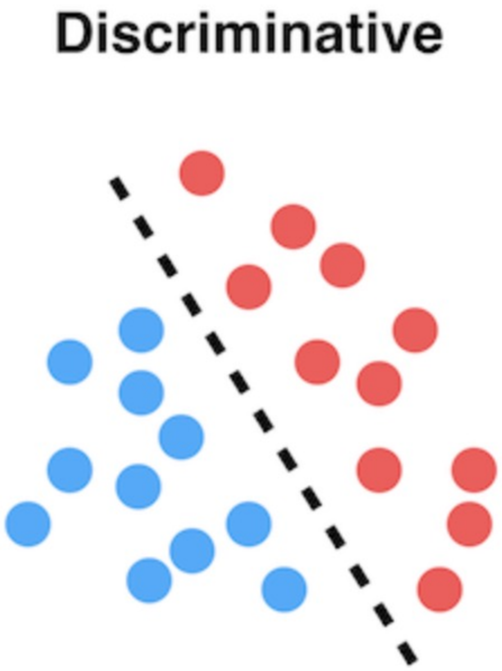
Discriminative models

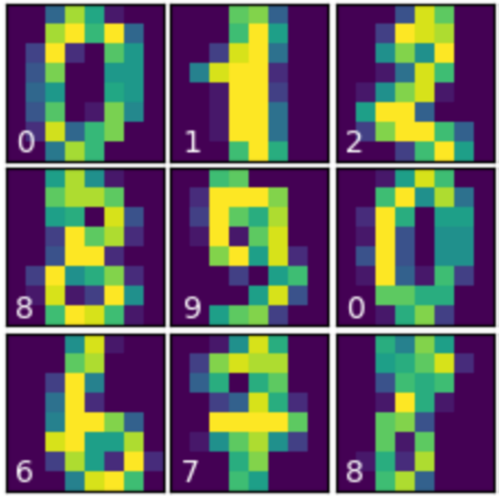
The set-up

- These models are used for most supervised **classification or regression problems**. E.g., you might want to construct a classifier that recognises handwritten digits. For that, we would use a dataset of labelled images (e.g., MNIST).

- From this, the model **learns the boundaries** between the classes. These boundaries are then used to discriminate an input and predict its class.

Mathematically speaking: the model learns the conditional probability P(y|x) of the output y given the input x.



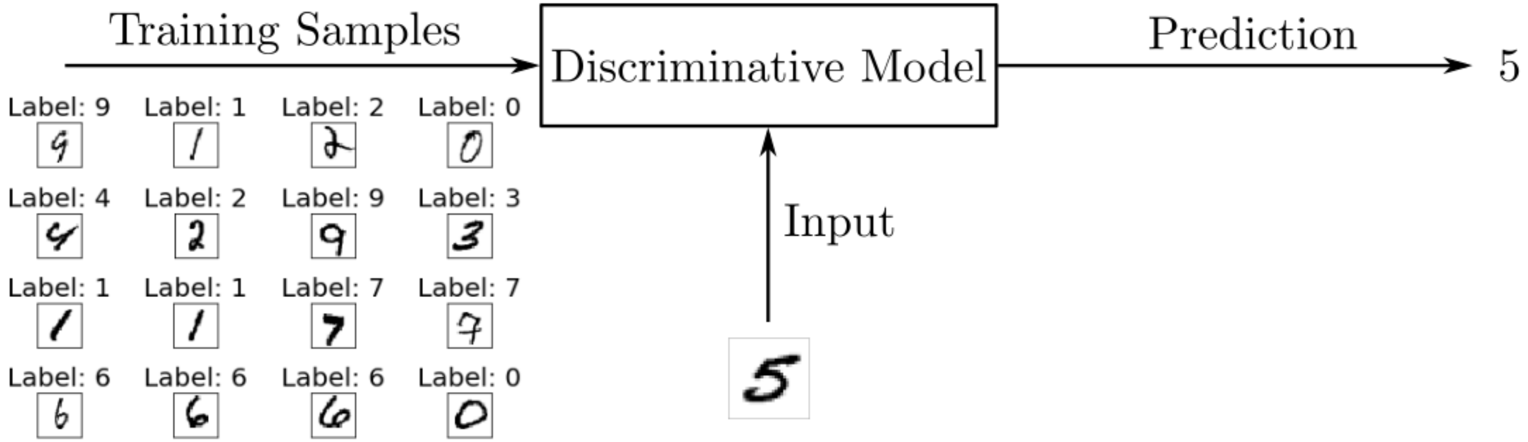


Discriminative models

Training

- To train our classifier, we would use an algorithm to adjust the model's parameters. The goal is to **minimise a loss function** so that we learn the conditional probability distribution P(y|x).

- Once trained, we can use the discriminative model to classify a new digit by estimating the most probably digit:



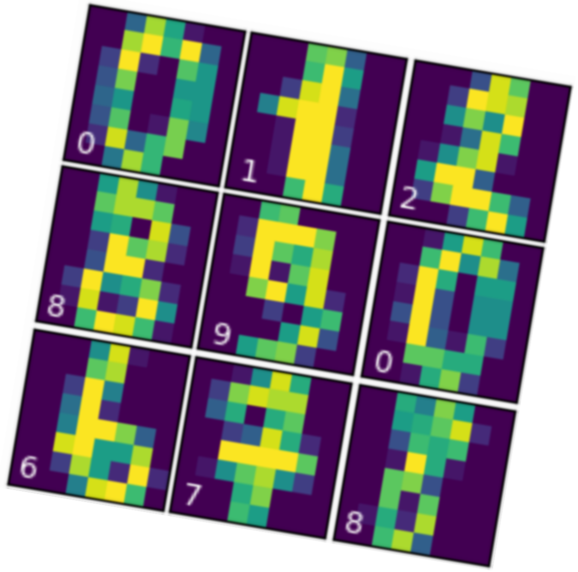
Training Samples

Discriminative Model

Prediction

5

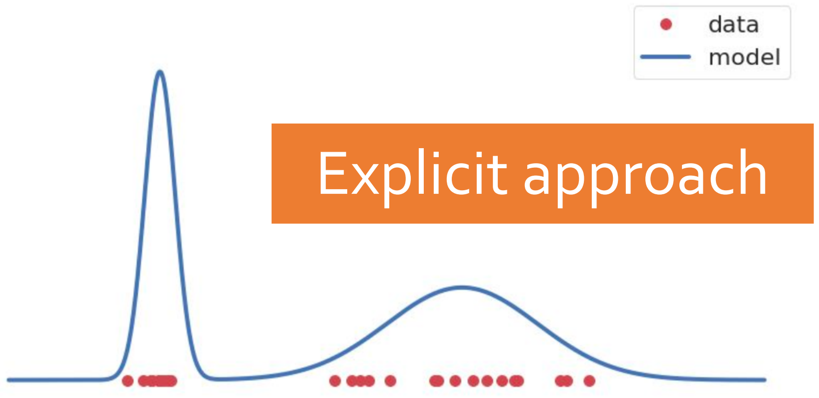
Input



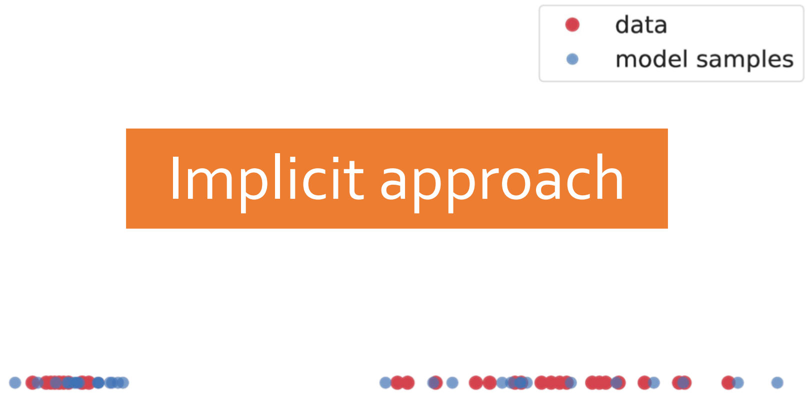
Generative models

The set-up

- Generative models, like GANs, are trained to describe how a dataset is generated in terms of a probabilistic model. By sampling from a generative model, we then generate new data.



Explicit approach



Implicit approach

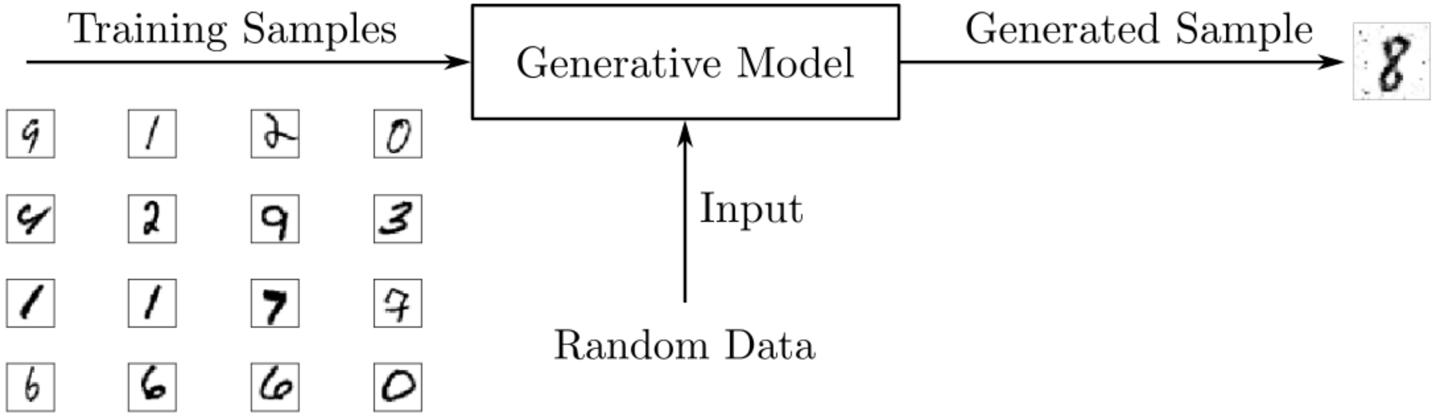
- While discriminative models are used for supervised learning, generative models are often used with unlabelled datasets and can be seen as a form of **unsupervised learning**.

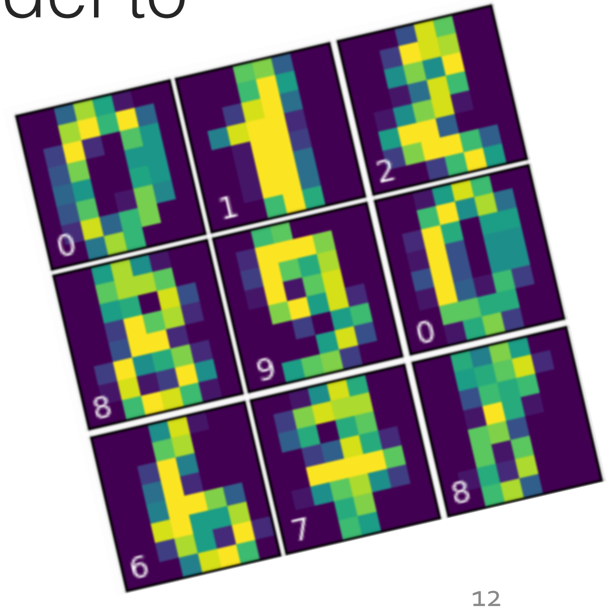
Generative models

Training

- For a dataset of hand-written digits, we would train our generative model to **generate new digits**. During training, we would use an algorithm to adjust the model's parameters to minimise a loss function and learn the **probability distribution** of the training set.

- Once trained, we can then use the generative model to create new samples as illustrated below:





Generative models

We can distinguish

EXPLICIT LIKELIHOOD MODELS

with access to underlying probability density distribution:

- Maximum likelihood methods
  - PPCA / factor analysis / mixture models
  - PixelCNN / PixelRNN
  - Wavenet
  - Autoregressive language models
- Approximate maximum likelihood methods
  - Boltzmann machines
  - Variational autoencoders

IMPLICIT MODELS

without access to likelihoods:

- Generative adversarial networks
- Moment matching machines

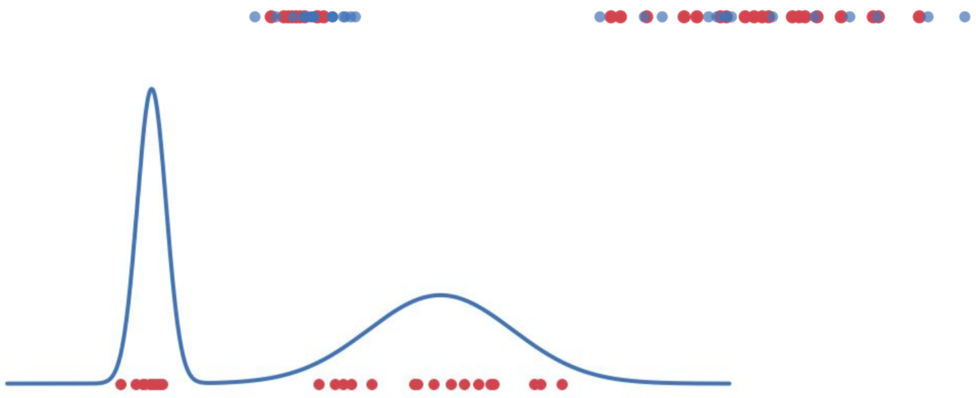
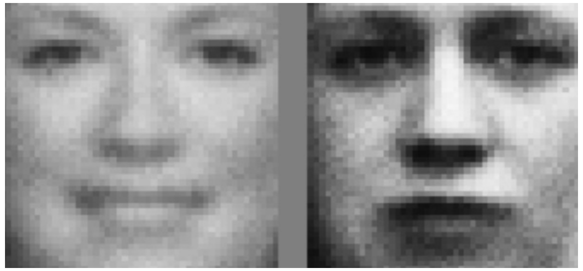


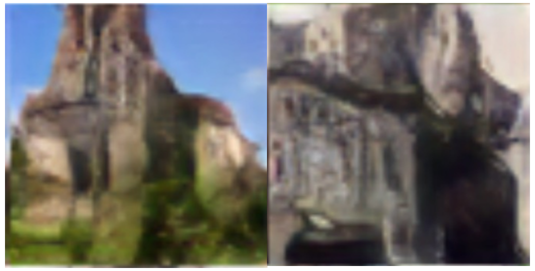
Image generation with GANs

Some examples from key references

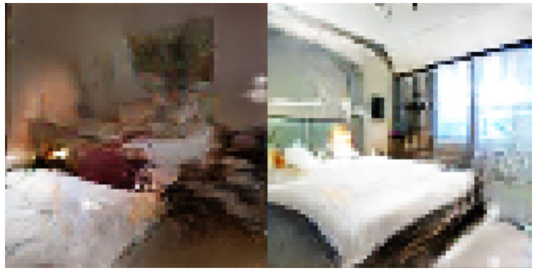
What can you see?



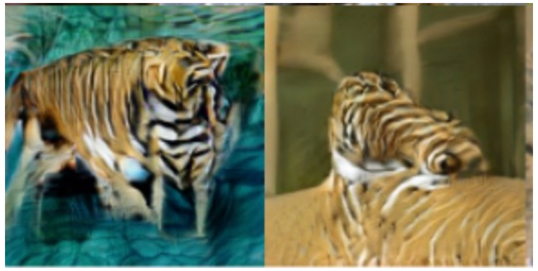
Goodfellow et al. (2014)



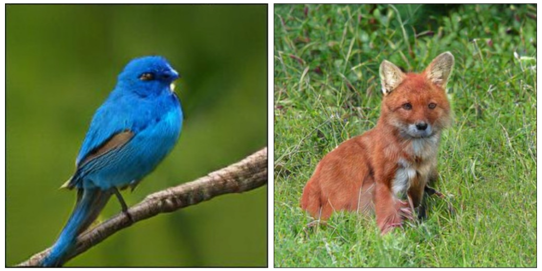
Denton et al. (2015)



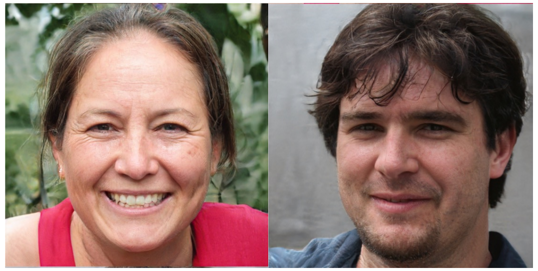
Radford et al. (2016)



Miyato et al. (2018)



Brock et al. (2019)



Karras et al. (2019)

GANs

The key idea

- These networks provide a path to sophisticated **domain-specific data augmentation** and a solution to problems that require a generative approach, such as image-to-image translation.

GANs learn an implicit model (no access to the likelihood) through a two-player game.

