Advanced Deep Learning:
- Generative Models
- Physics-Based Deep Learning

Sorbonne Université – Masters DAC et M2A. PatrickGallinari,
patrick.gallinari@sorbonne-universite.fr, https://pages.isir.upmc.fr/gallinari/

Year 2022-2023

Advanced Deep learning

- Generative models
  - Generative Adversarial Networks
  - Variational Auto-Encoders
  - Diffusion models
  - Flow models

- Physics Based Deep Learning
  - Neural Networks and Ordinary Differential Equation solvers
  - Neural Networks for spatio-temporal dynamics
    - Context: AI4Science
    - Four problems
      - Discovering dynamics from data
      - NNs as surogate models for solving Partial Differential Equations
      - Incorporating physical knowledge in statistical dynamics models
      - Generalization for agnostic ML models for dynamics modeling

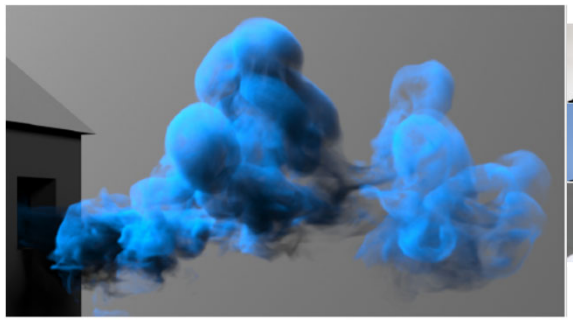
Generative models

Generative Adversarial Networks
Variational Auto-Encoders
Diffusion models
Flow models

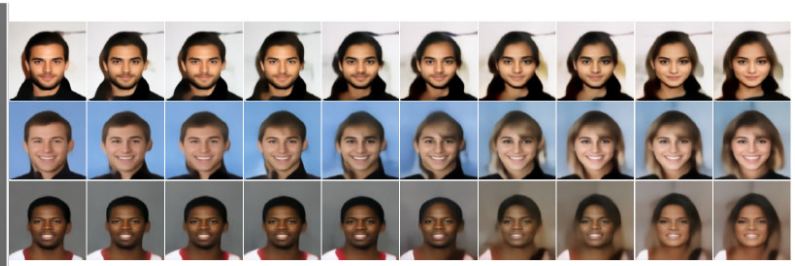
Generative models

- Objective
  - Learn a probability distribution model from data samples
    - Given $x^1, \ldots, x^N \in R^n$ learn to approximate their underlying distribution $\mathcal{X}$
    - For complex distributions, there is no analytical form, and for large size spaces (R^n) approximate methods (e.g. MCMC) might fail
    - Deep generative models recently attacked this problem with the objective of handling large dimensions and complex distributions

https://en.wikipedia.org/wiki/Edmond_de_Belamy
432 k$ Christies in 2018



Xie et al. 2019
artificial smoke



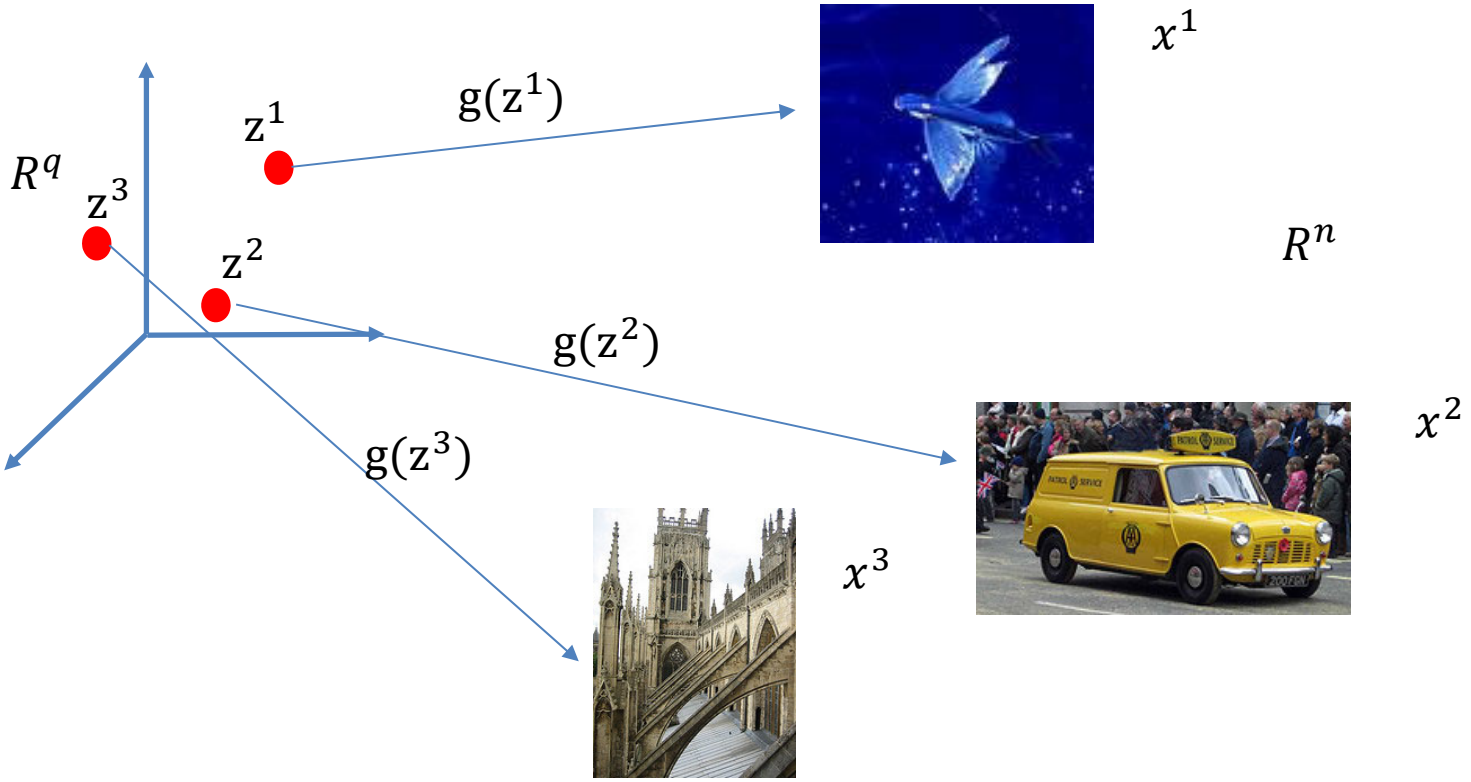
De Bezenac et al. 2021
Generating female images from male ones

Generative models

- Objective
  - General setup of deep generative models
    - Learn a generator nework $g_\theta: R^q \rightarrow R^n$ that transforms a latent distribution $\mathcal{Z} \subset R^q$ to match a target distribution $\mathcal{X}$
      - $\mathcal{Z}$ is usually a simple distribution e.g. Gaussian from which it is easy to sample, $q < n$
      - This is unlike traditional statistics where an analytic expression for the distribution is sought
    - Once trained the generator can be used for:
      - Sampling from the latent space:
        - $z \in R^q \sim \mathcal{Z}$ and then generate synthetic data via $g_\theta(.), g_\theta(z) \in R^n$
      - When possible, density estimation $p_\theta(x) = \int \quad p_\theta(x|z)p_Z(z)dz$
        - with $p_\theta(x|z)$ a function of g_θ

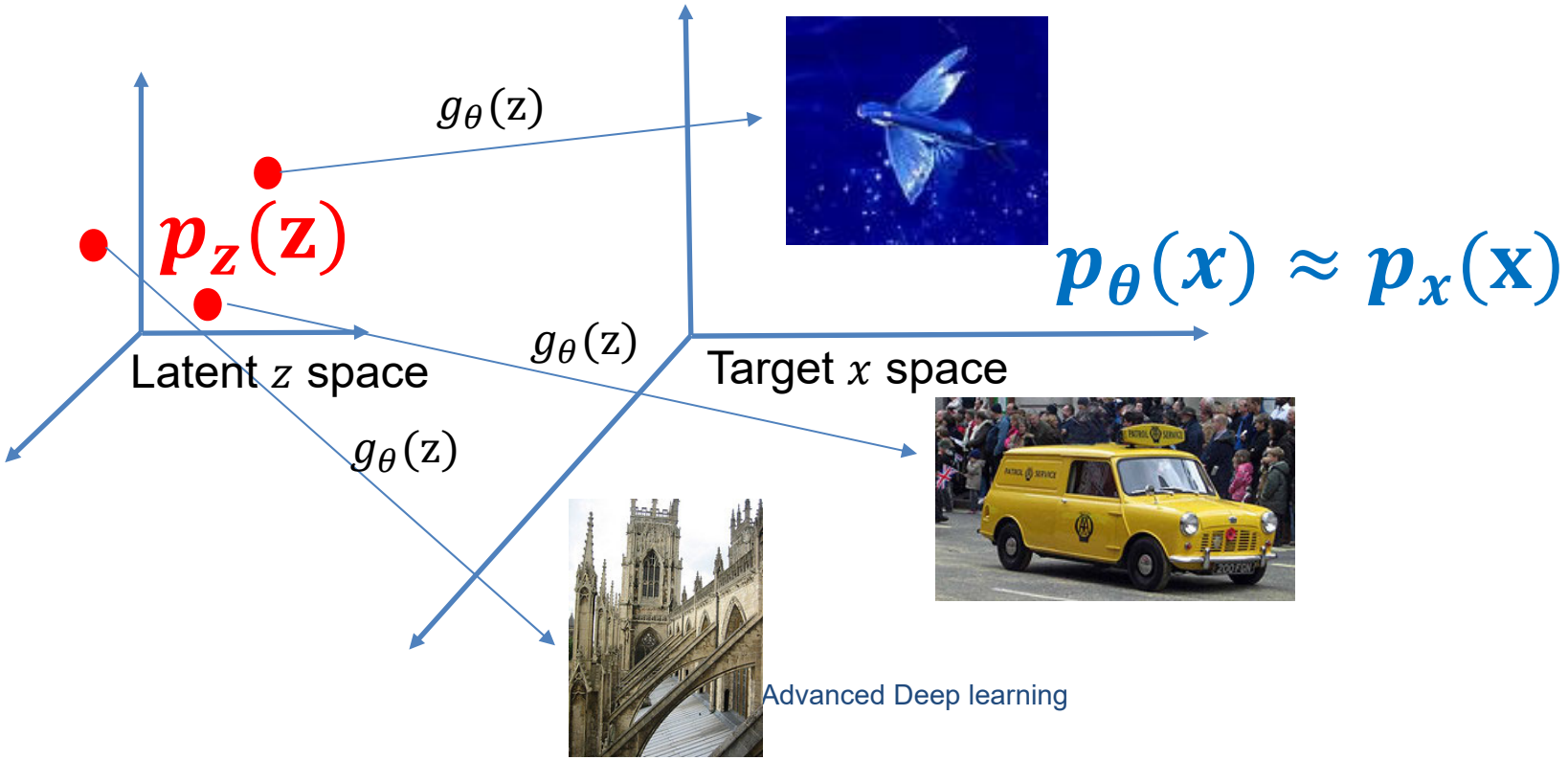
Generative models intuition

- Let $\{z^1, \ldots, z^N\}, z^i \in R^q$ and $\{x^1, \ldots, x^N\}, x^i \in R^n$, two sets of points in different spaces
  - Provided a sufficiently powerful model $g(x)$, it should be possible to learn complex deterministic mappings associating the two sets:



Generative models intuition

- Given distributions on a latent space $p_z(\mathrm{z})$, and on the data space $p_x(x)$, it is possible to map $p_z(\mathrm{z})$ onto $p_x(x)$?
  - g_θ defines a distribution on the target space $p_x\big(g_\theta(z)\big) = p_\theta(x)$
    - $p_\theta(x)$ is the generated data distribution, objective: $p_\theta(x) \approx p_x(x)$
  - Data generation: sample $\mathrm{z}\sim Z$, transform with g_θ, $g_\theta(z)$



Generative models intuition

- What we want: organize the latent space according to some characteristics of the observations (images)

