Machine Learning: Lecture #5

Jennifer Ngadiuba (Fermilab)

University of Pavia, May 8-12 2023

Overview of the lectures

- **Day 1:**
  - Introduction to Machine Learning fundamentals
  - Linear Models
- **Day 2:**
  - Neural Networks
  - Deep Neural Networks
  - Convolutional Neural Networks
- **Day 3:**
  - Recurrent Neural Networks
  - Graph Neural Networks (part 1)
- **Day 4:**
  - Graph Neural Networks (part 2)
  - Transformers
- **Day 5:**
  - Unsupervised learning
  - Autoencoders
  - Generative Models
    - Variational Autoencoders
    - Generative Adversarial Networks
  - Anomaly detection

Hands on sessions each day will closely follow the lectures topics

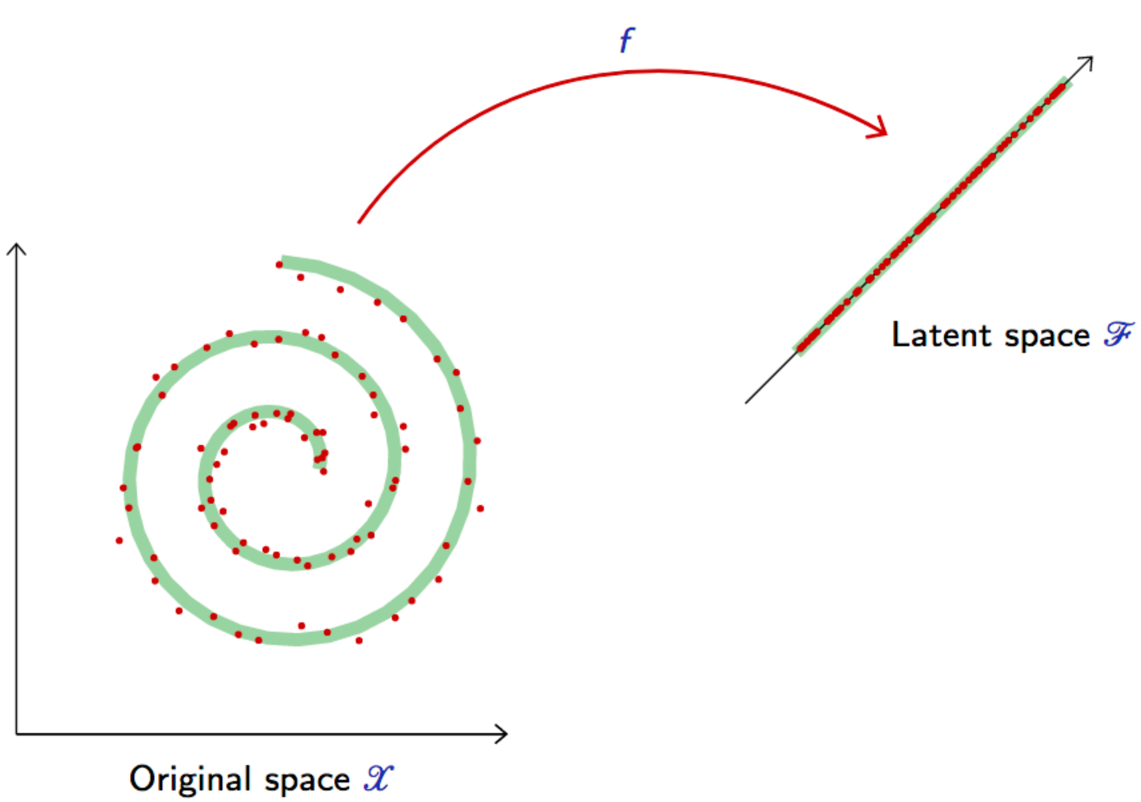
Unsupervised Learning

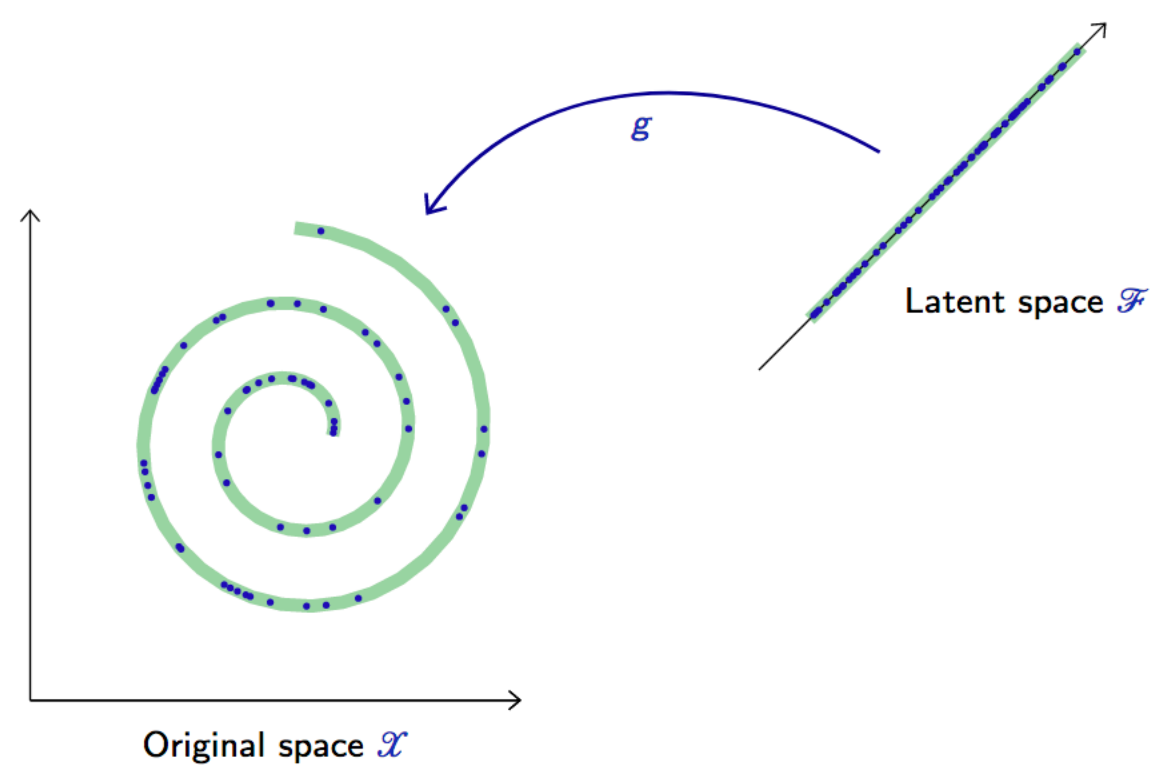
Beyond regression and classification

- Not all tasks are predicting a label from features, as in classification and regression, e.g.
  - data simulation
  - density estimation
  - anomaly detection
  - denoising
  - data compression, …
- Often you do not have labels → **Unsupervised Learning**
- Often framed as **modelling the lower dimensional "meaningful degrees of freedom"** that describe the data

Meaningful representations

- How can we find the "meaningful degrees of freedom" in the data? → **dimensionality reduction/compression**:
  - Can we compress the data to a latent space with smaller number of dimensions, and still recover the original data from this latent space representation?
  - Latent space must encode and retain the important information about the data
  - Can we learn this compression and latent space





Autoencoders

Autoencoders

- Autoencoders map a space to itself through a compression, $x \to z \to \hat{x}$, and should be close to the identity on the data
  - Data: $x \in \mathcal{X}$ Latent space: $z \in \mathcal{F}$
  - **Encoder:** Map from $\mathcal{X}$ to a lower dimensional latent space $\mathcal{F}$
    - parametrize as neural network $f_\theta(x)$ with parameters θ
  - **Decoder:** Map from latent space $\mathcal{F}$ back to data space $\mathcal{X}$
    - parametrize as neural network $g_\psi(z)$ with parameters ψ
- **Reconstruction loss:** *mean squared error (MSE)* between data and encoded-decoded data

$$\mathcal{L}_{\text{reco}}(\theta, \psi) = \frac{1}{N} \sum_n \| \, x_n - g_\psi(f_\theta(x_n)) \, \|^2$$

- Minimize this loss over parameters of encoder (θ) and decoder (ψ)

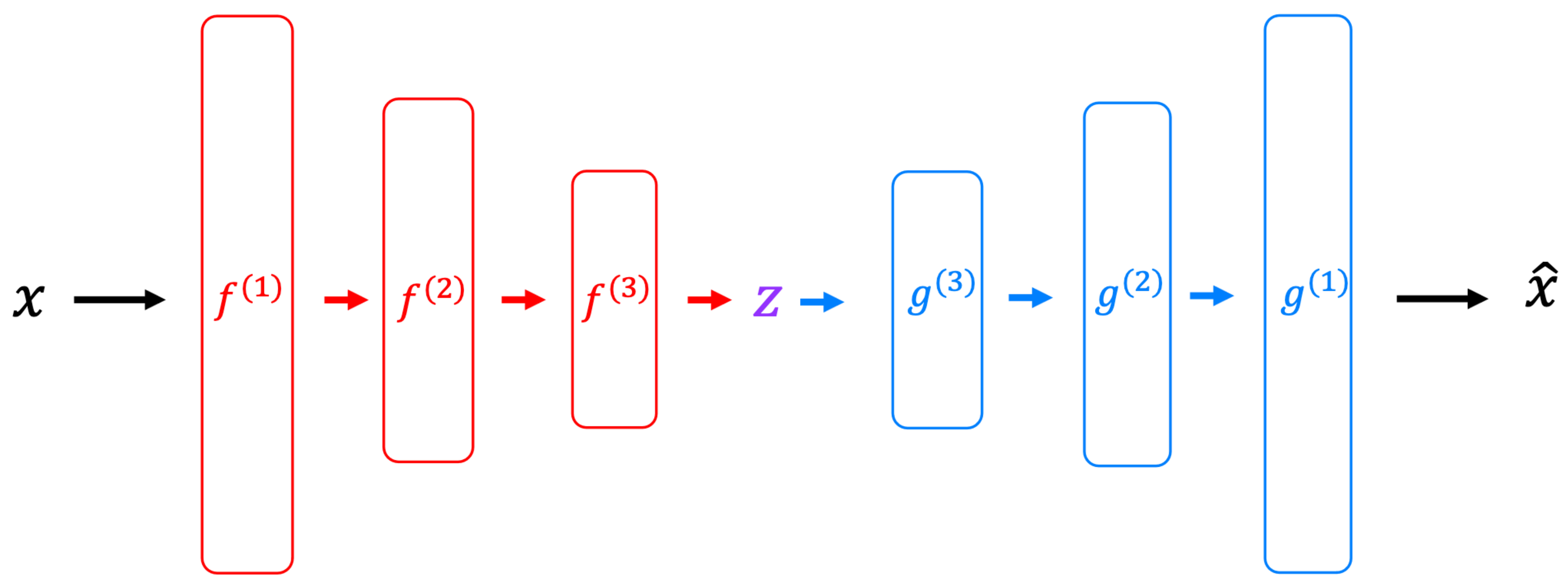
Deep Autoencoders

- If the latent space is of lower dimension, the autoencoder has to capture a "good" parametrization, and in particular dependencies between attributes → **Deep Autoencoders**

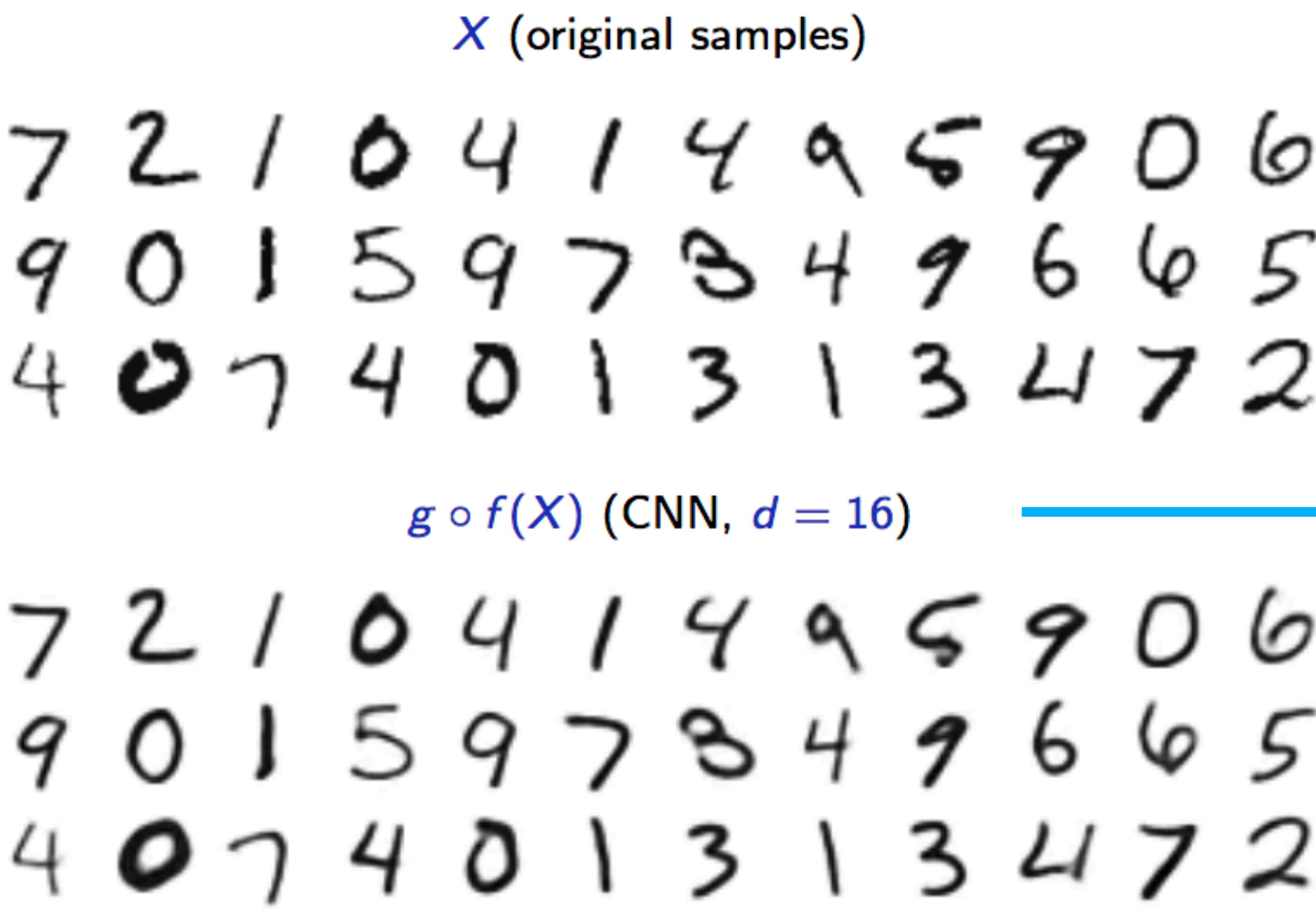
- When f_θ and g_ψ are multiple neural network layers, can learn complex mappings between $\mathscr{X}$ and $\mathscr{F}$

