Generative Machine Learning in HEP: Simulation and beyond

inverted CSC
CERN
School of Computing

Francesco Vaselli
francesco.vaselli@cern.ch

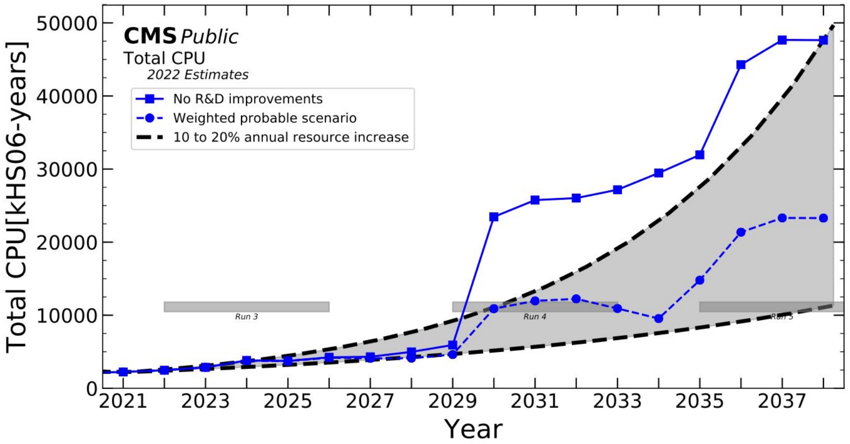
INFN
Istituto Nazionale di Fisica Nucleare
SCUOLA NORMALE SUPERIORE

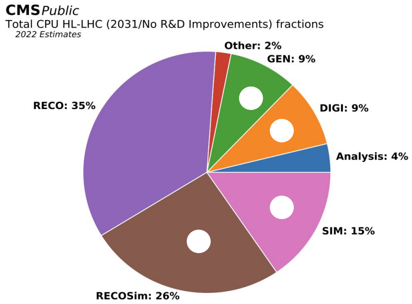
Simulation is a fundamental part of HEP

We need simulations to:

- notice unusual variations and new, previously unseen phenomena
- measuring processes known for being extremely rare

At the same time, simulations are computationally expensive, and the need is expected to increase!





Our current, trustworthy Simulation Frameworks

Fantastic traction on the ground
(adherence to data)

Reliable, *can* and *will* do the heavy lifting for your analysis

You can take lots of luggage
(many details)

Steady and accurate

Can we go faster if we renounce some comforts?

Extremely fast

Great traction on the track
(adherence to data)

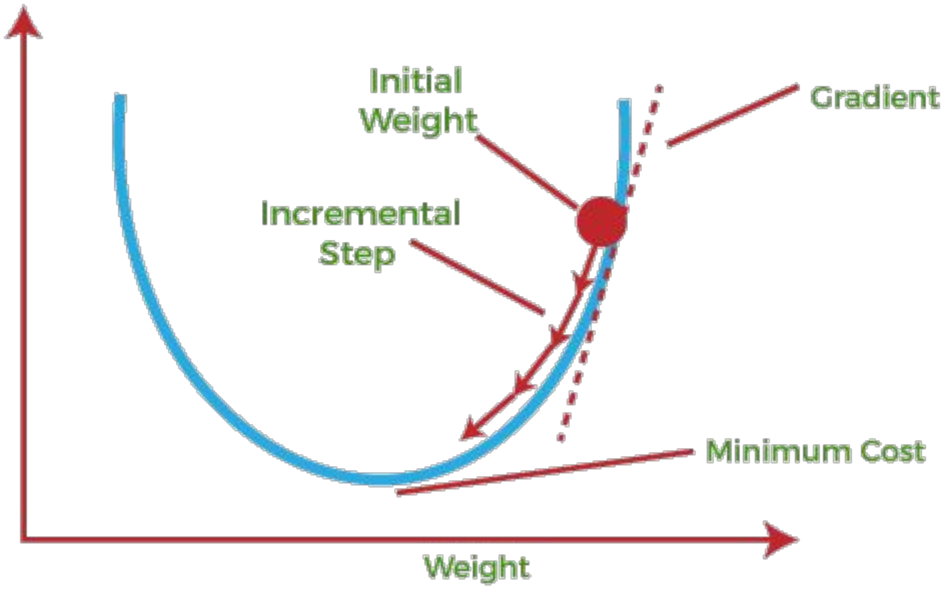
Can’t carry many things
with you!
(less details)

Machine Learning can be one way to do that!

Use some network (ensemble of learnable weights)

Find good *Loss function* describing how far is the network output from the optimal solution

Update the weights to minimize the Loss!



Outline

Core concepts

Generative Adversarial Networks

Variational Autoencoders

Normalizing Flows

Diffusion Models

Applications to HEP

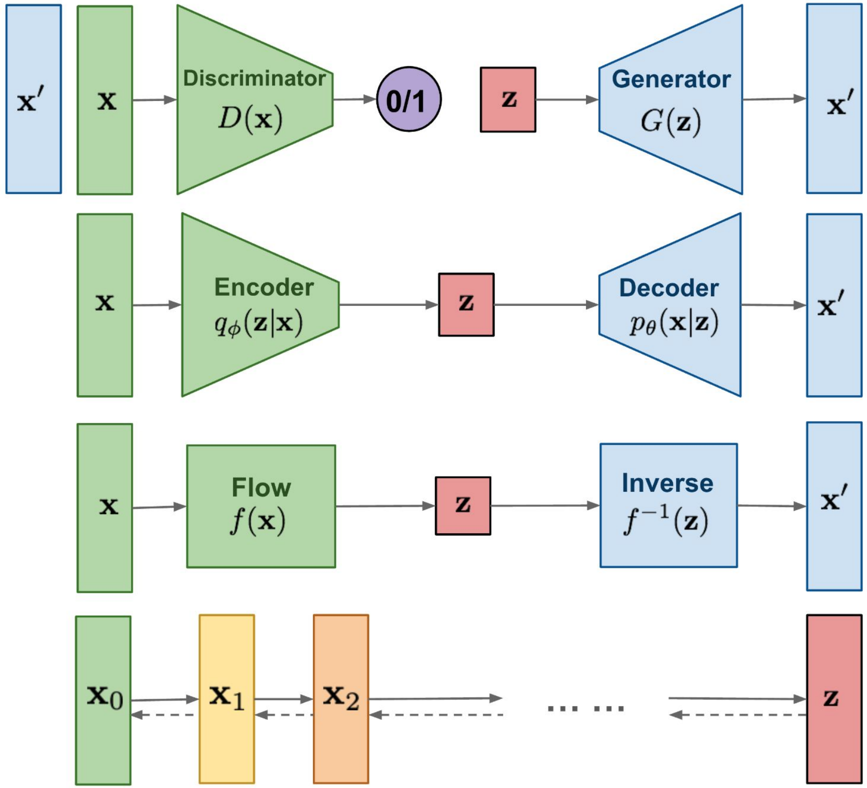
Generative Models in one slide

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

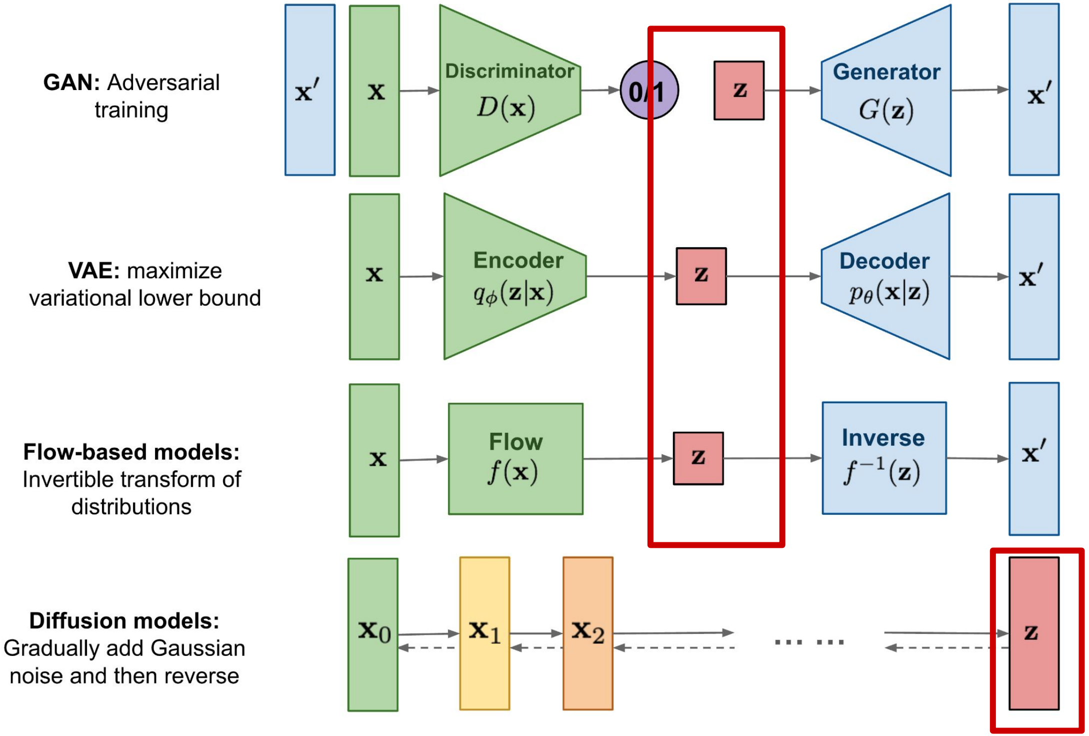
Diffusion models: Gradually add Gaussian noise and then reverse



Scheme from Lilian Weng's [blog post]

Common building blocks!

Latent space



Common building blocks!

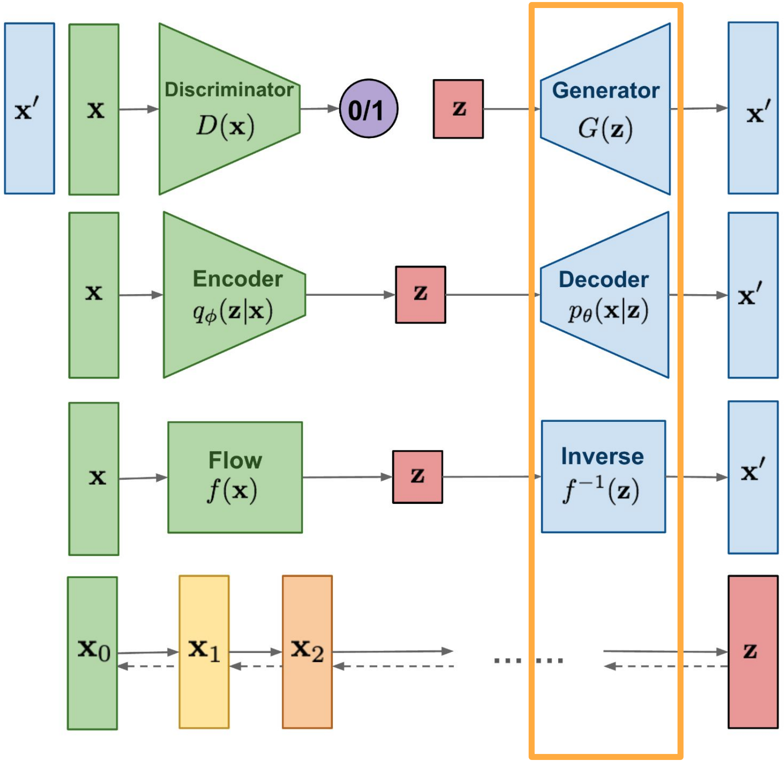
Generative Models

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

Diffusion models: Gradually add Gaussian noise and then reverse



Common building blocks!