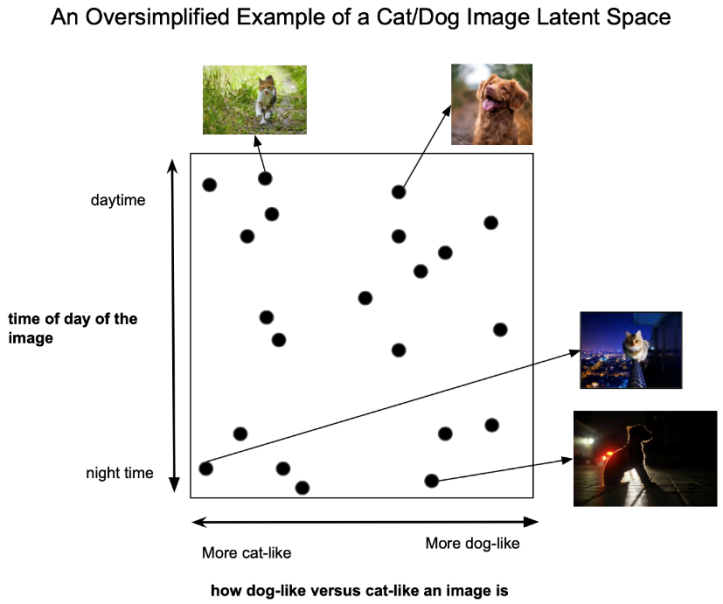
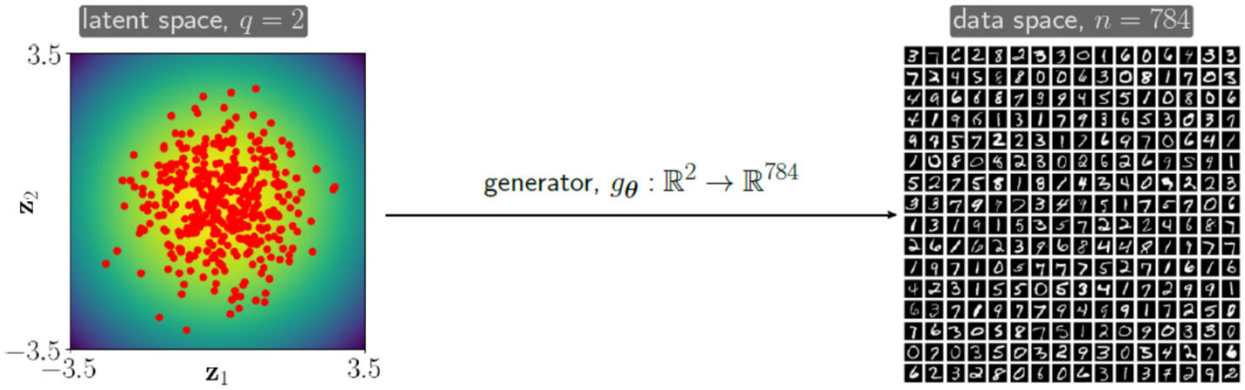


Fig.: https://ml.berkeley.edu/blog/posts/vq-vae/

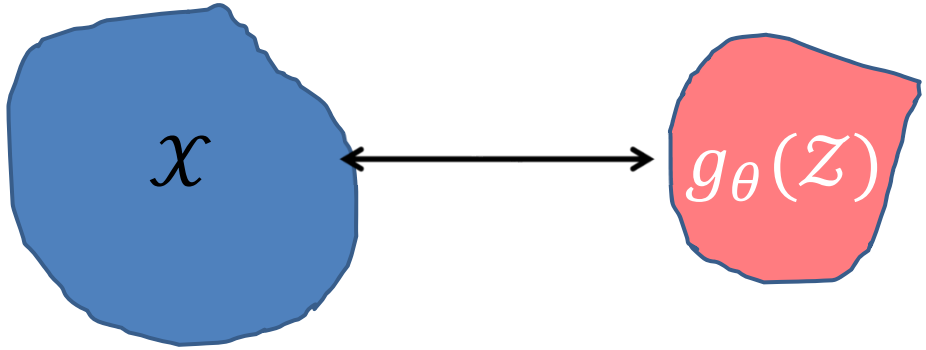
- See also the demos @
  - https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

Generative models intuition

- Data generation: sample z~Z, transform with g_θ, $g_\theta(z)$



- Important issue
  - How to compare predicted distribution $p_\theta(x)$ and target distribution $p_X(x)$?



Course objective

- Introduce three popular families of generative models
  - Joint requirements
    - Learn a generator g_θ from samples so that distribution $g_\theta(\mathcal{Z})$ is close to data distribution $\mathcal{X}, p_\theta(x) \approx p_x(x)$
    - This requires measuring the similarity between $g_\theta(\mathcal{Z})$ and $\mathcal{X}$
      - Different similarities are used for each family
- Three families
  - Generative Adversarial Networks
    - $g_\theta: R^q \rightarrow R^n, q \ll n$
    - Can approximate any distribution (no density hypothesis)
    - Similarity between generated and target distribution is measured via a discriminator or transport cost in the data space
  - Variational autoencoders
    - $g_\theta: R^q \rightarrow R^n, q \ll n$
    - Trained to maximize a lower bound of the samples' likelihood
    - Hyp: a density function explains the data
  - Flow models -> To be replaced by diffusion models
    - $g_\theta: R^n \rightarrow R^n$ is diffeomorphic $q = n$, invertibilty simplifies the problem
    - Trained to maximize the likelihood of the samples
    - Hyp: a density function explains the data

Variational Auto-Encoders

After Kingma D., Welling M., Auto-Encoding Variational Bayes, ICLR 2014

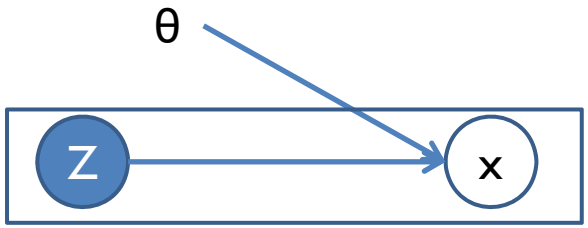
Plus some blogs – see the references

Prerequisite KL divergence

- Kullback Leibler divergence
  - Measure of the difference between two distributions p and q
  - Continuous variables
    - $D_{KL}(p(y)||q(y)) = \int_y (\log\frac{p(y)}{q(y)})p(y)dy$
  - Discrete variables
    - $D_{KL}(p(y)||q(y)) = \sum_i (\log\frac{p(y_i)}{q(y_i)})p(y_i)$
- Property
  - $D_{KL}(p(y)||q(y)) \geq 0$
  - $D_{KL}(p(y)||q(y)) = 0$ iff $p = q$
    - $D_{KL}(p(y)||q(y)) = -E_{p(y)}\left[log\frac{q(y)}{p(y)}\right] \geq -\log E_{p(y)}\left[\frac{q(y)}{p(y)}\right] = 0$
      - the first inequality is obtained via Jensen inequality:
      - For a convex function f, $f(E[x]) \leq E[f(x)]$, and $-\log x$ is a convex function
  - note: D_{KL} is asymmetric, symmetric versions exist, e.g. Jensen-Shannon divergence

Preliminaries – Variational methods

- Generative latent variable model
- Let us suppose available a joint model on the observed and latent variables $p_\theta(x, z)$



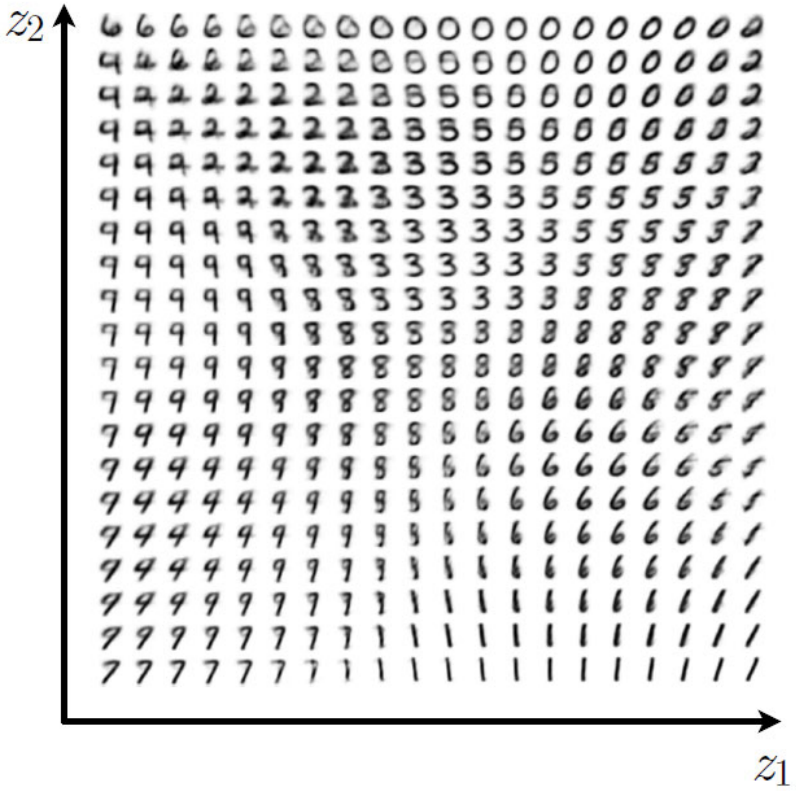
- The observations x are generated by the following process
  - Sample from $z \sim p_\theta(z)$ - $p_\theta(z)$ is the prior
  - generate $p_\theta(x|z)$ - $p_\theta(x|z)$ is the likelihood

- Training objective
  - We want to optimize the likelihood of the observed data
    - $p(x) = \int p(x|z)p(z)dz$ - $p(x)$is called the evidence
    - Computing the integral requires evaluating over all the configurations of latent variables,
    - This is often intractable
    - In order to narrow the sampling space, one may use importance sampling, i.e. sampling important z instead of sampling blindly from the prior
    - Let us introduce a sampling function $q_\Phi(z|x)$

VAEs - Intuition

- Intuitively, z might correspond to the factors conditioning the generation of the data

MNIST:



Frey Face dataset:

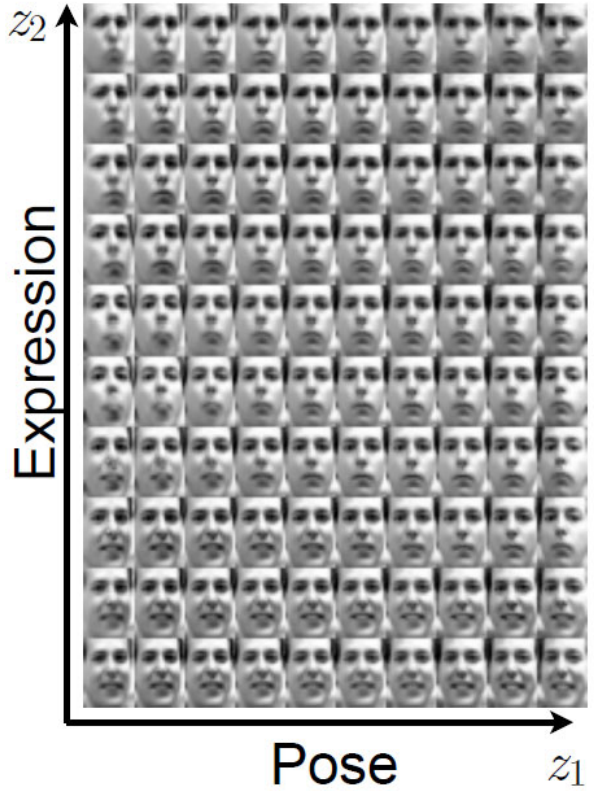


Fig. (Kingma 2015)

VAE
Loss criterion – summary

- The log likelihood for data point x can be decomposed as
  - $\log p_\theta(x) = D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + V_L(\theta, \phi; x)$
  - with
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
- Why is it useful?
  - $D_{KL}(.||.) \geq 0$, then $V_L(\theta, \phi; x)$ is a lower bound of $\log p_\theta(x)$
  - in order to maximize $\log p_\theta(x)$, we will maximize $V_L(\theta, \phi; x)$
- $V_L(\theta, \phi; x)$ is called the ELBO: Evidence Lower Bound
  - With some appropriate choice of $q_\phi(z|x)$ this is amenable to a computationable form
  - $q_\phi(z|x)$ approximates the intractable posterior $p_\theta(z|x)$
  - This method is called variational inference
    - In general inference denotes the compuations of hidden variables given observed ones (e.g. infering the class of an object)
- Note
  - Because each representation z is associated to a unique x, the loss likelihood can be decomposed for each point – this is what we do here
  - The global log likelihood is then the summation of these individual losses

VAE
Loss criterion – summary

- Note
  - Because each representation z is associated to a unique x, the log likelihood can be decomposed for each point – this is what we do here
  - The global log likelihood is then the summation of these individual losses

VAE
Loss criterion – summary

- Variational lower bound:
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Remarks
    - $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ is a **reconstruction** term
      - Measures how well the datum x can be reconstructed from latent representation z
    - $D_{KL}(q_\phi(z|x)||p(z))$ is a **regularization** term:
      - Forces the learned distribution $q_\phi(z|x)$ to stay close to the prior $p(z)$
        - Otherwise a trivial solution would be to learn a Dirac distribution for $q_\phi(z|x)$
        - We want the z to be close in the latent space for similar xs
      - $p(z)$ has usually a simple form e.g. $\mathcal{N}(0, I)$, then $q_\phi(z|x)$ is also forced to remain simple

VAE details
Derivation of the loss function

- $\boldsymbol{log\, p_\theta\,(x) = \mathbf{D}_{KL}(q_\phi(z|x)||p_\theta(\mathbf{z}|\mathbf{x})) + V_L(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})}$