  - f_θ and g_ψ can be MLP, CNN, RNN, GNN, …

  - choice of network architecture will depend on data



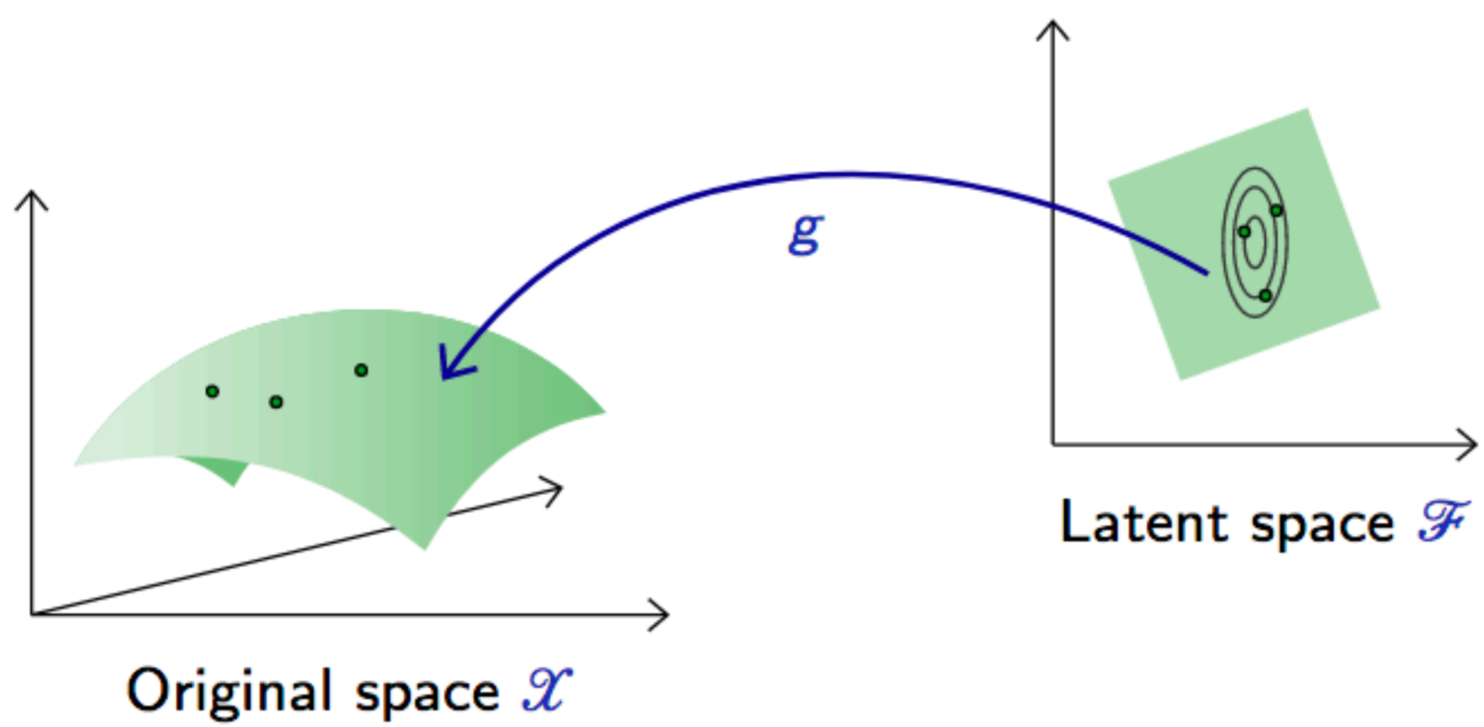
Deep Convolutional AE



f_θ and g_ψ are each 5 convolutional layers

New data generation

- Can we sample in latent space and decode to generate new data?
- What distribution to sample from in latent space?
    - try Gaussian with mean and variance learned from data



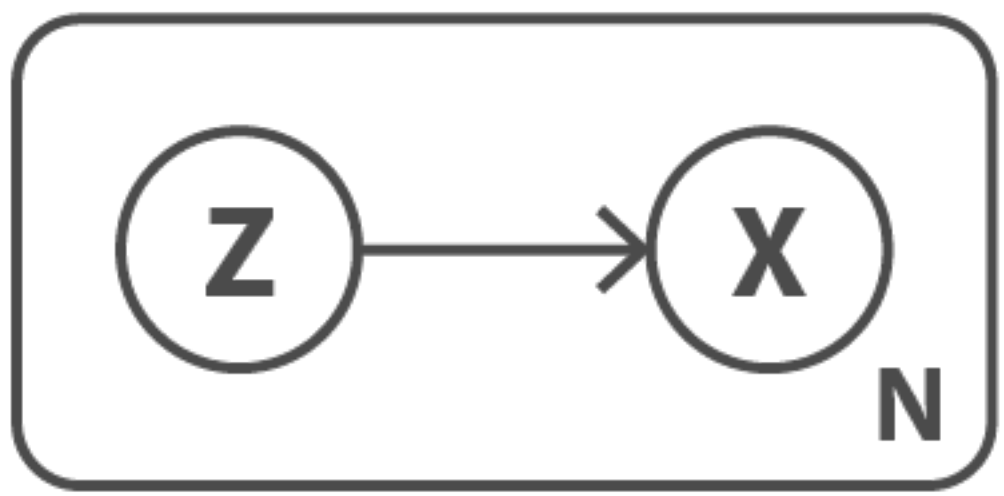
Generative models

Generative models

- Generative models aim to:
  - Learn a distribution $p(x)$ that explains the data
  - Draw samples of plausible data points
- Explicit Models:
  - can evaluate the density $p(x)$ of a data point x
  - e.g., normalizing flows
- Implicit Models:
  - can only sample from $p(x)$ but not evaluate density
  - e.g., variational autoencoders and generative adversarial networks

Probabilistic autoencoder

- Assume a prior distribution $p(z)$ of the latent space
- Sample a latent representation z from $p(z)$
- Generate new data x from the conditional likelihood distribution $p(x|z)$



Probabilistic autoencoder

- Assume a prior distribution $p(z)$ of the latent space
- Sample a latent representation z from $p(z)$
- Generate new data x from the conditional likelihood distribution $p(x|z)$
- We can now redefine our notions of encoder and decoder → probabilistic versions:
  - **Decoder** → $p(x|z)$: describes the distribution of the decoded variable given the encoded one
  - **Encoder** → $q(z|x)$: describes the distribution of the encoded variable given the decoded one

$$x \rightarrow q(z|x) \xrightarrow[\textit{sample}]{} z \rightarrow p(x|z)$$

We call this Variational Autoencoder

[introduced by Kingma and Welling at ICLR '14]

Variational Autoencoder

- Assume $p(z)$ is a standard Gaussian distribution $\mathcal{N}$ → the VAE encoder has two outputs: the mean and variance of the gaussian distribution

$$f_\psi(x) = \{\mu_\psi(x), \sigma_\psi(x)\}$$

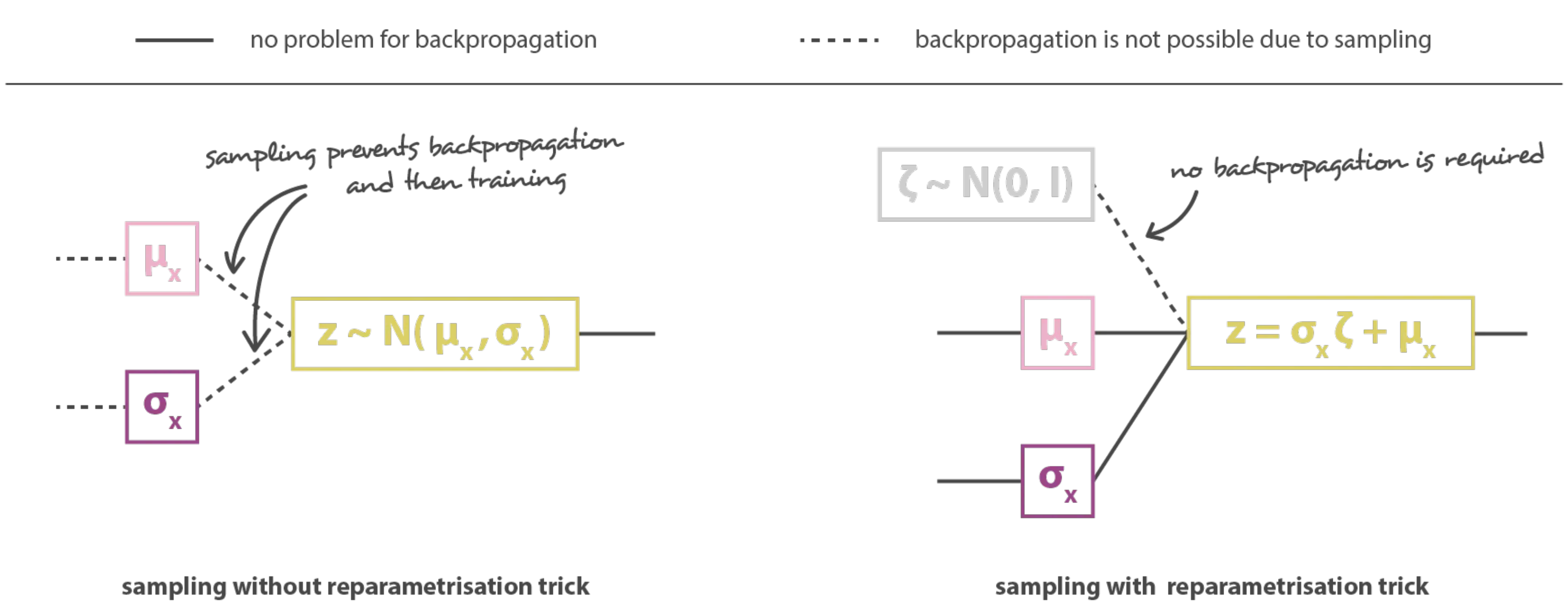
  - where ψ are parameters of the neural networks

- The probability of a point in latent space is $p_\psi(z \,|\, x) = \mathcal{N}(z \,|\, \mu_\psi(x), \sigma_\psi(x))$

- Draw a sample from the latent space using the so-called *re-parametrization trick*

$$z = \sigma_\psi(x) \cdot \epsilon + \mu_\psi(x) \quad \text{where} \quad \epsilon = \mathcal{N}(0,I)$$

Allows to make the gradient descent possible despite the random sampling that occurs halfway of the architecture



Variational Autoencoder

- Following the same approach the VAE decoder has also two outputs

$$g_\theta(z) = \{\mu_\theta(z), \sigma_\theta(z)\}$$

- Likelihood of an observation x is

$$p_\theta(x \,|\, z) = \mathcal{N}(x \,|\, \mu_\theta(z), \sigma_\theta(z))$$

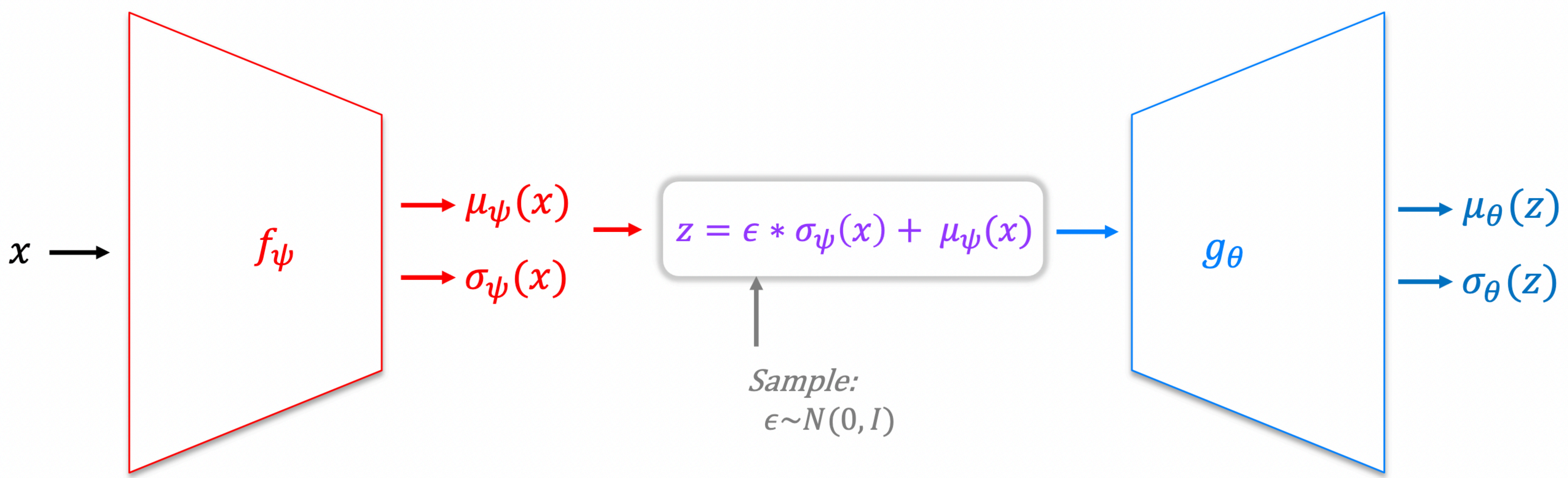
Variational Autoencoder

- Following the same approach the VAE decoder has also two outputs

$$g_\theta(z) = \{\mu_\theta(z), \sigma_\theta(z)\}$$

- Likelihood of an observation x is

$$p_\theta(x \,|\, z) = \mathcal{N}(x \,|\, \mu_\theta(z), \sigma_\theta(z))$$



The VAE loss

- **Reconstruction loss:** maximize expected log-likelihood of decoding x from encodings of x

$$\mathscr{L}_{\text{reco}} = \mathbb{E}_{z \sim q(z|x)}[\log p(x \mid z)] \approx \frac{1}{N} \sum_{z_i \sim q(z|x)} \log p(x \mid z_i)$$

The VAE loss

- **Reconstruction loss:** maximize expected log-likelihood of decoding x from encodings of x

$$\mathscr{L}_{\text{reco}} = \mathbb{E}_{z \sim q(z|x)}[\log p(x \mid z)] \approx \frac{1}{N} \sum_{z_i \sim q(z|x)} \log p(x \mid z_i)$$

- We want to make sure that the latent space distribution $q(z \mid x)$ remains a gaussian during optimization, that is it does not collapse to a single point

- In other words, need to ensure that **the encoder is consistent with our prior** $p(z)$

The VAE loss

- **Reconstruction loss:** maximize expected log-likelihood of decoding x from encodings of x

$$\mathscr{L}_{\text{reco}} = \mathbb{E}_{z \sim q(z|x)}[\log p(x \mid z)] \approx \frac{1}{N} \sum_{z_i \sim q(z|x)} \log p(x \mid z_i)$$

- We want to make sure that the latent space distribution $q(z \mid x)$ remains a gaussian during optimization, that is it does not collapse to a single point

- In other words, need to ensure that **the encoder is consistent with our prior** $p(z)$

- The difference between two distributions can be measured with the **Kullback–Leibler divergence**

$$D_{KL} = [q(z \mid x) \mid p(z)] = \mathbb{E}_{q(Z|x)}\left[\log \frac{q(z \mid x)}{p(z)}\right] = \int q(z \mid x) \log \frac{q(z \mid x)}{p(z)} dz$$

The VAE loss

- **Reconstruction loss:** maximize expected log-likelihood of decoding x from encodings of x

$$\mathscr{L}_{\text{reco}} = \mathbb{E}_{z \sim q(z|x)}[\log p(x\,|\,z)] \approx \frac{1}{N} \sum_{z_i \sim q(z|x)} \log p(x\,|\,z_i)$$

- We want to make sure that the latent space distribution $q(z\,|\,x)$ remains a gaussian during optimization, that is it does not collapse to a single point

- In other words, need to ensure that **the encoder is consistent with our prior** $p(z)$

- The difference between two distributions can be measured with the **Kullback–Leibler divergence**

$$D_{KL} = [q(z\,|\,x)\,|\,p(z)] = \mathbb{E}_{q(Z|x)}\left[\log \frac{q(z\,|\,x)}{p(z)}\right] = \int q(z\,|\,x) \log \frac{q(z\,|\,x)}{p(z)} dz$$

- **The full VAE objective:**

$$\max_{\theta,\psi} \mathscr{L}(\theta,\psi) = \max_{\theta,\psi}[\ \mathbb{E}_{q_\psi(Z|x)}[\log p_\theta(x, z)] - D_{KL}(q_\psi(z\,|\,x)\,|\,p(z))\ ]$$

reconstruction loss

latent space regularization

The VAE loss

- **Reconstruction loss:** maximize expected log-likelihood of decoding x from encodings of x

$$\mathscr{L}_{\text{reco}} = \mathbb{E}_{z \sim q(z|x)}[\log p(x \mid z)] \approx \frac{1}{N} \sum_{z_i \sim q(z|x)} \log p(x \mid z_i)$$