GANs

Generator vs. discriminator

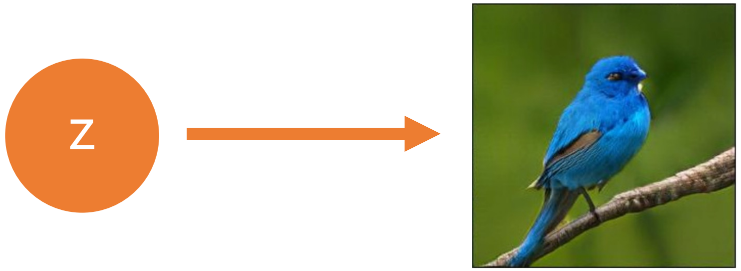
- The **two players** interacting in a GAN are the following:

DISCRIMINATOR

Learns to distinguish between real and generated (fake) data

GENERATOR

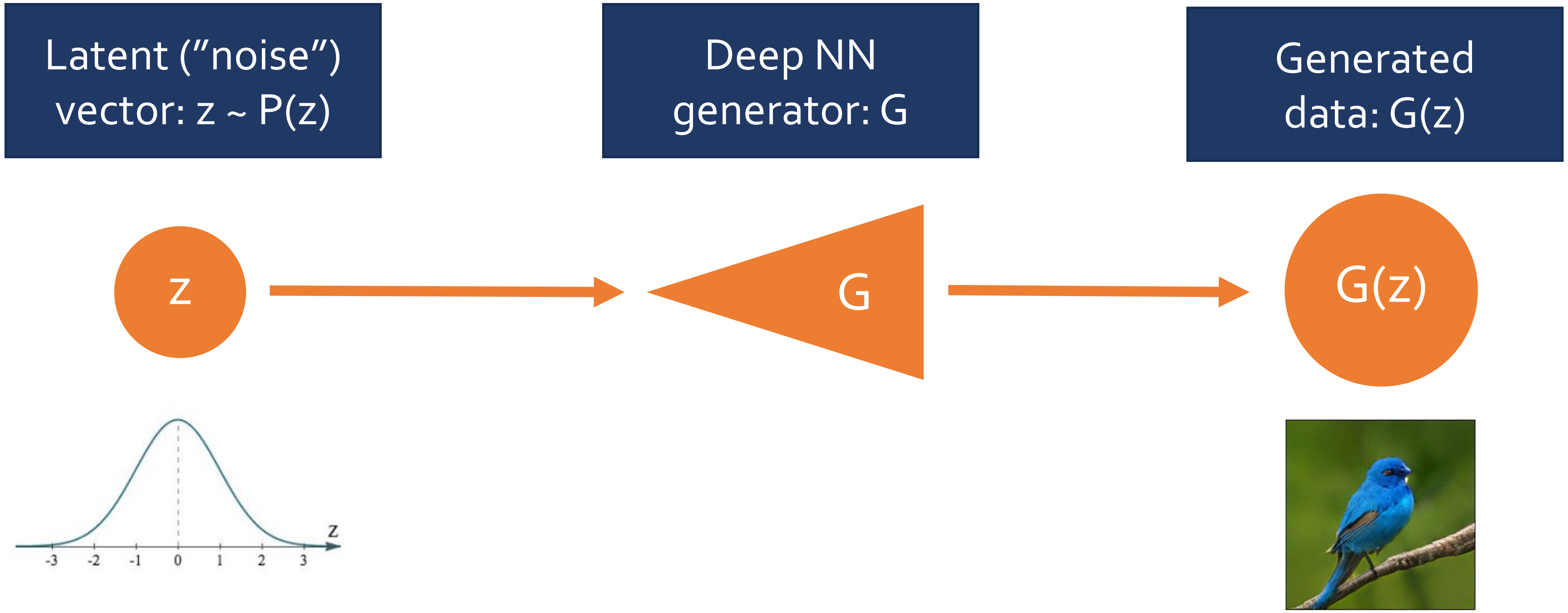
Learns to generate data to “fool” the discriminator.



Generator

The set-up

- The generator is a deep neural network that learns to generate data from noise. Initially, the output will look nothing like the desired output (e.g., bird image). This makes it easy to identify the output as ”fake”.



Latent (”noise”) vector: z ~ P(z)

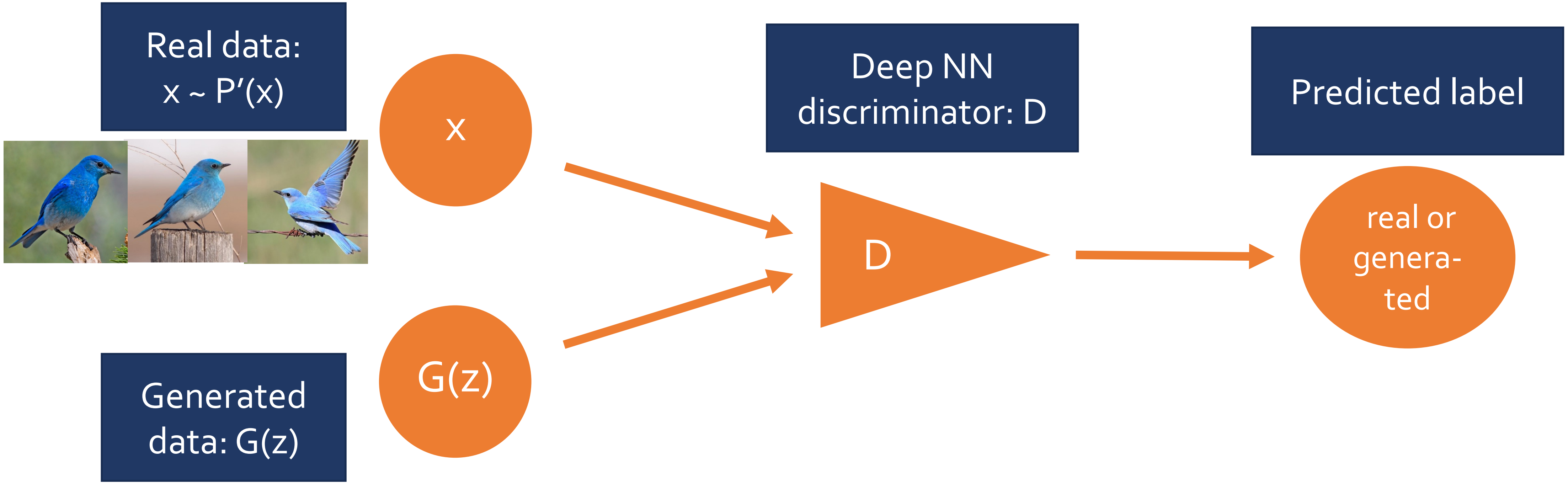
Deep NN generator: G

Generated data: G(z)

Discriminator

The set-up

- The discriminator has access to real data and knows what the true output should look like. It then tries to distinguish these true data samples from those created by the generator.



Real data: x ~ P′(x)

x

Generated data: G(z)

G(z)

Deep NN discriminator: D

D

Predicted label

real or genera-ted

~~Discriminator~~ Teacher

Different naming

- To use **less adversarial (negative) language**, the discriminator is now also referred to as **the teacher**. The generator makes the teacher "happy" by making the generated data look real.

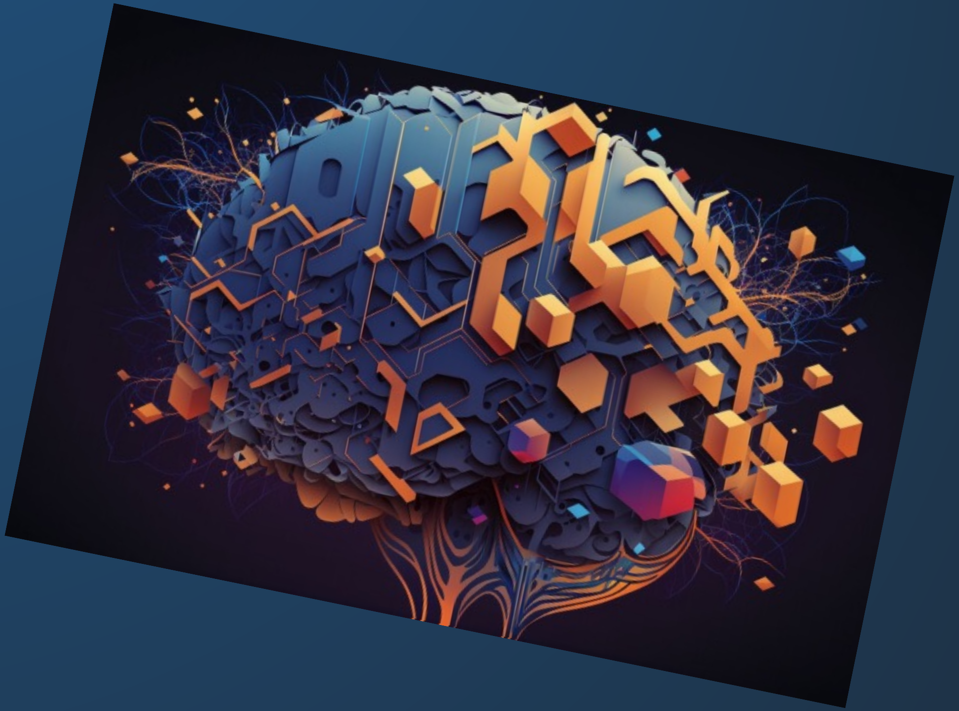
The teacher tells the generator how to improve and produce more realistic output!! Due to the iterative interplay between both components, GANs learn to produce more and more realistic output.

Introduction and key concepts

How do GANs work?

Types of GANs and their applications

Summary



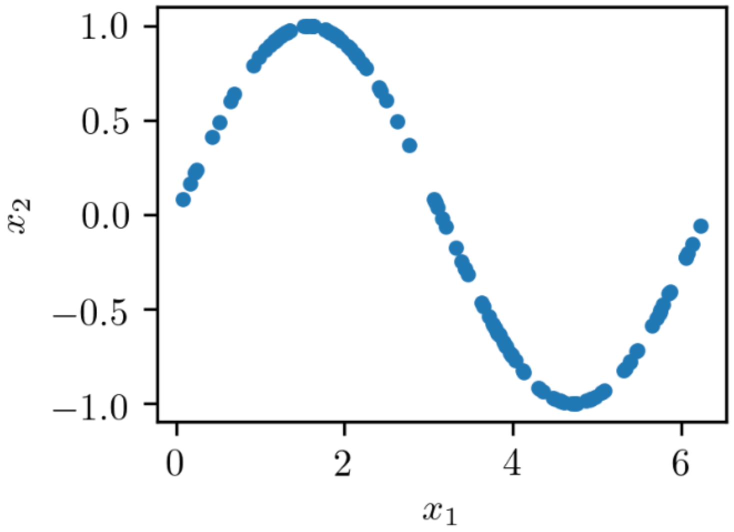
GAN architecture

A toy example I

- The generator (G) estimates the probability distribution of the real samples to provide generated samples resembling real data. The discriminator (D), in turn, is trained to estimate the probability that a given sample came from the real data rather than being generated.

- The two compete against each other: G tries to get better at fooling D, while D tries to get better at identifying samples generated by G.

Let's consider a toy example with a dataset composed of two-dimensional samples (x_1, x_2) with x_1 in the interval from 0 to 2π and $x_2 = \sin(x_1)$, as illustrated in the figure.

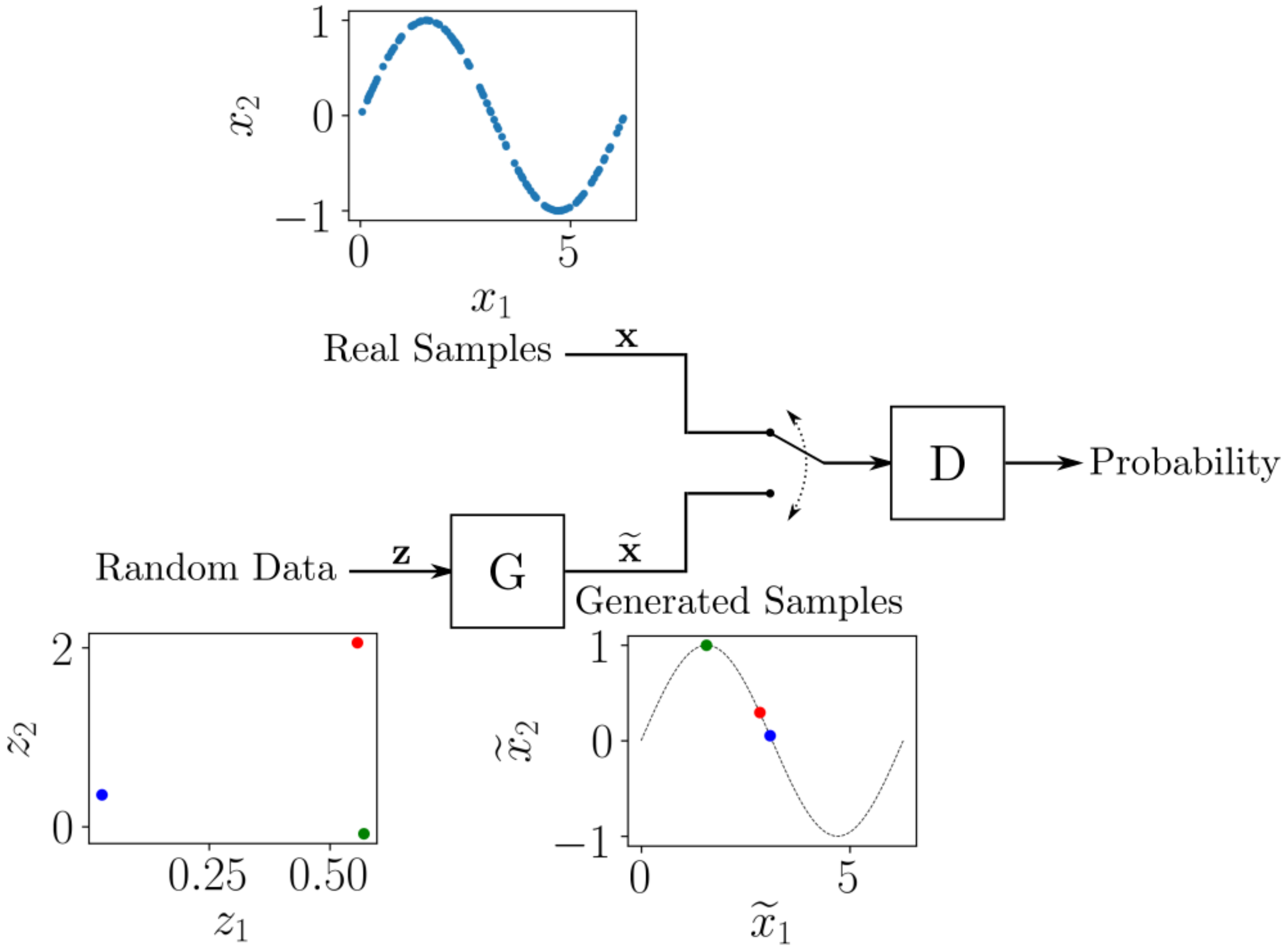


GAN architecture

A toy example II

- The purpose of a GAN is now to generate pairs $\tilde{\mathrm{x}} = (\tilde{\mathrm{x}}_1, \tilde{\mathrm{x}}_2)$ from random data $\mathrm{z} = (\mathrm{z}_1, \mathrm{z}_2)$, that resemble the samples of the initial dataset $\mathrm{x} = (\mathrm{x}_1, \mathrm{x}_2)$.

- The general structure of this architecture is illustrated in the sketch on the right-hand side.

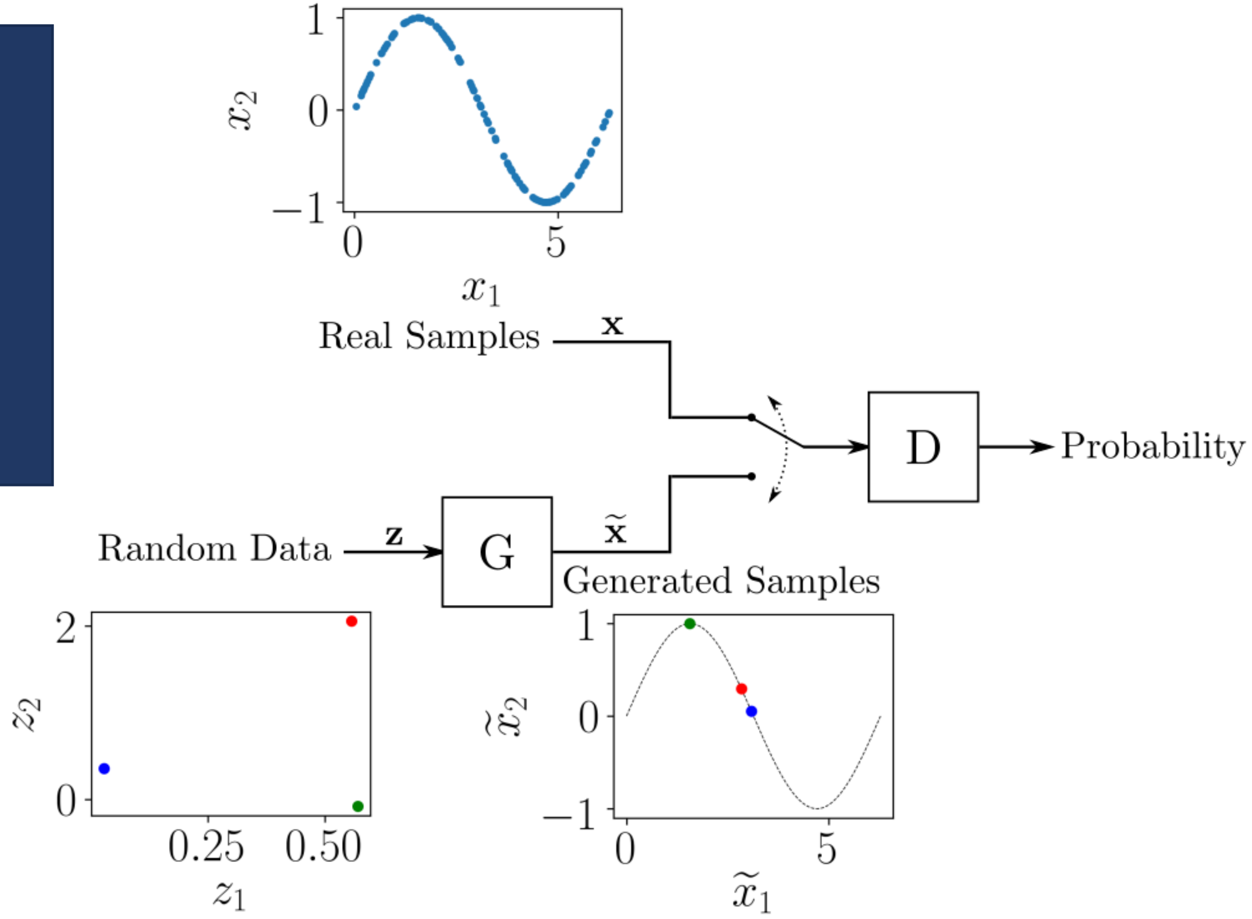


GAN architecture

Generator neural network

The structure of the neural network G is arbitrary. This allows us to use multilayer perceptrons (MLPs), convolutional neural network (CNNs), or any other structure.