Proof

- $\log p_\theta\,(x) = \int_z\,(\log p(x))q(z|x)\,dz \qquad (\int_z\,q(z|x)\,dz = 1)$
- $\log p_\theta(x) = \int_z\,(\log \frac{p(x,z)}{p(z|x)})q(z|x)\,dz$
- $\log p_\theta(x) = \int_z\,(\log \frac{p(x,z)}{q(z|x)}\frac{q(z|x)}{p(z|x)})q(z|x)\,dz$
- $\log p_\theta(x) = \int_z\,(\log \frac{p(x,z)}{q(z|x)})q(z|x)\,dz + \int_z\,(\log \frac{q(z|x)}{p(z|x)})q(z|x)\,dz$
- $\log p_\theta(x) = E_{q(z|x)}[\log \mathrm{p(x,z)} - \log q(z|x)] + D_{KL}(q(z|x)||p(z|x))$

$$\log p_\theta(x) = V_L(\theta, \phi; x) + \mathrm{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$$

with

$$V_L(\theta, \phi; x) = E_{q(z|x)}[\log\ \mathrm{p}_\theta(x, z) - \log q_\phi(z|x)]$$

- Maximizing $\log p_\theta\,(x)$ is equivalent to maximizing $V_L(\theta, \phi; x)$ (and minimizing $\mathrm{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$
- $V_L(\theta, \phi; x)$ is called an Evidence Lower Bound (ELBO)

VAE details
Derivation of the loss function

- $\boldsymbol{V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[log\, p_\theta(x|z)]}$

Proof:

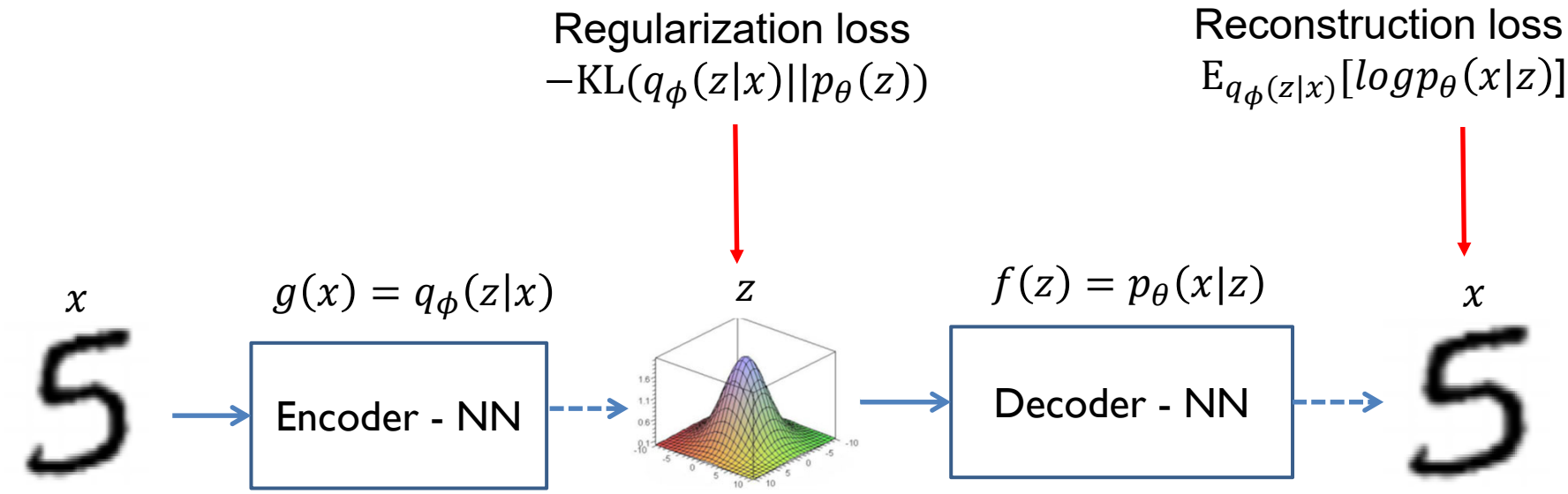
- $V_L(\theta, \phi; x) = \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x, z) - \log q_\phi(z|x)]$
- $V_L(\theta, \phi; x) = \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)]$
- $V_L(\theta, \phi; x) = -\mathrm{D}_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z)) + \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x|z)]$

VAE
Loss criterion – summary

- Variational lower bound:
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - This form provides a link with a NN implementation
    - The generative $p_\theta(x|z)$ and inference $q_\phi(z|x)$ modules are implemented by NNs
    - They will be trained to maximize the reconstruction error for each (z, x): $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ term
    - The inference module $q_\phi(z|x)$ will be constrained to remain close to the prior $p(z)$: $-\mathrm{D}_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z)) \approx 0$

VAE
Loss - summary

- Loss function in the NN model



- Training performed via Stochastic gradient
  - This requires an analytical expression for the loss functions and for gradient computations
    - ⇢ Sampling
    - → deterministic

VAE- reparametrization trick

- Training with stochastic units: reparametrization trick
  - Not possible to propagate the gradient through stochastic units (the zs and xs are generated via sampling)
  - Solution
    - Parametrize z as a deterministic transformation of a random variable ϵ: $z = g_\phi(x, \epsilon)$ with $\epsilon \sim p(\epsilon)$ independent of ϕ, e.g. $\epsilon \sim N(0,1)$
    - Example
      - If $z \sim \mathcal{N}(\mu, \sigma)$, it can be reparameterized by $z = \mu + \sigma \odot \epsilon$, with $\epsilon \sim \mathcal{N}(0,1)$, with $\odot$ pointwise multiplication (μ, σ are vectors here)
      - For the NN implementation we have: $z = \mu_z(x) + \sigma_z(x) \odot \epsilon_z$
    - This will allow the derivative to « pass » through the z
    - For the derivative, the sampling operation is regarded as a deterministic operation with an extra input z, whose distribution does not involve variables needed in the derivation

VAE - reparametrization trick

- Reparametrization (fig. from D. Kingma)

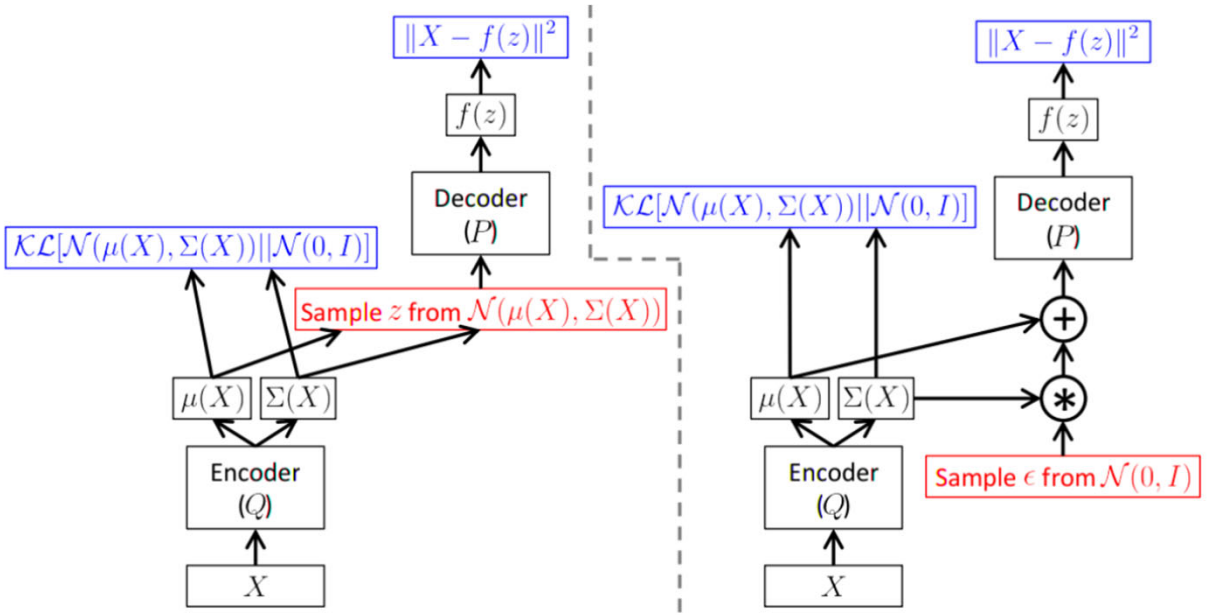


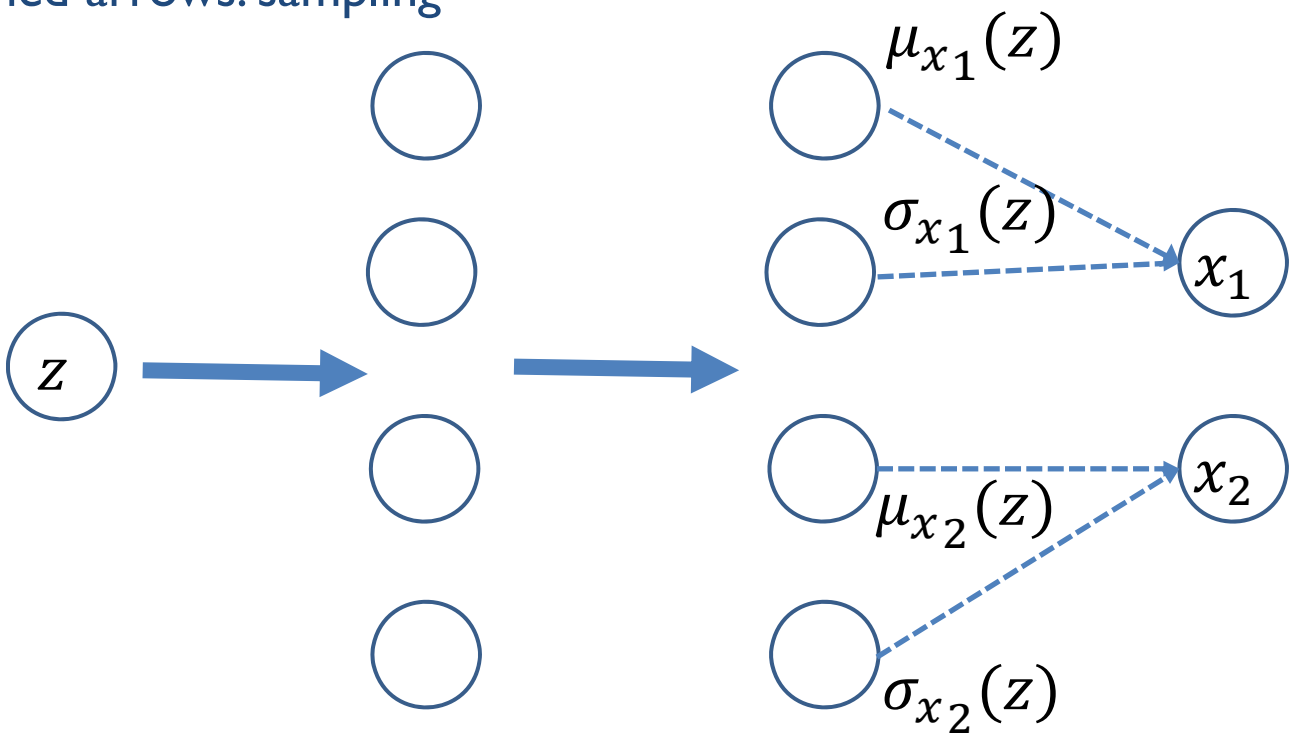
Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

VAE
Exemple: Gaussian priors and posteriors

- Special case: gaussian priors and posteriors
- Hyp:
  - $p(z) = \mathcal{N}(0, I)$
  - $p_\theta(x|z) = \mathcal{N}(\mu(z), \sigma(z)), \sigma(z)$ diagonal matrix, $x \in R^D$
  - $q_\phi(z|x) = \mathcal{N}(\mu(x),\ \sigma(x)), \sigma(x)$ diagonal matrix, $z \in R^J$