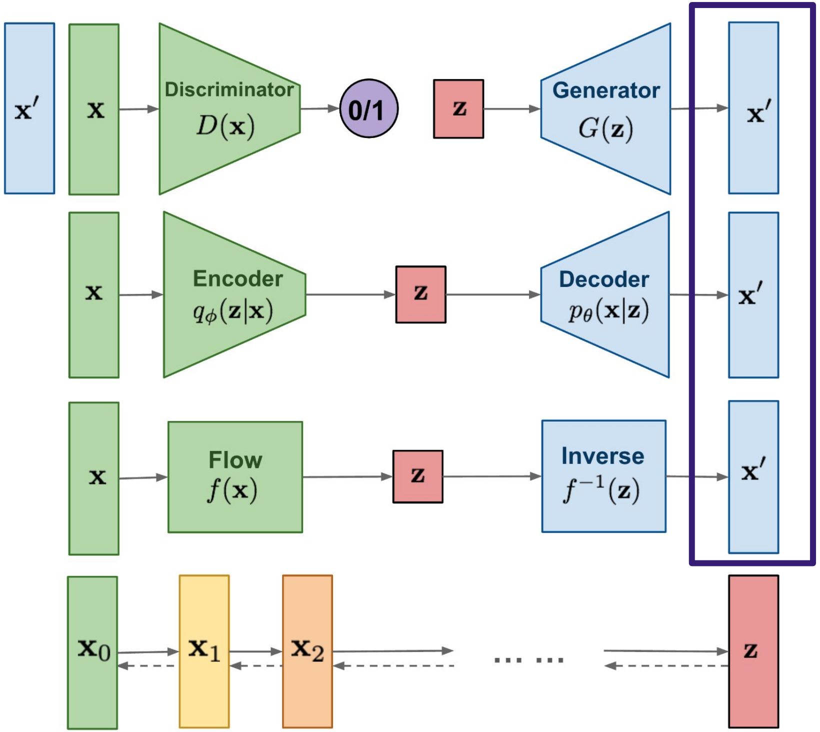
Synthetic data

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

Diffusion models: Gradually add Gaussian noise and then reverse



A common strategy for defining the loss

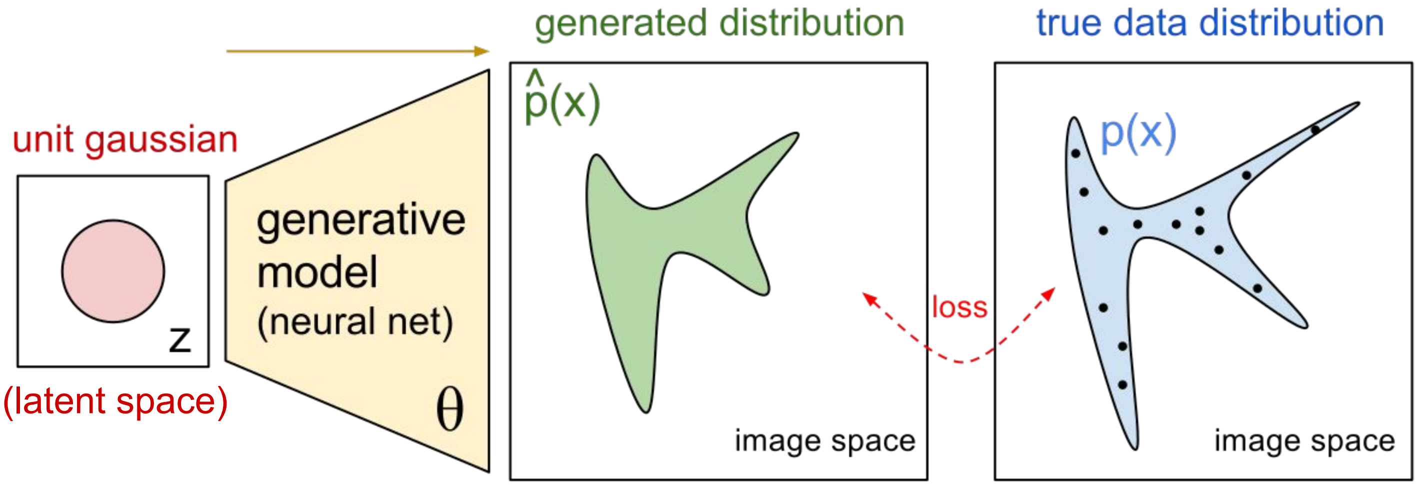
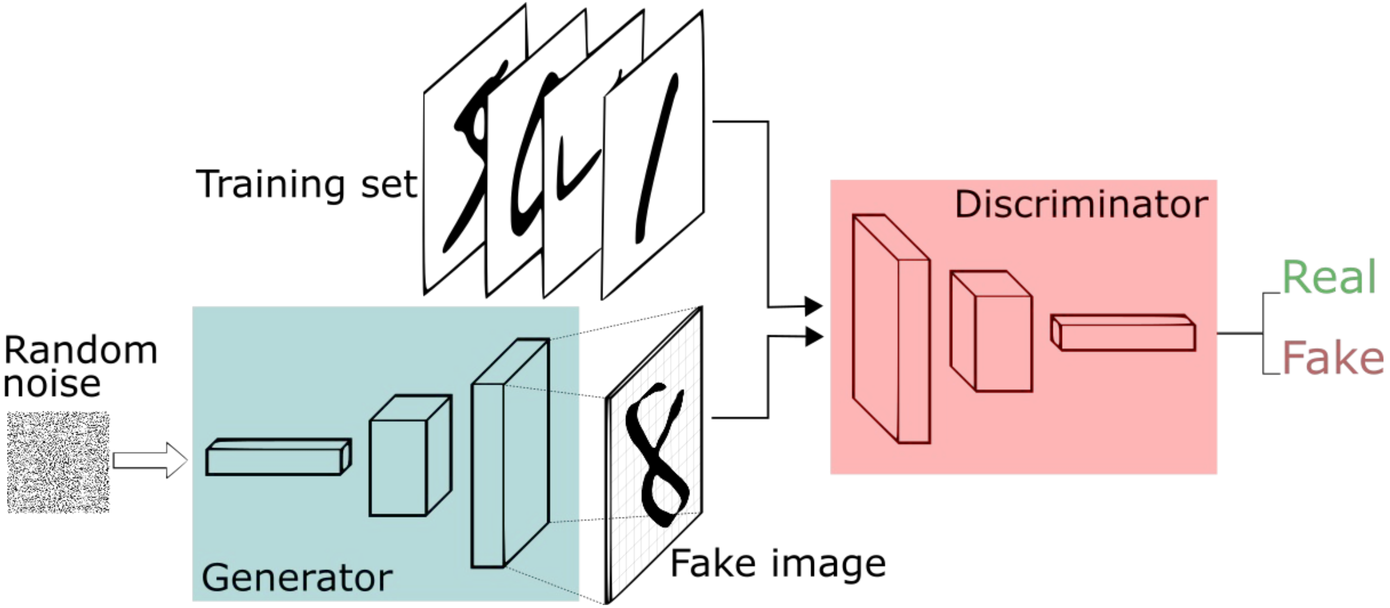


Figure from the OpenAI blog post

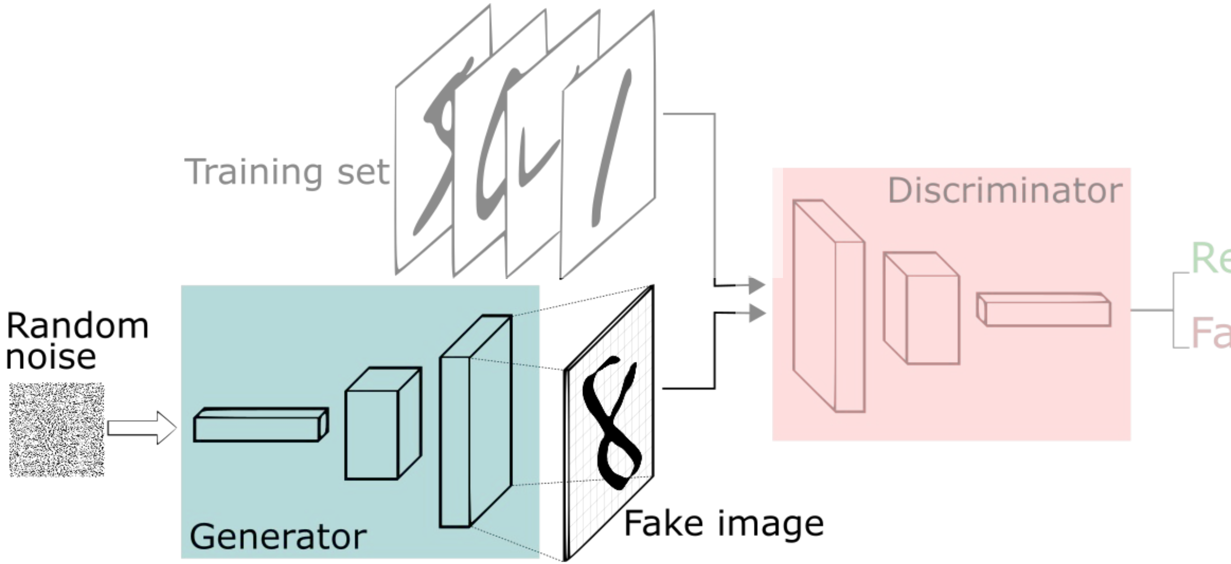
How a good idea can change a whole field

Generative Adversarial Networks: game-theory inspired training dynamic



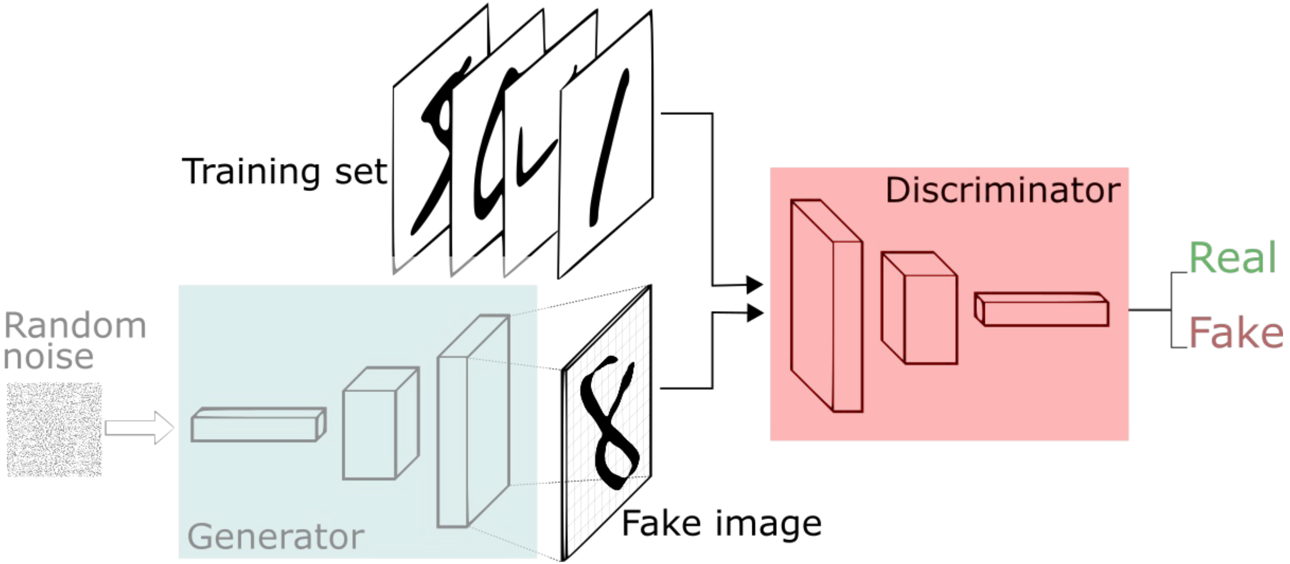
Generative Adversarial Networks: game-theory inspired training dynamic