- We want to make sure that the latent space distribution $q(z \mid x)$ remains a gaussian during optimization, that is it does not collapse to a single point

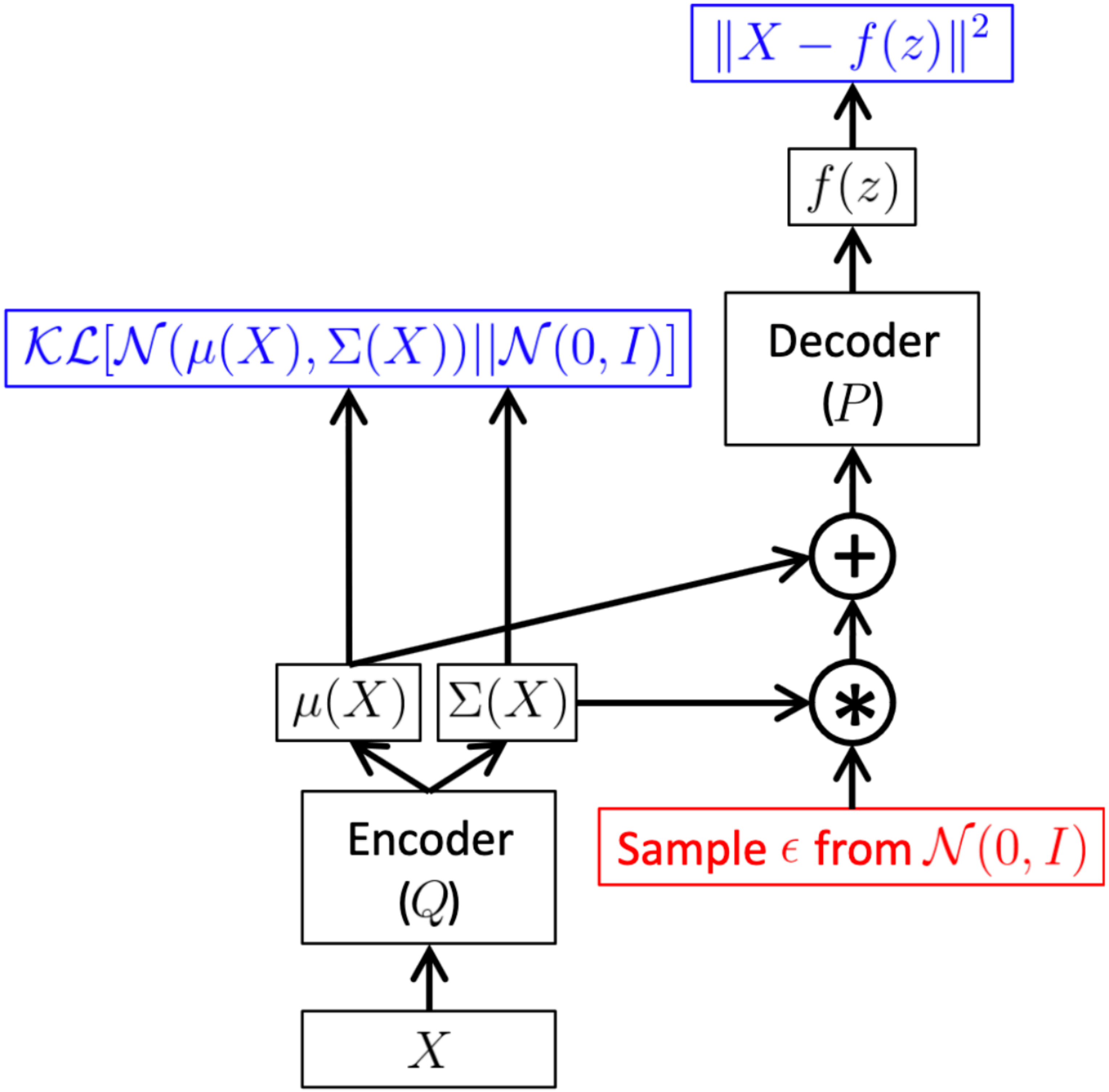
- In other words, need to ensure that **the encoder is consistent with our prior** $p(z)$

- The difference between two distributions can be measured with the **Kullback–Leibler divergence**

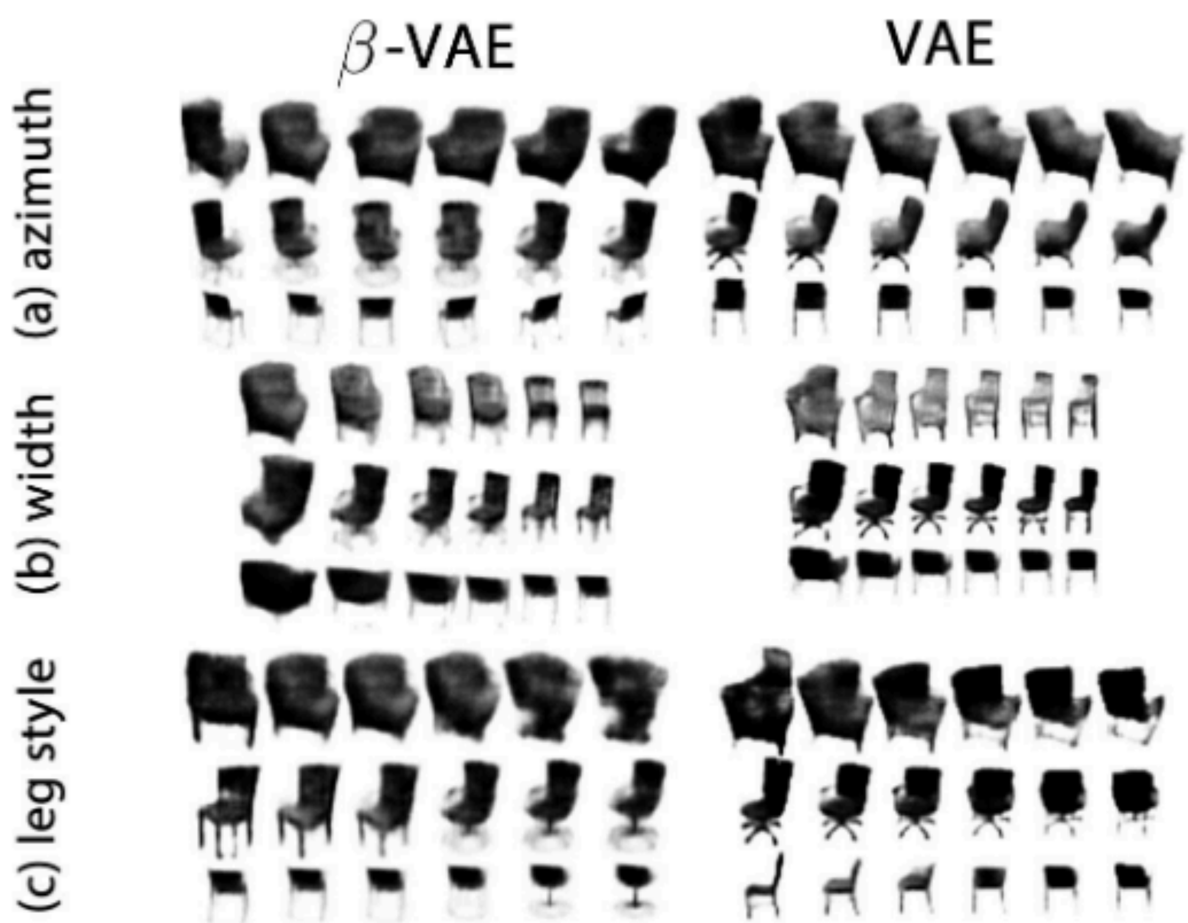
$$D_{KL} = [q(z \mid x) \mid p(z)] = \mathbb{E}_{q(Z|x)}\left[\log \frac{q(z \mid x)}{p(z)}\right] = \int q(z \mid x) \log \frac{q(z \mid x)}{p(z)} dz$$

- The KL loss can be computed analytically: $D_{KL} = \frac{1}{2} \sum [1 + \log(\sigma_\theta^2(x)) - \mu_\theta^2(x) - \sigma_\theta^2(x)]$

The full VAE model



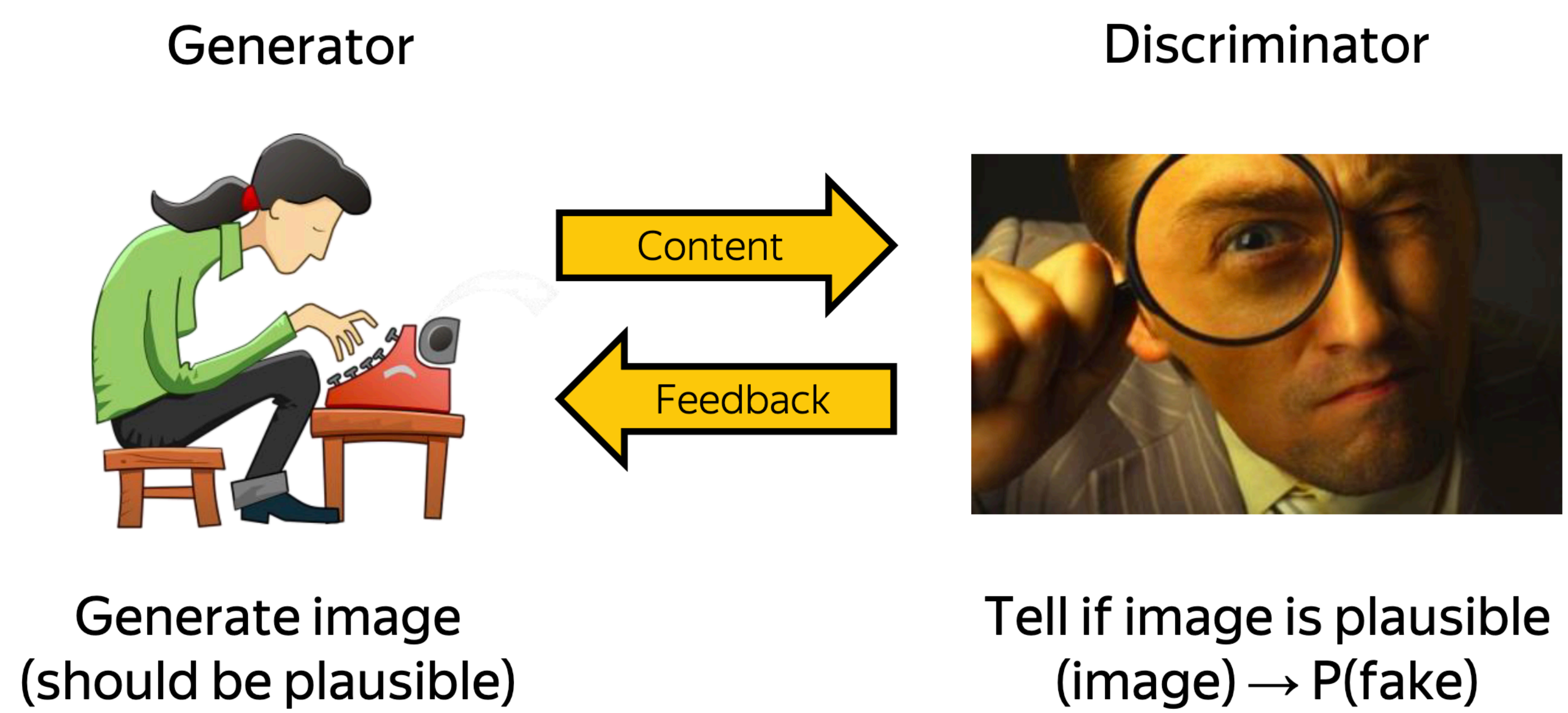
Examples



Higgins et al., [2017]

Another way to generate data

- Formulate as a two player game
- One player tries to output data that looks as real
- Another player tries to compare real and fake data
- In this case we need:
    - A **generator** that can produce samples
    - A **measure** of not too far from the real data



Generative Adversarial Network (GAN)

- **Generator NN** $g_\theta(z)$ with parameters θ
  - map sample from known $p(z)$ to sample in data space

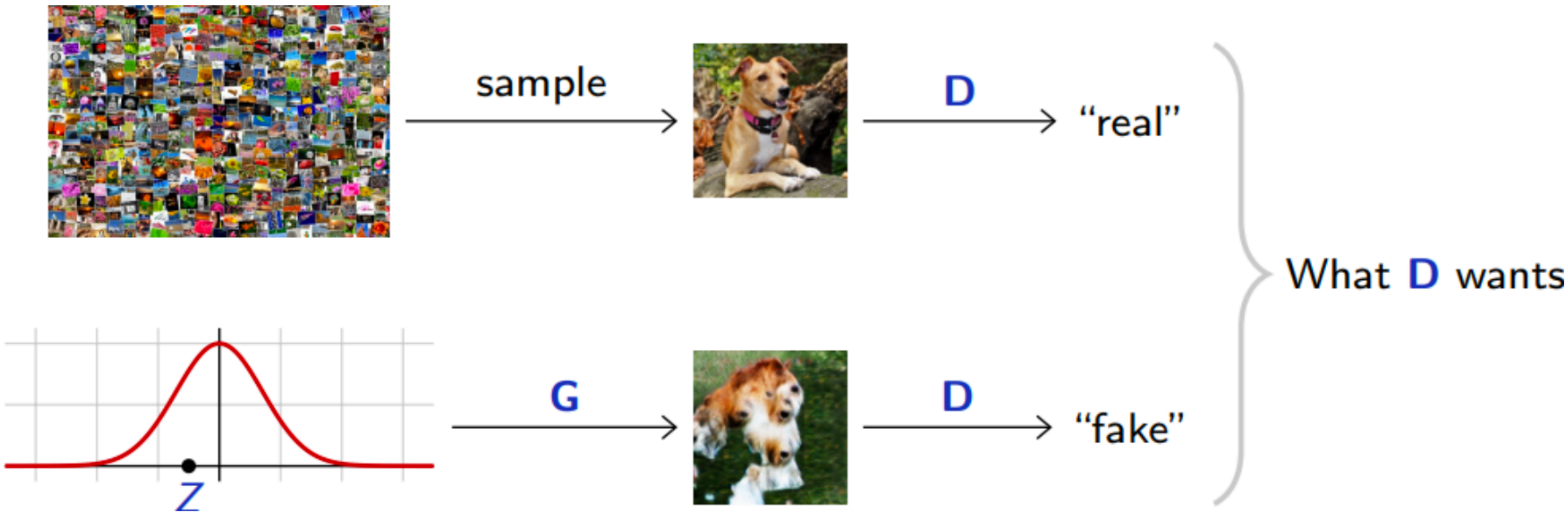
$$x = g_\theta(z) \quad \text{with} \quad z \sim p(z)$$

  - we don't know what the generated distribution $p_\theta(x)$ is, but we can sample from it → *implicit model*

- **Discriminator NN** $d_\phi(x)$ with parameters ϕ
  - classifier trained to distinguish between real and fake data
  - it learns to predict $p(y = \text{real} \,|\, x)$
  - it is our measure of *not too far from the real data*

Introduced by Ian Goodfellow et al. at NIPS '14

GAN setup



- Generator's goal is to produce *fake* data that tricks the discriminator to think it is *real* data
- Discriminator wants to miss-classify data as real or fake as little as possible
- The setup is **adversarial** because the two networks have opposing objectives

GAN objective

- Data
  - Real data samples: $\{x_i, y_i = 1\}$
  - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$
- For a fixed generator, can train discriminator by minimizing the cross entropy

$$
\begin{aligned}
L(\phi) &= -\frac{1}{2N}\sum_{i=1}^{N}\Big[y_i \log d_\phi(x_i) + (1-\tilde{y}_i)\log(1-d_\phi(\tilde{x}_i)\ )\Big] \\
&= -\frac{1}{2N}\sum_{i=1}^{N}\Big[\log d_\phi(x_i) + \log(1-d_\phi(g_\theta(z_i))\ )\Big] \\
&= -\mathbb{E}_{x\sim p_{\mathrm{data}}(x)}\Big[\log d_\phi(x)\Big] - \mathbb{E}_{z\sim p(z)}\Big[\log(1-d_\phi(g_\theta(z))\ )\Big]
\end{aligned}
$$

GAN objective

- However, generator isn't fixed… have to train it!