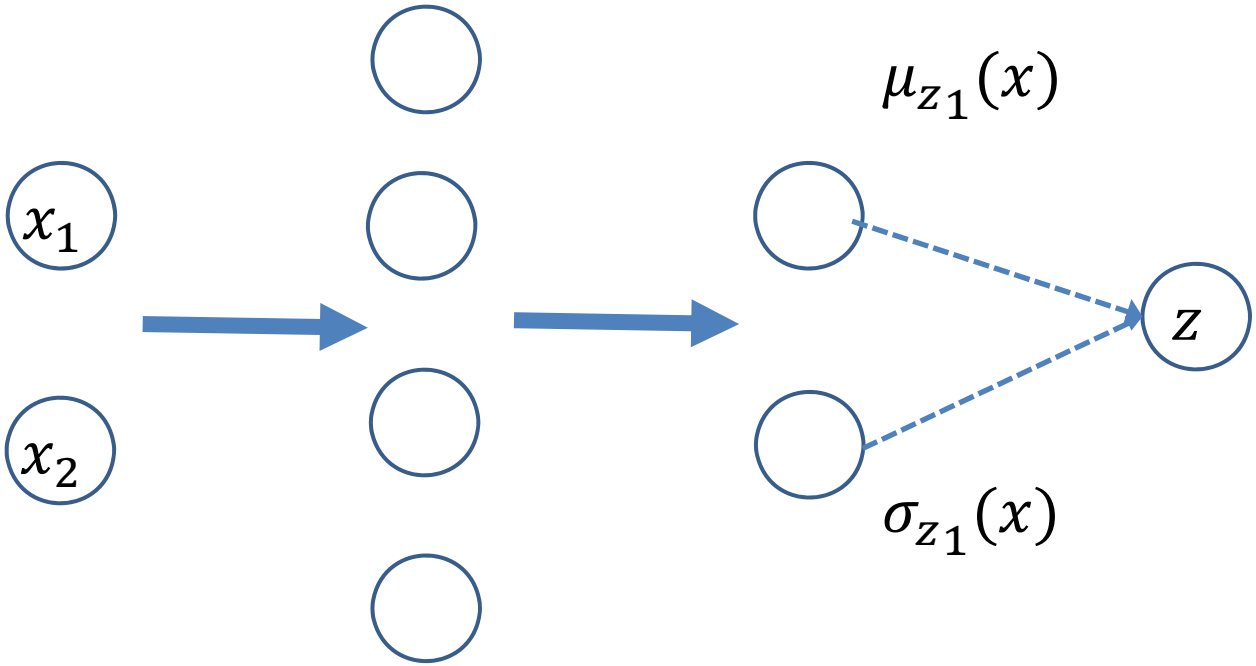
VAE
Exemple: Gaussian priors and posteriors - illustration

- Decoder:
  - in the example z is 1 dimensional and x is 2 dimensional, f is a 1 hidden layer MLP with gaussian output units and tanh hidden units
  - full arrows: deterministic
  - dashed arrows: sampling



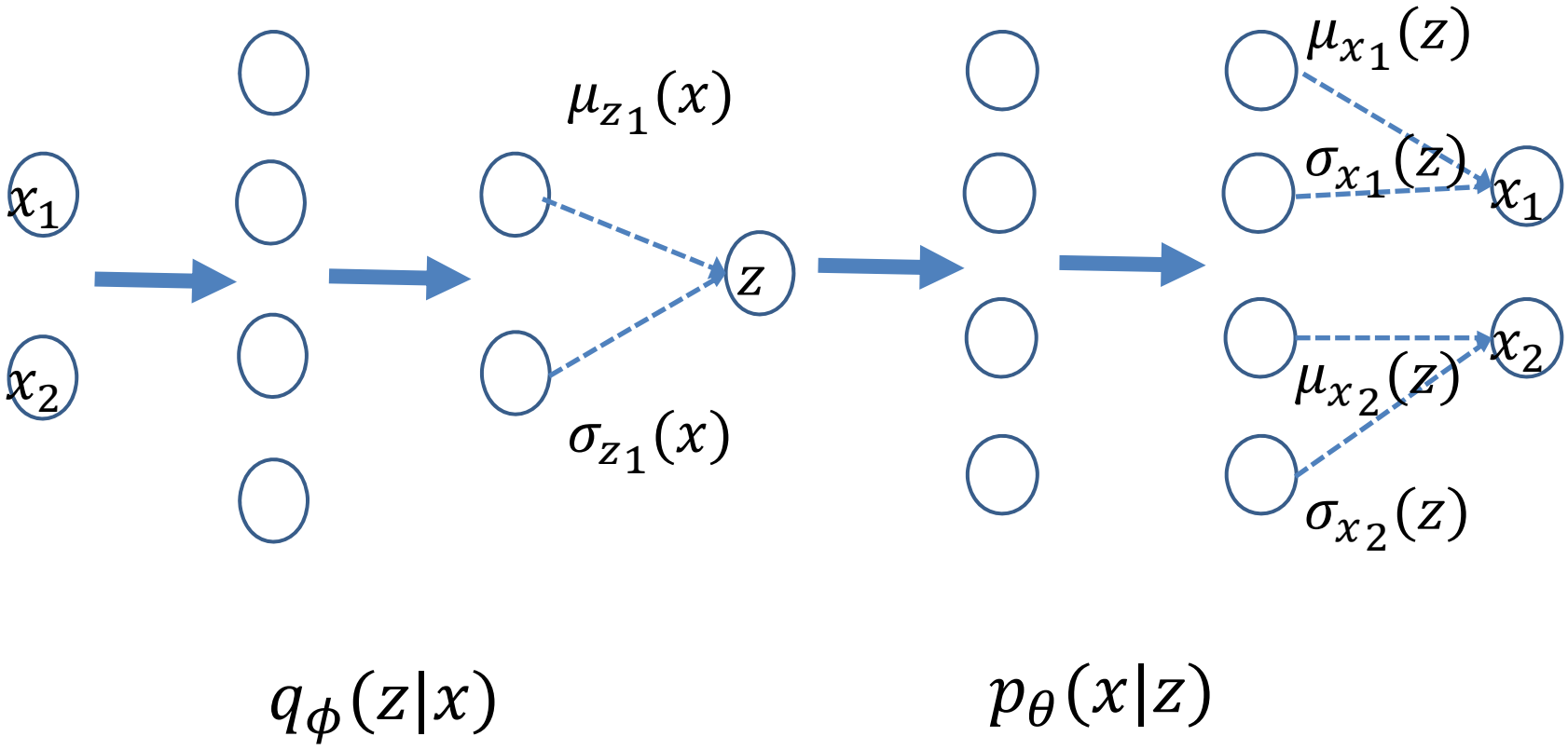
VAE
Gaussian priors and posteriors - illustration

- Encoder
  - in the example z is 1 dimensional and x is 2 dimensional, g is a 1 hidden layer MLP with gaussian output units and tanh hidden units
  - full arrows: deterministic
  - dashed arrows: sampling



VAE
Gaussian priors and posteriors

- Putting it all together



VAE details
for Gaussian priors and posteriors

VAE – instanciation example
Gaussian priors and posteriors

- Special case: gaussian priors and posteriors
- Hyp:
  - $p(z) = \mathcal{N}(0, I)$
  - $p_\theta(x|z) = \mathcal{N}(\mu(z), \sigma(z)), \sigma(z)$diagonal matrix, $x \in R^D$
  - $q_\phi(z|x) = \mathcal{N}(\mu(x),\ \sigma(x)), \sigma(x)$ diagonal matrix, $z \in R^J$

- Variational lower bound
  - $V_L(\theta, \phi; x) = -KL(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - In this case, $\mathrm{D}_{KL}(q_\phi(z|x)||p(z))$ has an analytic expression (see next slide)
    - $-D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left(\left(\sigma_{z_j}\right)^2\right) - \left(\mu_{z_j}\right)^2 - \left(\sigma_{z_j}\right)^2)$
  - $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ is estimated using Monte Carlo sampling
    - $E_{q_\phi(z|x)}[\log p_\theta(x|z)] \simeq \frac{1}{L}\sum_{l=1}^{L}\log(p_\theta(x|z^{(l)})$
    - $\log(p_\theta\left(x\middle|z^{(l)}\right) = -\sum_{j=1}^{D}\frac{1}{2}\log(\sigma_{x_j}^2\left(z^{(l)}\right)) + \frac{(x_j - \mu_{x_j}(z^{(l)}))^2}{2\sigma_{x_j}^2(z^{(l)})}$
    - i.e. L samples with $z^{(l)} = g_\phi\left(x, \epsilon^{(l)}\right)$

VAE - instanciation example
Gaussian priors and posteriors (demos next slides)

- If $z \in R^J$: $-KL(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left((\sigma_j)^2\right) - (\mu_j)^2 - (\sigma_j)^2)$
- proof
  - $KL(q_\phi(z)||p(z)) = \int \; q_\phi(z) \log \frac{q_\phi(z)}{p(z)} dz$
- Consider the 1 dimensional case
  - $\int q_\phi(z) \log p(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; 0,1) dz$
  - $\int q_\phi(z) \log p(z) dz = -\frac{1}{2}\log(2\pi) - \frac{1}{2}(\mu^2 + \sigma^2)$
    - Property of 2nd order moment of a Gaussian
  - $\int q_\phi(z) \log q_\phi(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; \mu, \sigma) dz$
  - $\int q_\phi(z) \log q_\phi(z) dz = -\frac{1}{2}\log(2\pi) - \frac{1}{2}(1 + \log \sigma^2)$
  - ……
  - Since both ddp are diagonals, extension to J dimensions is straightforward, hence the result

VAE - instanciation example
Gaussian priors and posteriors – demos for the 1 dimensional case

- Remember $q_\phi(z|x) = \mathcal{N}(\mu(x),\ \sigma(x))$
- Then $\int q_\phi(z) \log p(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; 0,1) dz$
- $$= E_{q_\Phi}[\log \mathcal{N}(z; 0,1)]$$
  $$= E_{q_\Phi}\left[\log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right)\right)\right]$$
  $$= E_{q_\Phi}\left[-\frac{1}{2}\log 2\pi - \frac{z^2}{2}\right]$$
  $$= -\frac{1}{2}\log 2\pi - \frac{1}{2} E_{q_\Phi}\ [z^2]$$
- What is the value of $E_q[z^2]$?
  - $E_{q_\Phi}[(z-\mu)^2] = \sigma^2$
  - $E_{q_\Phi}[z^2] - 2E_{q_\Phi}[z\mu] + \mu^2 = \sigma^2$
    - $E_{q_\Phi}[z\mu] = \mu^2$
  - $E_{q_\Phi}[z^2] = \mu^2 + \sigma^2$
- Then $\int \boldsymbol{q_\phi(z)\, log\, p(z) dz = -\frac{1}{2} log\, 2\pi - \frac{1}{2}(\mu^2 + \sigma^2)}$

VAE - instanciation example
Gaussian priors and posteriors – demos for the 1 dimensional case

- $\int q_\phi(z) \log q_\phi(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; \mu, \sigma) dz$

$$= E_{q_\Phi}[\log(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right))]$$

$$= -\frac{1}{2}\log 2\pi - \log\sigma - E_{q_\Phi}[\frac{(z-\mu)^2}{2\sigma^2}]$$

$$= -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma^2 - \frac{1}{2}$$

$$= -\frac{1}{2}\log 2\pi - \frac{1}{2}(\log\sigma^2 + 1)$$

VAE - instanciation example
Gaussian priors and posteriors

- Loss
- Regularization term
  - $-KL(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left(\left(\sigma_j\right)^2\right) - \left(\mu_j\right)^2 - \left(\sigma_j\right)^2)$
- Reproduction term
  - $\log\left(p(x|z)\right) = \sum_{j=1}^{D}\frac{1}{2}\log(\sigma_j^2(z)) + \frac{(x_j - \mu_j(z))^2}{2\sigma_j^2(z)}$
- Training
  - Mini batch or pure stochastic
    - Repeat
      - $x \leftarrow$ random point or minibatch
      - $\epsilon \leftarrow$ sample from $p(\epsilon)$ for each x
      - $\theta \leftarrow \nabla_\theta V_L(\theta, \phi; x, g(\epsilon, \phi))$
      - $\phi \leftarrow \nabla_\phi V_L(\theta, \phi; x, g(\epsilon, \phi))$
    - Until convergence

- References
  - Nice blog explaining VAEs
    - https://lilianweng.github.io/posts/2018-08-12-vae/
    - https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
    - https://www.fenghz.xyz/vector-quantization-based-generative-model/
  - This blog on variational inference
    - https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-inference-25a8aa9bce29

Learning discrete distributions: VQ-VAE (highlights)

- So far we considered continuous latent distributions
- There are several instances were discrete distributions are more appropriate
  - Text data, objects in images (color, size, orientation,…), etc
  - There are several algorithms, e.g. transformers designed to work with discrete data
  - Teaser: Dall-e – makes use of a discrete VAE together with transformers in order to generate diverse images
    - https://openai.com/blog/dall-e/, https://openai.com/dall-e-2/
    - https://gpt3demo.com/apps/openai-dall-e
    - https://www.craiyon.com/ (mini version of Dall-e)

Learning discrete distributions: VQ-VAE

- What is a discrete latent distribution?