The only requirement is that the dimensions of the input and output match the dimensions of the latent space and the real data, respectively.

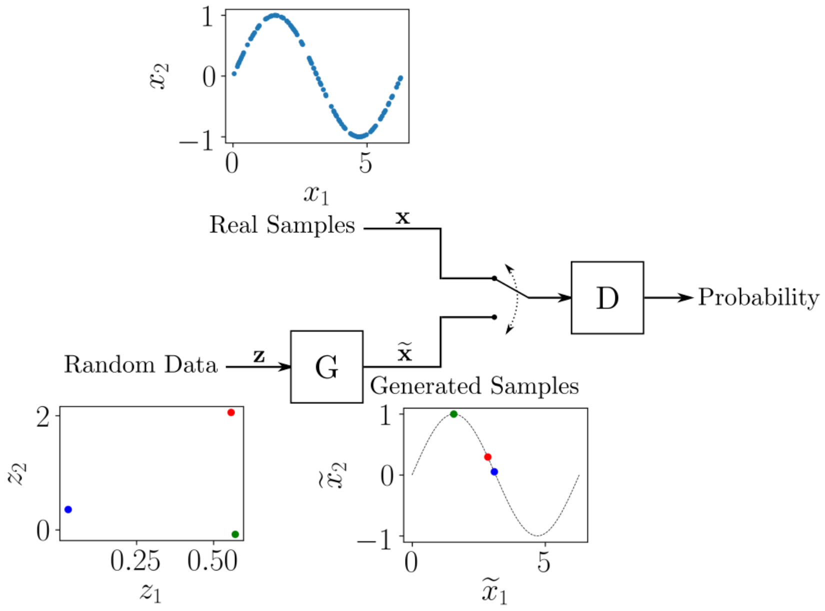


GAN architecture

Discriminator neural network

- The discriminator D is fed either real samples from the training dataset or generated samples provided by the generator G. Its role is to estimate the probability that the input is real.

- The training is performed such that D outputs 1 when it sees a real sample and 0 when it is fed a generated sample.



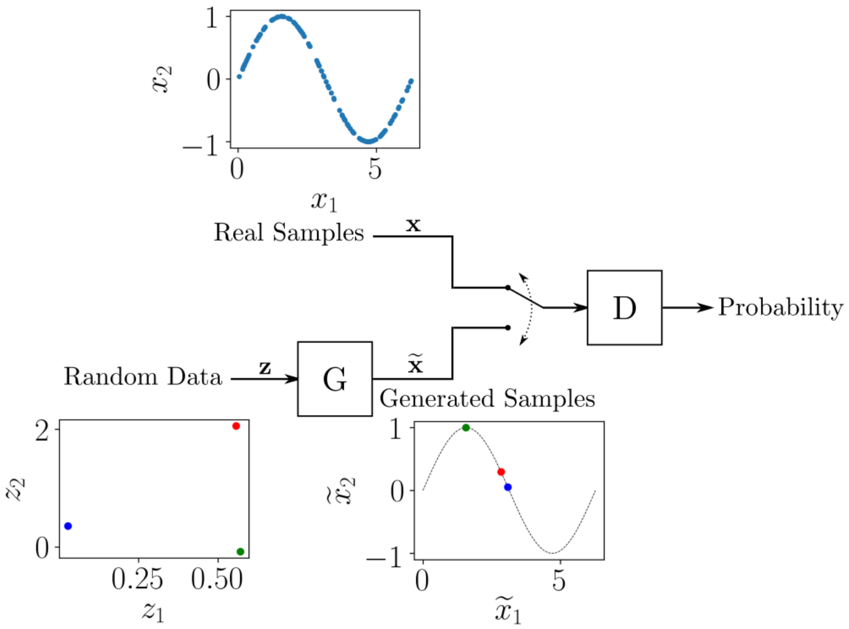
As for G, we can choose an arbitrary network structure for D as long as it respects the necessary input and output dimensions.

In our example, the input is two-dimensional, while the output of the binary discriminator is typically a scalar ranging from 0 to 1.

GAN optimisation

Min-max game

The GAN training process consists of a two-player min-max game in which D is trained to minimise the discrimination error between real and generated samples, while G is trained to maximise the probability of D making a mistake.



- Although the dataset containing the real data is not labelled, the training processes for D and G are performed in a **supervised way**.
- At certain stages in the training, the parameters of D and G are updated. For the original GAN proposal, the parameters of D are updated k times, while those of G are updated only once per training step (see below).

GAN optimisation

The mathematics

- What we just described in a contextual way, we can mathematically express as follows, where V denotes the value function of our optimisation problem:

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{Log-probability that D correctly predicts that real data samples x are real}} + \underbrace{\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{Log-probability that D correctly predicts that generated data samples G(z) are generated}}$$

GAN optimisation

The mathematics

- What we just described in a contextual way, we can mathematically express as follows, where V denotes the value function of our optimisation problem:

$$\min_{G}\max_{D} V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{Log-probability that D correctly predicts that real data samples x are real}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}_{\text{Log-probability that D correctly predicts that generated data samples G(z) are generated}}$$

Discriminator’s (D) goal: maximise prediction accuracy

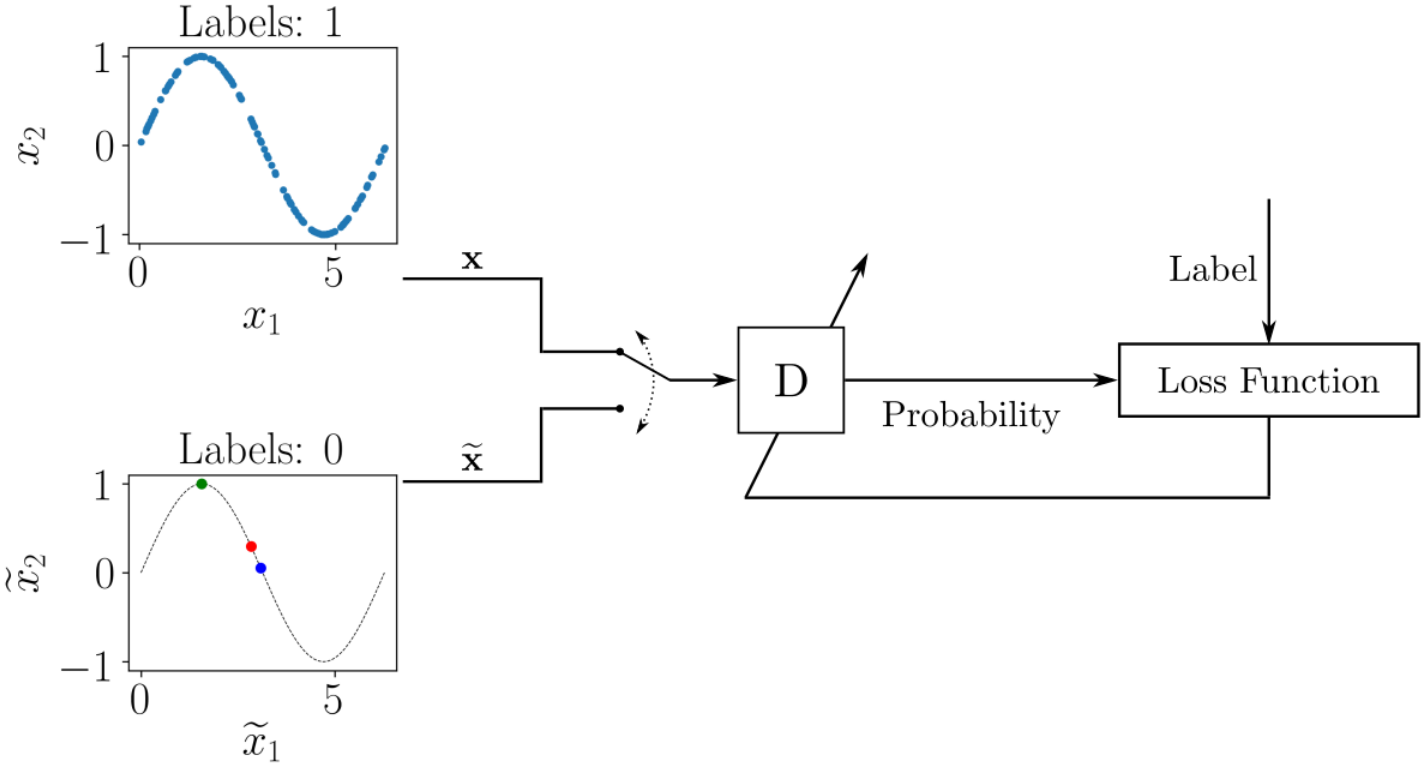
Generator’s (G) goal: minimise prediction accuracy by fooling D into believing its outputs G(z) are real as often as possible.

GAN optimisation

The training process I

- To train D, at each iteration, we label some real samples taken from the training data as 1 and some generated samples from G as 0.

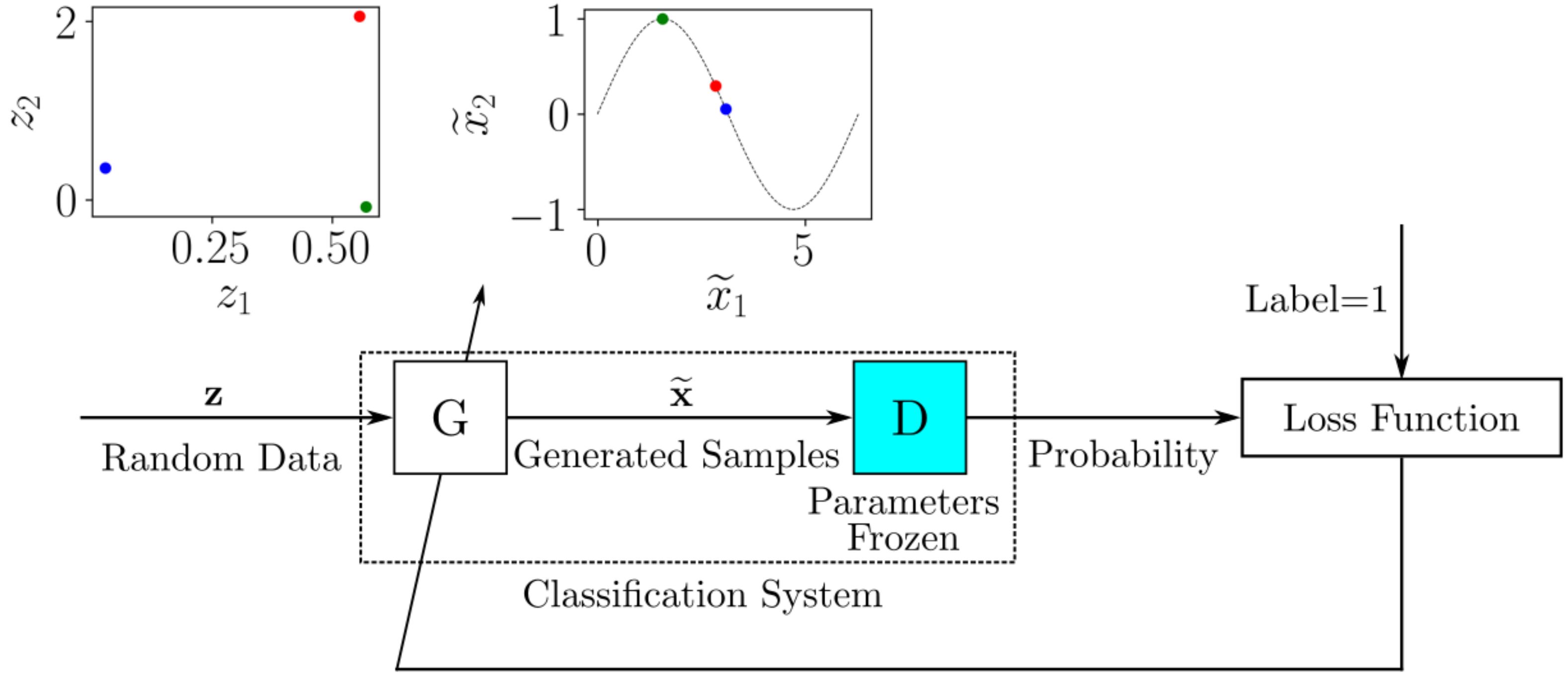
This way, we can use a conventional supervised training framework to update the parameters of D to minimise our loss function for each batch of training data.



GAN optimisation

The training process II

- After the parameters of the discriminator D have been updated, we then train the generator G to produce better generated samples.



Note that the output of G is connected to D, whose parameters are kept frozen during the updating of G.

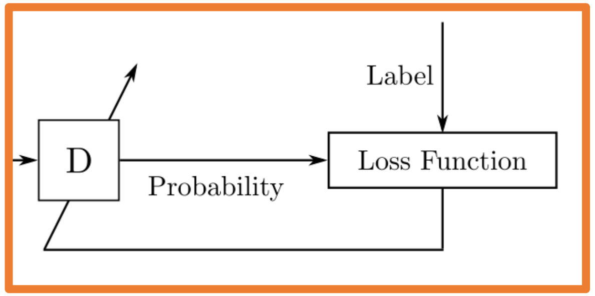
GAN optimisation

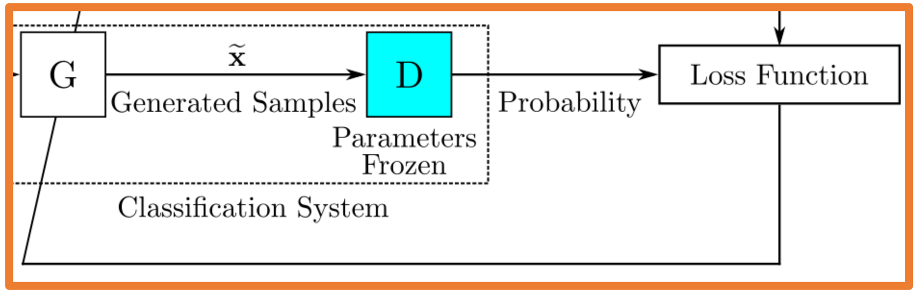
The original algorithm by Goodfellow et al. (2014)