Generator G:
outputs synthetic samples given noise as input (brings in stochasticity)

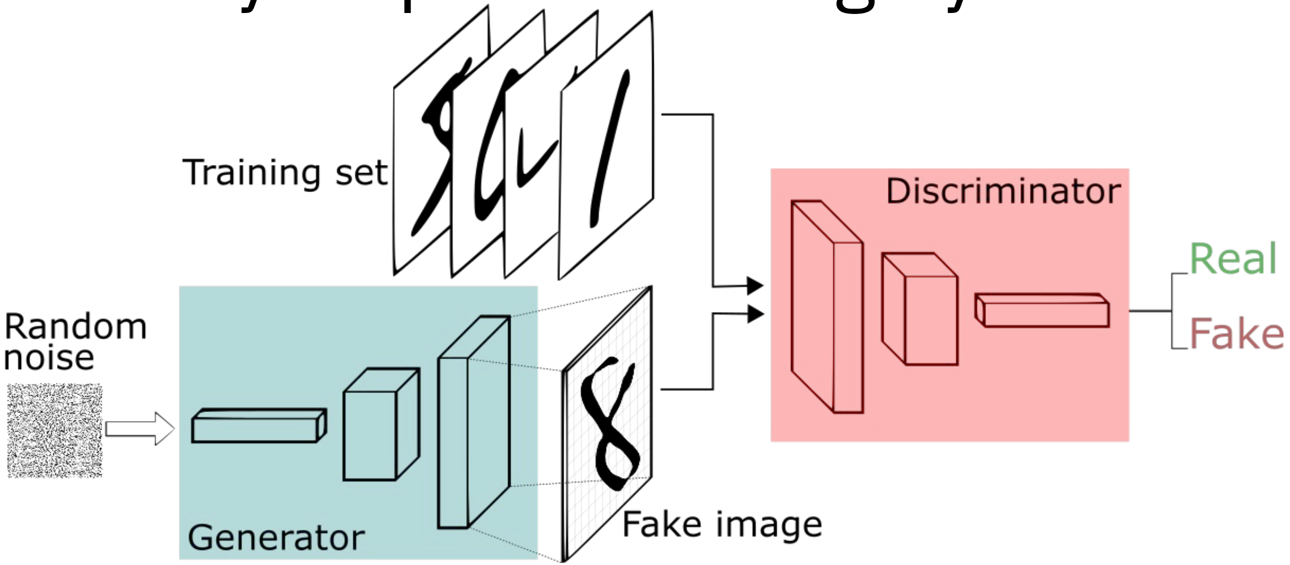


Generative Adversarial Networks: game-theory inspired training dynamic

Discriminator D: estimates the probability of a given sample coming from the real dataset

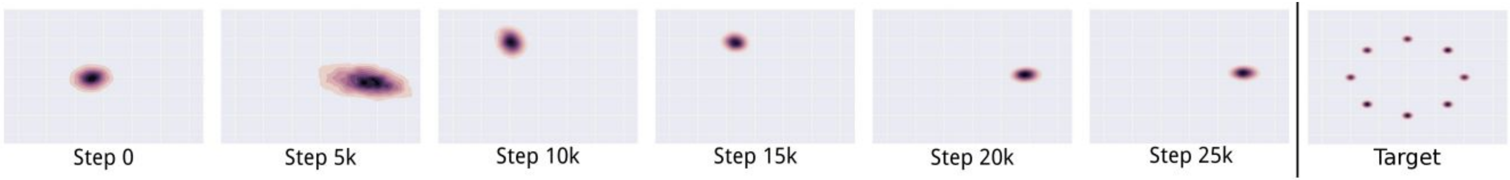


Generative Adversarial Networks: game-theory inspired training dynamic



$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Unstable training dynamics:
Mode collapse



Training instability (mode collapse)

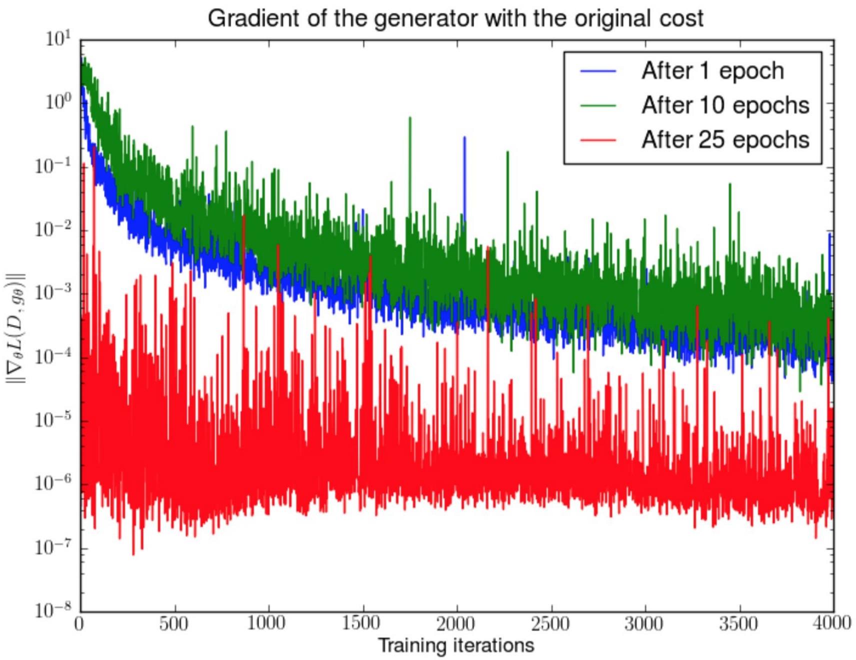
Example of mode collapse taken from [arXiv:1611.02163]

Unstable training dynamics:
vanishing gradients

When the discriminator is perfect:

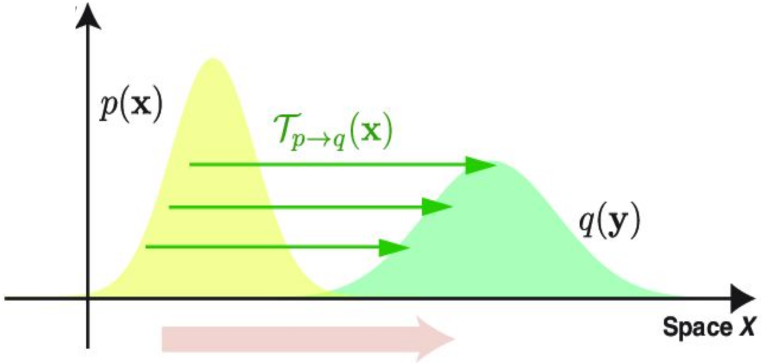
- D(x_True)->1
- D(x_Gen)->0

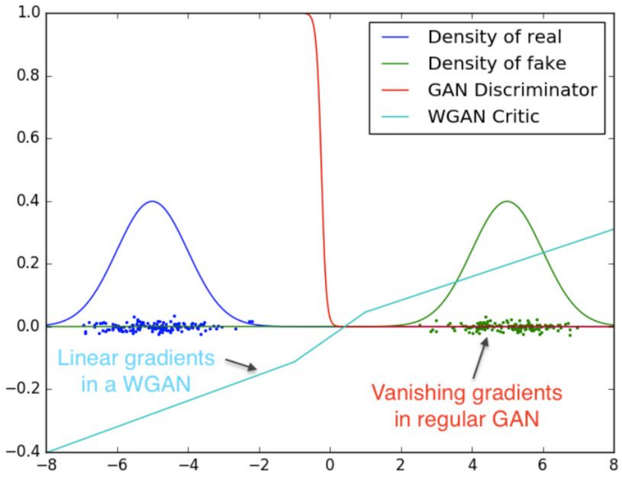
L(x)->0!!!
No improvements



Example of vanishing gradients from https://arxiv.org/pdf/1701.04862

Wasserstein GAN as an improvement to training





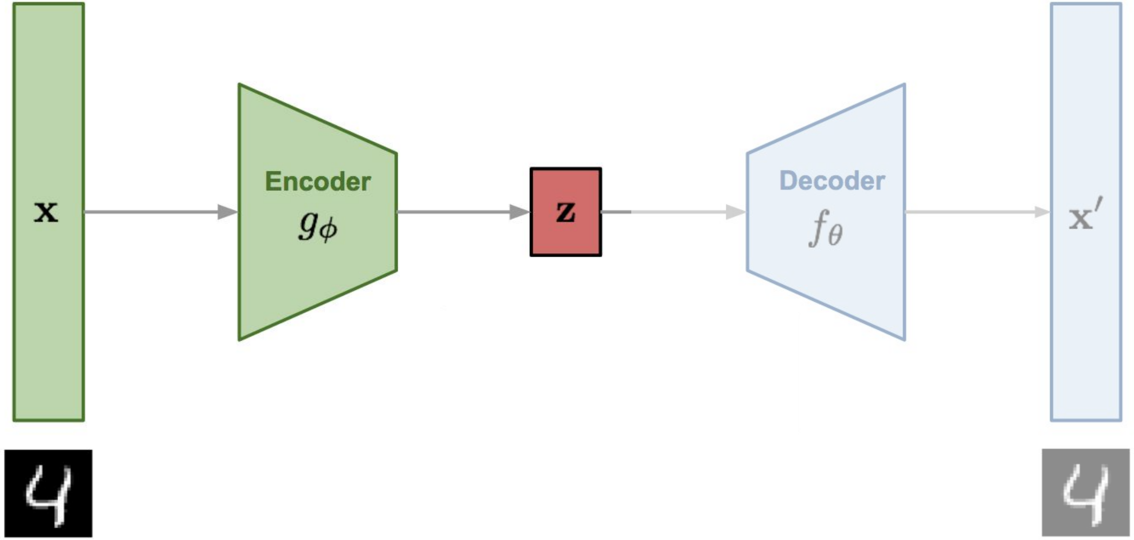
Loss function now measures the *Wasserstein distance*