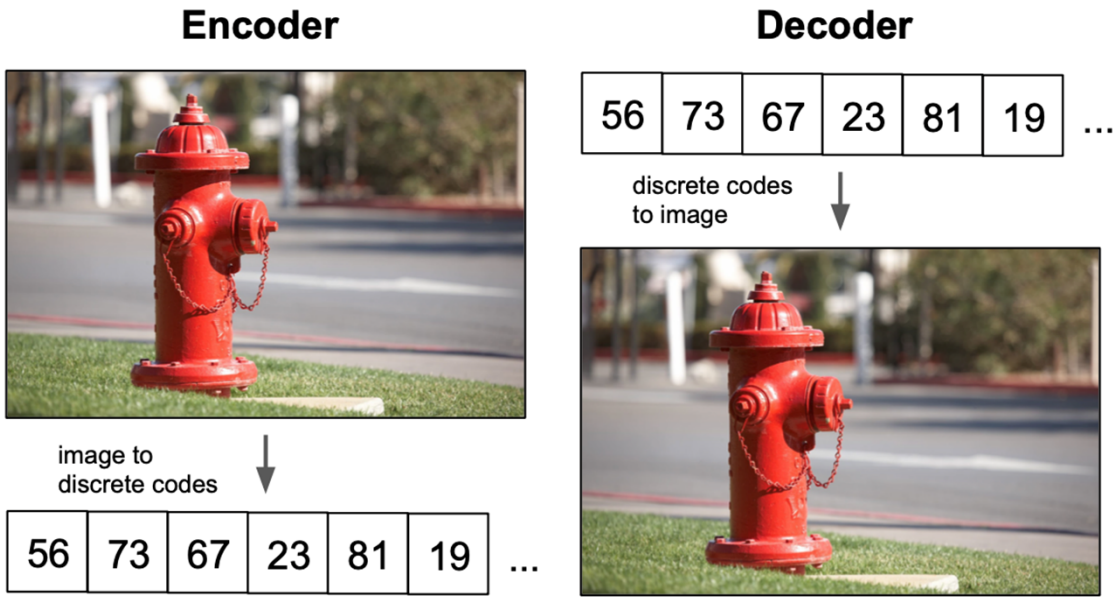
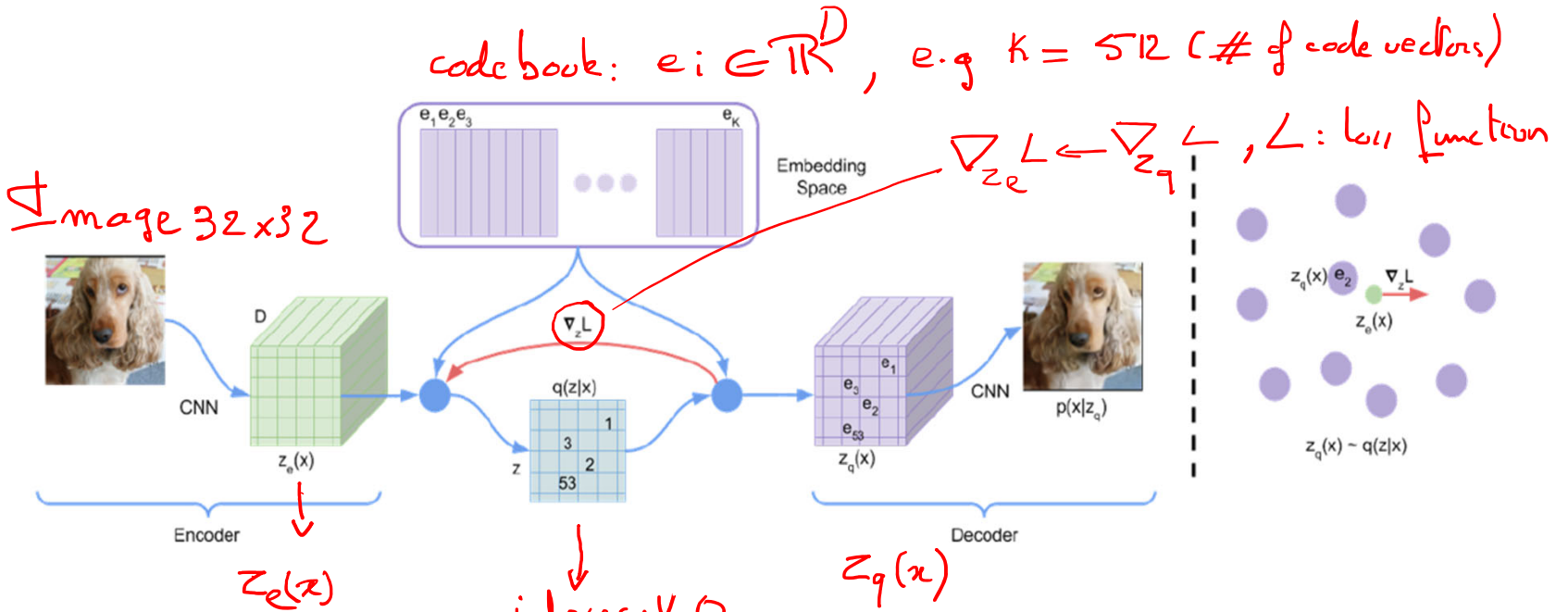


Fig: https://ml.berkeley.edu/blog/posts/vq-vae/

Learning discrete distributions: VQ-VAE

- VQ-VAE modifies the vanilla VAE by adding a discrete codebook of vectors to the VAE - It is used to quantize the VAE bottleneck
  - General scheme: VQ-VAE paper - https://arxiv.org/pdf/1711.00937.pdf



Learning discrete distributions: VQ-VAE

- Loss function
  - $L = \| x - Dec(zq(x)) \|^2 + \| sg(ze(x)) - zq(x) \|^2 + \beta \| ze(x) - sg(zq(x)) \|^2$
  - With $sg()$ stop gradient
    - $\| x - Dec(zq(x)) \|^2$: train **decoder** and **encoder**
    - $\| sg(ze(x)) - zq(x) \|^2$: train the **codebook** $e = zq(x)$
    - $\| ze(x) - sg(zq(x)) \|^2$: train **encoder**, forces $ze(x)$ to stay close to $e = zq(x)$
      - This is because the codebook does not train as fast as the encoder and the decoder
        - Prevents the encoder values to diverge

- Gradients
  - Since it is not possible to compute the gradtient through the VQ component, it is proposed to simply copy the gradient w.r.t. zq to ze
  - $\nabla_{ze(x)} \| x - Dec(zq(x)) \|^2 = \nabla_{zq(x)} \| x - Dec(zq(x)) \|^2$
  - This is called straight-through gradient
- Note
  - This is an incomplete description, the model requires additional steps
  - Dall-e makes use of a slightly different discrete VAE (called dVAE)

- References
  - Nice blog explaining VAEs
    - https://lilianweng.github.io/posts/2018-08-12-vae/
    - https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
    - https://www.fenghz.xyz/vector-quantization-based-generative-model/
  - This blog on variational inference
    - https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-inference-25a8aa9bce29

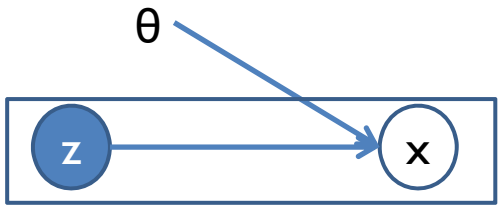
Generative Adversial Networks - GANs

Ian J. Goodfellow, et al. 2014

There has been a strong hype for GANs for several years - O(1000) GAN papers on Arxiv

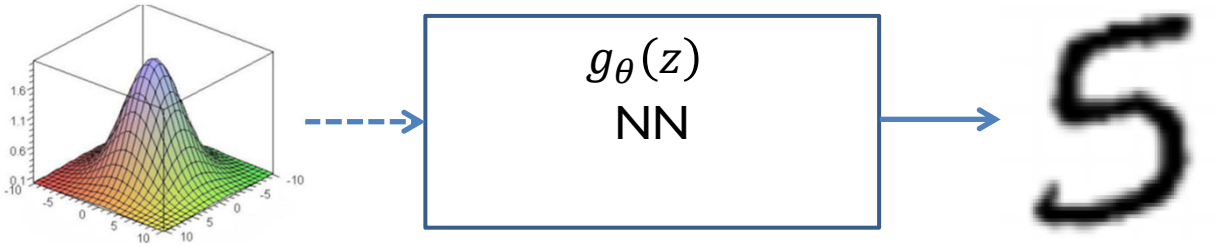
GANs

- Generative latent variable model



  - Given Samples $x^1, \ldots, x^N \in R^n$, with $x \sim \mathcal{X}$, latent space distribution $z \sim \mathcal{Z}$ e.g $z \sim \mathcal{N}(0, \mathrm{I})$, use a NN to learn a possibly complex mapping $g_\theta : R^q \rightarrow R^n$ such that:

$$p_x\big(g_\theta(z)\big) = p_\theta(x) \approx p_x(x)$$



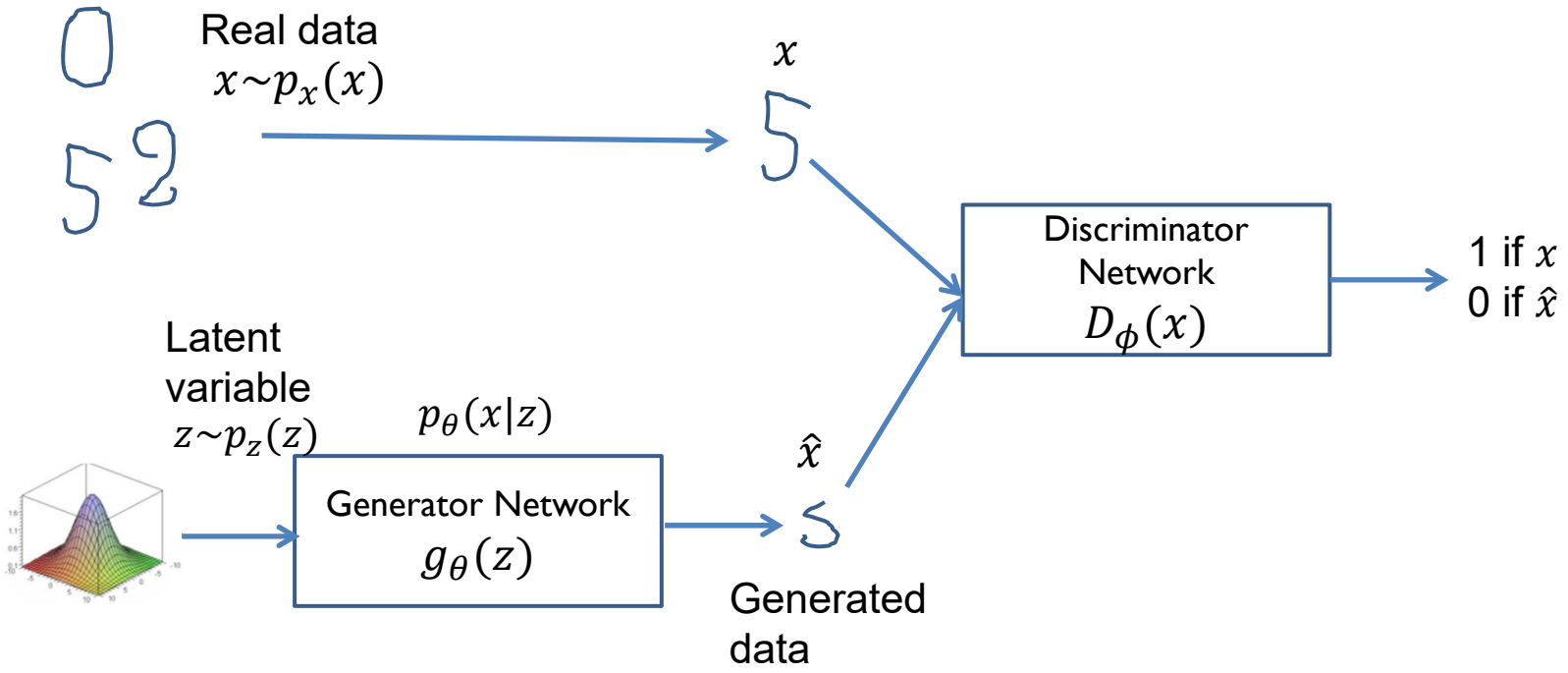
- Different solutions for measuring the similarity between $p_\theta(x)$ and $p_x(x)$
  - In this course: binary classification
- Note:
  - Once trained, sample from z directly generates the samples $g_\theta(z)$
  - Different from VAEs and Flows where the NN $g_\theta(.)$ generate distribution parameters

GANs – Adversarial training as binary classification

- Principle
  - A **generative** network generates data after sampling from a latent distribution
  - A **discriminant** network tells if the data comes from the generative network or from real samples
    - The discriminator will be used to measure the distance between the distributions $p_\theta(x)$ and $p_x(x)$
  - The two networks are trained together
    - The generative network tries to fool the discriminator, while the discriminator tries to distinguish between true and artificially generated data
    - The problem is formulated as a MinMax game
    - The Discriminator will force the Generator to be « clever » and learn the data distribution
- Note
  - No hypothesis on the existence of a density function
    - i.e. no density estimate (Flows), no lower bound (VAEs)