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 $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
              $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
          $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$
end for
```

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





GAN optimisation

The output

- During training, as we update the parameters of the discriminator D and the generator G, we expect that generated samples created by G will more and more closely resemble real data. At the same time, D will have more and more trouble distinguishing between real and generated data.

Once the generator G does a good enough job to fool the discriminator D, we expect the output probability to be close to 1.

GAN optimisation

The output

- During training, as we update the parameters of the disc tor D and the generator G, we exp ated samples created by G losely resemble real data. will have more and more troub between real and generated data.

In the tutorial, we will see how to implement a basic GAN architecture.

Once the generator G does a good enough job to fool the discriminator D, we expect the output probability to be close to 1.

Divergence minimisation

Kullback-Leibler (KL) divergence

- The general goal of GANs to optimise the same loss function “in two different direction” has a connection to the **game theory literature** (see Nash equilibria, GT strategies or fictitious play).

$$\min_G \max_D V(D, G)$$

- We can also connect the objective of generative models to minimising a divergence or distance. The most commonly used statistical distance measure is the **Kullback-Leibler (KL) divergence** (inspired by entropy):

$$\mathrm{D_{KL}}(\mathrm{P}||Q) = \int_{-\infty}^{\infty} \mathrm{p(x)log}\left(\frac{\mathrm{p(x)}}{\mathrm{q(x)}}\right) \mathrm{dx},$$

with probability densities p(x), q(x).

The KL divergence D_{KL} measures how two probability distributions, P and Q, differ from each other.

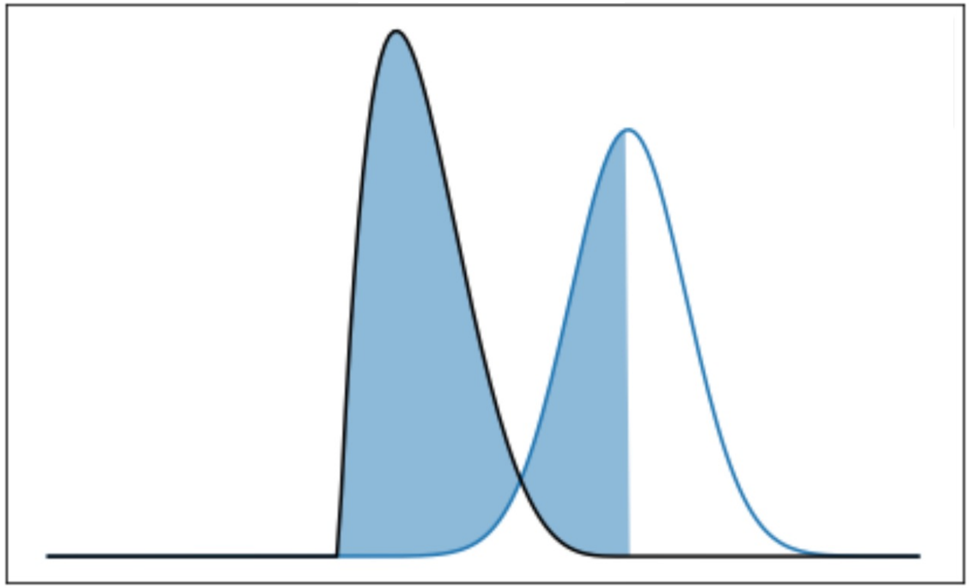
Divergence minimisation

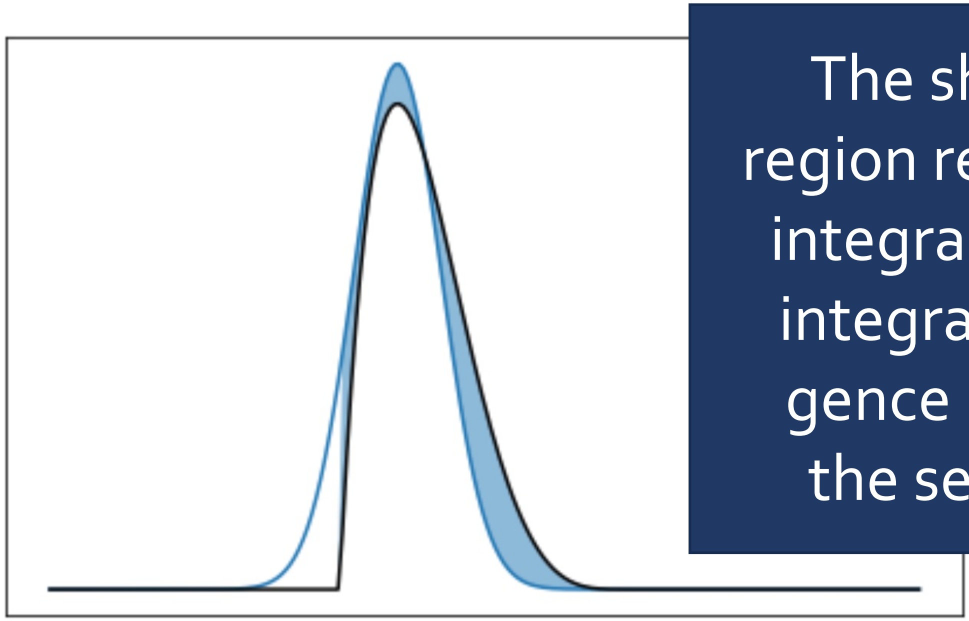
KL divergence example

$$\mathrm{D_{KL}}(\mathrm{P}||Q) = \int_{-\infty}^{\infty} \mathrm{p(x)log}\left(\frac{\mathrm{p(x)}}{\mathrm{q(x)}}\right)\mathrm{dx}$$

- To illustrate how the KL divergence operates, let’s look at a ”target” distribution P shown in black and two “test” distributions Q in blue:

The divergence is zero when p(x) = 0, and large when p(x) and q(x) overlap.

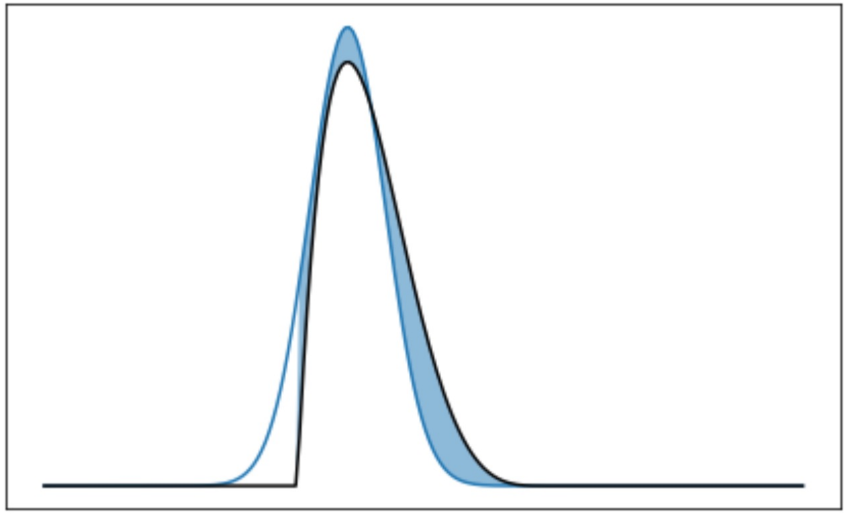




The shaded blue region represents the integrand of the KL integral. The divergence is smaller in the second case.

Divergence minimisation

Connection to maximum likelihood



- Minimising the KL divergence implies:

For $D_{KL}(P||Q) = 0$, we have $P = Q$.

- We can also show that minimising the KL divergence for two distributions P(x|θ^*) and P(x|θ) is equivalent to maximum likelihood estimation for P(x|θ) (i.e., finding the optimal parameters θ that best describe the data x):

$$\theta_{KL} = \arg\min_\theta D_{KL}[P(x|\theta^*)||P(x|\theta)] = \arg\max_\theta P(x|\theta) = \theta_{MLE}$$

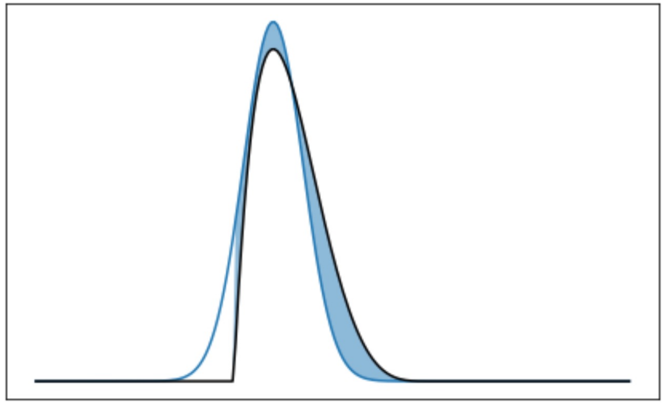
Divergence minimisation

Jensen Shannon (JS) divergence

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)\log\left(\frac{p(x)}{q(x)}\right) dx$$

- Note that the KL divergence is **not symmetric** and, hence, not a true measure of distance, because

$$D_{KL}(P||Q) \neq D_{KL}(Q||P).$$



- We can, however, define an alternative divergence that satisfies the symmetry condition and always has a finite value, the so-called **Jensen Shannon (JS) divergence**, which is defined as

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M) \quad \text{where} \quad M = \frac{1}{2}(P+Q)$$

Divergence minimisation

Connection to GANs

- Let us return to our optimisation problem for GANs:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- If the discriminator D(x) in our GAN is optimal, it is possible to show that the generator minimises the JS divergence between the real and the generated data distributions (see Goodfellow et al, 2014).

However, in practice D(x) is typically not optimal because (i) our computational resources are limited, and (ii) we only have access to samples and not the true data distribution.

Divergence minimisation

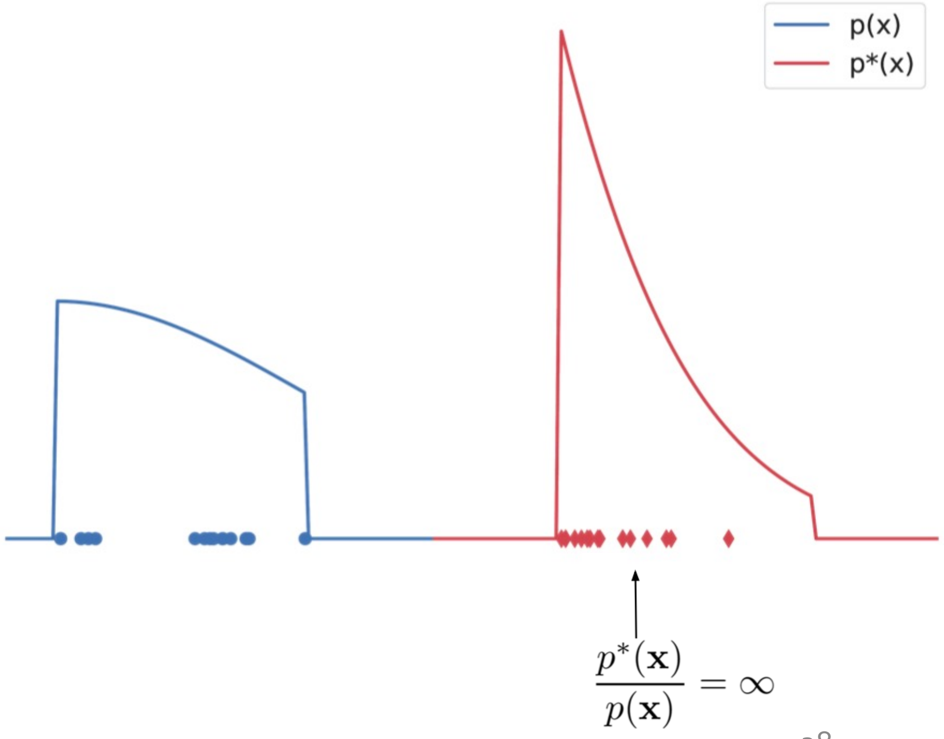
Issue with the KL and JS divergence

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

- Let us consider the following case, where we have two distributions p(x)* and p(x) that do not overlap along x, i.e., they have no overlapping support. In this case, we find for our two divergences:

$$D_{KL}(P^*||P) = \infty \quad \text{and} \quad D_{JS}(P^*||P) = \log 2$$

- If the distributions of our true and generated data do not overlap, our network does not “receive a signal” or learn.



$$\frac{p^*(\mathbf{x})}{p(\mathbf{x})} = \infty$$

Divergence minimisation

Issue with the KL and JS divergence

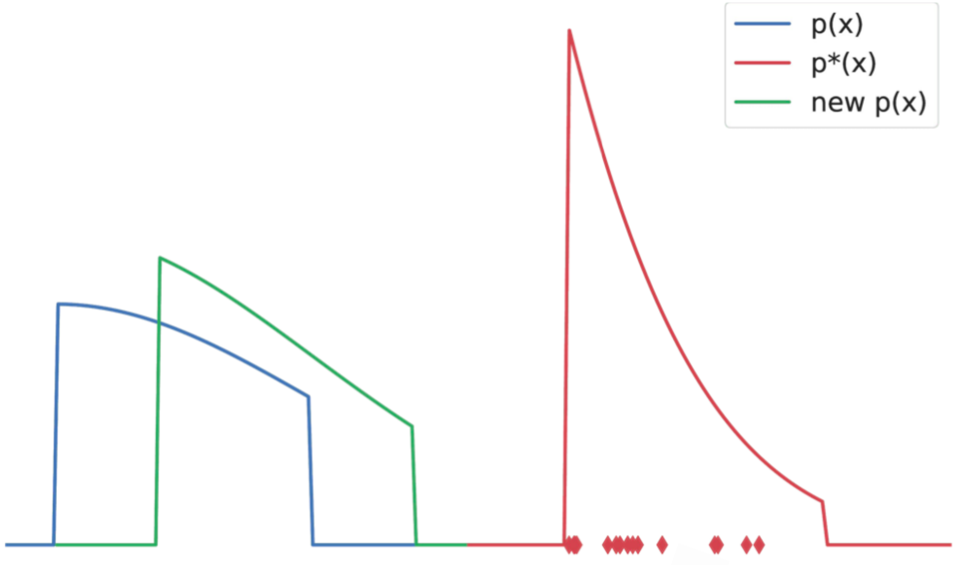
$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

- Let us consider the following case, where we have two distributions p(x)* and p(x) that do not overlap along x, i.e., they have no overlapping support. In this case, we find for our two divergences:

$$D_{KL}(P^*||P) = \infty \quad \text{and} \quad D_{JS}(P^*||P) = \log 2$$

- If the distributions of our true and generated data do not overlap, our network does not "receive a signal" or learn.

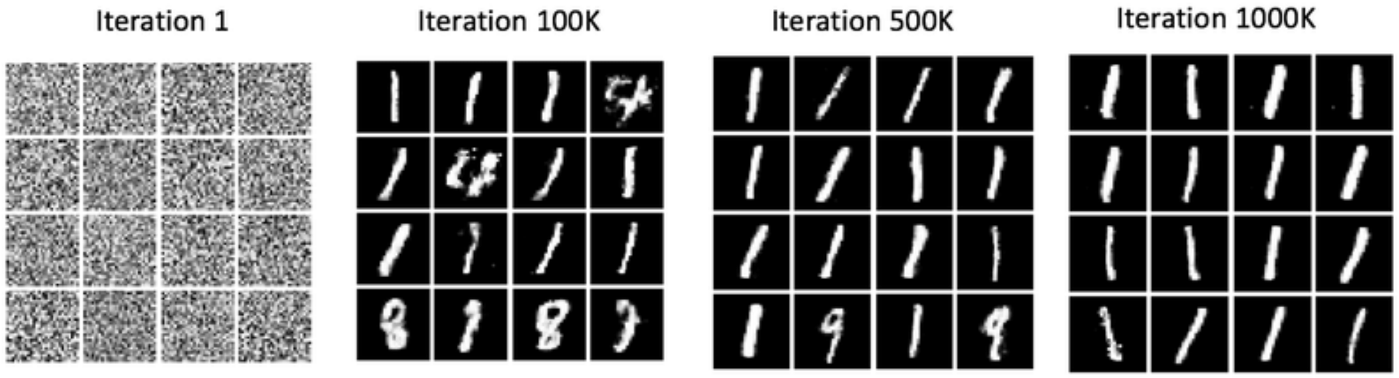
- Moving p "closer" to the true p* does not change the values of our divergences.



Mode collapse

Another issue with GANs

- Typically, we want our GAN to produce a wide variety of outputs. However, it is possible that the generator finds a particularly “good” output that is fooling the discriminator very well. This can happen when the discriminator is trapped in a local minimum (e.g., difficulty to distinguish digits 1 and 9).
- The generator then starts to produce the same output (the same mode) over and over. The discriminator’s best strategy would be to always label that output as “fake” to encourage the generator to move away from this output, but the local minimum prevents that from happening.

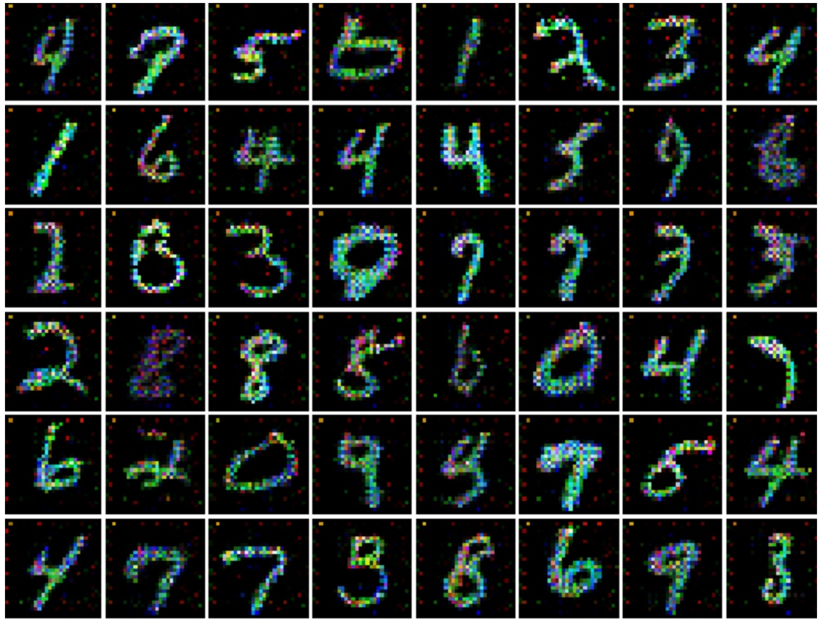


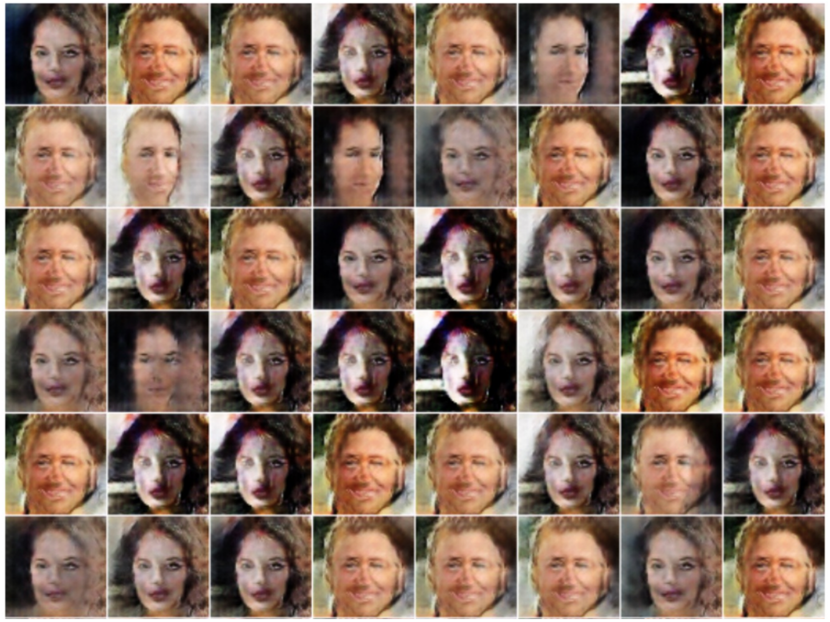
- As a result, the generator keeps producing the same output and does not learn to represent the full data distribution.

Mode collapse

Two more examples

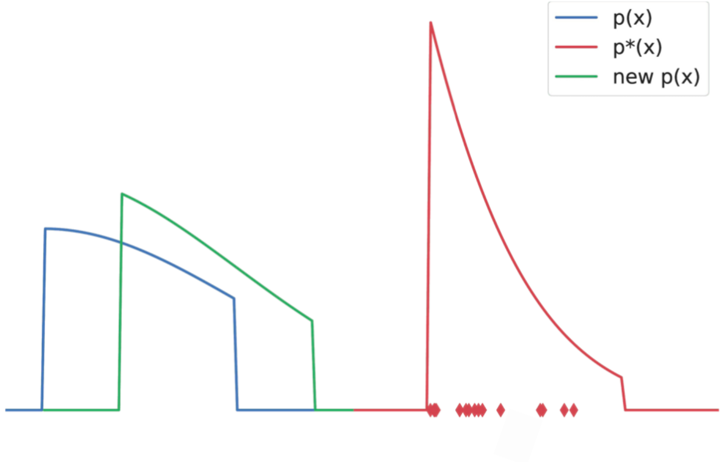
- On the left, the GAN fails to produce digits with a single colour. Instead, all digits are generated with a **mixture of the individual colours**. On the right, we see that the faces seem to be **generated from three modes** and are barely recognisable as faces. See Fedus et al. (2018) for details.





Distance measures

Other alternatives I



- To help with these issues in the original GAN formulation, we can use **different divergences.**

- A method that respects the geometry of the underlying space and captures the distance of two probability distributions whose support does not intersect is the so-called **Wasserstein distance**

$$D_{WS}(P||Q) = \sup_{||f||_L \leq 1} \left[\mathbb{E}_{x \sim p(x)}[f(x)] - \mathbb{E}_{y \sim q(y)}[f(y)] \right]$$

- In a **Wasserstein GAN**, we optimise (Arjovsky et al, 2017):

$$\min_{G} \max_{||D||_L \leq 1} \left[\mathbb{E}_{x \sim p_{data}(x)}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))] \right]$$

It has been empirically shown that the Wasserstein GAN generally avoids mode collapse.

Distance measures

Other alternatives II

- For completeness, let us highlight two more alternatives:
  - **MMD GANs** based on the **Maximum Mean Discrepancy** (MMD), which encodes the difference between feature means:

$$D_{\mathrm{MMD}}(P||Q) = \sup_{||f||_{\mathcal{H}} \leq 1} \left[\mathbb{E}_{x \sim p(x)}[f(x)] - \mathbb{E}_{y \sim q(y)}[f(y)] \right]$$

  - **f-GANs** based on **f-divergences** (generalisations of the KL divergence), which again encode the differences between two probability distributions:

$$D_f(P||Q) = \int_{-\infty}^{\infty} q(x) f\left(\frac{p(x)}{q(x)}\right) dx$$

Key idea: we can create GAN training criteria based on various divergences and distances.

GANs

Divergence minimisers? I

- Remember that for **implicit models**, we do not have access to the probability distribution of our real data, $p_{data}(x)$. We only have samples. So, we **cannot explicitly calculate** our earlier divergence measures.

$$D_f(P||Q) = \int_{-\infty}^{\infty} q(x) f\left(\frac{p(x)}{q(x)}\right) dx$$

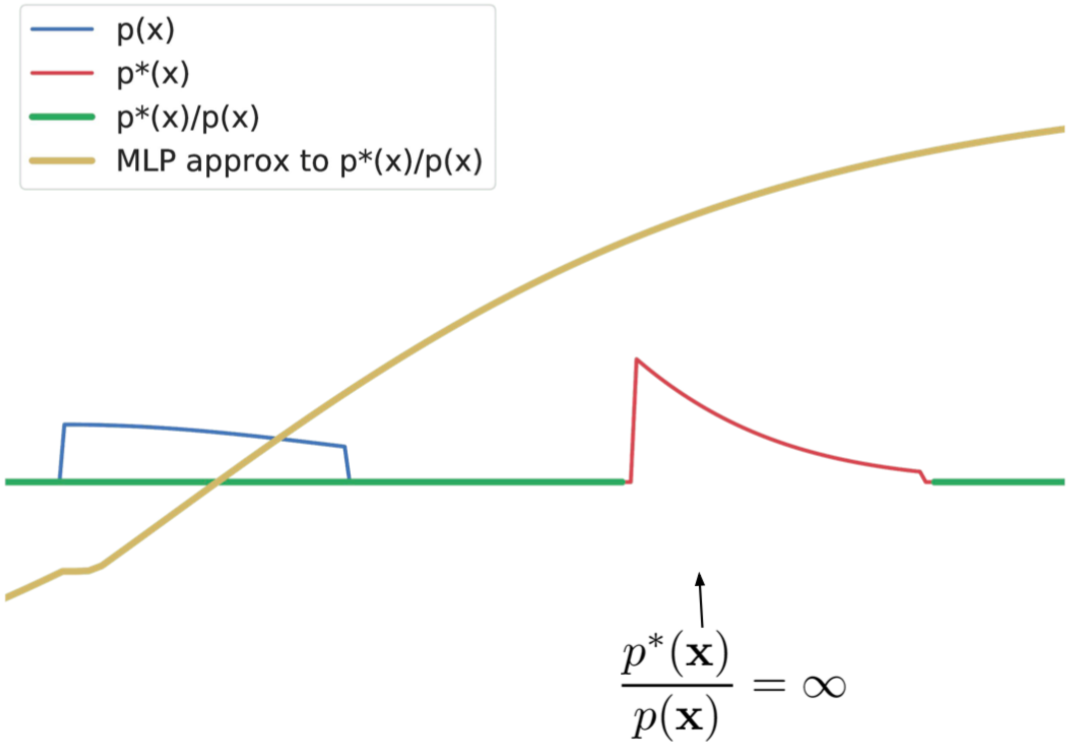
- Instead, we can use a trick and replace the intractable divergences by **lower bounds** (so-called variational lower bounds; see Nowozin et al. *f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization* (2016) for details) that only **depend on our samples**, not the underlying distributions.

GANs

Divergence minimisers? II

$$\min_{G} \boxed{\max_{D} V(D, G)}$$

- The key role of the discriminator is then to learn how to **distinguish between samples** from two distributions (not the distributions themselves).



$$\frac{p^*(\mathbf{x})}{p(\mathbf{x})} = \infty$$

- GANs do not do divergence minimisation in practice, but D learns a **“distance”** (i.e., boundary) between the data and model distributions. These distances then provide useful gradients to update the model.

- Another advantage is that GANs learn **smooth distances** and do not fail even in those cases, where the underlying divergences would.

GANs vs. divergence minimisation

Key differences

GANs

- PROs
  - Easy access to generated samples
  - Smooth learned loss function
  - Empirically the underlying loss matters less than the NN architecture, the training regime and the data used
- CONs
  - Dynamics are difficult to analyse
  - Optimal convergence cannot be guaranteed

DIVERGENCE MINIMISATION

- PROs
  - Optimal convergence is guaranteed
  - Analysis of loss properties is straight forward
- CONs
  - Access to good samples is very difficult

Evaluating GANs

The main issue

- Now that we know how GANs work in practice, we can ask ourselves:

When can we say that a GAN is performing well?

- We could focus, e.g., on the following metrics:
  - How good is the quality of the produced samples?
  - How good is our GAN at generalising to different tasks?
  - Is our network learning meaningful patterns (representation learning)?