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

Figure taken from
arXiv:1701.07875

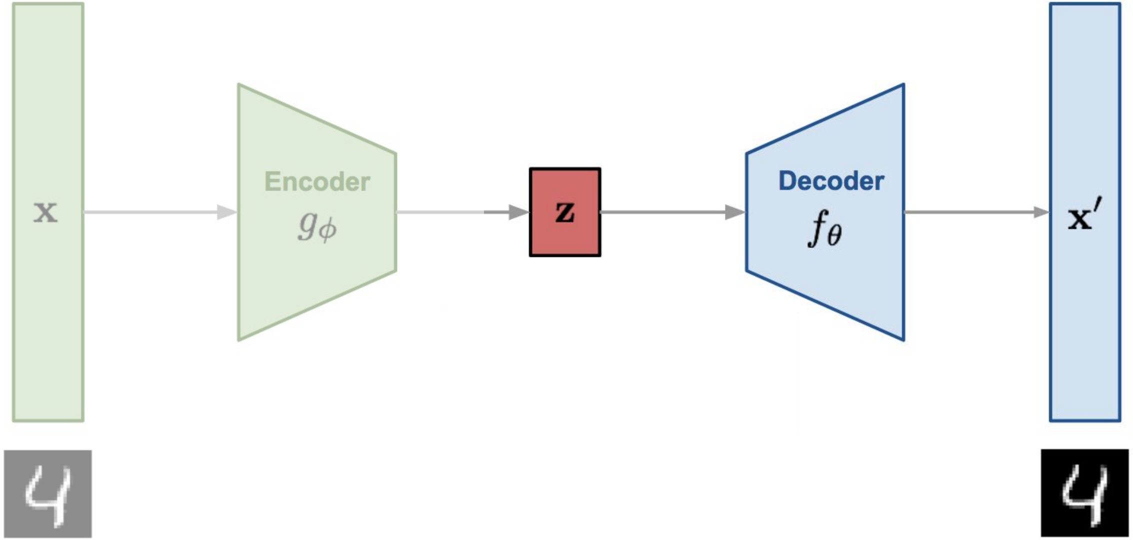
From Adversarial to Autoencoding

Autoencoders can be a great starting point



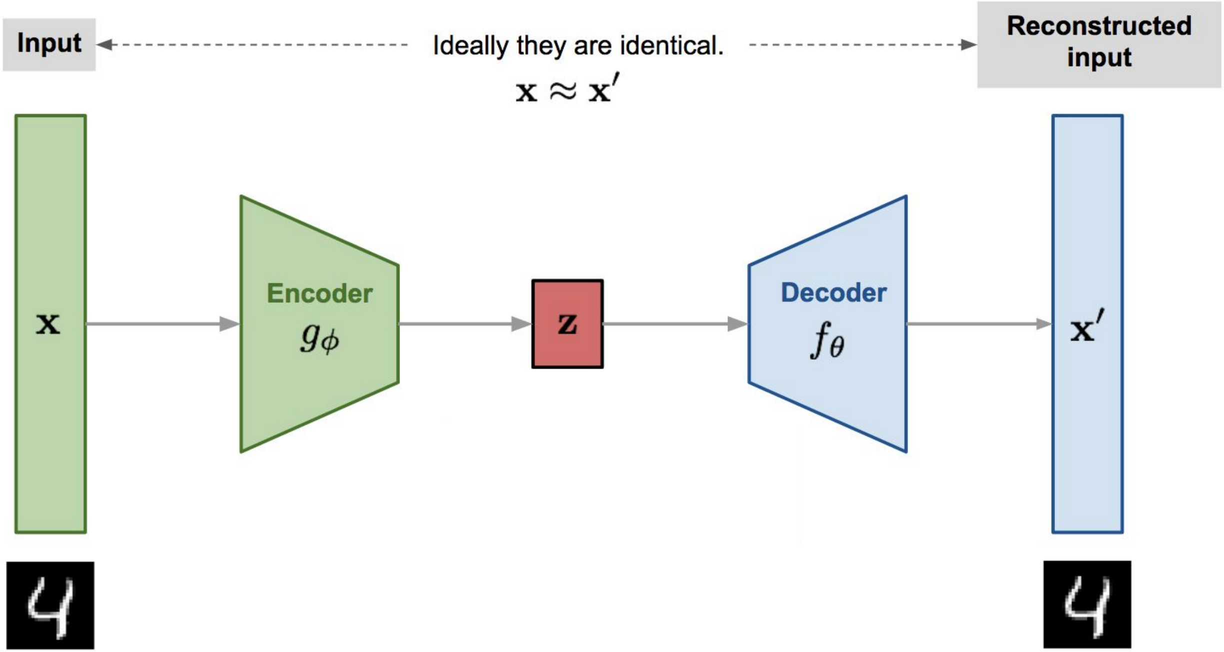
Encoder network: Encodes **x** into low dimensional representation **z**!

Autoencoders can be a great starting point



Decoder network: Decodes **z** into original representation **x**!

Autoencoders can be a great starting point



Reconstruction Loss $L_{\text{AE}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$

Why can't I use this for generation?

An irregular latent space is useless!

from
Joseph Rocca’s
blog post

How to regularize the latent space

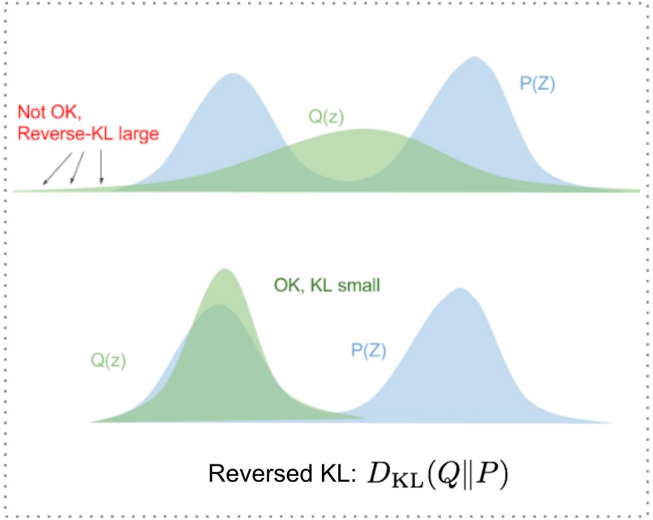
AE	**x** $\rightarrow$	**z = E(x)** $\rightarrow$	**x' = D(z)**
Variational AE	**x** $\rightarrow$	**z ~ p(z|x)** $\rightarrow$	**x' ~ p(x|z)**
		Probabilistic Encoder	Probabilistic Decoder

We need to add a term to the loss to make it a probabilistic model

Use Kullback-Leibler divergence to quantify the distance between NN and “Posterior”

Measures how much information is lost if the distribution Y is used to represent X.

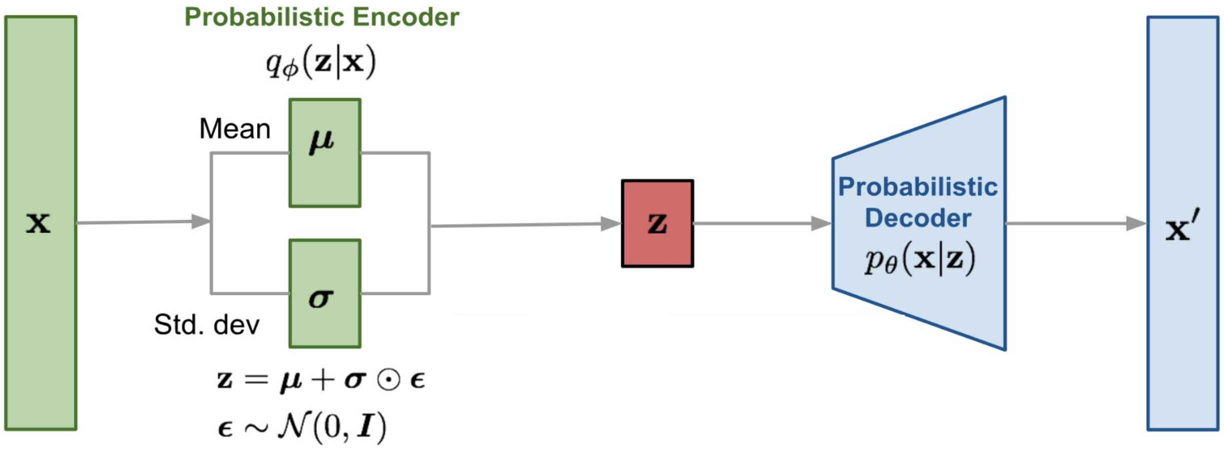
Total loss = RECO + VAE
= Evidence Lower Bound (ELBO)



Finally, a *variational* autoencoder!

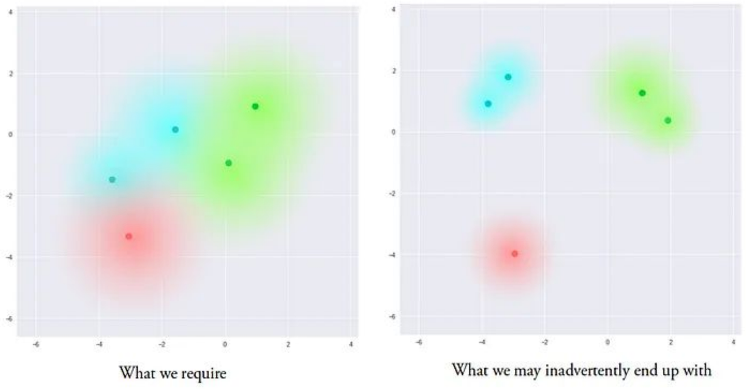
Good, flexible Latent Space Representation

Total loss = RECO + VAE = Evidence Lower Bound (ELBO)



Known issues affecting latent space and samples

- Mode Collapse
- Blurriness in Outputs
- Complex Training
- Limited Expressiveness



Images from the Enoch Kan's [blog post]

Wait, I really liked the idea of learning a *pdf*!

Why limit ourselves to just one sample?

GAN

FLOW

from "Why I Stopped Using GAN — ECCV 2020"

The basic idea: change of variables

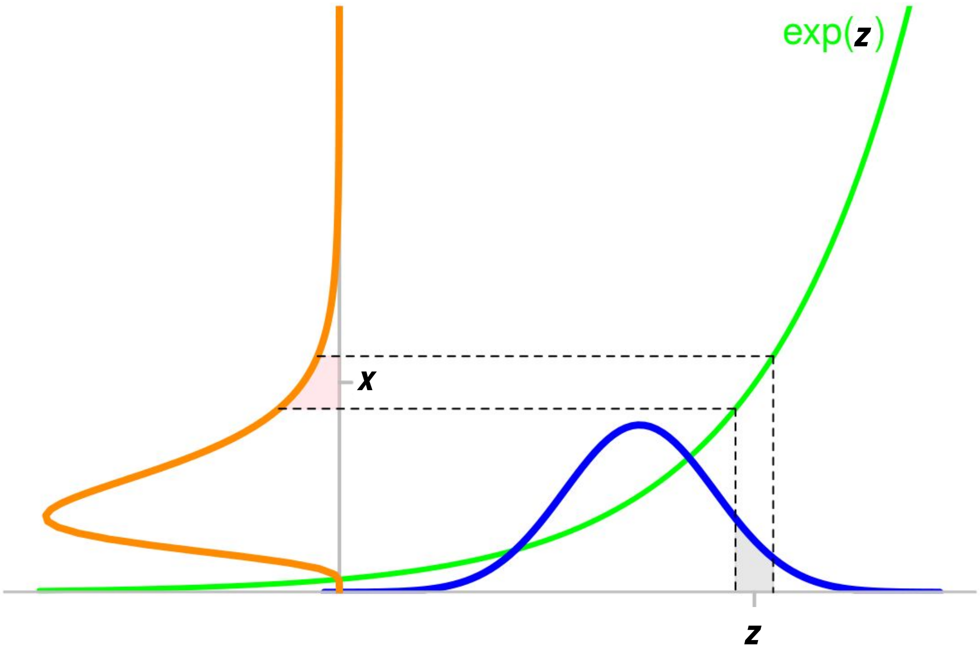
We define a transform f such that:

$$\mathbf{x} = f(\mathbf{z})$$
$$\mathbf{z} = f^{-1}(\mathbf{x})$$

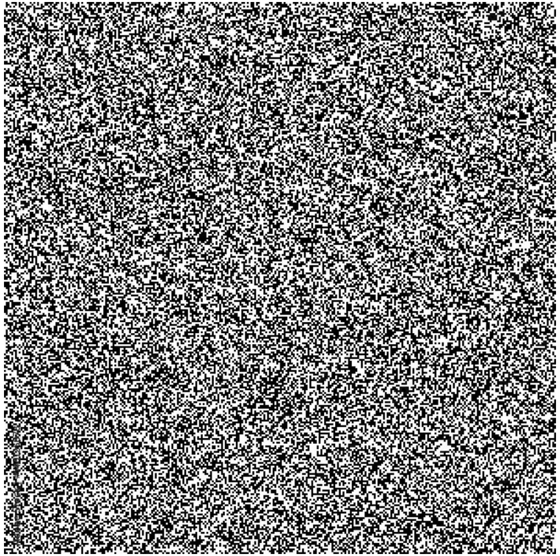
The two pdfs are related:

$$p_x(\mathbf{x})d\mathbf{x} = p_z(\mathbf{z})d\mathbf{z}$$

$$p_x(\mathbf{x}) = p_z(\boxed{f^{-1}(\mathbf{x})})\det\boxed{\left|\frac{d\mathbf{z}}{d\mathbf{x}}\right|}$$