GANs – Adversarial training as binary classification
Intuition - Training

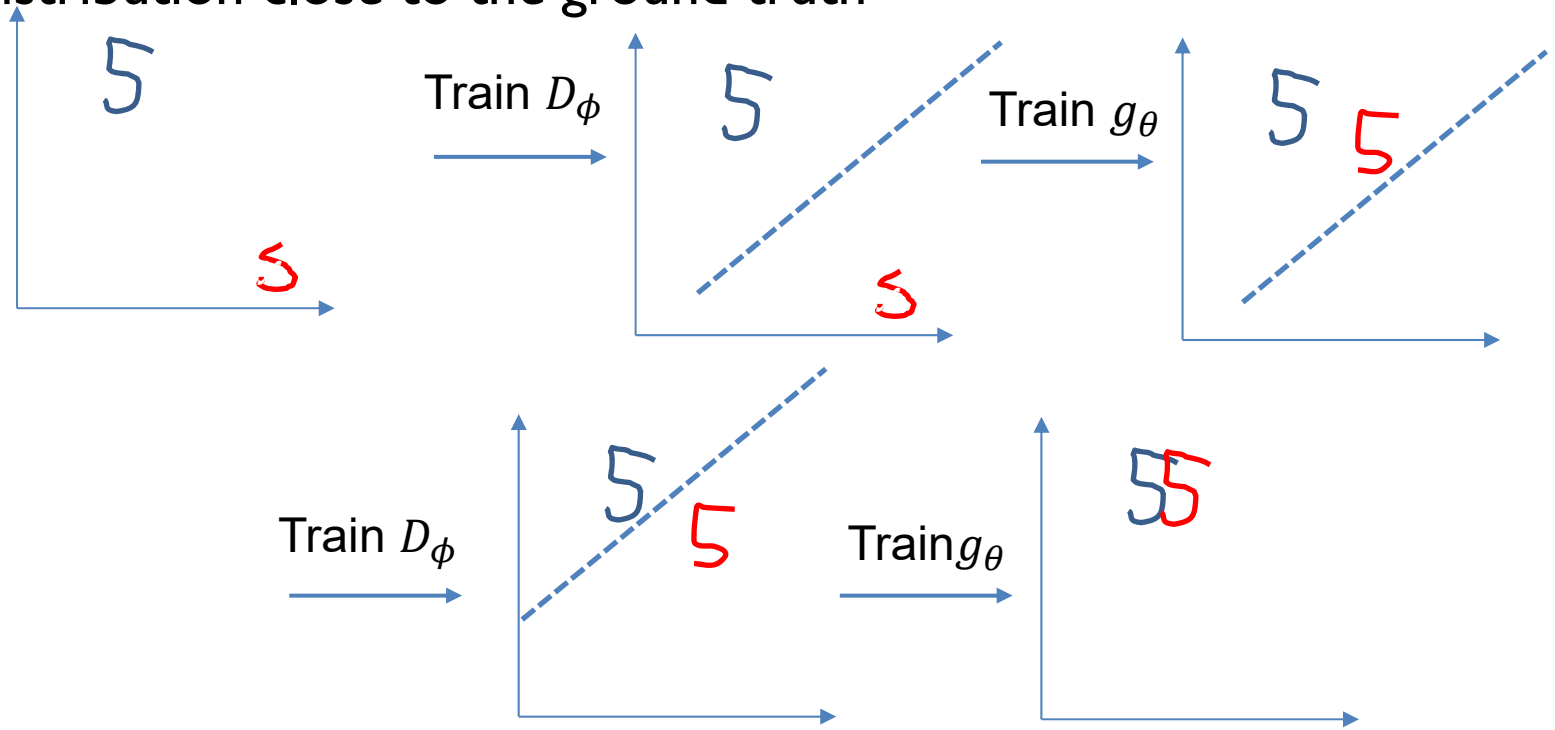
- Discriminator is presented alternatively with true (x) and fake ($\hat{x} = g_\theta(z)$)data



D_ϕ and g_θ are typically MLPs/Deep CNNs/…

GAN – Adversarial training as binary classification
Intuition - Training

- Algorithm alternates between optimizing D_ϕ (separate true and generated data) and g_θ (generate data as close as possible to true examples) – Once trained, G should be able to generate data witha distribution close to the ground truth



GANs - Adversarial training as binary classification
Loss function (Goodfellow et al. 2014)

- $x \sim p_x(x)$ distribution over data x
- $z \sim p_z(z)$ prior on z, usually a simple distribution (e.g. Normal distribution)
- Loss
  - $\min_{\theta} \max_{\phi} L(D_\phi, g_\theta) = E_{x \sim p_x(x)}[log D_\phi(x)] + E_{z \sim p_z(z)}[\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)]$
    - $g_\theta : R^q \rightarrow R^n$ mapping from the latent (z) space to the data (x) space
    - $D_\phi : R^n \rightarrow [0,1]$ probability that x comes from the data rather than from the generator g_θ
    - If g_θ is fixed, $L(D_\phi, g_\theta)$ is a classical binary cross entropy for D_ϕ, distinguishing real and fake examples
  - Note:
    - Training is equivalent to find D_{ϕ^*}, g_{θ^*} such that
      - $D_{\phi^*} \in arg \max_{\phi} L(D_\phi, g_{\theta^*})$ and $g_{\theta^*} \in arg \min_{\theta} L(D_{\phi^*}, g_\theta)$
      - Saddle point problem
        - instability
- Practical training algorithm
  - Alternates optimizing (maximizing) w.r.t. D_ϕ optimizing (minimizing) w.r.t. g_θ

Adversarial training as binary classification
Training GANs

- Training alternates optimization (SGD) on D_ϕ and g_θ
  - In the alternating scheme, g_θ usually requires more steps than D_ϕ+ different batch sizes
- It is known to be highly unstable with two pathological problems
  - Oscillation: no convergence
  - Mode collapse: g collapses on a few modes only of the target distribution (produces the same few patterns for all z samplings)
  - Low dimensional supports (Arjovsky 2017): $p_x$and p_θ may lie on low dimensional manifold that do not intersect.
    - It is then easy to find a discriminator, without p_θ close to p_x
  - Lots of heuristics, lots of theory, but
    - Behavior is still largely unexplained, best practice is based on heuristics

GAN- Adversarial training as binary classification Equilibrium analysis (Goodfellow et al. 2014)

- The seminal GAN paper provides an analysis of the solution that could be obtained at equilibrium
- Let us define
  - $L(D_\phi, g_\theta) = E_{x\sim p_x(x)}[logD_\phi(x)] + E_{x\sim p_\theta(x)}[\log\left(1 - D_\phi(x)\right)]$
    - with $p_x(x)$ the true data distribution and $p_\theta(x)$ the distribution of generated data
      - Note that this is equivalent to the $L(D, G)$ definition on the slide before
- If g_θ and D_ϕ have sufficient capacity
  - Computing $\underset{\theta}{argmin}\ g^* = \underset{\theta}{argmin} \max_\phi L\left(D_\phi, g_\theta\right)$
  - Is equivalent to compute
    - $g^* = argmin_\theta D_{JS}(p_x, p_\theta)$ with $D_{JS}(,)$ the Jenson-Shannon dissimilarity measure between distributions
    - The loss function of a GAN quantifies the similarity between the real sample distribution and the generative data distribution by JSD when the discriminator is optimal

GAN- Adversarial training as binary classification
Equilibrium analysis (Goodfellow et al. 2014)

- If the optimum is reached
  - $D_\phi(x) = \frac{1}{2}$ for all $x \rightarrow$ Equilibrium
- In practice equilibrium is never reached
- Note
  - Maximize $\log\left(D_\phi\left(g_\theta(z)\right)\right)$ instead of minimizing $\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)$ provides stronger gradients and is used in practice, i.e. $\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)$ is replaced by $-\log\left(D_\phi\left(g_\theta(z)\right)\right)$

GAN equilibrium analysis (Goodfellow et al. 2014)
Prerequisite KL divergence

- Kullback Leibler divergence
  - Measure of the difference between two distributions p and q
  - Continuous variables
    - $D_{KL}(p(y)||q(y)) = \int_y (\log\frac{p(y)}{q(y)})p(y)dy$
  - Discrete variables
    - $D_{KL}(p(y)||q(y)) = \sum_i (\log\frac{p(y_i)}{q(y_i)})p(y_i)$
- Property
  - $D_{KL}(p(y)||q(y)) \geq 0$
  - $D_{KL}(p(y)||q(y)) = 0$ iff $p = q$
    - $D_{KL}(p(y)||q(y)) = -E_{p(y)}\left[log\frac{q(y)}{p(y)}\right] \geq -\log E_{p(y)}\left[\frac{q(y)}{p(y)}\right] \geq 0$
      - where the first inequality is obtained via Jensen inequality
  - note: D_{KL} is asymmetric, symmetric versions exist, e.g. Jensen-Shannon divergence

GAN equilibrium analysis (Goodfellow et al. 2014) - proof

- For a given generator g, the optimal discriminator is
  - $\mathrm{D}^*(x) = \frac{p_X(x)}{p_X(x)+p_\theta(x)}$
    - Let $f(y) = a\,log(y) + b\,log(1-y)$, with $a, b, y > 0$
    - $\frac{df}{dy} = \frac{a}{y} - \frac{b}{1-y}, \frac{df}{dy} = 0 \Leftrightarrow y = \frac{a}{a+b}$ and this is a max
    - $Max_D\, L(D,G) = E_{x\sim p_X(x)}[logD(x)] + E_{\mathrm{x}\sim p_\theta(x)}[\log(1 - D(x))]$ is then obtained for:
      - $\mathrm{D}^*(x) = \frac{p_X(x)}{p_X(x)+p_\theta(x)}$

GAN equilibrium analysis (Goodfellow et al. 2014) - proof

- Let $C(g) = \max_D L(g, D) = L(g, D^*)$
- It si easily verified that:
  - $C(g) = -\log 4 + KL\left(p_X(x); \frac{p_X(x)+p_\theta(x)}{2}\right) + KL\left(p_\theta(x); \frac{p_X(x)+p_\theta(x)}{2}\right)$
  - Since $KL(p; q) \geq 0$ and $KL(p; q) = 0$ iff $p = q$
    - $C(g)$ is minimum for $p_\theta = p_X$ with $D^*(x) = \frac{1}{2}$
    - At equilibrium, GAN training optimises Jenson-Shannon Divergence, $JSD(p; q) = \frac{1}{2}KL\left(p; \frac{p+q}{2}\right) + \frac{1}{2}KL\left(q; \frac{p+q}{2}\right)$ between p_θ and p_X
- Summary
  - The loss function of a GAN quantifies the similarity between the real sample distribution and the generative data distribution by JSD when the discriminator is optimal
- Note
  - $\frac{p_X(x)}{p_\theta(x)} = \frac{p(x|y=1)}{p(x|y=0)} = k\frac{p(y=1|x)}{p(y=0|x)} = k\frac{D^*(x)}{1-D^*(x)}$ with $k = \frac{p(y=0)}{p(y=1)}$
  - The discriminator is used to implicitely measure the discrepancy between the distributions

Training GANs

- Training alternates optimization on D and G
  - In the alternating scheme, G usually requires more steps than D
- It is known to be highly unstable with two pathological problems
  - Oscillation: no convergence
  - Mode collapse: G collapses on a few modes only of the distribution (produces the same few patterns for all z samplings)
  - Low dimensional supports (Arjovsky 2017): $p_{data}$and p_g may lie on low dimensional manifold that do not intersect. It is then easy to find a discriminator, without training p_g to be close to p_{data}
  - Very large number of papers offering tentative solutions to these problems
    - e.g. recent developments concerning Wasserstein GANs (Arjovsky 2017)
  - This remain difficult and heuristic although various explanation heve been developped (e.g. stability of the generator – related to optimal transport or dynamics of the network – see course on ODE)
- Evaluation
  - What could we evaluate?
  - No natural criterion
    - Very often beauty of the generated patterns!