No single evaluation metric captures all desired properties. In practice, we have to evaluate the GAN performance based on our end goal (e.g., classification accuracy in semi-supervised settings, human evaluation).

Evaluating GANs

Metrics: Inception Score (IS)

- A key problem when evaluating GANs is that for these implicit models, we do not have access to the likelihoods (very expensive to approximate).

- To evaluate the GAN output, we can run the generated images through the **Inception network** that was pretrained on the **ImageNet** dataset. We then **compare the distribution of labels** obtained from real data with the distribution of labels from our samples using the KL divergence.

+ measures the sample quality and diversity
+ correlates with human evaluation
- does not measure differences beyond labels
- depends on quality of pretrained classifier

Evaluating GANs

Metrics: Inception Score (IS)

- A key problem when evaluating GANs is that for these implicit models, we do not have ac to the likelihoods (very expensive to approximate).

- To evaluate t n the generated images through the **Inceptio** **ImageNet** dataset. We then **compa** from real data w our samples using the K

If our GAN performs well, the distribution of generated labels should be different. This corresponds to a larger KL divergence: A large Inception Score reflects good GAN performance!

+ measures the sample quality and diversity
+ correlates with human evaluation
- does not measure differences beyond labels
- depends on quality of pretrained classifier

Evaluating GANs

Metrics: Fréchet Inception Distance (FID)

- This approach also uses the pretrained Inception network. Instead of focusing on labels, we however run our samples and real data through the classifier to **compare distributions of features in the final pooling layer**. Both are compared using the so-called **Fréchet distance**.

+ measures the sample quality and diversity
+ correlates with human evaluation
+ measures feature level statistics (not just classes)
- depends on quality of pretrained classifier
- biased for a small number of samples

A small Fréchet Inception Distance reflects good GAN performance!

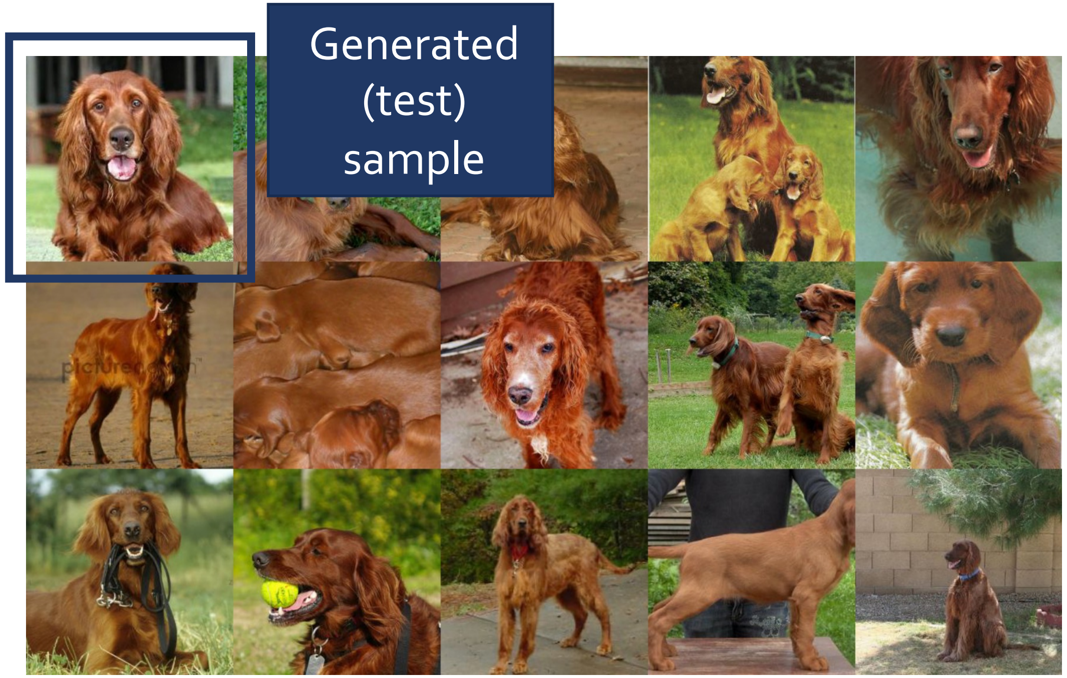
Evaluating GANs

Metrics: nearest neighbour classifier

- We can use a nearest neighbour classifier to compare a generated sample to every single training image and predict the closest training images (e.g., by comparing absolute pixel value difference).

- This gives a **qualitative measure** if **overfitting** takes place or not.

$$\left|\begin{array}{c}\text{test image}\\ \begin{array}{|c|c|c|c|}\hline 56 & 32 & 10 & 18\\ \hline 90 & 23 & 128 & 133\\ \hline 24 & 26 & 178 & 200\\ \hline 2 & 0 & 255 & 220\\ \hline\end{array}\end{array} - \begin{array}{c}\text{training image}\\ \begin{array}{|c|c|c|c|}\hline 10 & 20 & 24 & 17\\ \hline 8 & 10 & 89 & 100\\ \hline 12 & 16 & 178 & 170\\ \hline 4 & 32 & 233 & 112\\ \hline\end{array}\end{array}\right| = \begin{array}{c}\text{pixel-wise absolute value differences}\\ \begin{array}{|c|c|c|c|}\hline 46 & 12 & 14 & 1\\ \hline 82 & 13 & 39 & 33\\ \hline 12 & 10 & 0 & 30\\ \hline 2 & 32 & 22 & 108\\ \hline\end{array}\end{array} \rightarrow 456$$

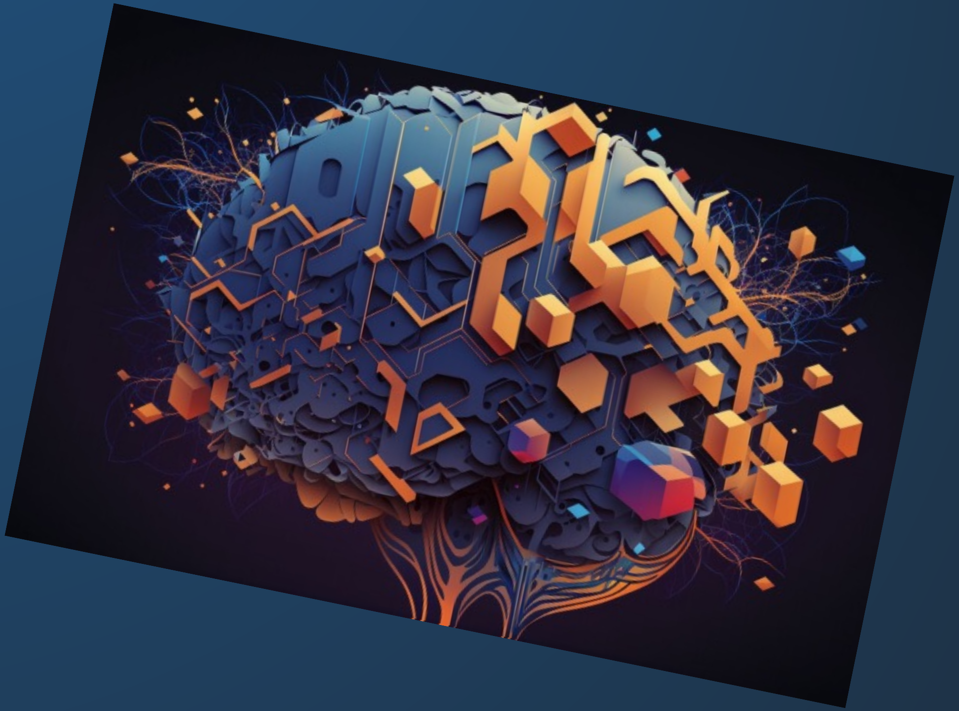


Introduction and key concepts

How do GANs work?

Types of GANs and their applications

Summary



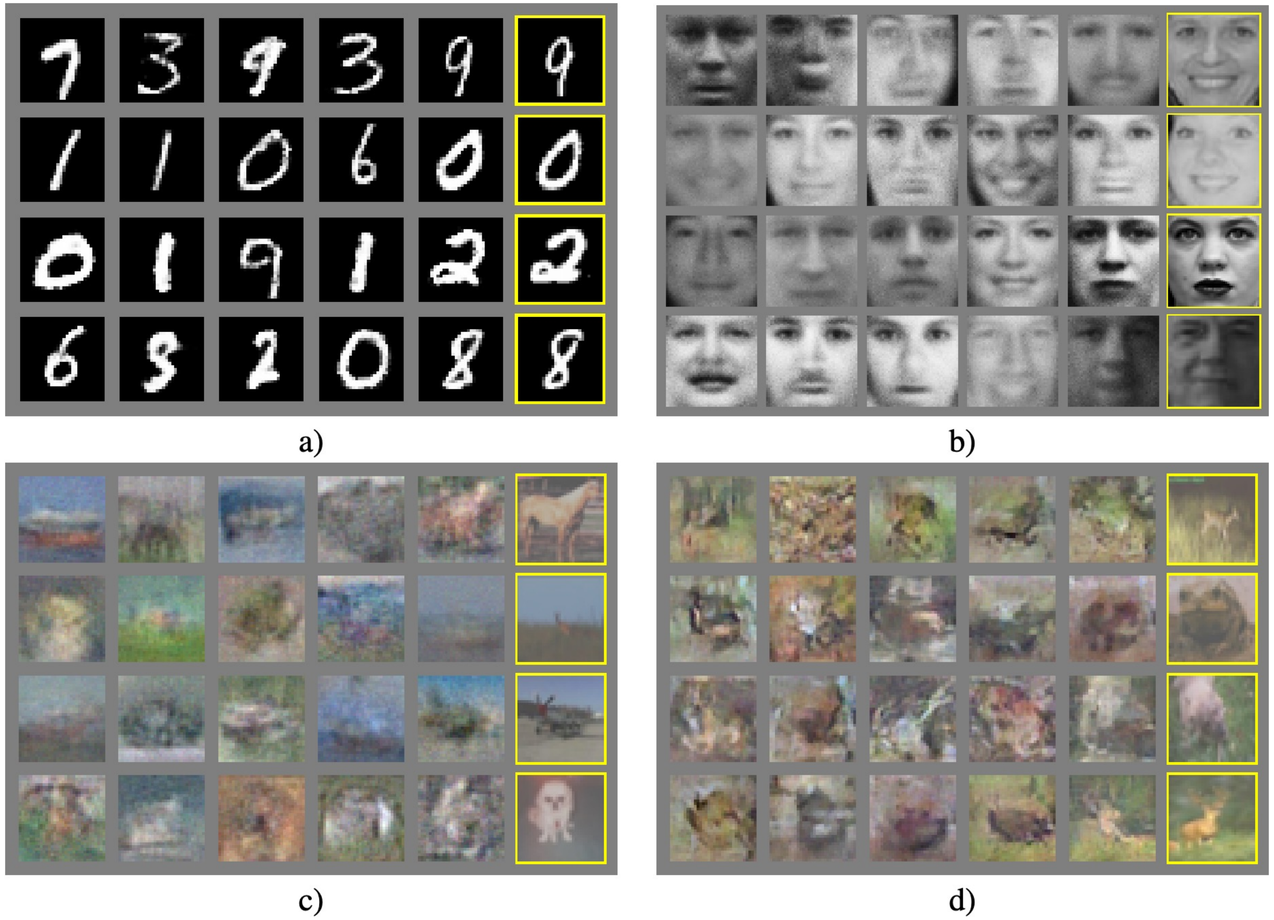
Original architecture

Goodfellow et al. (2014)

- The paper focused on **simple data structures** (images with 32x32 resolution) and simple models. For (a) (MNIST dataset), (b) (Toronto Face Database; TFD) and (c) (CIFAR-10 database) they applied a **simple MLP for the discriminator and generator**.

- For (d) (CIFAR-10), they used a **convolutional discriminator and deconvolutional generator**.

- For the training process, images were flattened to vectors; i.e., the spatial structure was ignored.



a)

b)

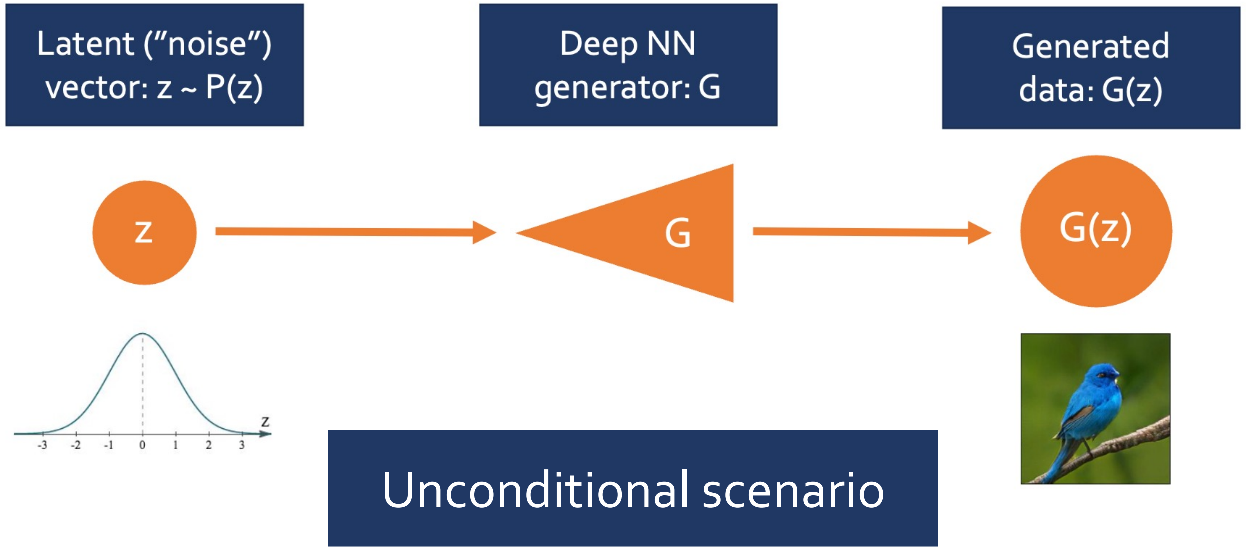
c)

d)

Unconditional vs. conditional GANs

The difference

- When introducing our generator, we focused on the scenario that the generator **produces data samples from random noise**. This allows us to capture the **full spread of the data distribution**. However, in this case, we do not have control over the kind of sample we produce.



Latent (”noise”) vector: z ~ P(z)

Deep NN generator: G

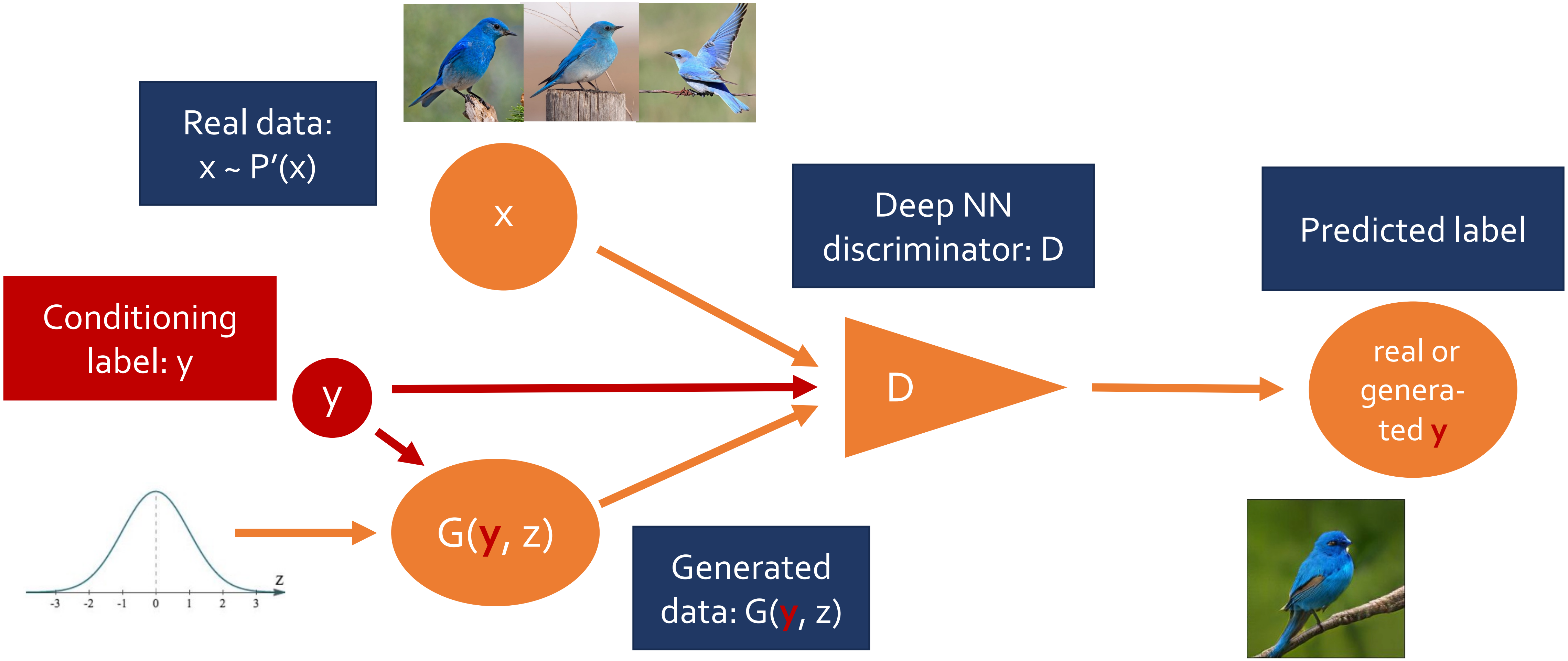
Generated data: G(z)

Unconditional scenario

- In **conditional GANs**, we can specify the kind of sample we want to output (i.e., bird vs. cat) by providing additional information (labels) to our generator.

Class-conditional GAN

Modification to our original flow chart



Real data:
x ~ P′(x)

Conditioning
label: y

Deep NN
discriminator: D

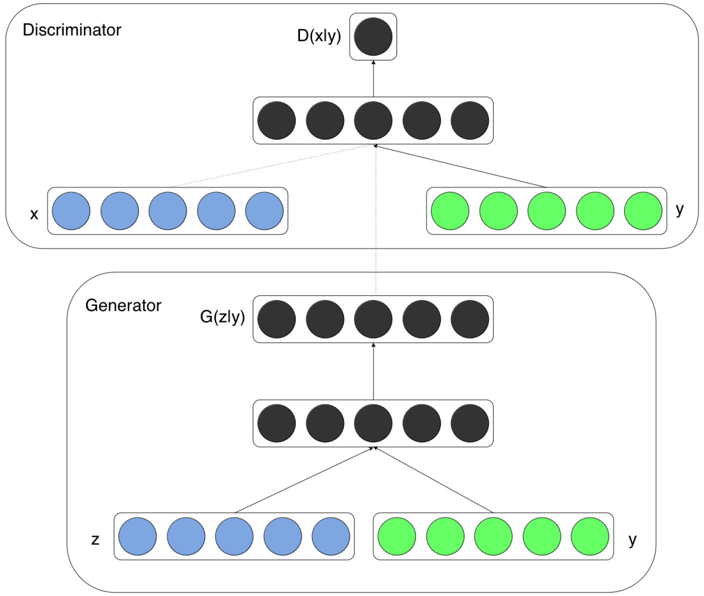
Predicted label

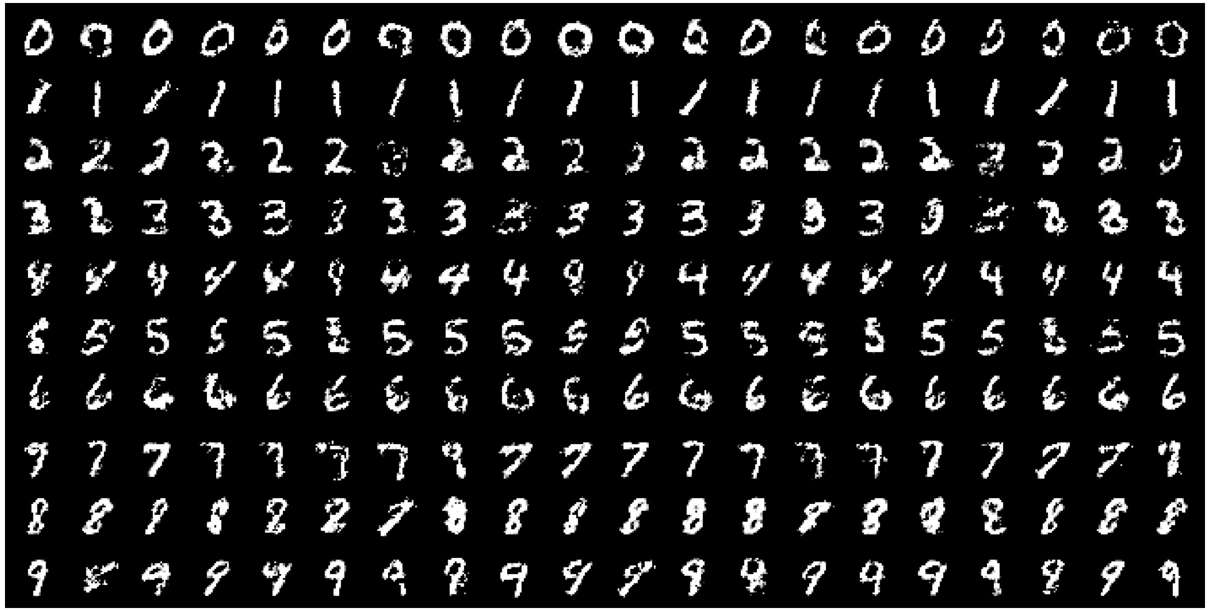
Generated
data: G(y, z)

Conditional GANs

Mirza & Osindero (2014)

- Mirza & Osindero generalised the original Goodfellow et al. algorithm to include additional conditional information, which was provided to the generator and the discriminator during training.





Laplacian GANs (LAPGANs) I

Denton et al. (2015)

- The authors introduced a generative parametric model that produces **high-quality samples** of images. To generate images from coarse to fine, they use a **cascade of CNNs** within a pyramid structure.