The basic idea: image generation



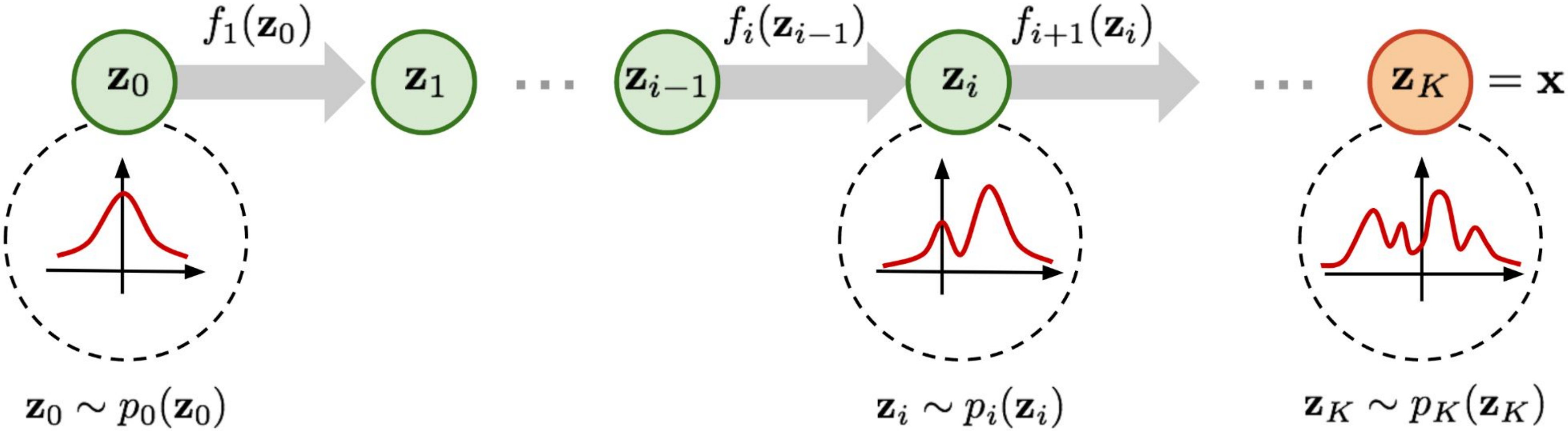
$$f(\mathbf{z}) \longrightarrow$$

$$\longleftarrow f^{-1}(\mathbf{x})$$

$p_{\mathbf{z}}(\mathbf{z})$

$p_{\mathbf{x}}(\mathbf{x})$

The basic idea: complex transforms



from Lilian Weng

We need just a few building blocks!

Task

Learn the ***f(z)*** to send $p_{\mathbf{z}}(\mathbf{z})$ into the (***unknown***) data distribution $p_{\mathbf{x}}(\mathbf{x})$

Pieces

- Basic distribution $p_{\mathbf{z}}(\mathbf{z})$, typically Gaussian
- Function called flow ***f(z) invertible*** and ***differentiable***, with ***tractable jacobian***

The usage is straightforward

Density evaluation

$$p_x(\mathbf{x}) = p_z(f^{-1}(\mathbf{x}))\,\det\left|\frac{d\mathbf{z}}{d\mathbf{x}}\right|$$

Sampling new data

- Sample from $p_{\mathbf{z}}(\mathbf{z})$ (Gaussian, trivial)
- Compute $\mathbf{x} = f(\mathbf{z})$ (fast)

The loss is explained from the change of variables!

$$p_x(\mathbf{x}) = p_z(f^{-1}(\mathbf{x})) \det \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right|$$

Invertible transform

Jacobian for Volume Correction

$$\log(p_x(x)) = \log(p_z(f^{-1}(\mathbf{x}))) + \log\left(\det \mathbb{J}_{f^{-1}}(\mathbf{x})\right)$$

$$\mathcal{L}(\phi) = -\mathbb{E}_{p_x^*(\mathbf{x})}[\log(p_z(f^{-1}(\mathbf{x};\ \phi))) + \log\left(\det \mathbb{J}_{f^{-1}}(\mathbf{x};\ \phi)\right)]$$

where ϕ are the parameters of *f(z)*

Flows building blocks: *transformations*

How do we transform the variables?
Various ways to do it (as long as the transformation is invertible!)

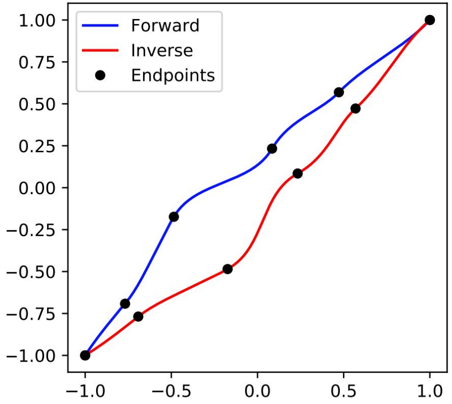
Each model is made up of multiple transformation blocks

This gives us an expressive final transformation with good correlations
between variables

Affine:

$$\tau(\mathrm{z}_i; \boldsymbol{h}_i) = \alpha_i \mathrm{z}_i + \beta_i$$

Splines:



Normalizing Flows are *powerful* GMs!

Efficient to sample from $p_{\mathbf{x}}(\mathbf{x})$

Efficient to evaluate $p_{\mathbf{x}}(\mathbf{x})$

Highly expressive

Useful latent representation

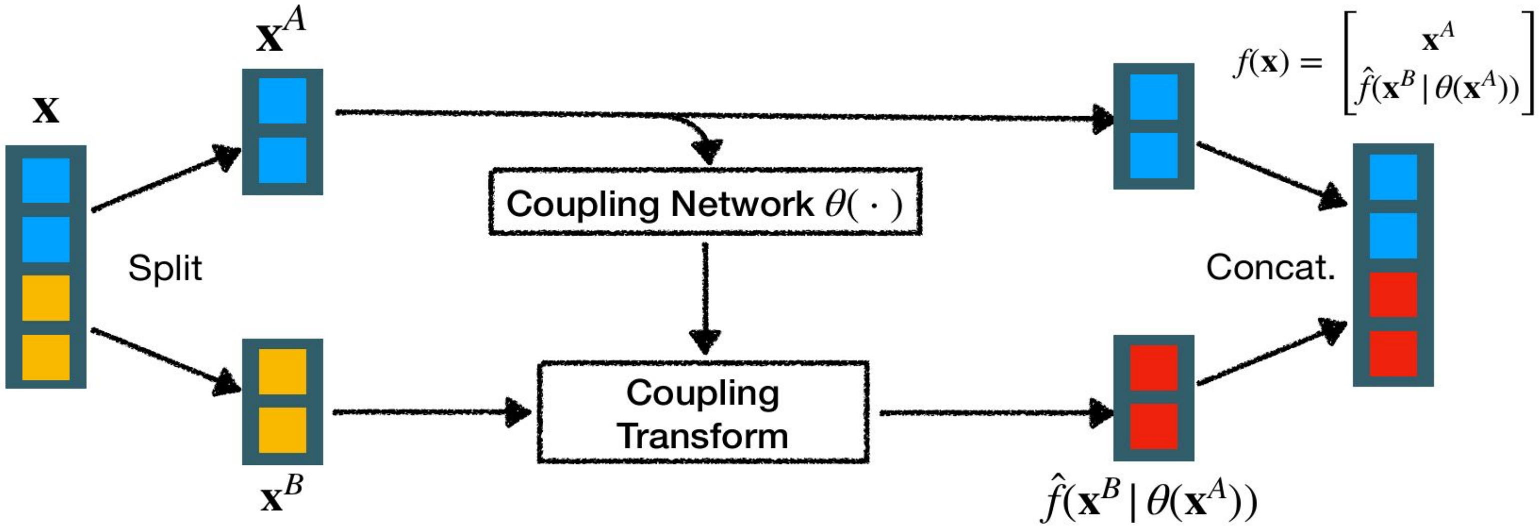
Straightforward to train

Normalizing Flows are *flawed* GMs!

Computation of the Jacobian is hard

Not defined to work for *discrete variables*!

Coupling layers for reducing jacobian complexity



$$\mathbb{J}_f(\mathbf{x};\,\phi) = \begin{pmatrix} \frac{\partial \vec{x}_{1:d}}{\partial \vec{z}_{1:d}} & \frac{\partial \vec{x}_{1:d}}{\partial \vec{z}_{d+1:D}} \\ \frac{\partial \vec{x}_{d+1:D}}{\partial \vec{z}_{1:d}} & \frac{\partial \vec{x}_{d+1:D}}{\partial \vec{z}_{d+1:D}} \end{pmatrix} = \begin{pmatrix} \mathbb{I} & 0 \\ A & \mathbb{J}^* \end{pmatrix}$$

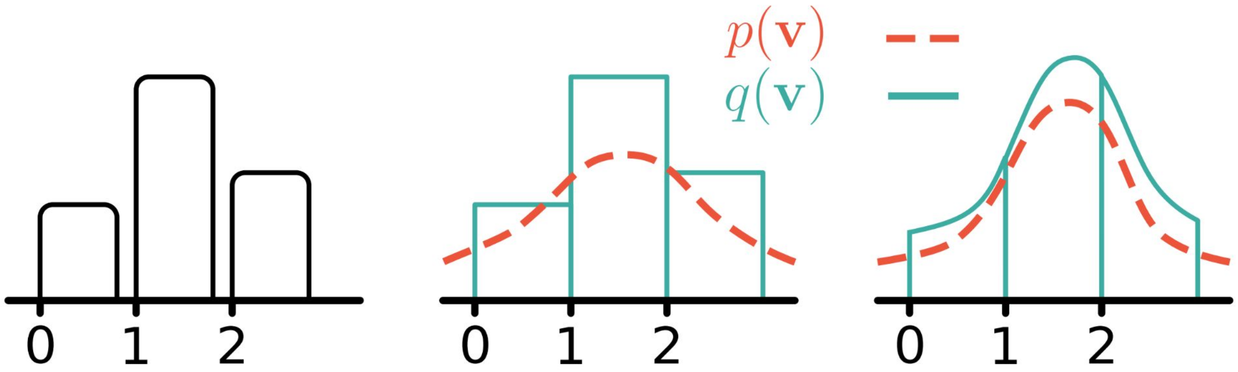
the Jacobian becomes triangular!

from Jason Yu

Dequantization can be used on discrete variables

Apply a Gaussian smearing

From discrete data to continuous!



(a) $P_{\text{data}}(\mathbf{x})$ (b) uniform $q(\mathbf{v}|\mathbf{x})$ (c) flexible $q(\mathbf{v}|\mathbf{x})$

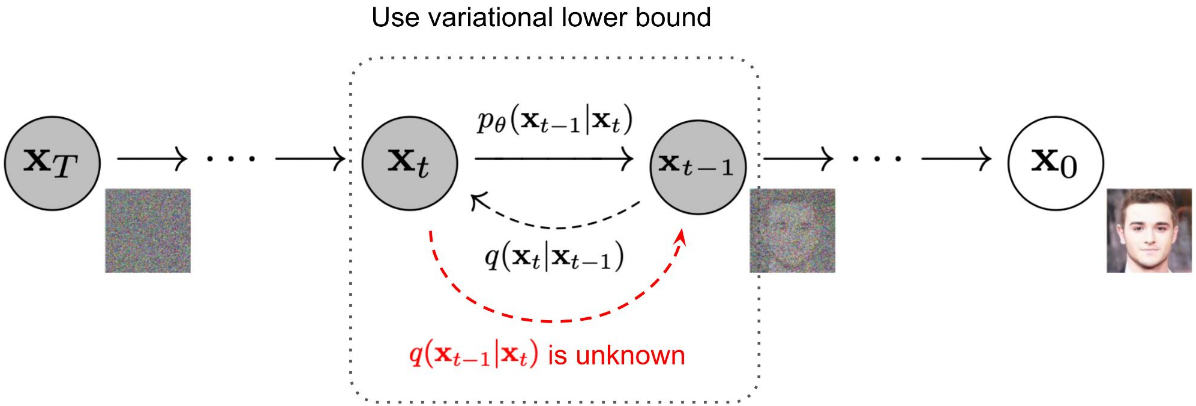
From arXiv:2001.11235

Increasing Realism step-by-step

A Markov-chain approach to generation!