- Consider objective as a function of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \Big[\log d_\phi(x) \Big] + \mathbb{E}_{z \sim p(z)} \Big[\log(1 - d_\phi(g_\theta(z))\ ) \Big]$$

  - the goal of the generator g_θ is to fool the discriminator, so the generative neural network is trained to **maximise the final classification error** $V(\phi, \theta)$

GAN objective

- However, generator isn't fixed... have to train it!

- Consider objective as a function of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\Big[\log d_\phi(x)\Big] + \mathbb{E}_{z \sim p(z)}\Big[\log(1 - d_\phi(g_\theta(z))\ )\Big]$$

  - the goal of the generator g_θ is to fool the discriminator, so the generative neural network is trained to **maximise the final classification error** $V(\phi, \theta)$

  - the goal of the discriminator d_ϕ is to detect fake generated data, so the discriminative neural network is trained to **minimise the final classification error** $V(\phi, \theta)$

GAN objective

- However, generator isn't fixed... have to train it!
- Consider objective as a function of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\Big[\log d_\phi(x)\Big] + \mathbb{E}_{z \sim p(z)}\Big[\log(1 - d_\phi(g_\theta(z))\ )\Big]$$

  - the goal of the generator g_θ is to fool the discriminator, so the generative neural network is trained to **maximise the final classification error** $V(\phi, \theta)$
  - the goal of the discriminator d_ϕ is to detect fake generated data, so the discriminative neural network is trained to **minimise the final classification error** $V(\phi, \theta)$

- So our optimization goal becomes:

$$\theta^* = \arg\min_{\phi}\max_{\theta} V(\phi, \theta)$$

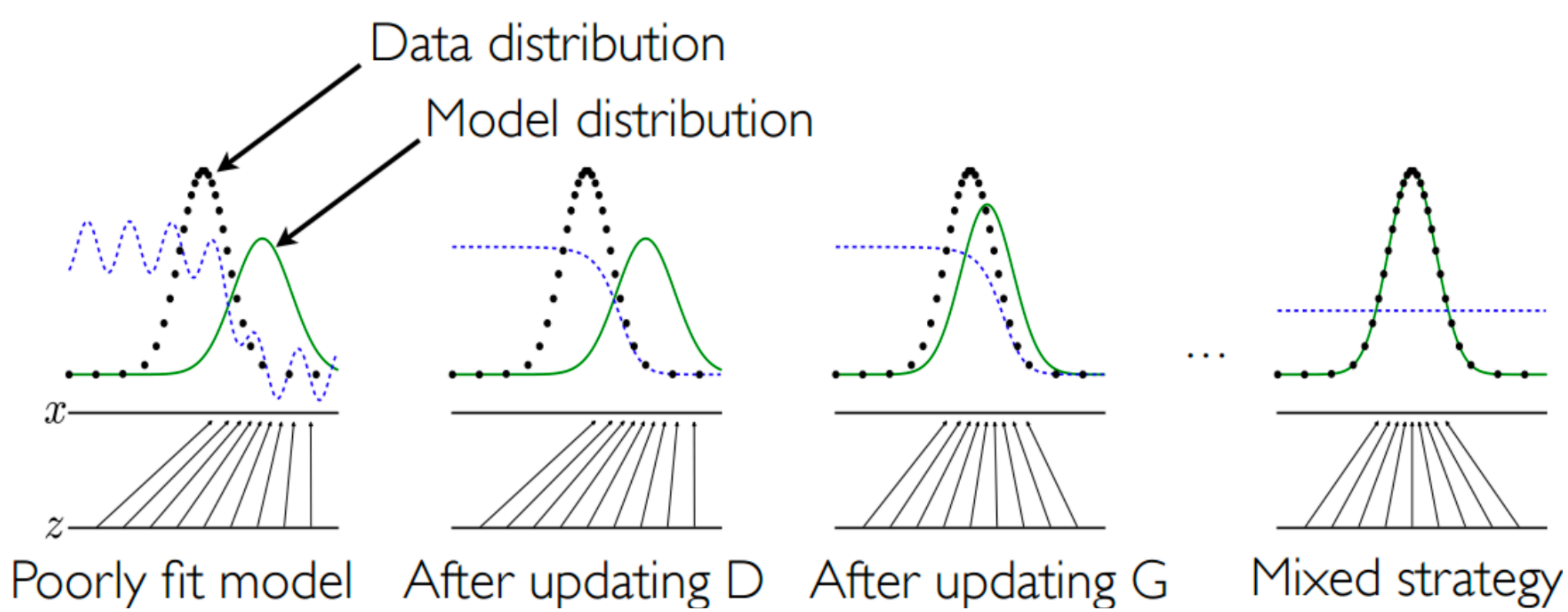
GAN training

- Alternating gradient descent to solve the min-max problem

$$\theta \leftarrow \theta - \gamma \nabla_\theta V(\phi,\ \theta) = \theta - \gamma \frac{\partial V}{\partial d} \frac{\partial (d_\phi)}{\partial g} \frac{\partial g_\theta}{\partial \theta}$$

$$\phi \leftarrow \phi - \gamma \nabla_\phi V(\phi,\ \theta) = \phi - \gamma \frac{\partial V}{\partial d} \frac{d(d_\phi)}{d\phi}$$

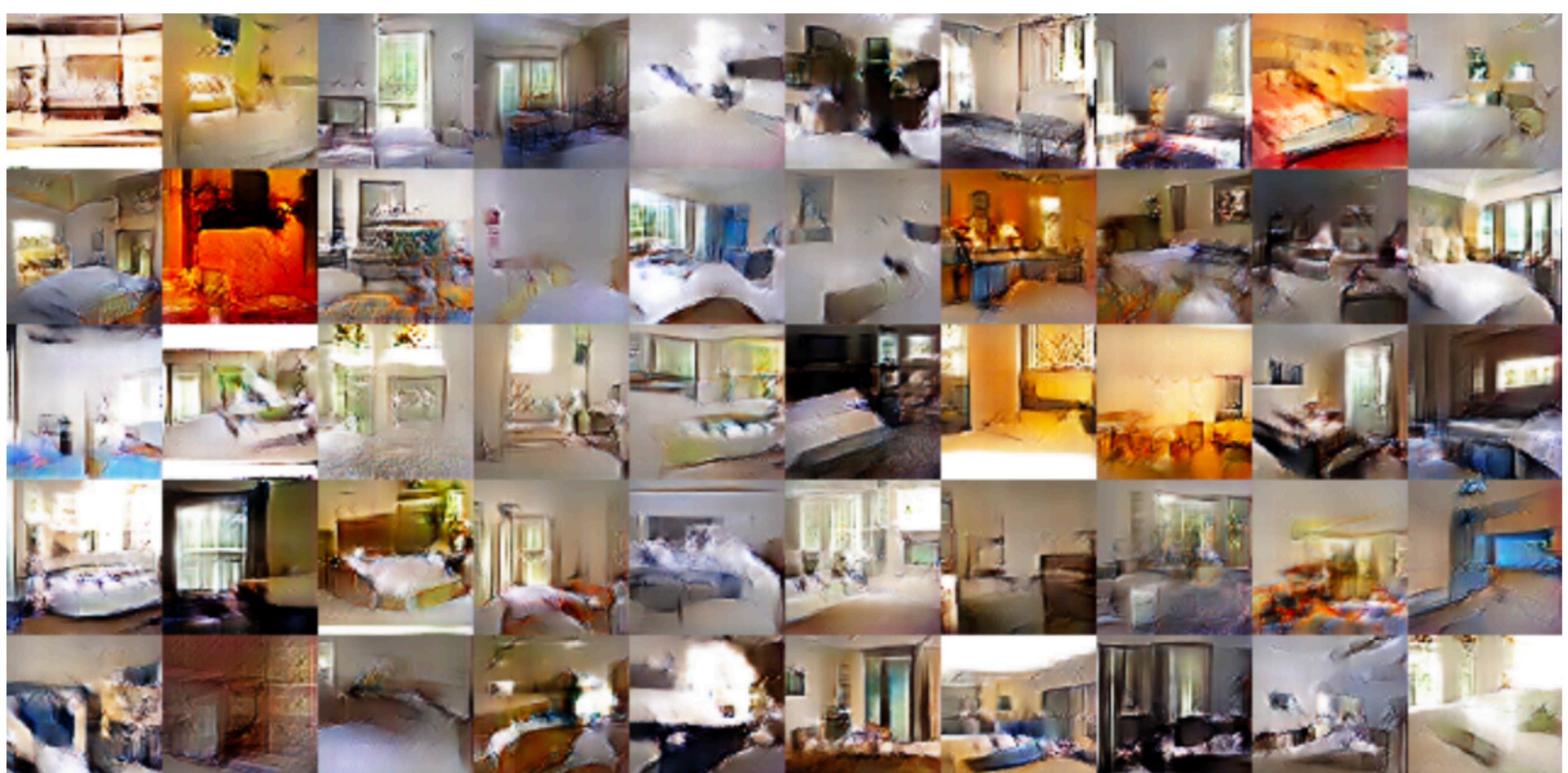
- For each θ step, take k steps in ϕ to keep discriminator near optimal



Examples

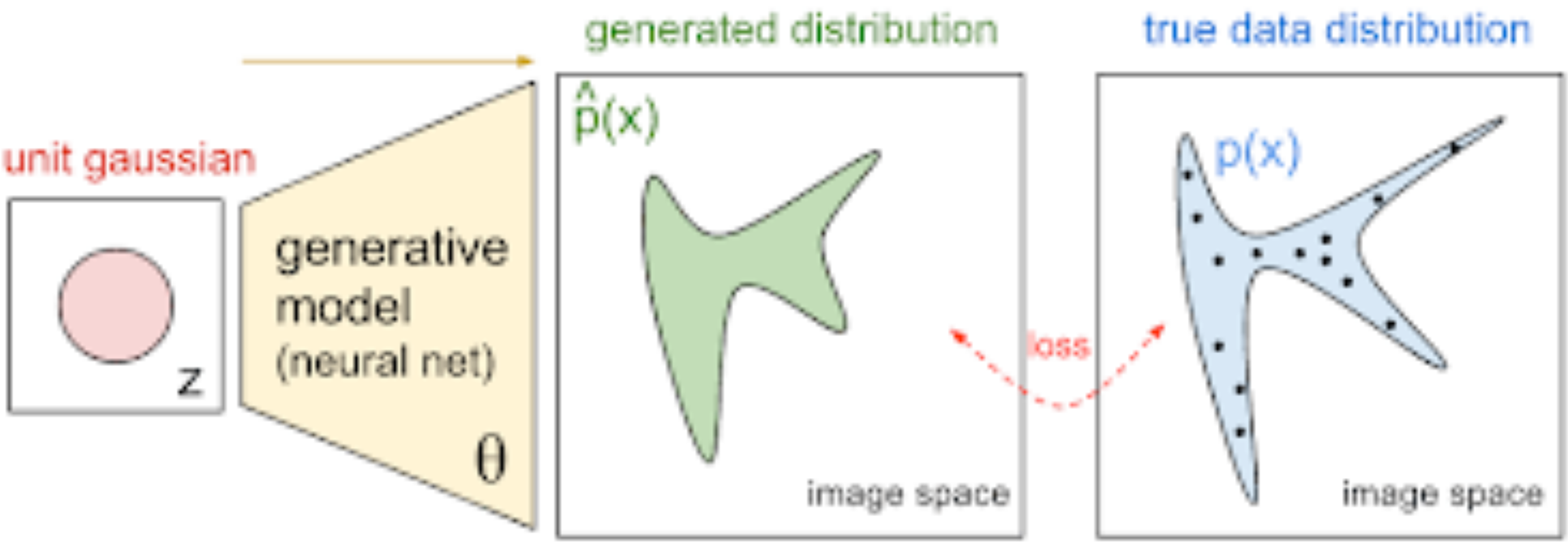
Goodfellow et. al., 2014

Not so good
Goodfellow 2016



Radford et al, 2015

Do we really learn a distribution?



How to measure it?

- We have learned that we can measure the distance between two distributions with the KL divergence

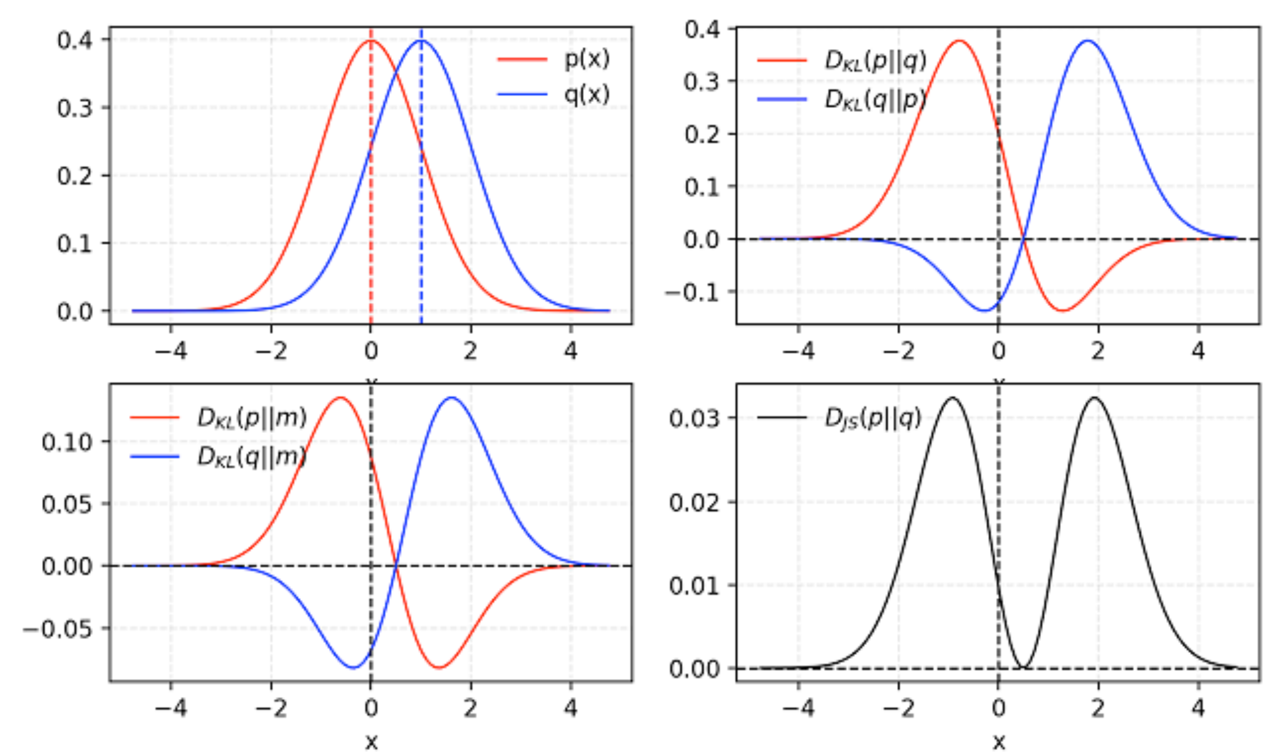
$$D_{KL} = [q(z \,|\, x) \,|\, p(z)] = \mathbb{E}_{q(Z|x)}\left[\log \frac{q(z \,|\, x)}{p(z)}\right] = \int q(z \,|\, x) \log \frac{q(z \,|\, x)}{p(z)} dz$$

- Is it good enough?