Objective functions

- A large number of alternative objective functions have been proposed, we will present two examples
  - Least Square GANs
  - Wasserstein GANs

Objective functions – Least Square GANS (Mao et al. 2017)

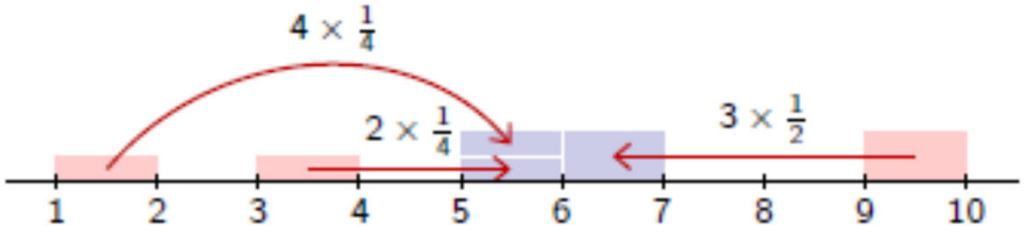
- If a generated sample is well classified but far from the real data distribution, there is no reason for the generator to be updated
- LS-GAN replaces the cross entropy loss with a LS loss which penalizes generated examples by moving them close to the real data distribution.
- The objective becomes
  - $L(D) = E_{x\sim p_X(x)}\left[(D(x)-b)^2\right] + E_{z\sim p_z(z)}\left[(D(g(z))-a)^2\right]$
  - $L(g) = E_{z\sim p_z(z)}\left[\left(D\left(g(z)\right)-c\right)^2\right]$
  - Where a, b are constants respectively associated to generated and real data and c is a value that g wants D to believe for the generated data.
  - They use for example $a = 0, b = c = 1$

Objective functions – Wasserstein GANs (Arjovski et al. 2017)

- Arjovski advocates that D_{KL} (or D_{JS}) might not be appropriate
- They suggest using the Wasserstein distance between the real and generated distributions (also known as Earth Moving Distance or EMD)
  - Intuitively, this is the minimum mass displacement to transform one distribution to the other
- Wassertein distance is defined as
  - $W(p_X, p_\theta) = \inf_{\gamma \in \Pi(p_X, p_\theta)} E_{(x,x')\sim\gamma}[\| x - x' \|]$
    - where $\Pi(p_X, p_\theta)$ is the set of distributions over X^2 , with $X \subset R^n$ the space of data, whose marginals are respectively $p_X(x)$ and $p_\theta(x)$, $\| x - x' \|$ is the Euclidean norm.
  - Intuitively,
    - $W(,)$ is the minimum amount of work required to transform $p_X(x)$ to $p_\theta(x)$ – see next slide
    - it makes sense to learn a generator g minimizing this metric
      - $g^* = argmin_G W(p_X, p_\theta)$

Wasserstein GANs (Arjovski et al. 2017)

- Earth Mover distance illustration
  - 2 distributions (pink (μ) and blue (μ'))
  - An elementary rectangle weights ¼
  - The figure illustrates the computation of $W(\mu, \mu')$, the Wasserstein distance between pink and blue: this is the earth mover distance to transport pink on blue. This is denoted as $\mu' = \#\mu$, μ' is the push forward of μ



$$\mu = \frac{1}{4}\mathbf{1}_{[1,2]} + \frac{1}{4}\mathbf{1}_{[3,4]} + \frac{1}{2}\mathbf{1}_{[9,10]} \qquad \mu' = \frac{1}{2}\mathbf{1}_{[5,7]}$$

$$W(\mu, \mu') = 4 \times \frac{1}{4} + 2 \times \frac{1}{4} + 3 \times \frac{1}{2} = 3$$

Fig. from F. Fleuret 2018

Objective functions – Wasserstein GANs (Arjovski et al. 2017)

- Let x and y respectively denote the variables from the source and the target distributions
- $p_X(x) = \int_y \gamma(x, y)dy$ is the amount of mass to move from x,

  $p_\theta(y) = \int_y \gamma(x, y)dx$ is the amount of mass to move to y
- Transport is defined as the amont of mass multiplied by the distance it moves, then the transport cost is: $\gamma(x, y). \| x - y \|$ and the minimum transport cost is $\inf_{\gamma \in \Pi(p_X, p_\theta)} E_{(x,x') \sim \gamma}[\| x - x' \|]$

Wasserstein GANs (Arjovski et al. 2017)
Optimal Transport interpretation

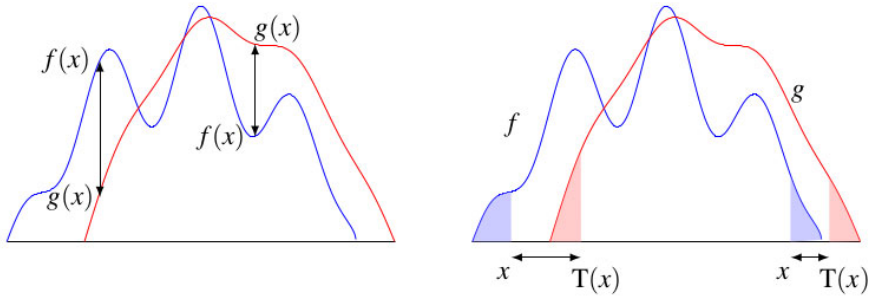


Fig. Santambrogio, 2015

- Left: standard ways to compute distance between functions (point distance)
- Right: Optimal Transport way
  - Seek the best map T which transports the blue distribution on the red one.
  - The smaller T , the closest f and g.
- Wasserstein distance is defined as $W(f,g) = \inf_{T|T\#f=g} \int_x |T(x) - x| dx$
- Which can be translated in:
  - "You look at all the ways to transport f on g with a map T (denoted $T\#f = g$).
  - For a given such transport map T, you look at the total distance you traveled on the x axis , that is $\int_x |T(x) - x| dx$.
  - Among all these transport maps, you look at the one which achieves the optimal (i.e. minimal) distance traveled. This minimal distance is called the Wasserstein distance between f and g."

Wasserstein GANs (Arjovsky et al. 2017)

- The $W(,)$ definition does not provide an operational way for learning G
- Arjovsky uses a duality theorem from Kantorovitch and Rubinstein, stating the following result:
  - $W(p_X, p_\theta) = \sup_{\|f\|_L \leq 1} E_{x \sim p_X}|f(x)| - E_{x \sim p_\theta}|f(x)|$
  - Where $f: X \to R$ is 1-Lipchitz, i.e. $|f(x) - f(y)| < 1 \parallel x - y \parallel, \forall\, x, y \in X$
    - i.e. $\parallel f \parallel_L \leq 1$ denotes the 1-Lipchitz functions
- Implementation
  - Using this result, one can look for a generator g and a critic f_w:
    - $g^* = argmin_g W(p_X, p_\theta)$
    - $g^* = argmin_g \sup_{\|f\|_L} E_{x \sim p_X}|f_w(x)| - E_{x \sim p_\theta}|f_w(x)|$
    - $g^* = argmin_g \sup_{\|f\|_L} E_{x \sim p_X}|f_w(x)| - E_{z \sim p_z}|f_w(G(z))|$
    - f_w is implemented via a NN with parameters w, it is called a critic because it does not classify but scores its inputs
    - In the original WGAN, f_w is made 1-Lipchitz by clipping the weights (Arjovski et al. 2017)
      - Better solutions were developed later

Wasserstein GANs (Arjovski et al. 2017)

- Algorithm
  - Alternate
    - Optimize f_w
    - Optimize g_θ

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, \ldots, n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of priors.
    $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

From Arjovski 2017

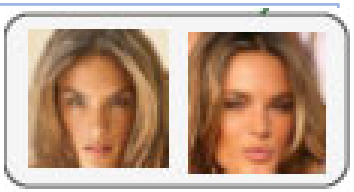
GANs examples
Deep Convolutional GANs (Radford 2015) - Image generation

- LSUN bedrooms dataset - over 3 million training examples

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

Fig. Radford 2015

Gan example
MULTI-VIEW DATA GENERATION WITHOUT VIEW SUPERVISION (Chen 2018 - Sorbonne)



- Objective
  - Generate images by disantangling content and view
    - Eg. Content 1 person, View: position, illumination, etc
  - 2 latent spaces: view and content
    - Generate image pairs: same item with 2 different views
    - Learn to discriminate between generated and real pairs

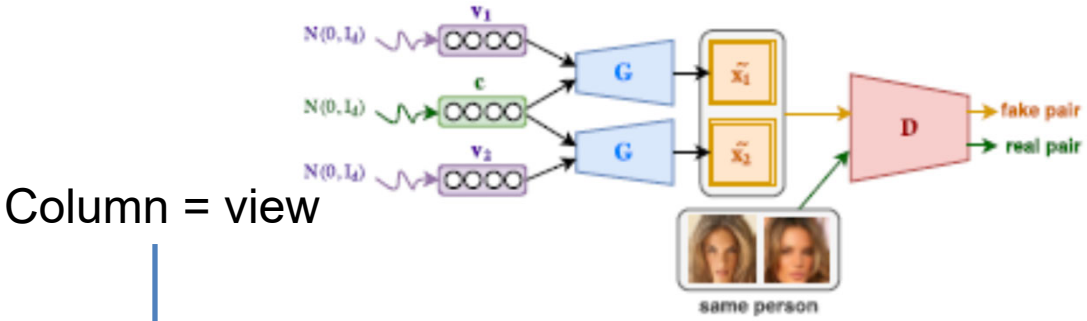
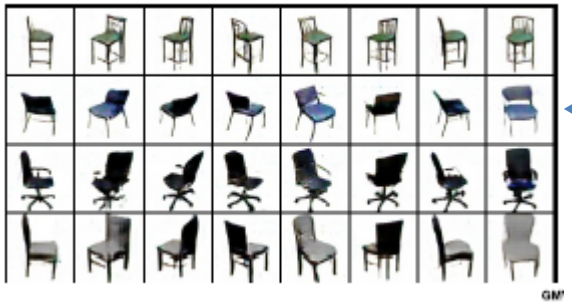


Fig. Chen 2018

Column = view

Column = view



1 row = 1 content

Conditional GANs (Mirza 2014)

- The initial GAN models distributions by sampling from the latent Z space
- Many applications require to condition the generation on some data
  - e.g.: text generation from images, in-painting, super-resolution, etc
- (Mirza 2014) proposed a simple extension of the original GAN formulation to a conditional setting:
  - Both the generator and the discriminator are conditioned on variable y – corresponding to the conditioning data

$$\min_g \max_D L(D,g) = E_{x\sim p_X(x)}[logD(x|y)] + E_{z\sim p(z)}[\log\Big(1 - D\big(g(z|y)\big)\Big)]$$

Conditional GANs (Mirza 2014)

$$\min_g \max_D L(D,g) = E_{x\sim p_X(x)}[logD(x|y)] + E_{z\sim p(z)}[\log\Big(1 - D\big(g(z|y)\big)\Big)]$$

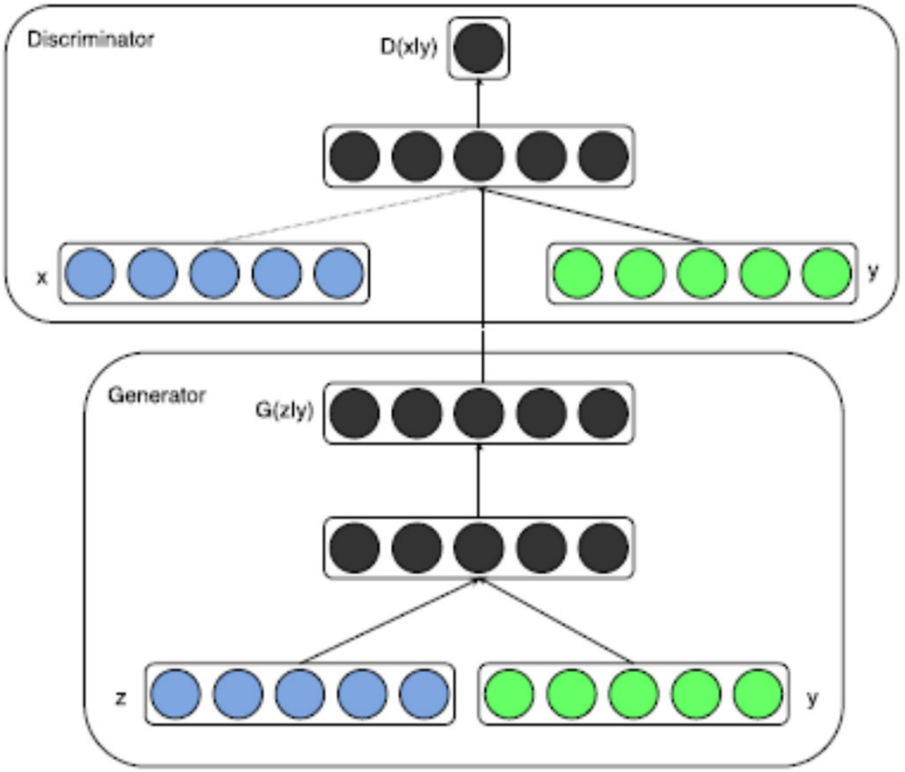


Fig. (Mirza 2014)

Conditional GANs example
Generating images from text (Reed 2016)

- Objective
  - Generate images from text caption
  - Model: GAN conditioned on text input
- Compare different GAN variants on image generation
- Image size 64x64

Fig. from Reed 2016

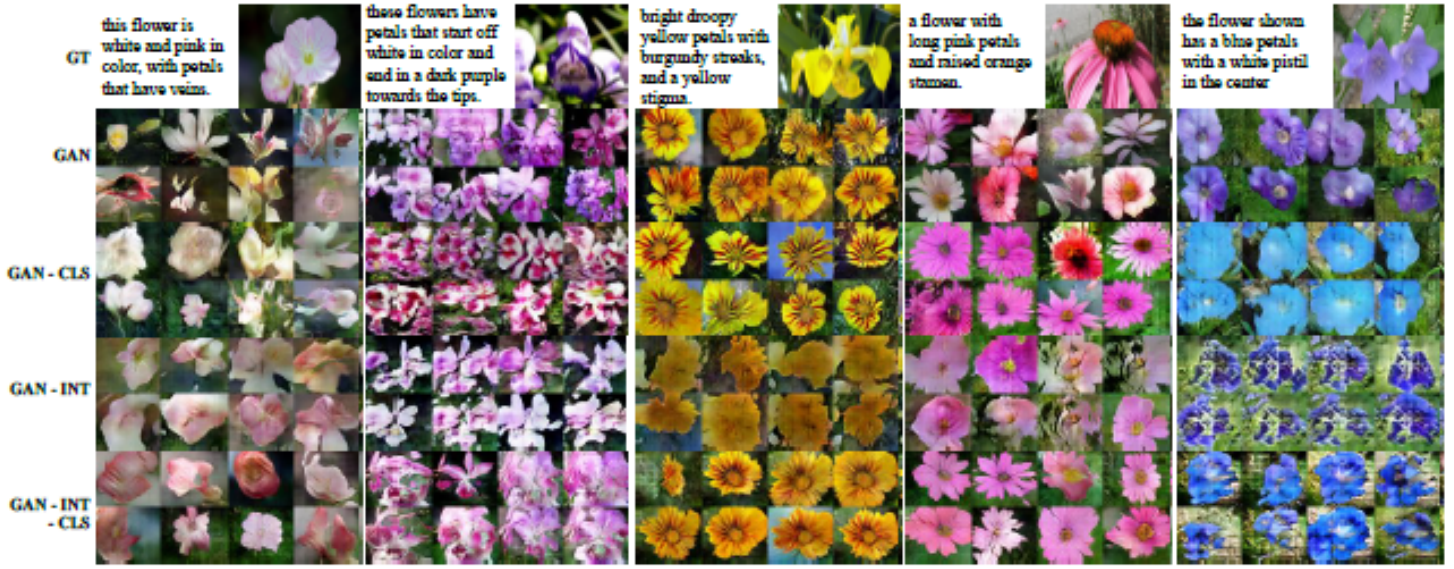
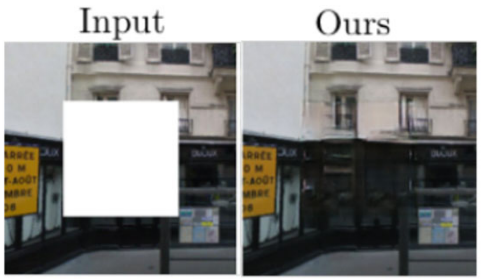


Figure 4. Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved.