Diffusion Models define a Markov chain that slowly adds random noise to data and then learn to reverse

We need to learn a model to approximate these conditional probabilities in order to run the reverse diffusion process.



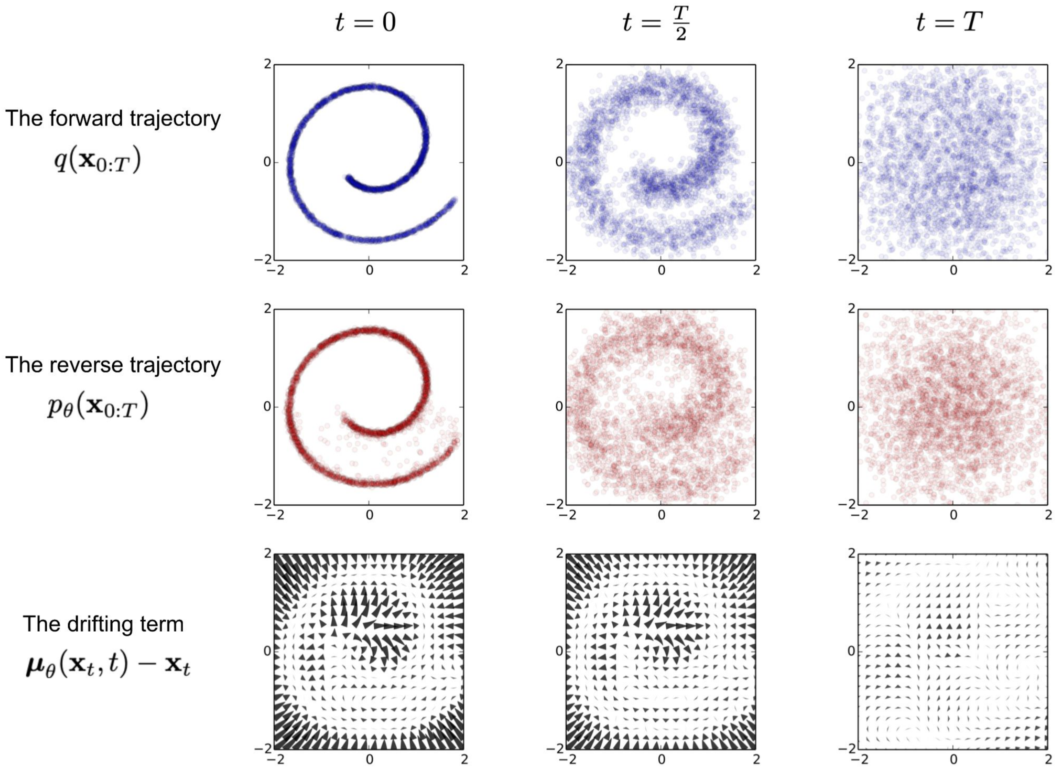
$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Output of NN

Model in action, pros and cons

Pros: Diffusion models are both *analytically tractable* and *flexible*

Cons: Diffusion models rely on a long Markov chain, *expensive* in terms of time and compute



How do I do *that*??!

An illustration of an avocado sitting in a therapist's chair, saying 'I just feel so empty inside' with a pit-sized hole in its centre. The therapist, a spoon, scribble notes

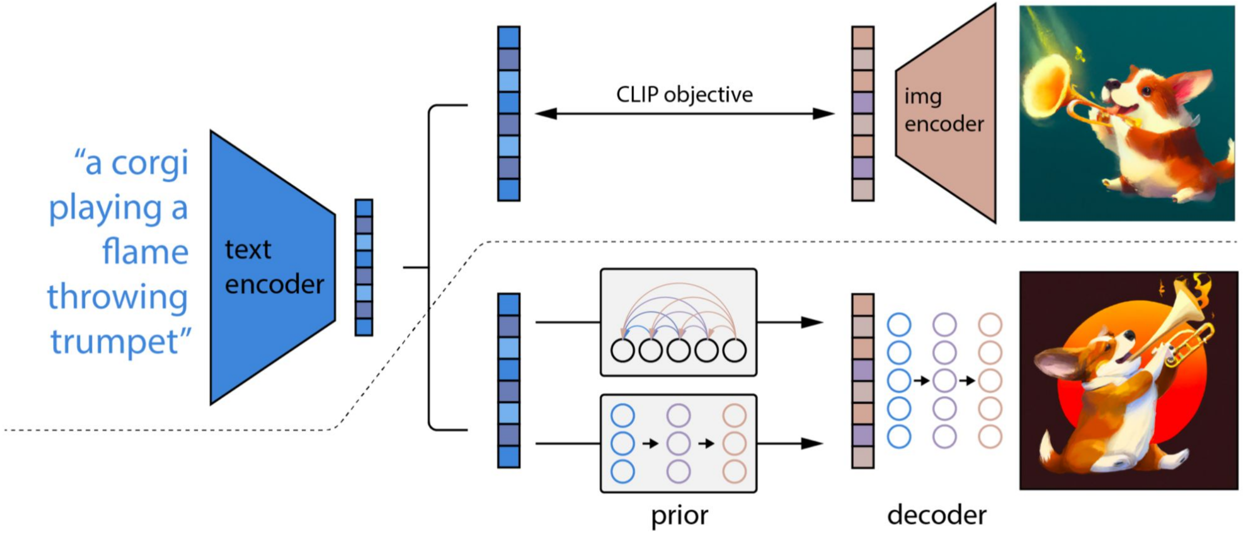
⟶ DALL·E 3 ⟶

Just so you know: text conditioning

Many approaches

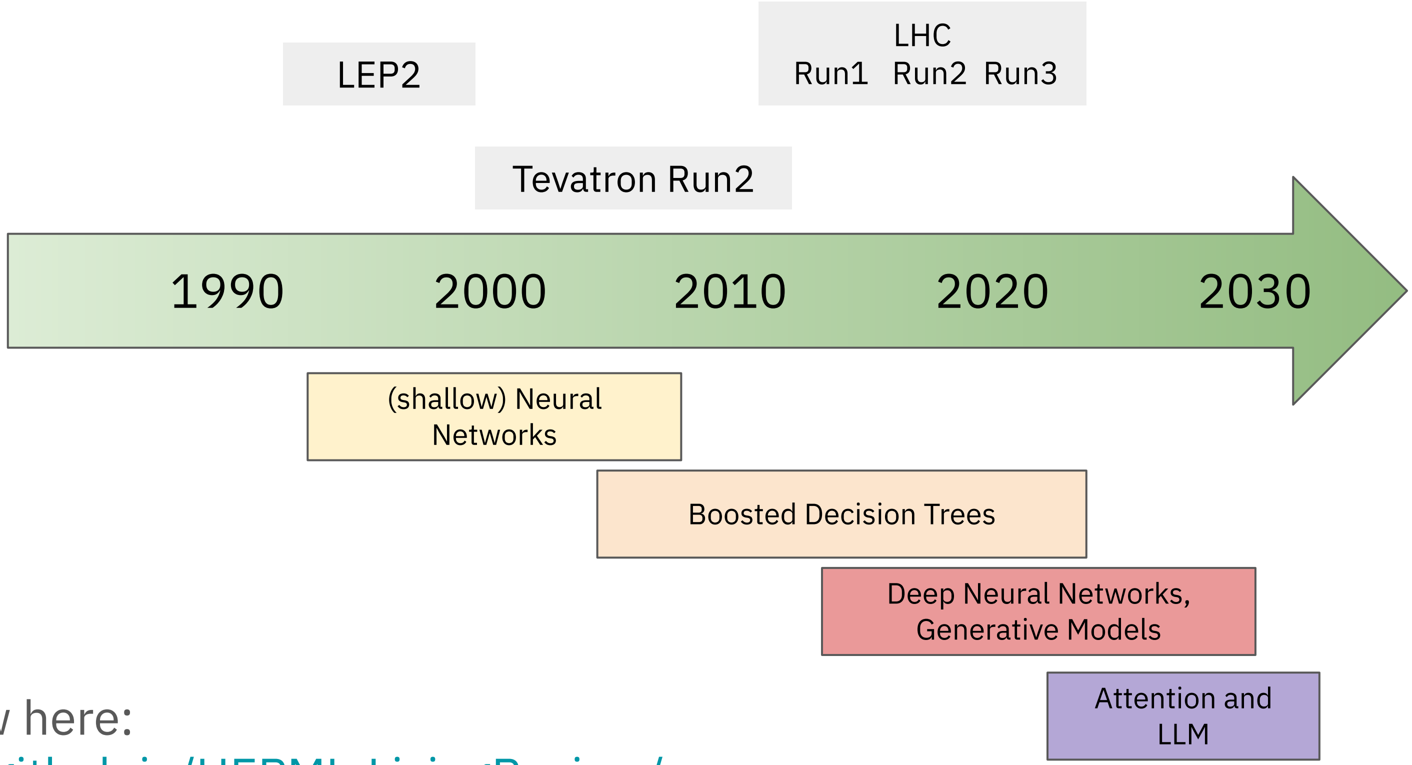
A CLIP (*Contrastive Language–Image Pre-training*) model learns to match a latent representation of the image given the associated label

The latent input is given to a diffusion decoder



Ramesh et al. 2022

Applications: data generation and beyond!

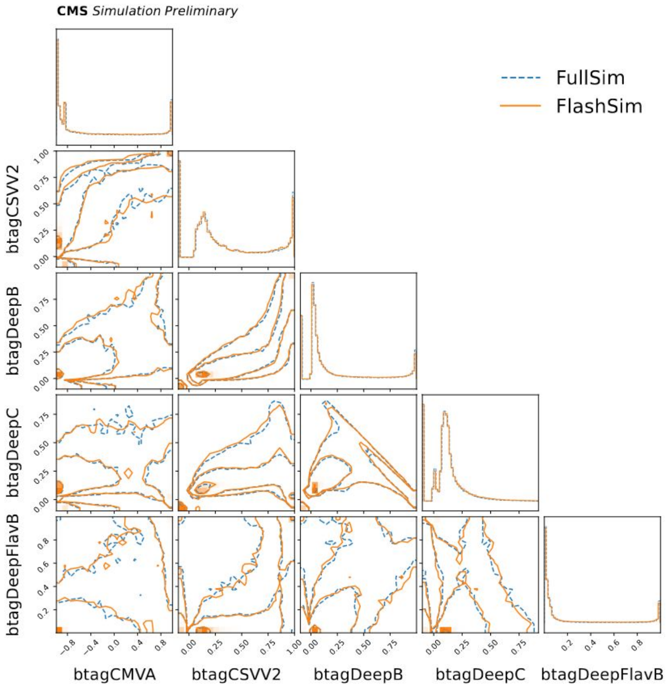


Amazing review here:
https://iml-wg.github.io/HEPML-LivingReview/

Flows for end-to-end simulation

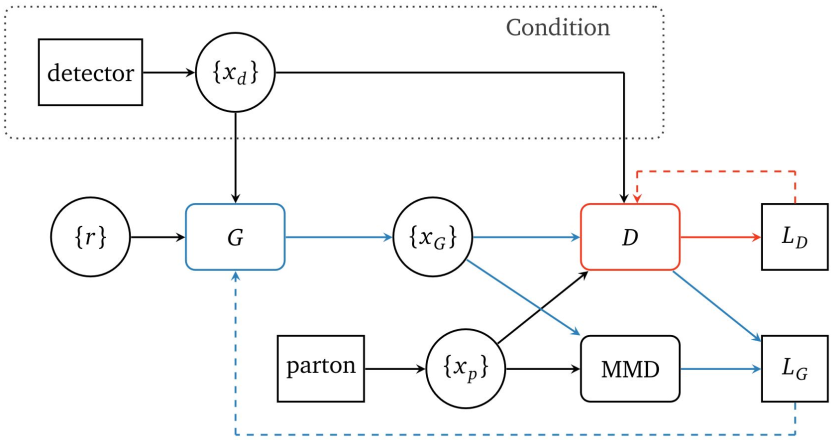
As the CMS FlashSim group, we are currently developing this type of approach