A better option: the JS divergence

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$$

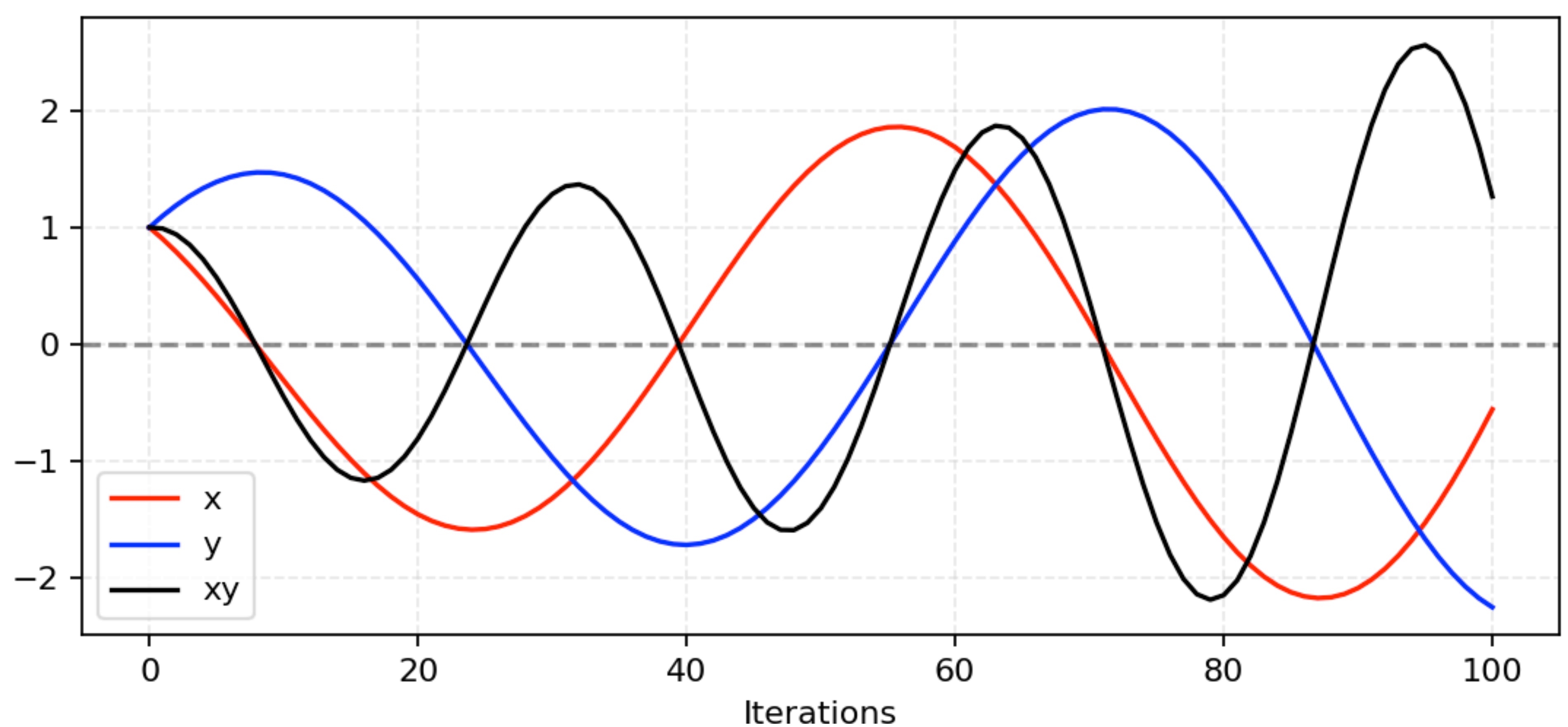
- JS = Jensen-Shannon
- Historically was the first used in GANs (if you hear just "GAN", it's likely optimizing JS)
- Why it's better for GANs? → mostly from empirical verification, but
  - JS divergence is symmetric and bounded
  - these are nice properties
- Some believe that one reason behind GANs success was switching the loss from KL to JS
- But it's not as easy…



$m = (p+q)/2$

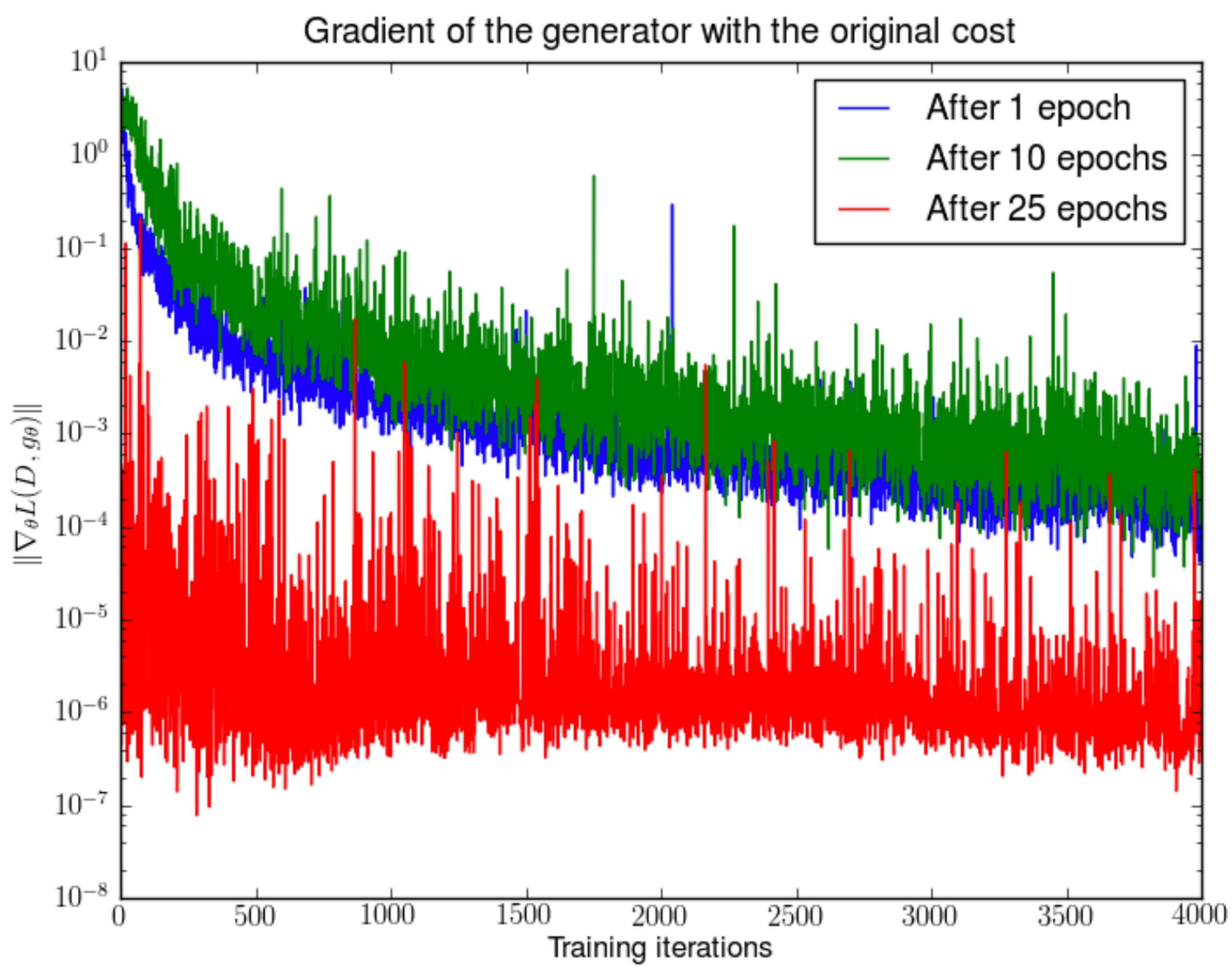
Challenges with GANs

- **Problem #1: Hard to achieve Nash equilibrium**
  - Player 1 takes control of x to minimize $f_1(x) = xy$
  - Player 2 at the same time constantly updates y to minimize $f_2(y) = -xy$
  - so x wants to become negative against f_2 objective and y wants to become positive against f_1 objective



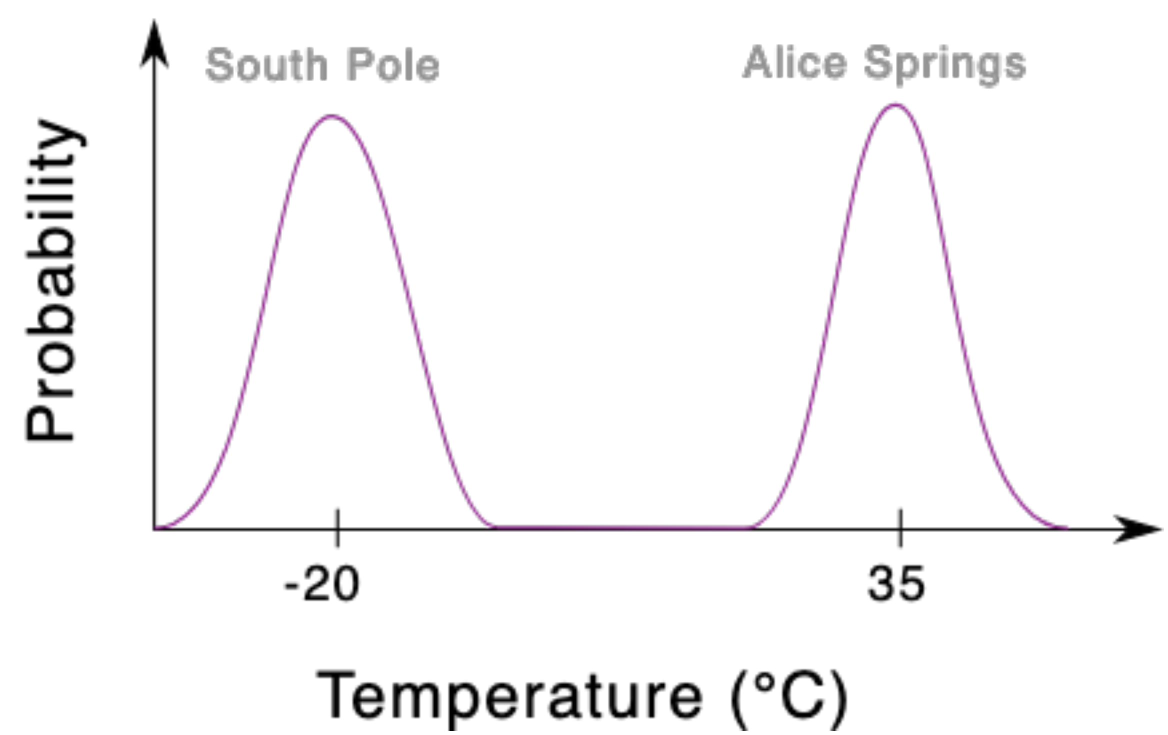
Challenges with GANs

- **Problem #2: Vanishing gradient**
  - if the generator is perfect than the loss function is 0
  - no gradient to update the loss during learning iterations



Challenges with GANs

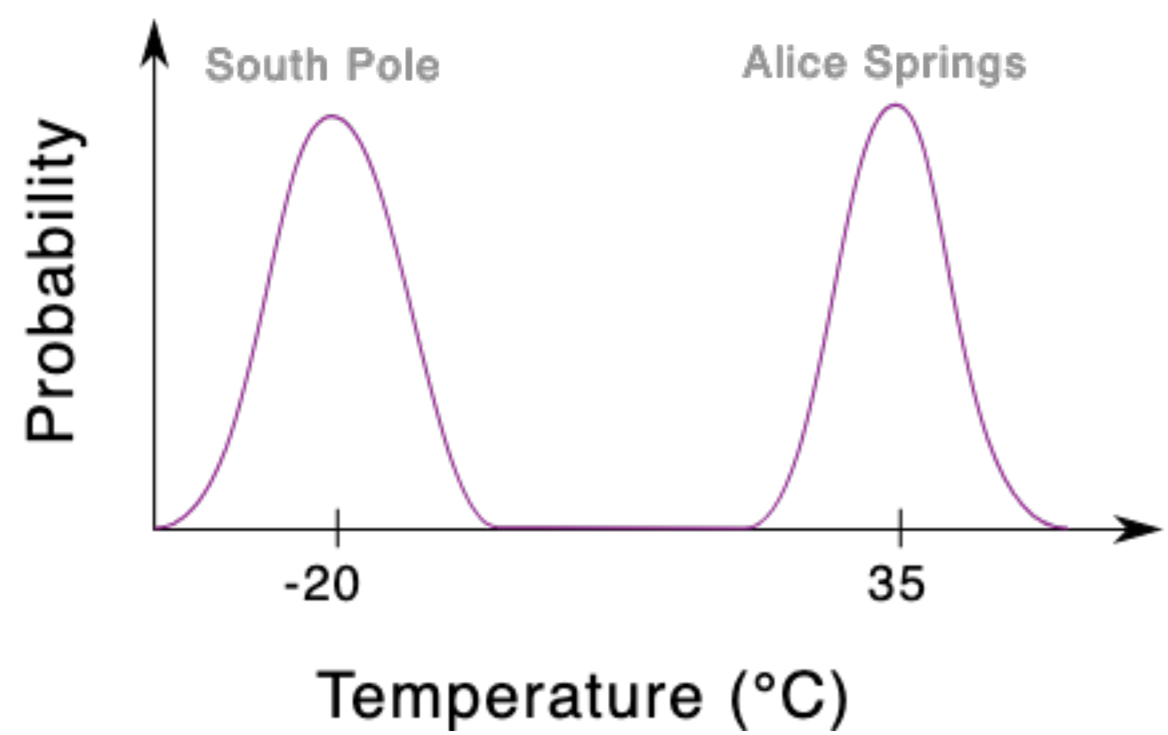
- **Problem #3: Mode collapse**
  - Example: dataset containing a mixture of summer day temperature readings from Alice Springs in central Australia (typically very hot) and the South Pole in Antarctica (typically very cold)



Challenges with GANs

- **Problem #3: Mode collapse**
  - Example: dataset containing a mixture of summer day temperature readings from Alice Springs in central Australia (typically very hot) and the South Pole in Antarctica (typically very cold)

    1. The generator learns that it can fool the discriminator into thinking that it is outputting realistic temperatures by producing values close to Antarctic temperatures.

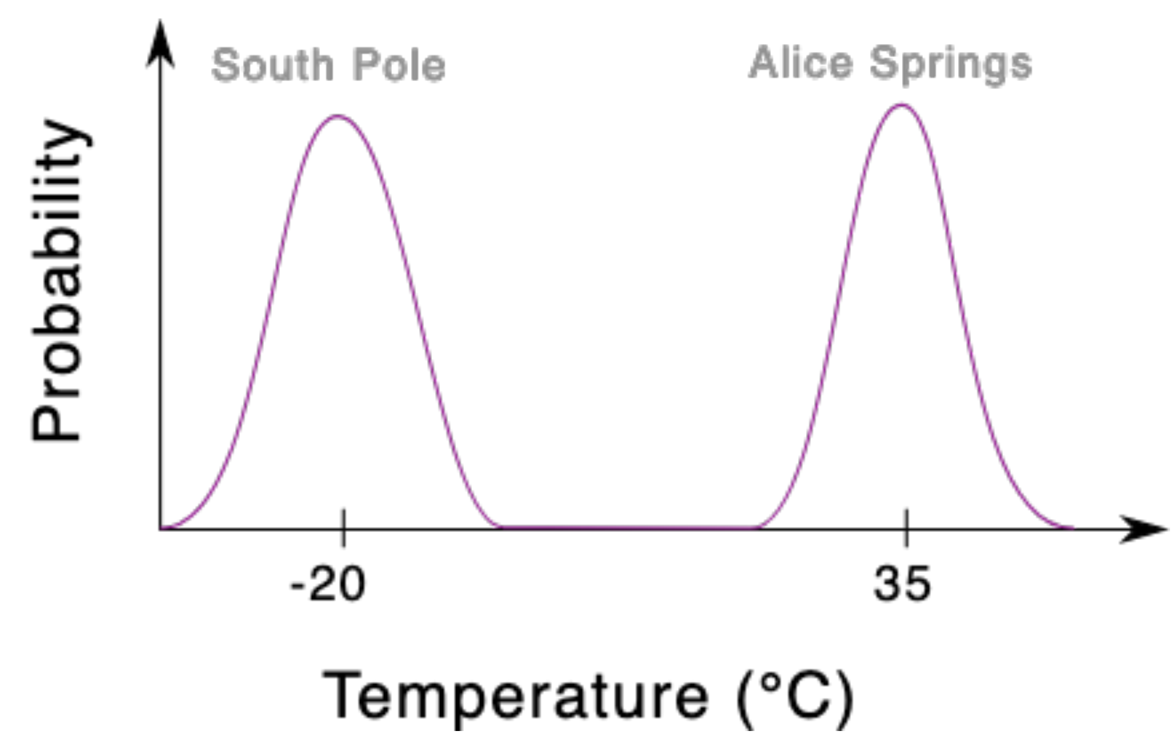


Challenges with GANs

- **Problem #3: Mode collapse**
  - Example: dataset containing a mixture of summer day temperature readings from Alice Springs in central Australia (typically very hot) and the South Pole in Antarctica (typically very cold)

    1. The generator learns that it can fool the discriminator into thinking that it is outputting realistic temperatures by producing values close to Antarctic temperatures.