Laplacian GANs (LAPGANs) I

Denton et al. (2015)

- The authors introduced a generative parametric model that produces **high-quality samples** of images. To generate images from coarse to fine, they use a **cascade of CNNs** within a pyramid structure.

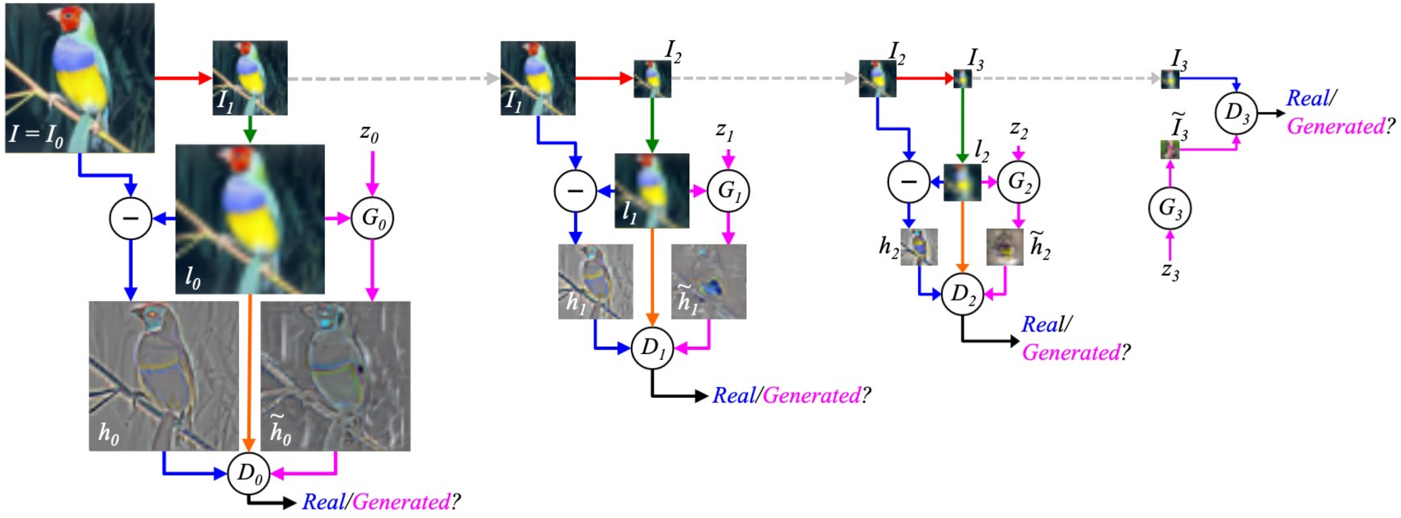


Figure 2: The training procedure for our LAPGAN model. Starting with a 64x64 input image I from our training set (top left): (i) we take $I_0 = I$ and blur and downsample it by a factor of two (red arrow) to produce I_1; (ii) we upsample I_1 by a factor of two (green arrow), giving a low-pass version l_0 of I_0; (iii) with equal probability we use l_0 to create *either* a real *or* a generated example for the discriminative model D_0. In the real case (blue arrows), we compute high-pass $h_0 = I_0 - l_0$ which is input to D_0 that computes the probability of it being real vs generated. In the generated case (magenta arrows), the generative network G_0 receives as input a random noise vector z_0 and l_0. It outputs a generated high-pass image $\tilde{h}_0 = G_0(z_0, l_0)$, which is input to D_0. In both the real/generated cases, D_0 also receives l_0 (orange arrow). Optimizing Eqn. 2, G_0 thus learns to generate realistic high-frequency structure $\tilde{h}_0$ consistent with the low-pass image l_0. The same procedure is repeated at scales 1 and 2, using I_1 and I_2. Note that the models at each level are trained independently. At level 3, I_3 is an 8×8 image, simple enough to be modeled directly with a standard GANs G_3 & D_3.

$$\min_G \max_D \mathbb{E}_{h,l\sim p_{\text{Data}}(\mathbf{h},\mathbf{l})}[\log D(h,l)] + \mathbb{E}_{z\sim p_{\text{Noise}}(\mathbf{z}),l\sim p_l(\mathbf{l})}[\log(1 - D(G(z,l),l))] \qquad (2)$$

Laplacian GANs (LAPGANs) II

Denton et al. (2015)

- At each pyramid level, generators are trained independently to model the corresponding high-frequency structure. Once trained, the generative models are used to reconstruct a final generated output:

Laplacian GANs (LAPGANs) II

Denton et al. (2015)

- At each pyramid level, generators are trained independently to model the corresponding high-frequency structure. Once trained, the generative models are used to reconstruct a final generated output:

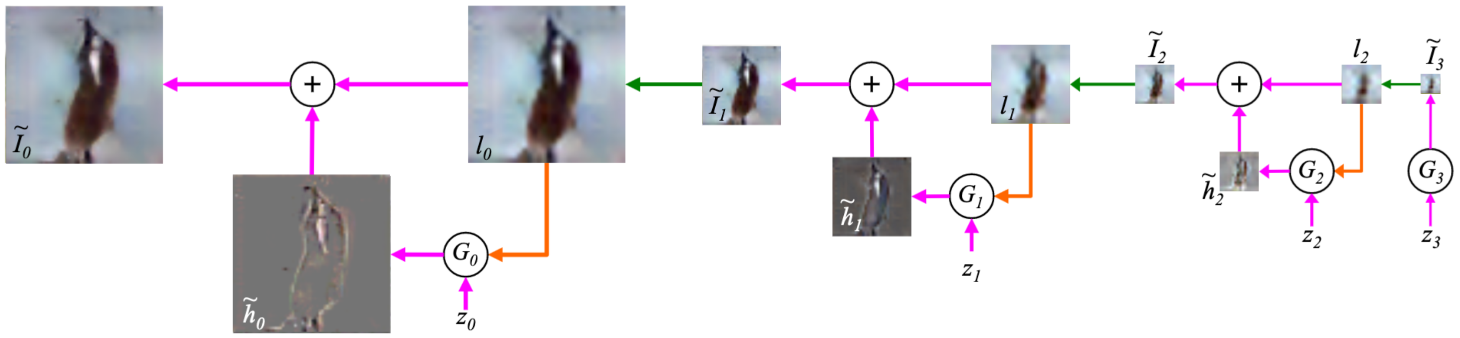


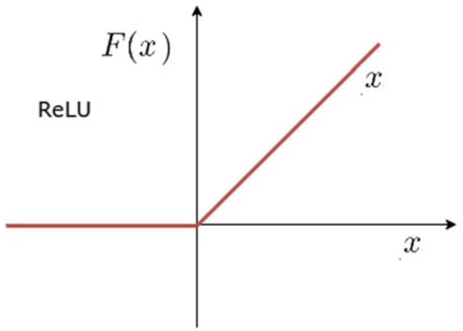
Figure 1: The sampling procedure for our LAPGAN model. We start with a noise sample z_3 (right side) and use a generative model G_3 to generate $\tilde{I}_3$. This is upsampled (green arrow) and then used as the conditioning variable (orange arrow) l_2 for the generative model at the next level, G_2. Together with another noise sample z_2, G_2 generates a difference image $\tilde{h}_2$ which is added to l_2 to create $\tilde{I}_2$. This process repeats across two subsequent levels to yield a final full resolution sample I_0.

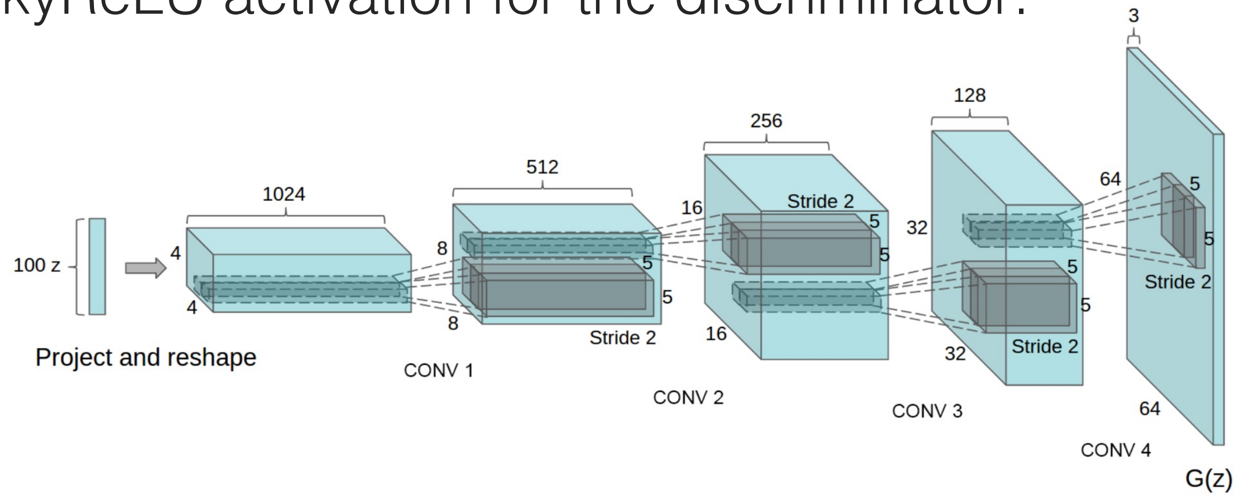
This method performs well at higher resolutions. Human evaluation showed that roughly 40% of generated images with the LAPGAN were identified as real.

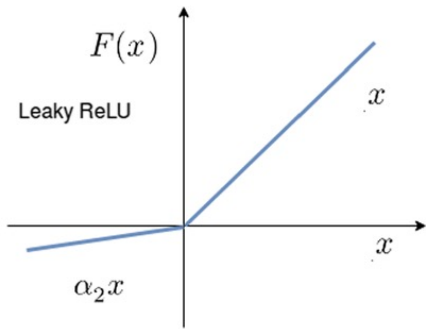
Deep Convolutional GANs (DCGANs) I

Radford et al. (2016)

- The authors introduce a GAN approach that uses deep CNNs for the generator and discriminator. They imposed certain architecture guidelines to stabilise the otherwise **very difficult training process**:
  - Batch normalisation in both G and D.
  - Removal of fully connected hidden layers.
  - Primarily ReLU activation for the generator.
  - LeakyReLU activation for the discriminator.



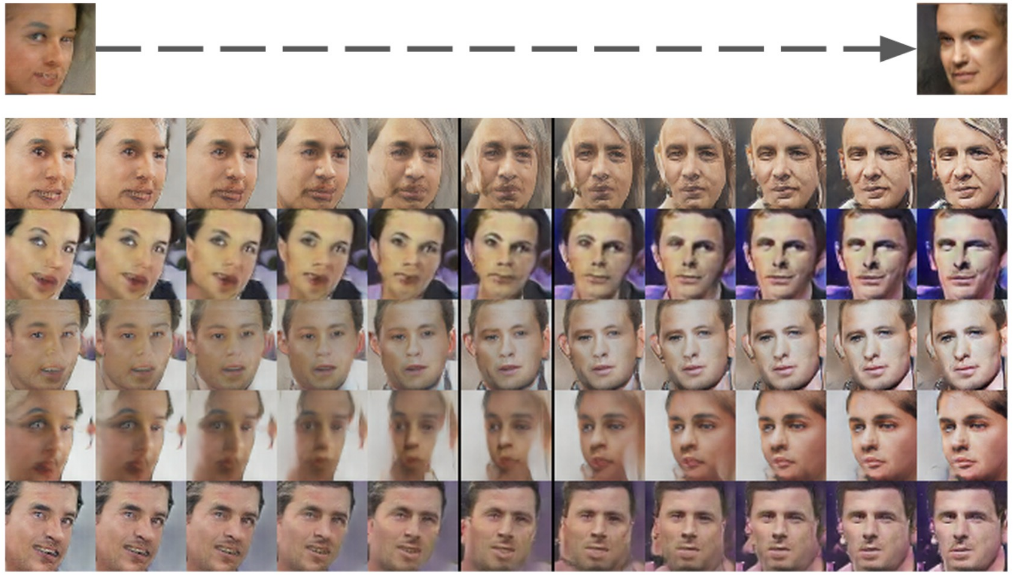


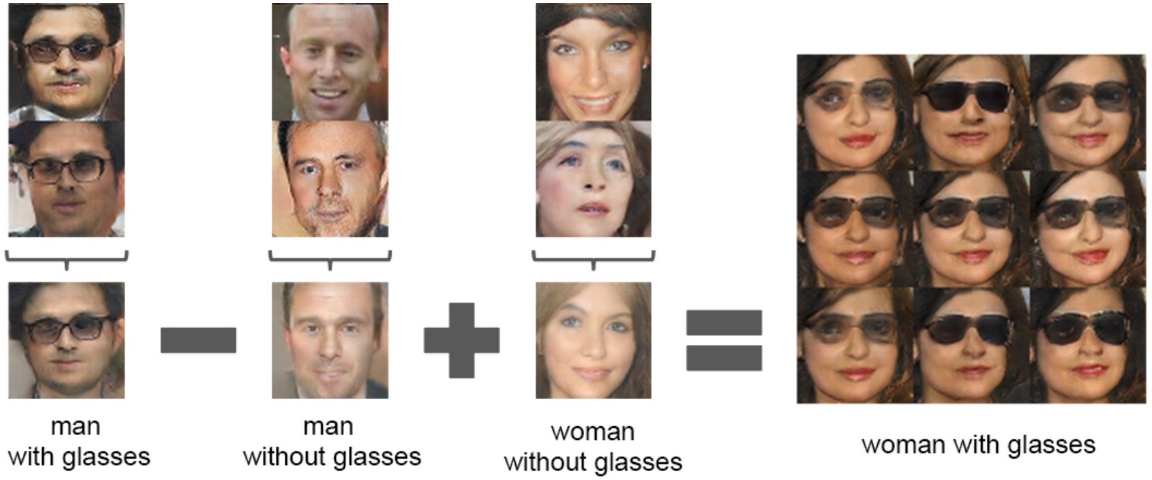


Deep Convolutional GANs (DCGANs) II

Radford et al. (2016)

- This approach also enabled the definition of a ”**turning vector**” for faces looking from left to right. By interpolating along this axis, the GAN samples was able to reliably transform their pose.





- The DCGAN generator’s noise/latent space also seems to have meaningful understanding of semantics.

Spectral normalisation (SN) GANs

Miyato et al. (2018)

- One of the key difficulties when using GANs is to stabilise the training of the discriminator. To overcome this issue, Miyato et al. introduced a **new normalisation** for the network weights based on the **weights' spectral properties** to restrict the choices of functions that can be used for the discriminator. This led to more diverse samples.

Tiger

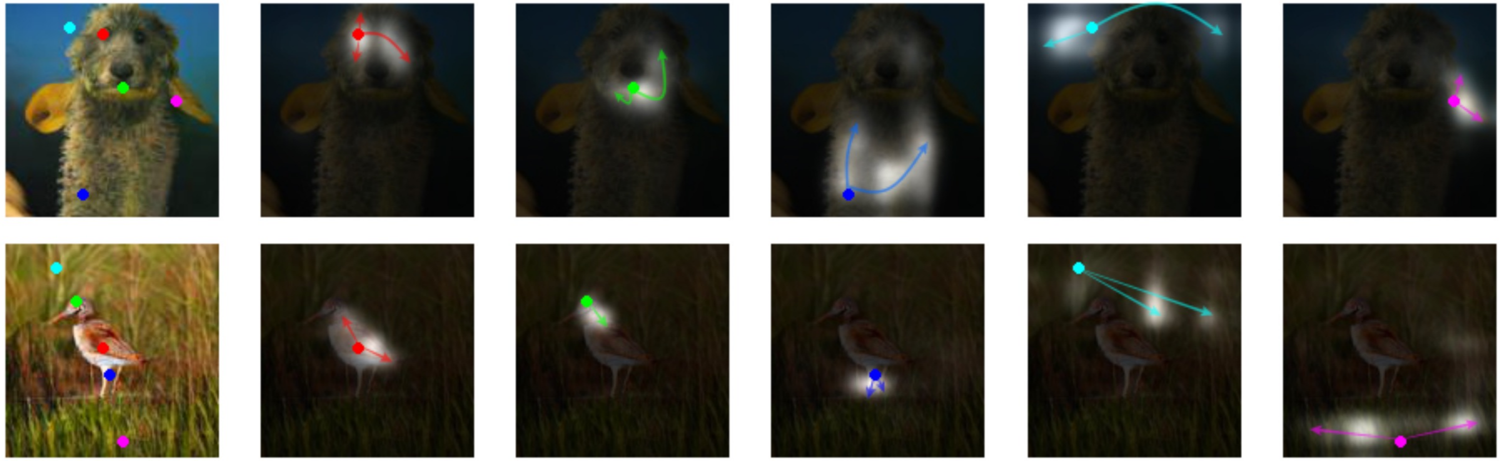
Pizza

Daisy

Self-Attention (SAGANs)

Zhang et al. (2019)

- Traditional convolutional GANs generated high-resolution details in images as a function of spatially local points only. To improve on this, SAGANs allow feature generation based on **global feature locations**. This is implemented via **self-attention** (updates are performed based on weighted sums of the features across all positions).
- Moreover, the discriminator checks that detailed features in distant portions of the image are consistent with each other.



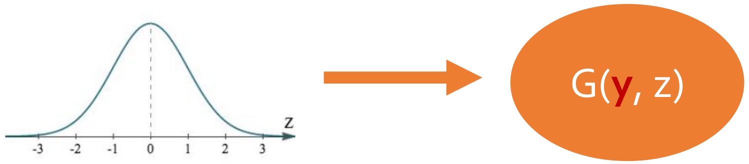
BigGAN I

Brock et al. (2019)

- Despite the advances discussed so far, by the beginning of 2019, it was still difficult to generate high-resolution samples with large diversity for complex datasets such as ImageNet. To solve this, Brock et al. **trained class-conditional GANs** for the first time **at large scale**. They increased