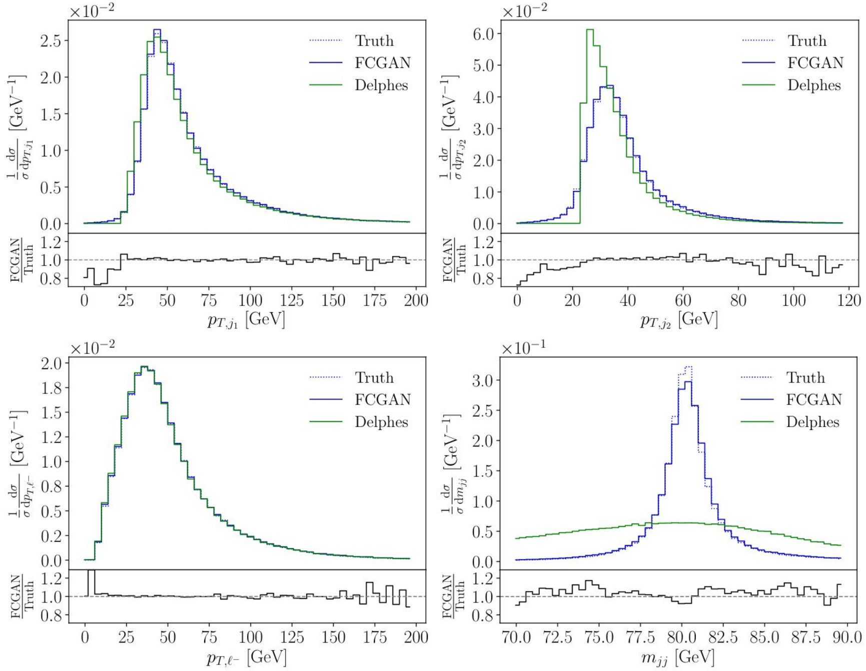
Several orders of magnitude of speedup and great accuracy



see https://cds.cern.ch/record/2858890?ln=it
and https://arxiv.org/abs/2402.13684

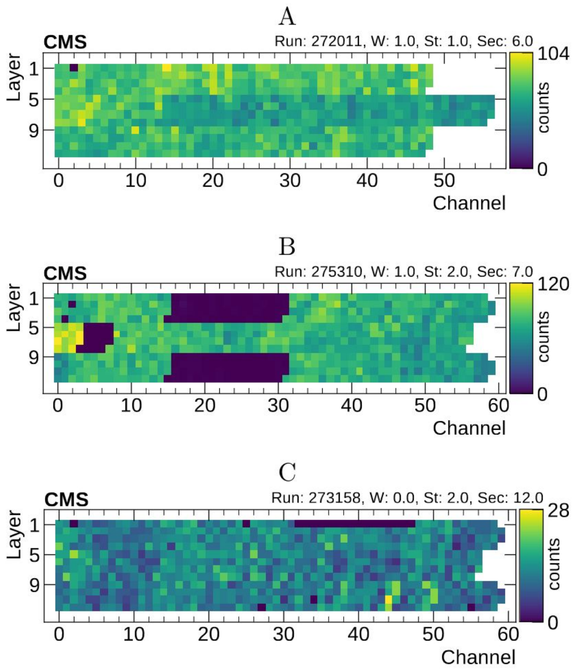
GAN for unfolding

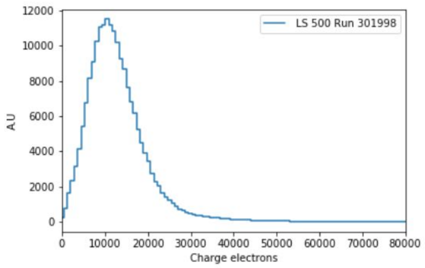


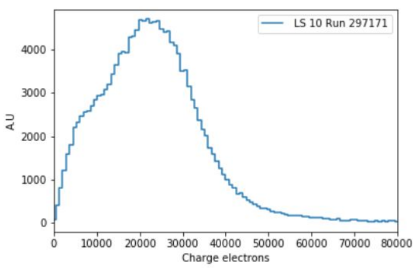


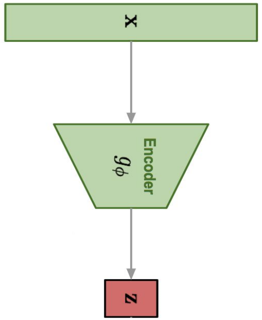
https://scipost.org/10.21468/SciPostPhys.8.4.070

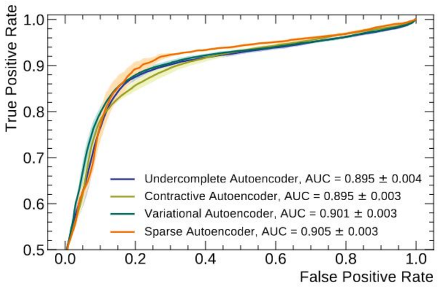
VAE for DQM and anomaly detection







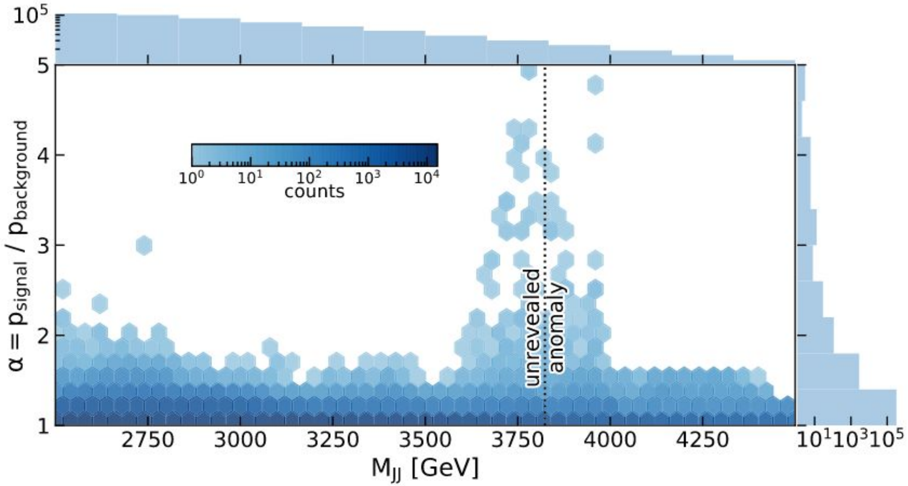


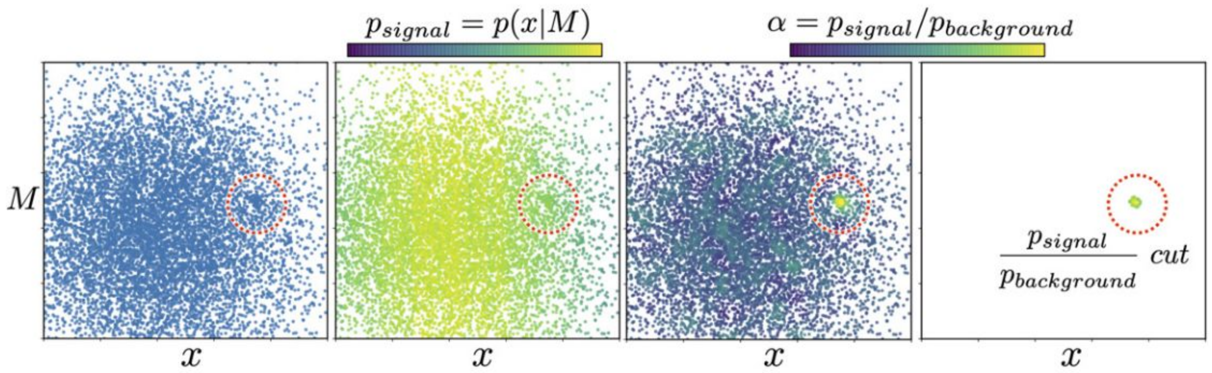


https://inria.hal.science/hal-03159873/document

Flows for anomaly detection

Idea: Model PDF of signal and background using Normalizing Flows





$$p_{bkg} = p(x|M \pm \Delta)$$

from LHC Olympics 2020

Conclusions

Generative models are a powerful tool at our disposal

Different models have specific advantages and drawbacks

Widespread adoption in many Physics use-cases and convincing results!

No readily available implementations for our problems, need to experiment! See you at the exercise?
https://github.com/francesco-vaselli/iCSC-exercise

Citations:
Thanks Lilian Weng!

Lilian Weng

OpenAI

Verified email at openai.com - Homepage

deep learning   machine learning   network science


```
@article{weng2021diffusion,
  title   = "What are diffusion models?",
  author  = "Weng, Lilian",
  journal = "lilianweng.github.io",
  year    = "2021",
  month   = "Jul",
  url     = "https://lilianweng.github.io/posts/2021-07-11-diffusion-models/"
}

@article{weng2017gan,
  title   = "From GAN to WGAN",
  author  = "Weng, Lilian",
  journal = "lilianweng.github.io",
  year    = "2017",
  url     = "https://lilianweng.github.io/posts/2017-08-20-gan/"
}

 @article{weng2018VAE,
  title   = "From Autoencoder to Beta-VAE",
  author  = "Weng, Lilian",
  journal = "lilianweng.github.io",
  year    = "2018",
  url     = "https://lilianweng.github.io/posts/2018-08-12-vae/"
}

@article{weng2018flow,
  title   = "Flow-based Deep Generative Models",
  author  = "Weng, Lilian",
  journal = "lilianweng.github.io",
  year    = "2018",
  url     = "https://lilianweng.github.io/posts/2018-10-13-flow-models/"
}
```

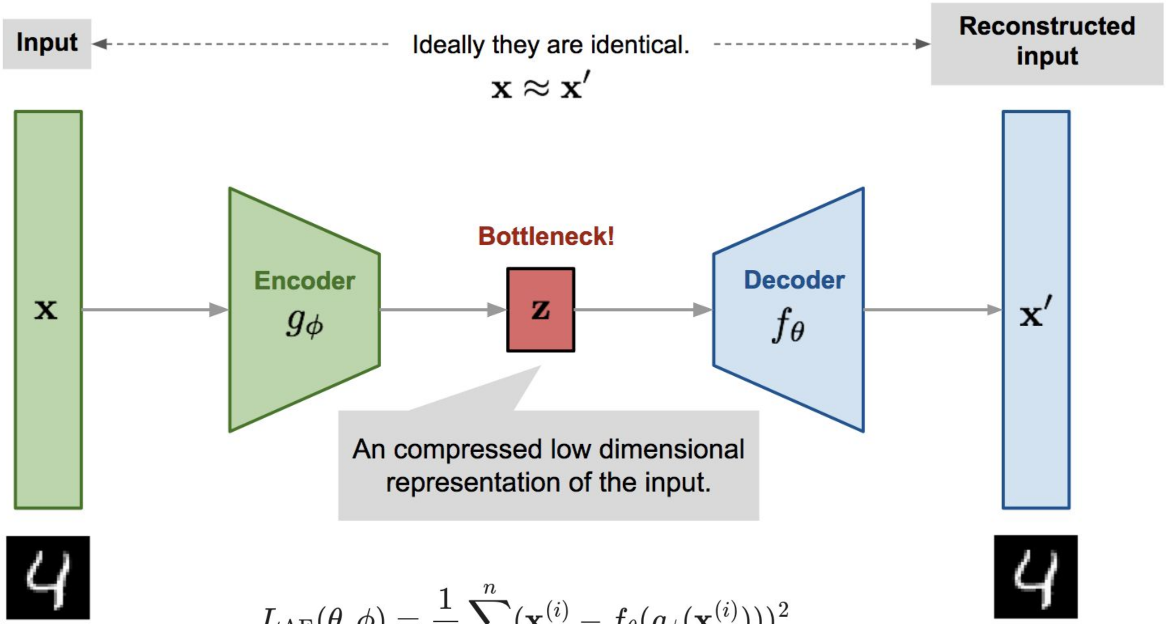
Backup

Autoencoders can be a great starting point

Encode **x** into low dimensional representation!