    2. The discriminator counters by learning that all Australian temperatures are real (not produced by the generator)



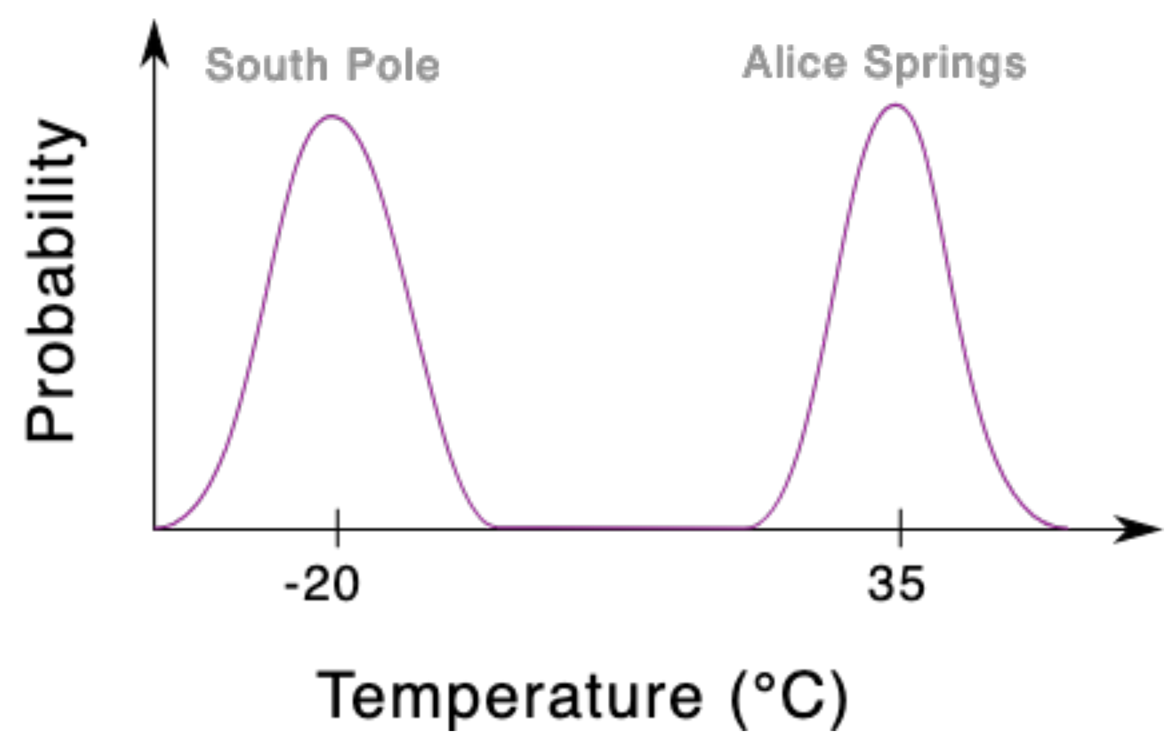
Challenges with GANs

- **Problem #3: Mode collapse**
  - Example: dataset containing a mixture of summer day temperature readings from Alice Springs in central Australia (typically very hot) and the South Pole in Antarctica (typically very cold)

    1. The generator learns that it can fool the discriminator into thinking that it is outputting realistic temperatures by producing values close to Antarctic temperatures.

    2. The discriminator counters by learning that all Australian temperatures are real (not produced by the generator)

    3. The generator exploits the discriminator by switching modes to produce values close to Australian temperatures instead, abandoning the Antarctic mode.



Challenges with GANs

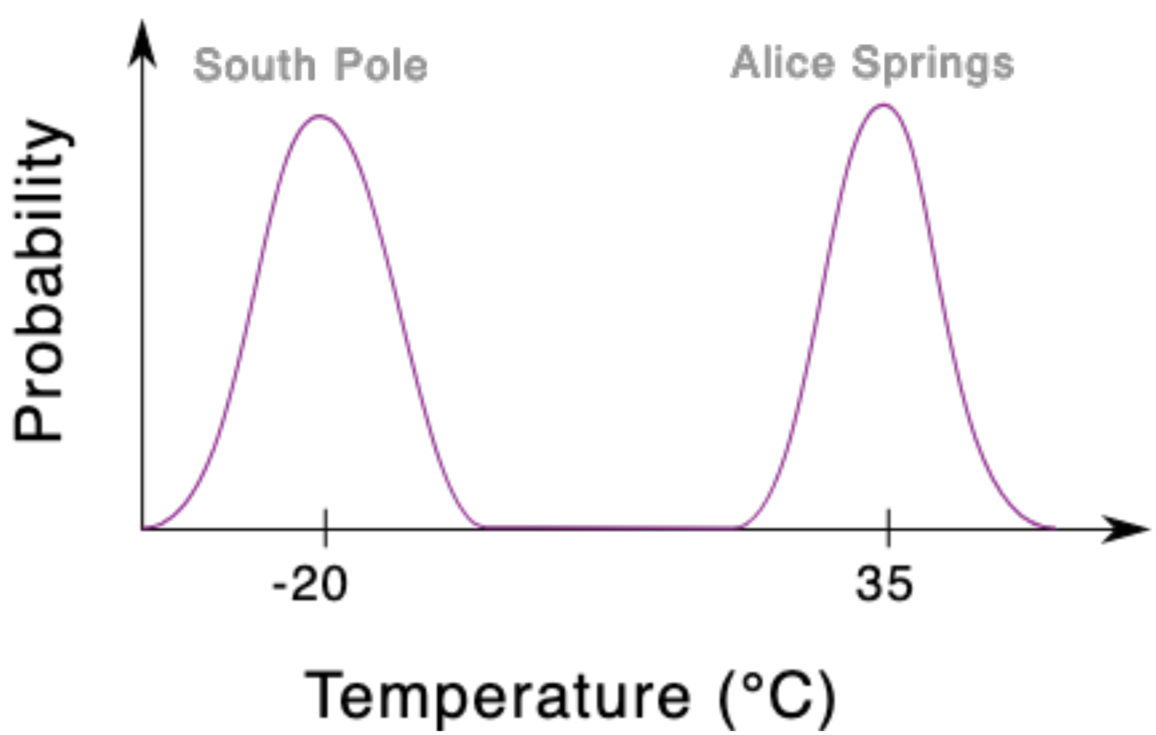
- **Problem #3: Mode collapse**
  - Example: dataset containing a mixture of summer day temperature readings from Alice Springs in central Australia (typically very hot) and the South Pole in Antarctica (typically very cold)

    1. The generator learns that it can fool the discriminator into thinking that it is outputting realistic temperatures by producing values close to Antarctic temperatures.

    2. The discriminator counters by learning that all Australian temperatures are real (not produced by the generator)

    3. The generator exploits the discriminator by switching modes to produce values close to Australian temperatures instead, abandoning the Antarctic mode.

    4. The discriminator now assumes that all Australian temperatures are fake and Antarctic temperatures are real.

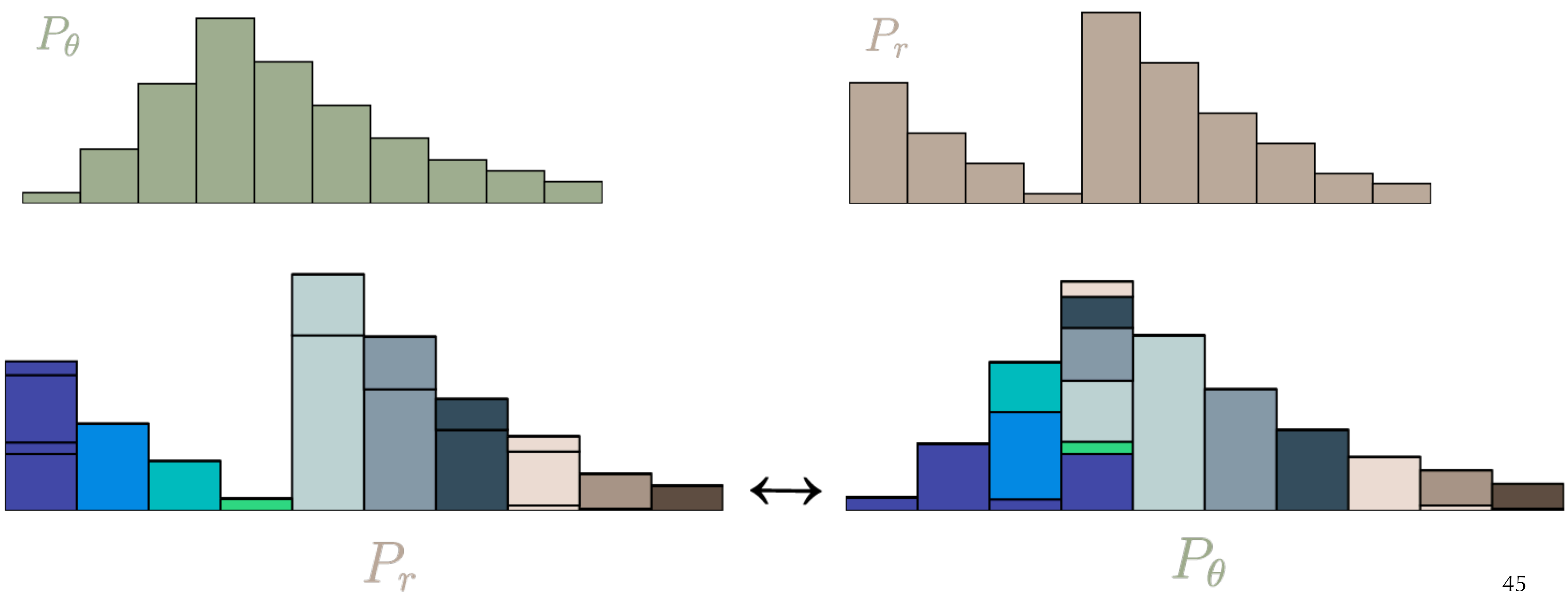


Challenges with GANs

- **Problem #4: No convergence metric**
  - when are generated data good enough?

The Wasserstein Distance

- Empirical observation: the **Wasserstein distance**, also known as the **earth mover's distance (EMD)**, has been shown to solve these problems
- Imagine the distributions have different heaps of a certain amount of earth, then the EMD is the minimal total amount of work it takes to transform one heap into the other
- Work is defined as the amount of earth in a chunk times the distance it was moved.



The Wasserstein Distance

- **Why the Wasserstein is better than JS or KL divergence?**
  - smooth measure, even for disjoint distributions
  - JS is constant with no gradient
  - KL would have given us infinity

Suppose we have two probability distributions, P and Q:

$$\forall (x,y) \in P, x = 0 \text{ and } y \sim U(0,1) \forall (x,y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0,1)$$

When $\theta \neq 0$:

$$D_{KL}(P\|Q) = \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

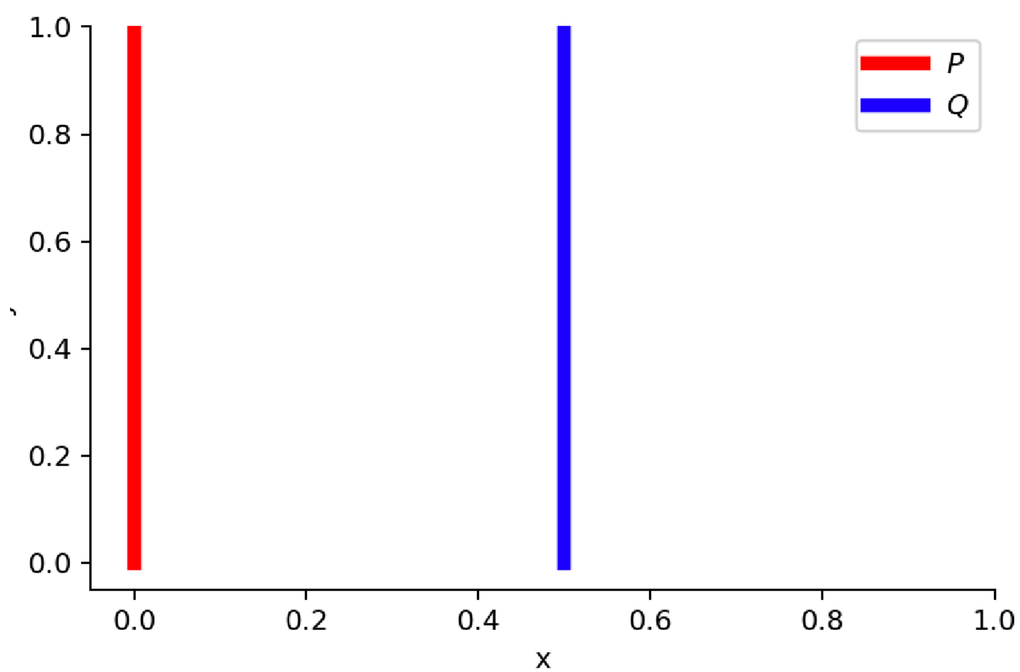
$$D_{KL}(Q\|P) = \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{JS}(P,Q) = \frac{1}{2}\left(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2$$

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

But when $\theta = 0$, two distributions are fully overlapped:

$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P,Q) = 0$$
$$W(P,Q) = 0 = |\theta|$$

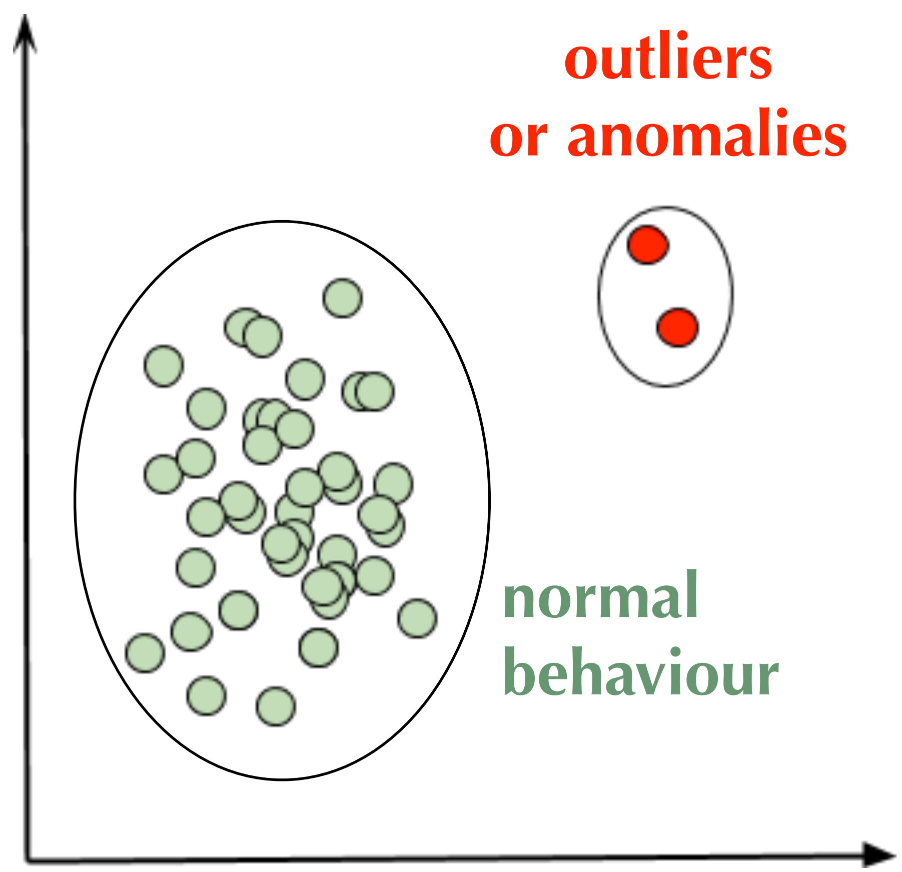


[source]