  - Batch size (2048)
  - Model parameters (>100 million)
  - Training dataset size (1.2 million images)
  - Resolution in images (128, 256, 512)

BigGAN II

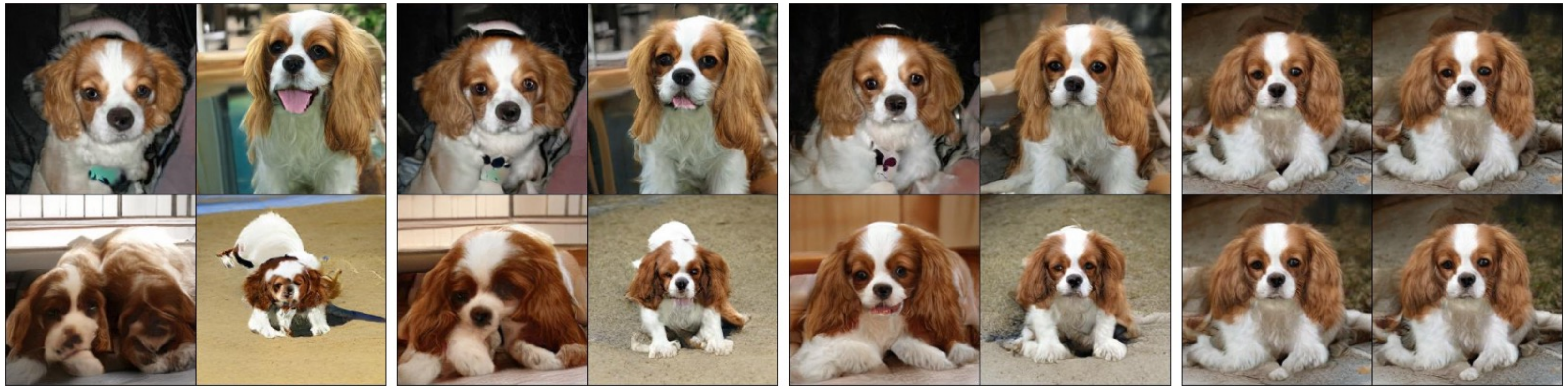
Brock et al. (2019)



- One of the key things the papers finds, is that because of the large sample size, they can use a so-called **"truncation trick"**, which refers to varying the scale of the noise, z, that is fed to the generator.

- For smaller (truncated) noise, they increase the fidelity of the images and create prototypical examples for each class. Larger noise on the other hand produces more variety and generates the full class distribution.

Less truncation (larger noise)

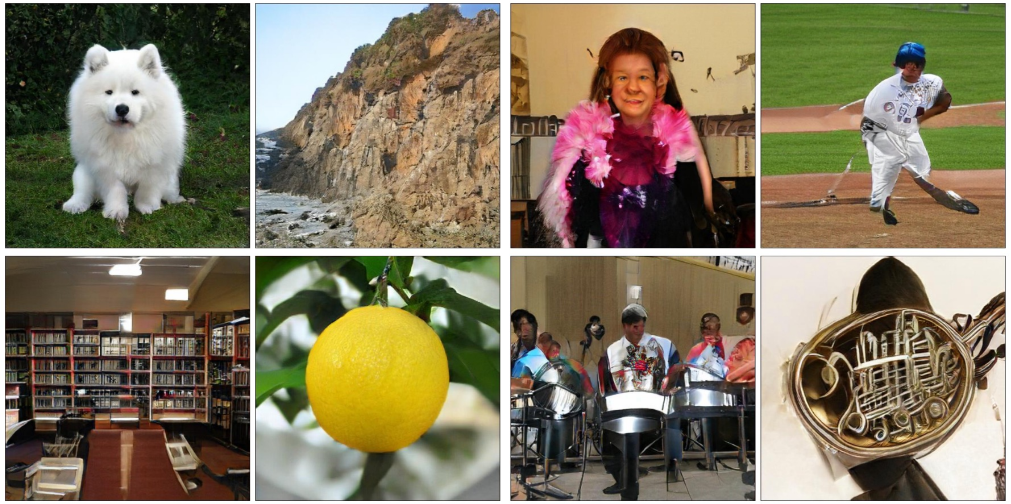


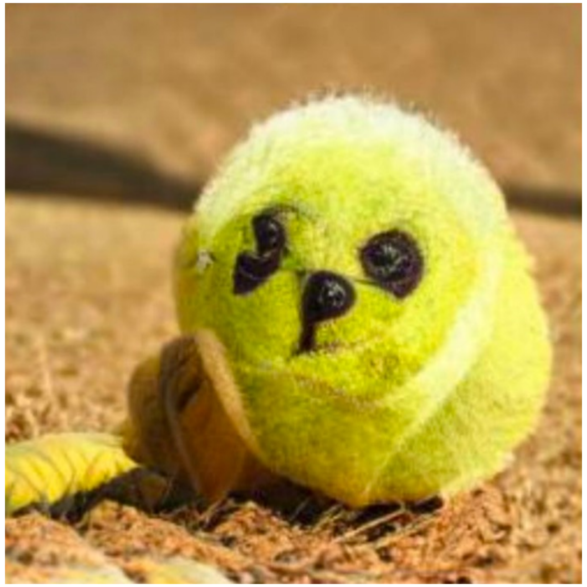
More truncation (smaller noise)

BigGAN III

Brock et al. (2019)

- In addition, the paper presents a **large empirical study** to build a reliable (aka stable) prescription for large-scale GAN training focusing on, e.g., different losses, spectral normalisation, self-attention, and other tricks.
- However, there are new cases where **BigGAN fails**: (i) some classes are easier to generate (e.g., dogs are common, and cliffs have a single texture) than others, which are more dynamic or structured. (ii) images from one class can contain properties of another (class leakage).



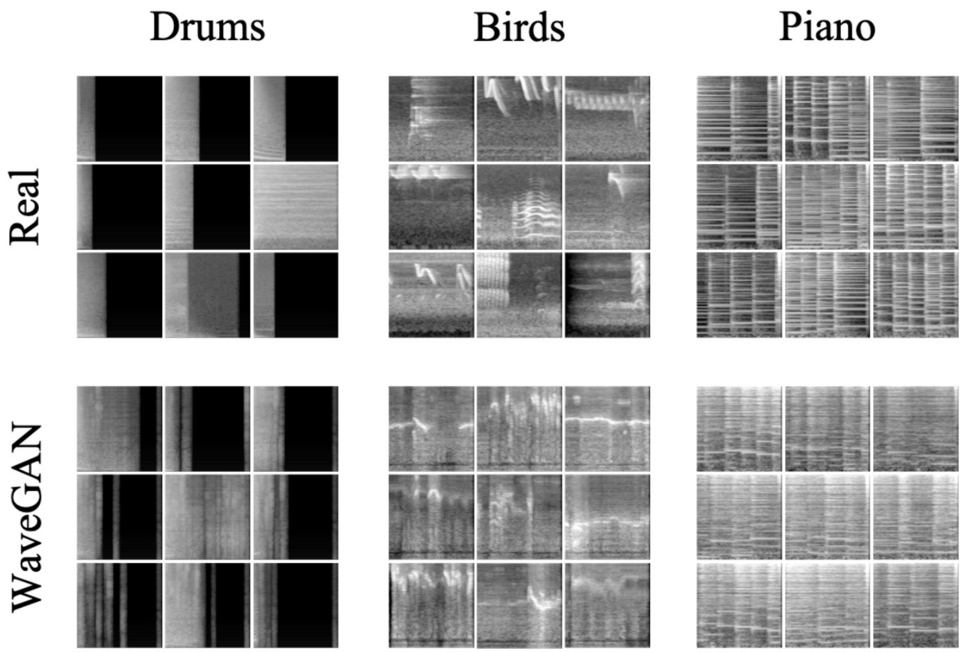


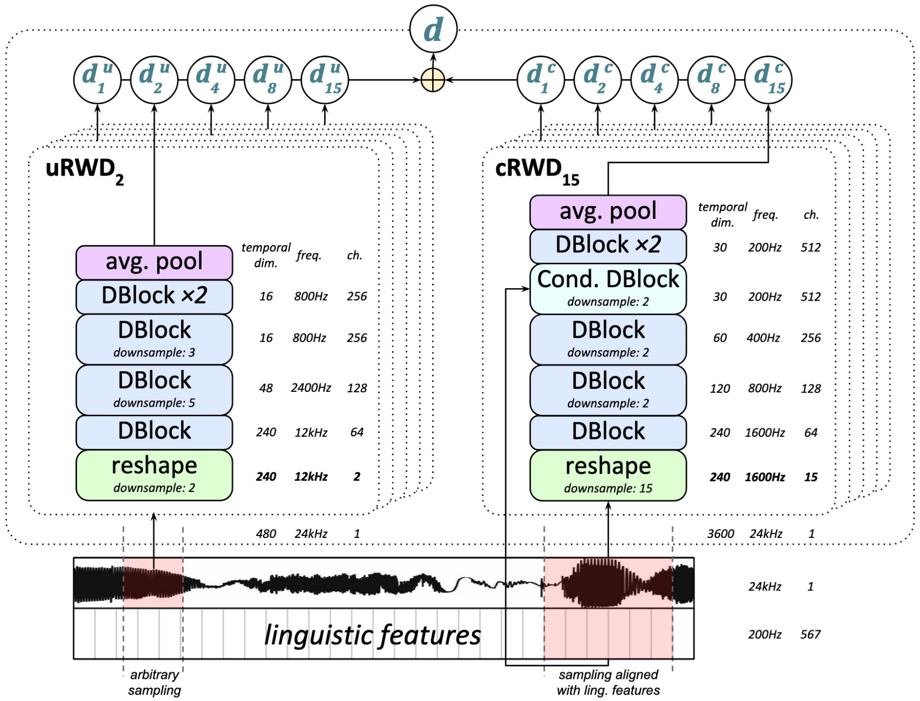
GANs for audio synthesis

A few examples

WaveGAN reproduces spectral characteristics from various audio signals with high-temporal resolution (Donahue et al. *Adversarial Audio Synthesis.* 2019)

GAN-TTS for Text-to-Speech combines a feedforward generator with an ensemble of Random Window Discriminators (Binkowksi et al. *High fidelity speech synthesis with adversarial networks.* 2019)





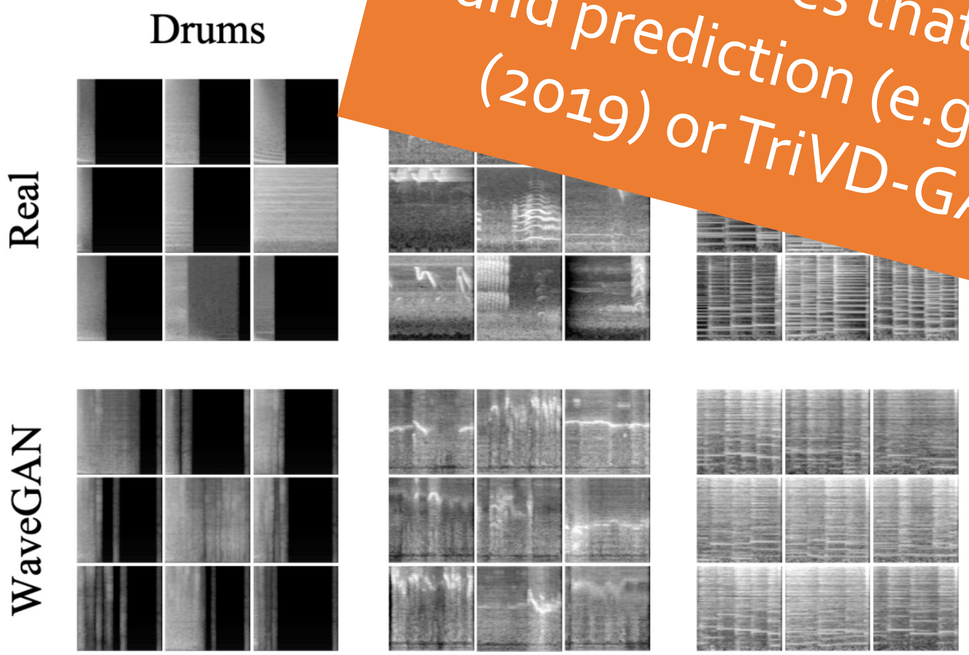
GANs for audio synthesis

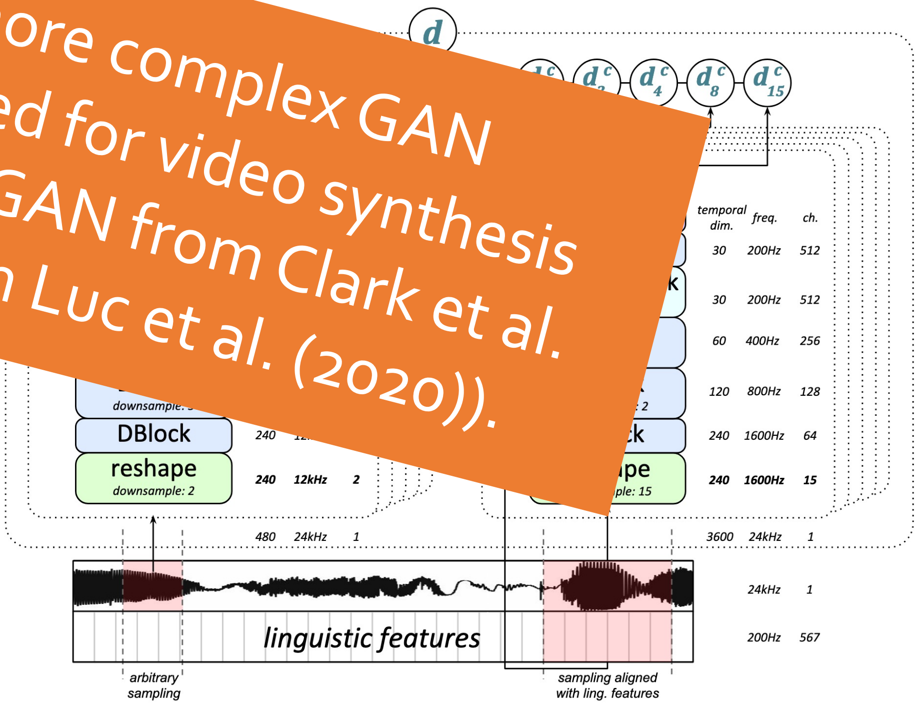
A few examples

WaveGAN reproduces spectral characteristics fr ious audio signals with high-tempor hue et al. *Adversaria*

GAN-TTS for Text-to-Speech combines a feedforward generator with an ensemble of Random Window Discriminators (Binkowksi et al. *High fidelity speech synthesis with adversarial networks.* 2019)

We now also have more complex GAN architectures that are used for video synthesis and prediction (e.g. DVD-GAN from Clark et al. (2019) or TriVD-GAN from Luc et al. (2020)).

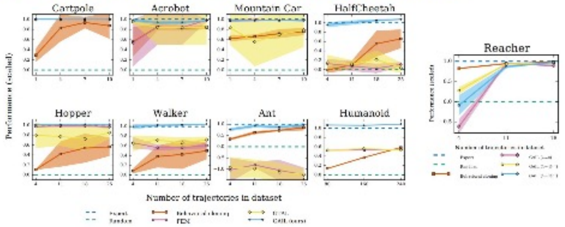




GANs are everywhere

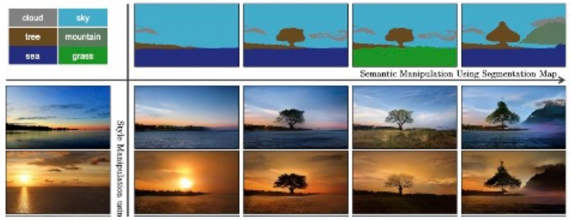
A few examples

RL (Imitation Learning): **GAIL**



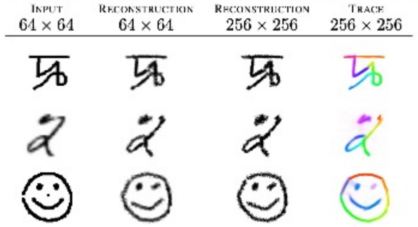
Ho and Erman. **Generative Adversarial Imitation Learning.** Neural Information Processing Systems **(2016)**

Image Editing: **GauGAN**



Park et al. **Semantic Image Synthesis with Spatially-Adaptive Normalization.** IEEE Conference on Computer Vision and Pattern Recognition **(2019)**

Program Synthesis: **SPIRAL**



Ganin et al. **Synthesizing Programs for Images using Reinforced Adversarial Learning.** International Conference on Machine Learning **(2018)**

Motion Transfer: **Everybody Dance Now**

Chan et al. **Everybody Dance Now.** International Conference on Computer Vision **(2019)**

Domain Adaptation: **DANN**

Ganin et al. **Domain-Adversarial Training of Neural Networks.** Journal of Machine Learning Research **(2016)**

Art: **Learning to See**

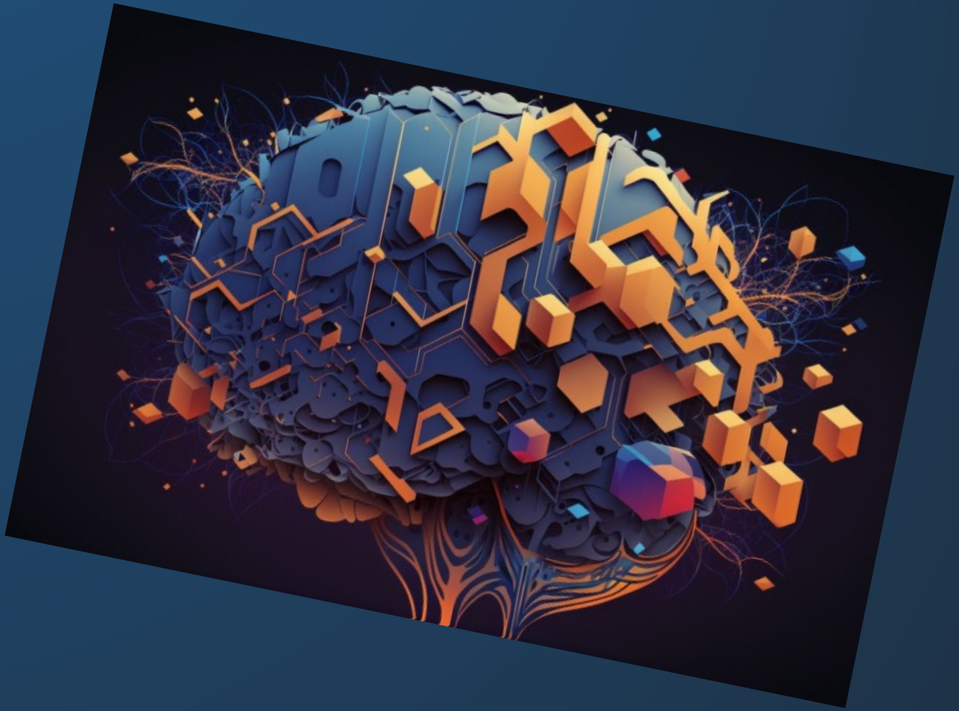
Akten. **Learning To See.** http://www.memo.tv/portfolio/learning-to-see/ **(2017**, accessed 2020)

Introduction and key concepts

How do GANs work?

Types of GANs and their applications

Summary



Summary I

- GANs are a type of **generative model** that create new samples by framing the problem as a supervised learning problem with two competing components (two-player game).

- The generator creates new samples that the discriminator (teacher) tries to identify as “fake”. By optimising both at the same time, the generated output gets better and better. The **discriminator** aims to **maximise** the prediction accuracy, while the **generator minimises it**.

- As a result, GANs learn to **approximate the probability distribution** of the real data. However, GANs are an **implicit framework**, where we do not have direct access to the underlying likelihoods.

Summary II

- **Different GAN training criteria** have been developed. These are inspired by certain **divergence and distance measures** to compare the similarity between (samples from) two probability distributions. However, GANs do not perform divergence minimisation in practice.

- **Evaluating the performance** of GANs is difficult as we do not have access to the likelihoods. Instead, different evaluation metrics are used (e.g., Inception Scores, Fréchet Inception Distance).

- Training GANs is challenging but the **field is rapidly evolving**, and a **lot of progress** has been made since 2014. GANs are now used for many tasks like image-to-image translation and generating realistic pictures of objects, people, etc. However, the difficulty in identifying outputs as fake also **poses serious risks** that need to be addressed.