Encoder network: It translates the original high-dimension input into the latent low-dimensional code. The input size is larger than the output size.
Decoder network: The decoder network recovers the data from the code

Why can't I use this for generation?



$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

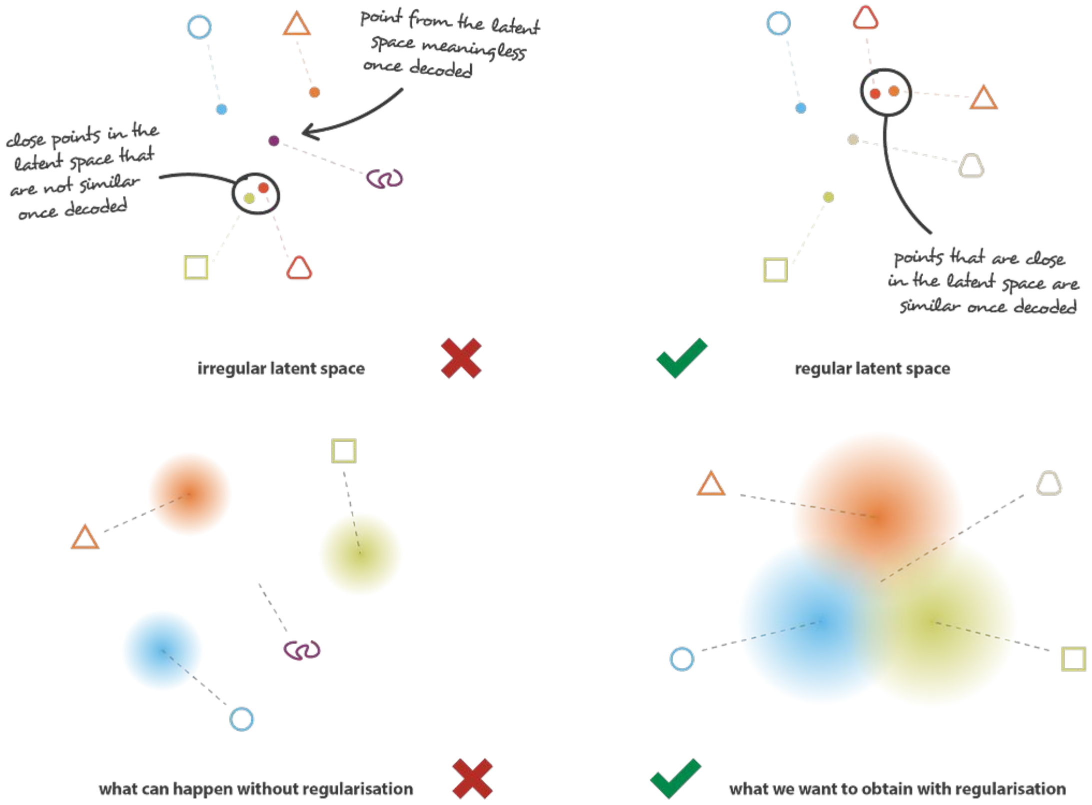
Reconstruction Loss

An irregular latent space is useless for generation!

Problem:
The autoencoder is solely trained to encode and decode with as few reconstruction loss as possible, no matter how the latent space is organised.

Meaningless points in latent space!

Random/useless samples!

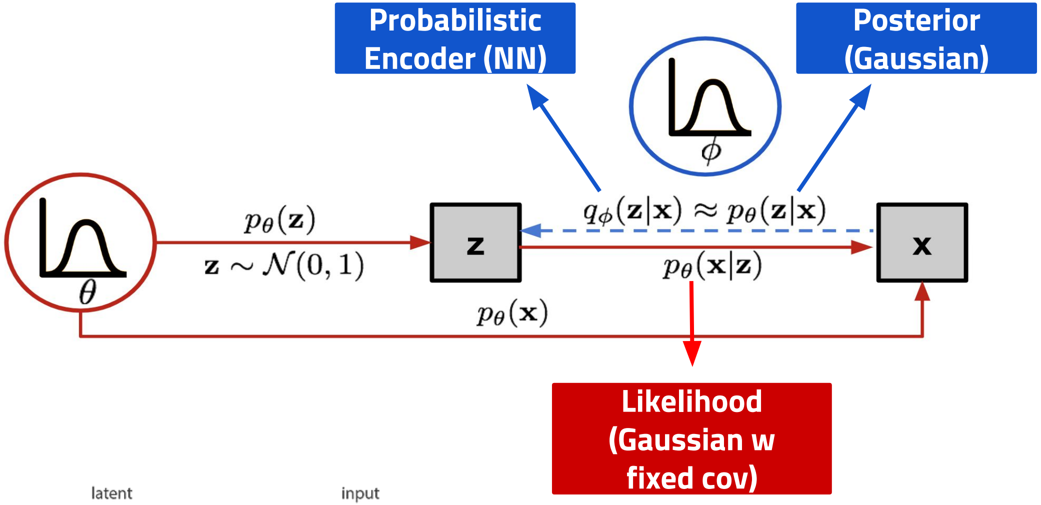


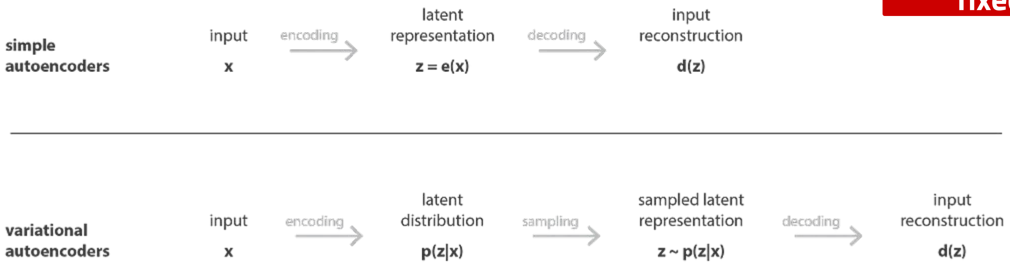
A graph model shows how we can regularize latent space!

We want to map the input into a distribution!

Prob encoder: learns to model conditional Gaussian dist given x

Prob decoder: learns to model mean of likelihood distribution given z



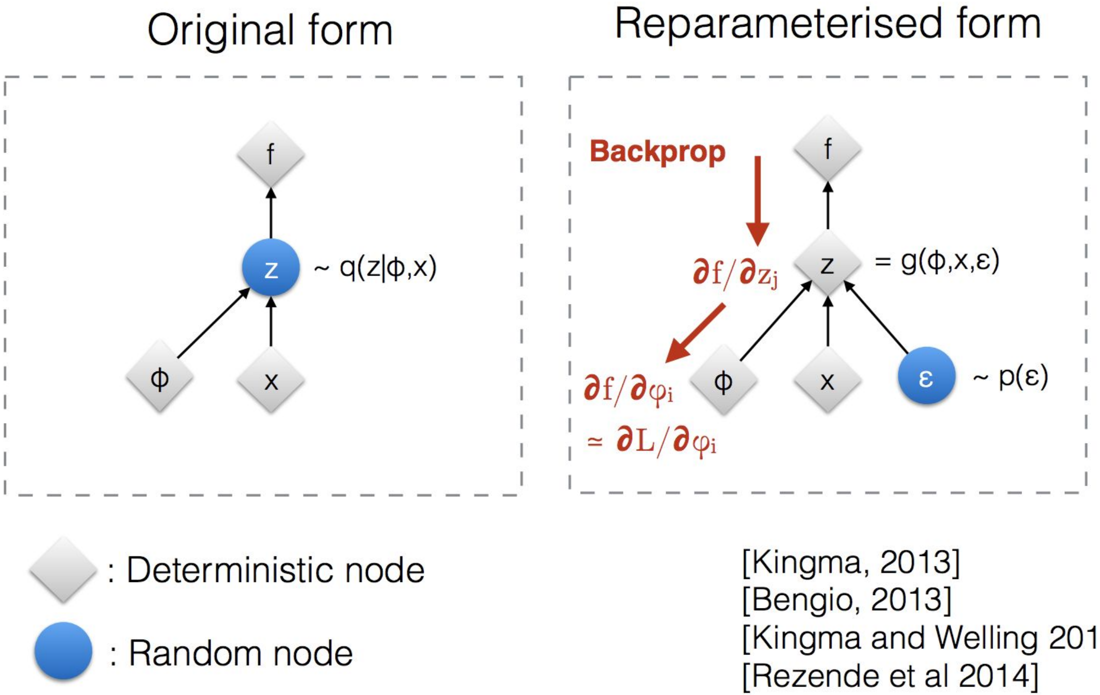


To train we need a small trick!

The expectation term in the loss function invokes generating samples from
$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$$

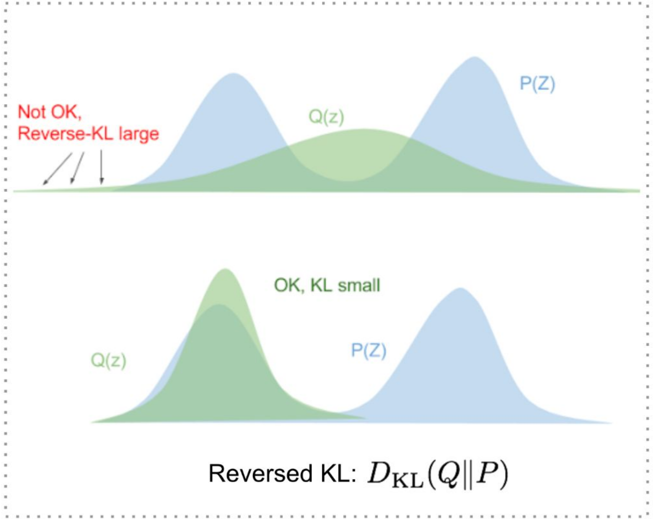
Sampling is a stochastic process and therefore we cannot backpropagate the gradient! To make it trainable, the reparameterization trick is introduced:

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$$



[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

A single loss is not enough!



Need to map NN output into the posterior

We can use Kullback-Leibler divergence to quantify the distance between these two distributions (NN vs Posterior)
$D_{\mathrm{KL}}(X|Y)$measures how much information is lost if the distribution Y is used to represent X.

Total loss = RECO + VAE
= Evidence Lower Bound (ELBO)

The "lower bound" part in the name comes from the fact that KL divergence is always non-negative and thus the loss is the lower bound of log(p(x))

$$
\begin{aligned}
L_{\mathrm{VAE}}(\theta,\phi) &= -\log p_\theta(\mathbf{x}) + \boxed{D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))} \\
&= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \\
\theta^*, \phi^* &= \arg\min_{\theta,\phi} L_{\mathrm{VAE}}
\end{aligned}
$$

$$
-L_{\mathrm{VAE}} = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x})
$$

Splines can be a smart choice for f(z)

Expressive

Admit analytical inverse,
fast to invert AND evaluate

We use ML to learn the optimal
disposition of points and derivatives

Just one of the possible choices!

Linear transformations (*Affine*)
are also used a lot:

$$f(z) = Wz + b$$