Anomaly detection

Anomaly detection

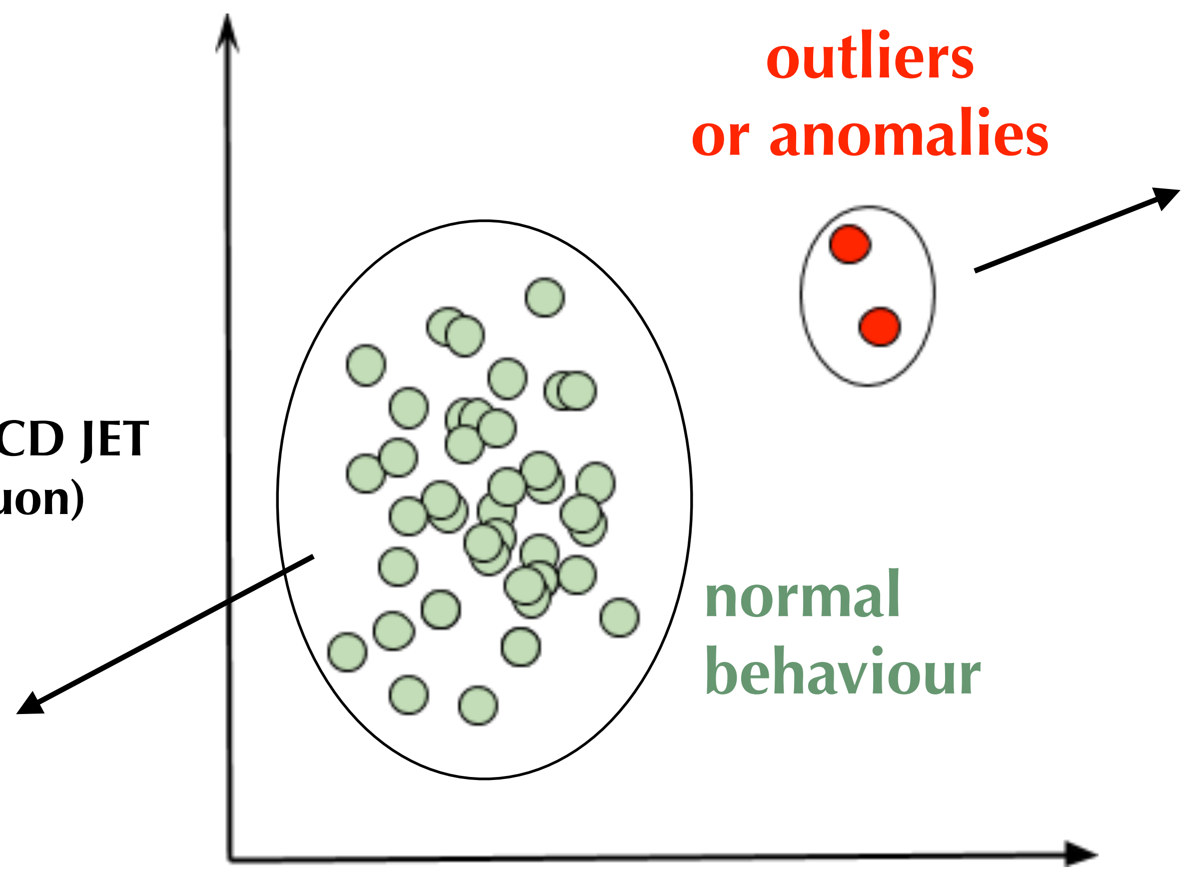
- The process of identifying rare events in data sets which deviate significantly from the majority of the data and do not conform to "normal" behaviour
- Often applied on unlabelled data to learn the normal behaviour → *unsupervised learning*

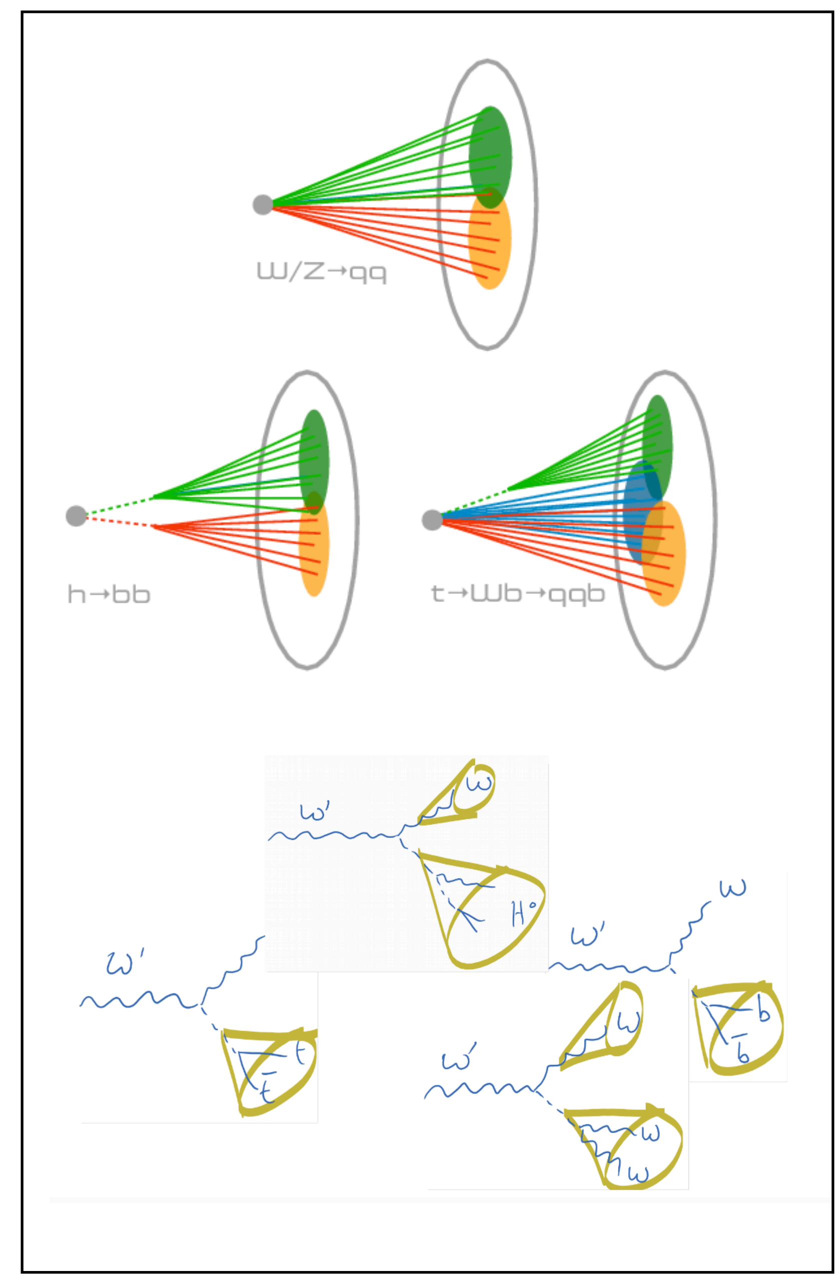


Anomaly detection in HEP

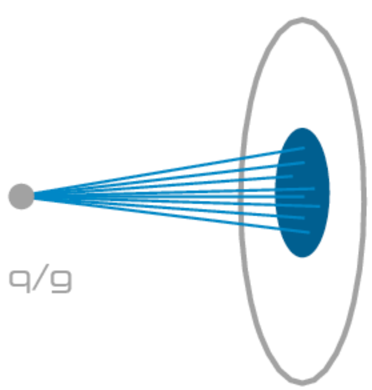
- **Model-agnostic searches for new physics at the LHC**
  - at object level — eg., jet identification

LARGE CLASS OF POTENTIAL SIGNAL JETS
(from unknown new physics)





BACKGROUND QCD JET
(single quark/gluon)

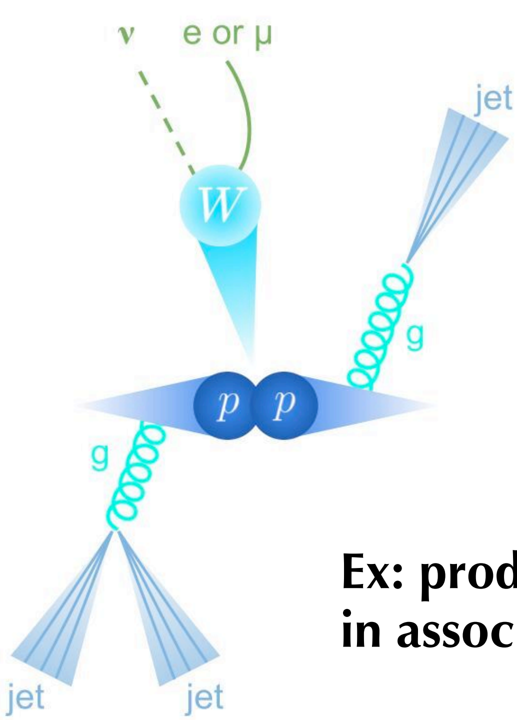


Currently applying this with Oz Aram et al. on CMS data!

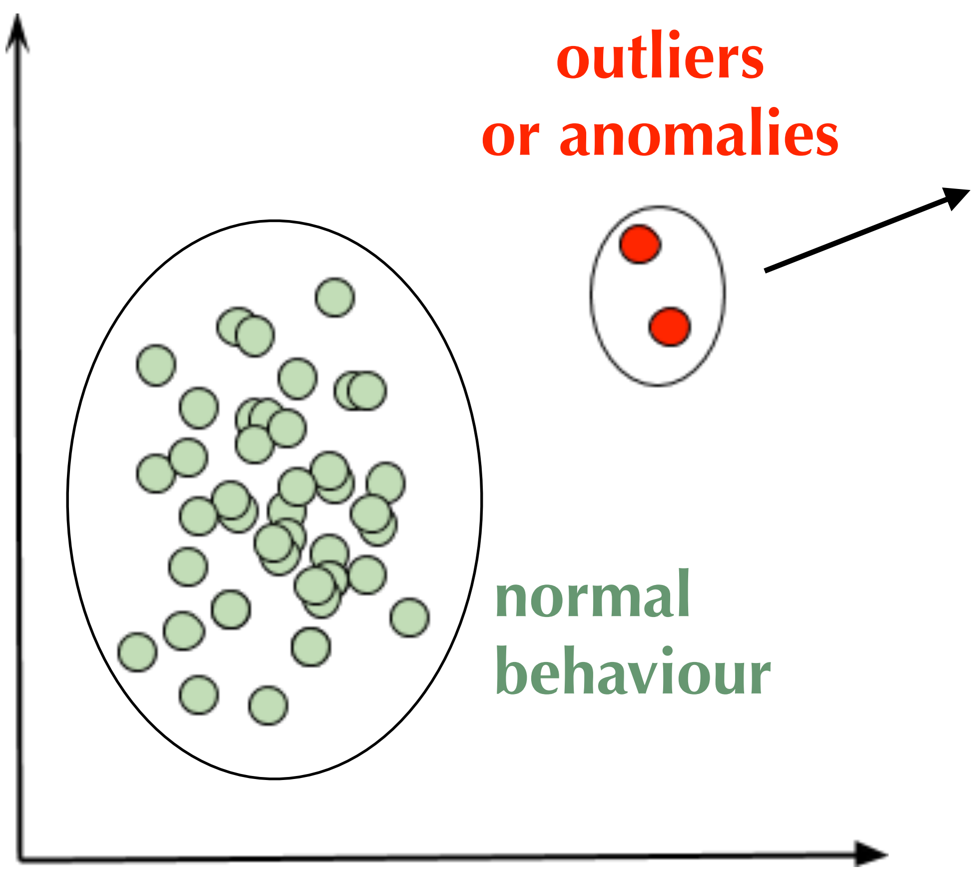
Anomaly detection in HEP

- **Model-agnostic searches for new physics at the LHC**
  - at object level — eg., jet identification
  - at event level — eg, number and properties of objects

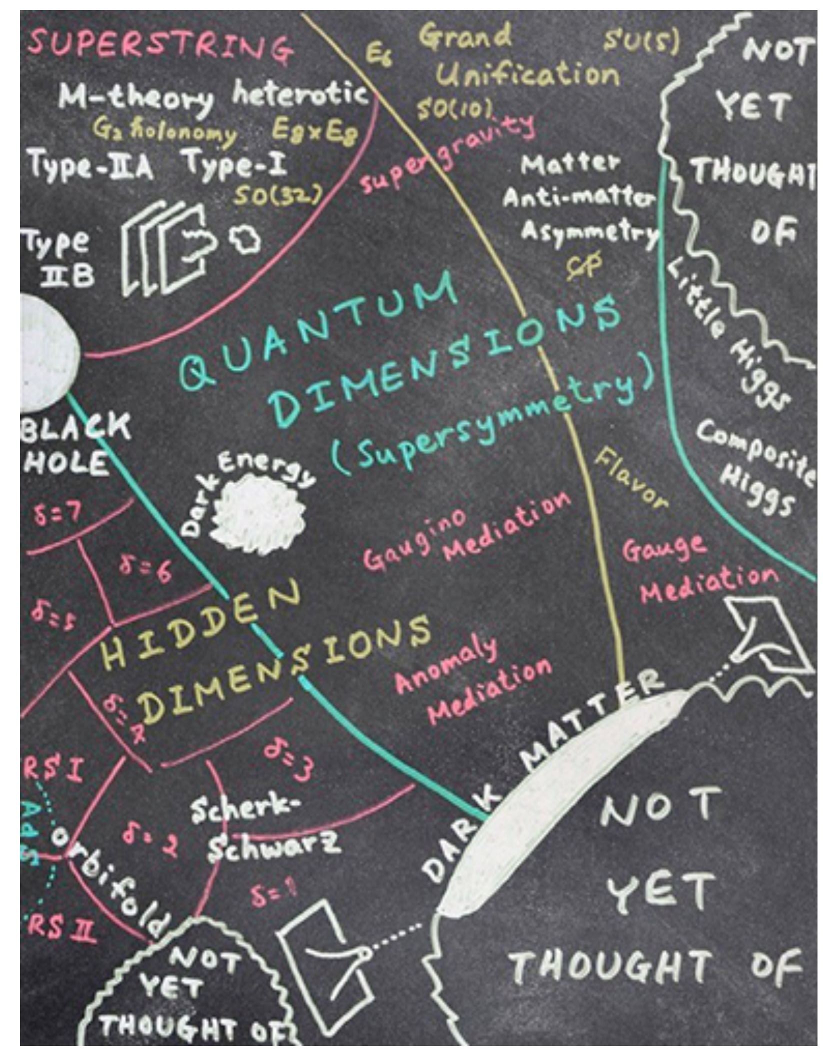
SOME STANDARD MODEL PROCESS



Ex: production of a W boson in association with 3 jets



LARGE CLASS OF POTENTIAL SIGNAL EVENTS (from unknown new physics)



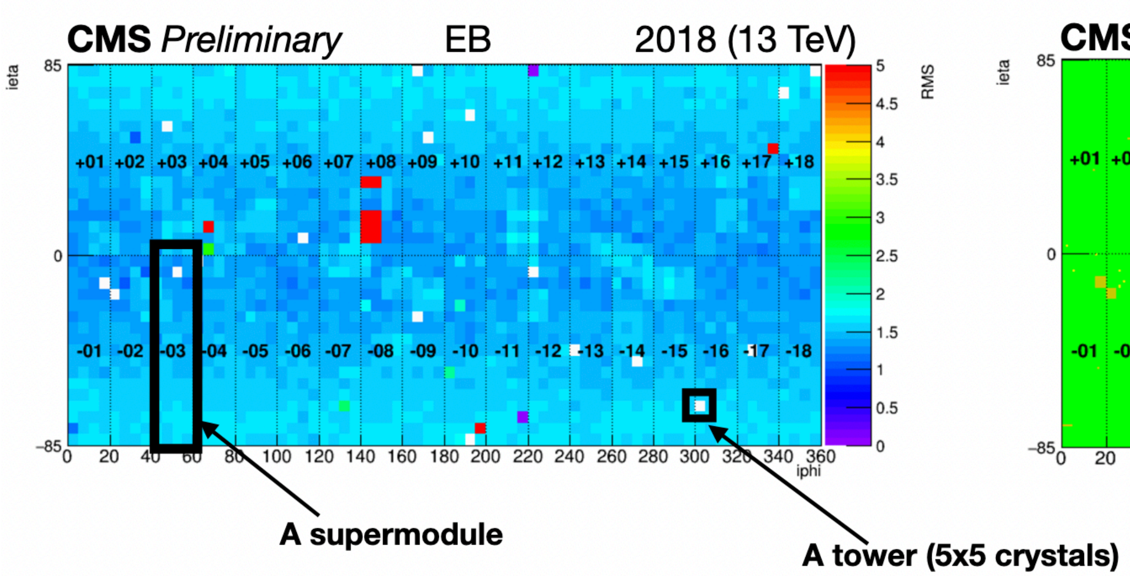
Anomaly detection in HEP

- **Model-agnostic searches for new physics at the LHC**
  - at object level — eg., jet identification
  - at event level — eg, number and properties of objects

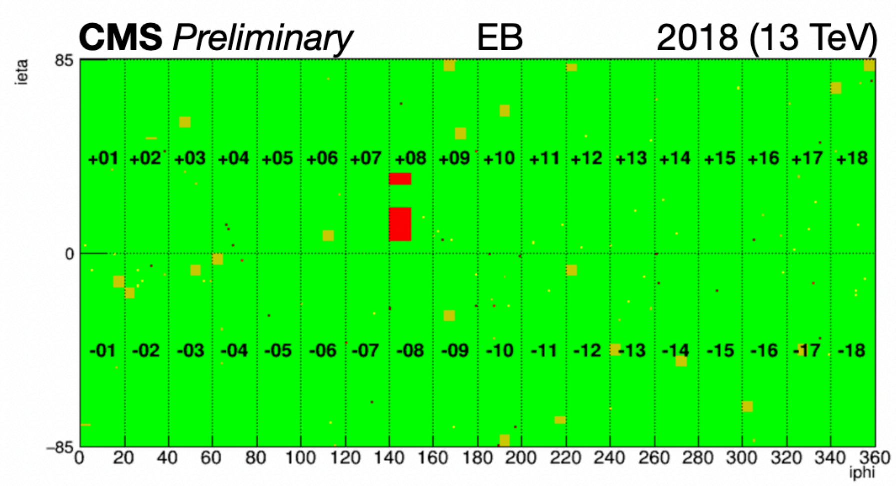
- **Detector health monitoring (online or offline)**

Example from CMS Electromagnetic Calorimeter

Occupancy histogram: collect statistics



Quality histogram: Assess quality



In the quality histogram:

GREEN = good

RED = bad

BROWN = known problem

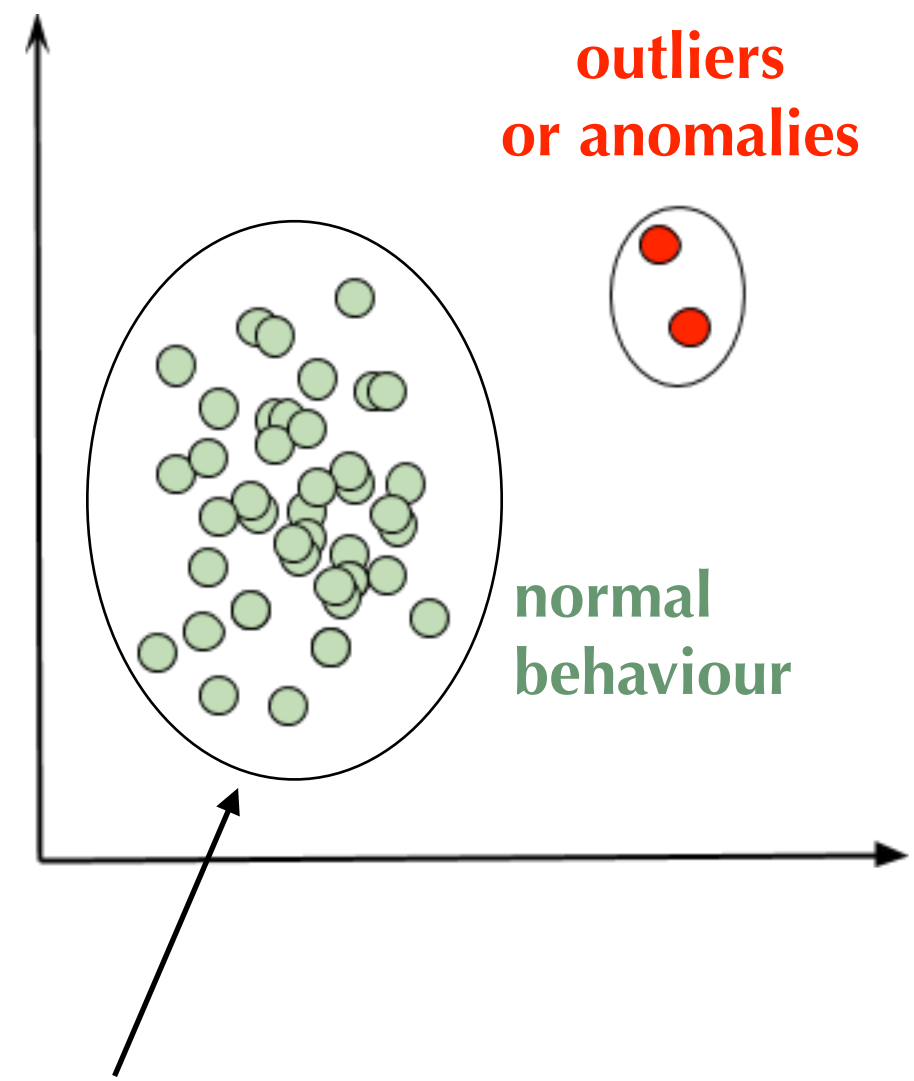
YELLOW = no data (which may or may not be problematic – depends on context).

https://indico.cern.ch/event/1106990/contributions/4998187/attachments/2535977/4364449/ML4DQM_ACAT2022_AbhiramiHarilal_final.pdf

Anomaly detection

- The process of identifying rare events in data sets which deviate significantly from the majority of the data and do not conform to "normal" behaviour

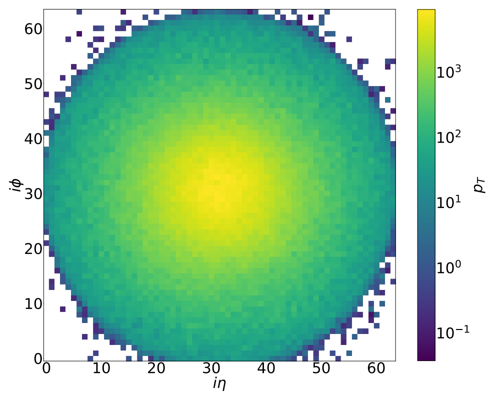
- Often applied on unlabelled data to learn the normal behaviour → *unsupervised learning*

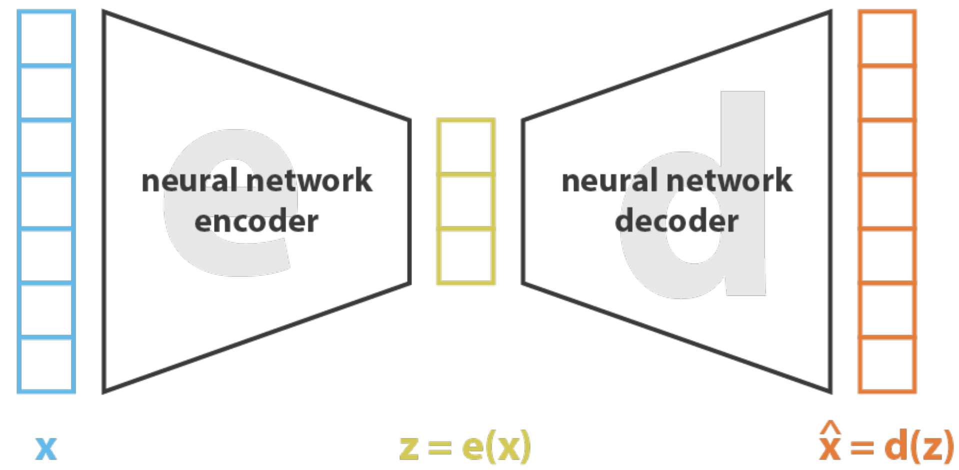


How do we learn the normal behaviour?

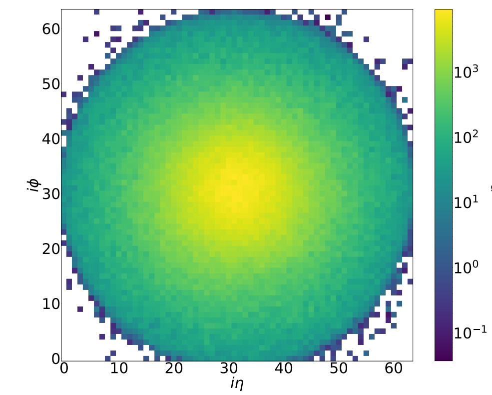
Anomaly detection with AE

e.g, jet images



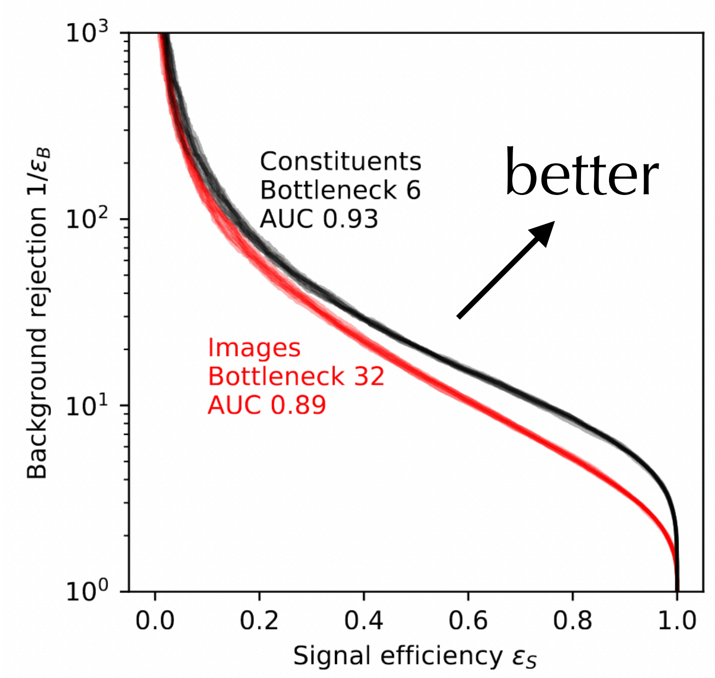


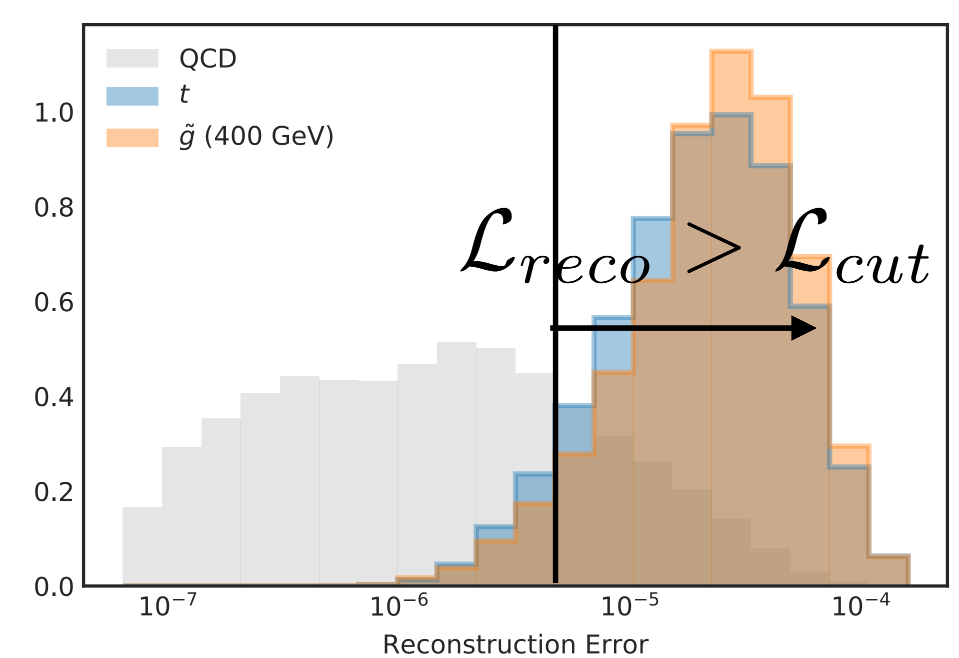
e.g, jet images



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

- One of the first applications was for signal-independent jet tagging using images
- Recently also point-cloud AE architectures were studied
  [see 2212.07347]





[*] Heimel et al.: SciPost Phys. 6, 030 (2019) , Farina et al.: Phys. Rev. D 101, 075021 (2020)

Concluding remarks

What to use? So many choices

- Once you know what you want to do…

- WHAT architecture should you use?
  - MLPs
  - CNNs
  - RNNs
  - GNNs
  - Transformers
  - ….

- The relational bias in your data will tell you!

No free lunch

- In the absence of prior knowledge, there is no a priori distinction between algorithms, no algorithm that will work best for every supervised learning problem
  - you can not say algorithm X will be better without knowing about the system
  - a model may work really well on one problem, and really poorly on another
  - this is why data scientists have to try lots of algorithms!
- However, scientists know the system — take a shortcut and use this knowledge to design algorithms
- Even with that prior knowledge, how to chose hyperparameters? How to compare different choices?
- There are some empirical heuristics that have been observed…

Practical advices

- You will likely need to try many algorithms and hyperparameters per each…
  - start with something simple!
  - use more complex algorithms along with your own learning curve
  - use cross validation to check for overcomplexity / overtraining
- For all the above… you first need to define your metrics!
- Check the literature
  - if you can cast your (HEP) problem as something in the ML / data science domain, there may be guidance on how to proceed
- Debug when needed
  - check bias and variance
  - check convergence
  - where is the error in my algorithm coming from

Potential fixes

- Fixes to try:
  - Get more training data — Fixes high variance
  - Try smaller feature set size — Fixes high variance
  - Try larger feature set size — Fixes high bias
  - Try different features — Fixes high bias
- Did the training converge?
  - Run gradient descent a few more iterations — Fixes optimization algorithm
    - or adjust learning rate
  - Try different optimization algorithm — Fixes optimization algorithm
- Is it the correct model / objective for the problem?
  - Try different regularization parameter value — Fixes optimization objective
  - Try different model — Fixes optimization objective
- You will often need to come up with your own diagnostics to understand what is happening to your algorithm

[Ng]

From M. Kagan lecture