Conditional GANs example – Pix2Pix
Image translation with cGANs (Isola 2016)

- Objective
  - Learn to « translate » images for a variety of tasks using a common framework
    - i.e. no task specific loss, but only adversarial training + conditioning
  - Tasks: semantic labels -> photos, edges -> photos, (inpainting) photo and missing pixels -> photos, etc



Conditional GANs example – Pix2Pix
Image translation with cGANs (Isola 2016)

- Loss function
  - Conditional GAN
- $\min_g \max_D L(D,g) = E_{\substack{x\sim p_X(x) \\ y\sim p(y)}}[logD(x,y)] + E_{\substack{z\sim p(z) \\ y\sim p(y)}}[\log\big(1 - D(g(z,y),y)\big)]$
  - Note: the formulation is slightly different from the conditional GAN model of (Mirza 2014): it makes explicit the sampling on y , but this is the same loss.
- This loss alone does not insure a correspondance between the conditioning variable y and the input data x
  - They add a loss term, its role is to keep the generated data $g(z,y)$ « close » to the conditioning variable y
  - $L_{L^1}(g) = E_{x,y,z}\|x - g(y,z)\|_1$
    - Where $\|.\|_1$ is the L^1 norm
- Final loss
  - $\min_g(\max_D L(D,g) + \lambda L_{L^1}(g))$

Conditional GANs example – Pix2Pix
Image translation with cGANs – Examples (Isola 2016)

Figure 15: Example results of our method on automatically detected edges→handbags, compared to ground truth.

Fig. (Isola 2016)

Conditional GANs example – Pix2Pix
Image translation with cGANs - Examples - (Isola 2016)

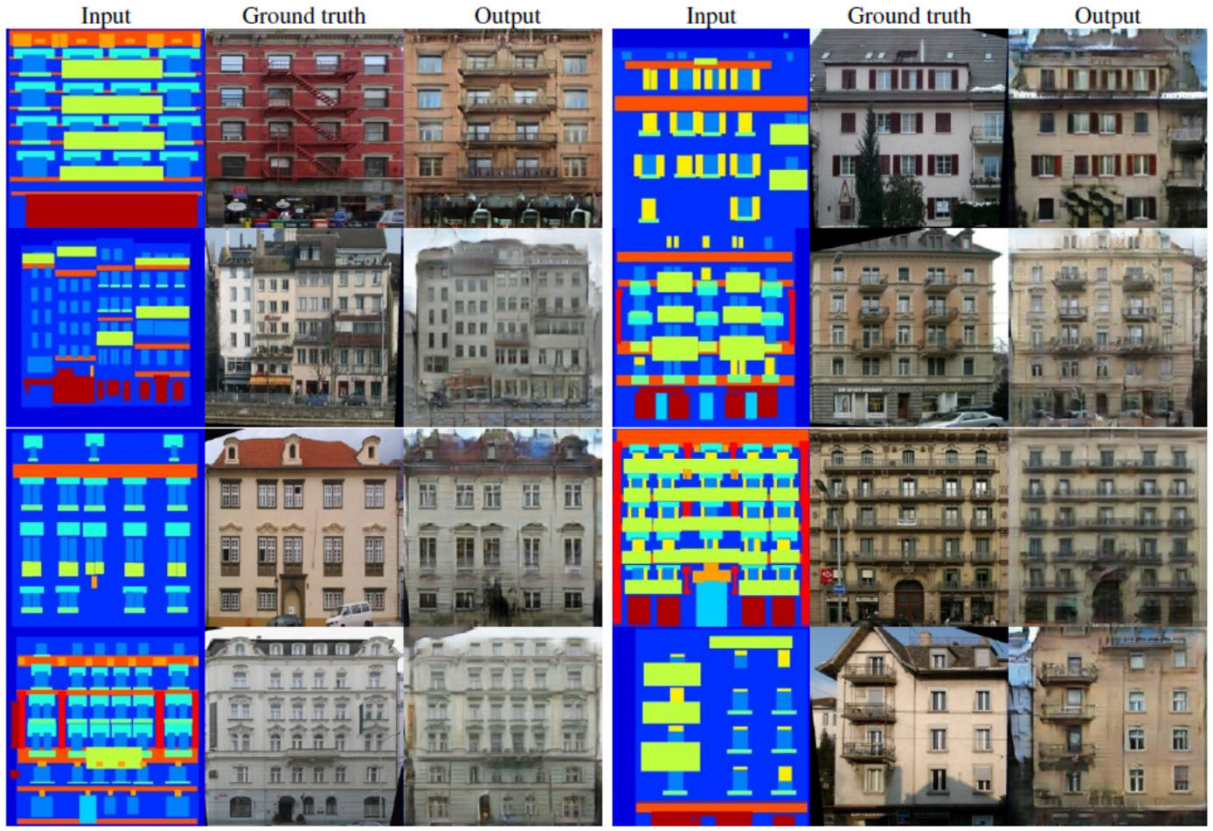


Figure 13: Example results of our method on facades labels→photo, compared to ground truth.

Fig. (Isola 2016)

Conditional GANs example – Pix2Pix
Image translation with cGANs – Examples - (Isola 2016)

- Failure examples

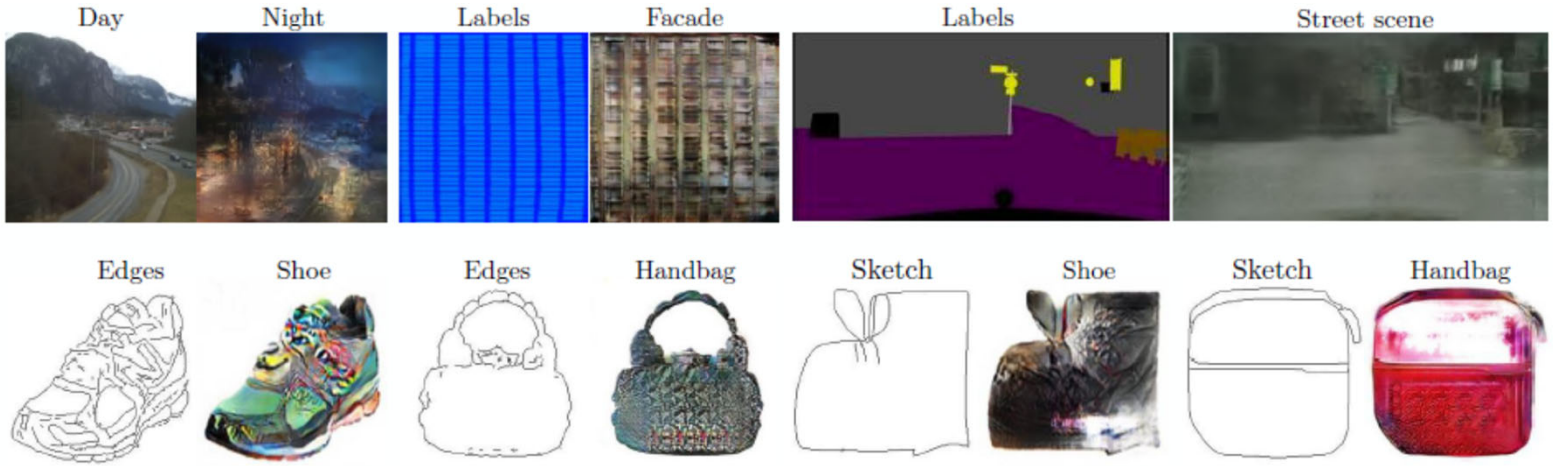


Figure 20: Example failure cases. Each pair of images shows input on the left and output on the right. These examples are selected as some of the worst results on our tasks. Common failures include artifacts in regions where the input image is sparse, and difficulty in handling unusual inputs. Please see `https://phillipi.github.io/pix2pix/` for more comprehensive results.

Fig. (Isola 2016)

Cycle GANs (Zhu 2017)

- Objective
  - Learn to « translate » images without aligned corpora
    - 2 corpora available with input and output samples, but no pair alignment between images
  - Given two unaligned corpora, a conditional GAN can learn a correspondance between the two distributions (by sampling the two distributions), however this does not guaranty a correspondance between input and output
- Approach
  - (Zhu 2017) proposed to add a « consistency » constraint similar to back translation in language
    - This idea has been already used for vision tasks in different contexts
    - Learn two generative mappings
      - $g: X \rightarrow Y$ and $f: Y \rightarrow X$ such that:
      - $f \circ g(x) \simeq x$ and g $\circ f(y) \simeq$y
    - and two discriminant functions D_Y and D_X

Cycle GANs (Zhu 2017)

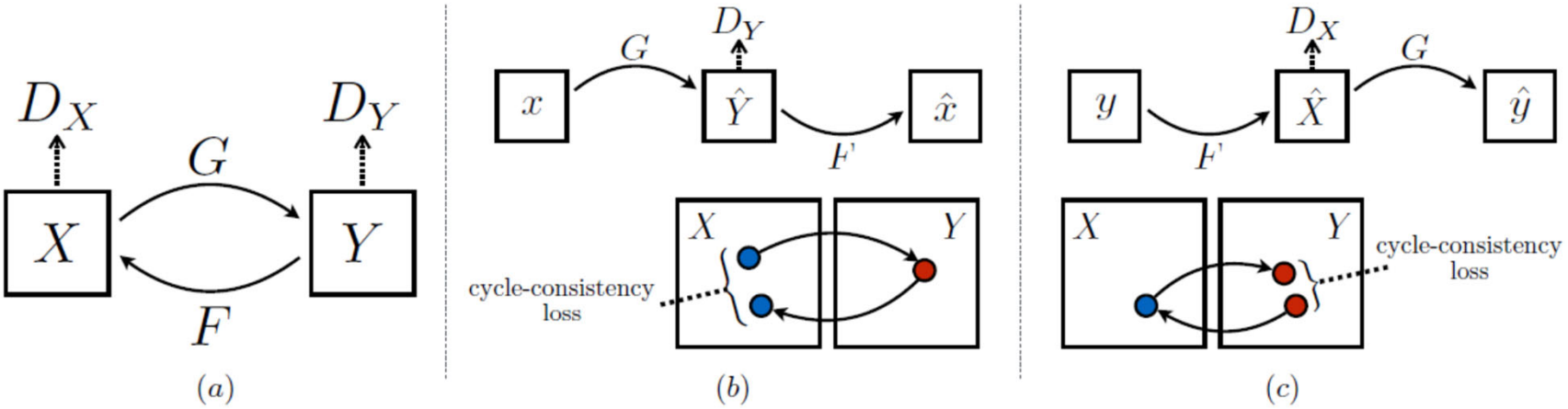


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, we introduce two "cycle consistency losses" that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Fig (Zhu 2017)

Cycle GANs (Zhu 2017)

- Training
  - The loss combines two conditional GAN losses (g, D_Y) and (f, D_X) and a cycle consistency loss
  - $L_{cycle}(f, g) = E_{p_X(x)}[\|f(g(x)) - x)\|_1] + E_{p_{data}(y)}[\|g(f(y)) - y)\|_1]$
  - $L(g, D_Y, f, D_X) = L(g, D_Y) + L(f, D_X) + L_{cycle}(f, g)$
  - Note: they replaced the usual $L(g, D_Y)$ and $L(f, D_X)$ term by a mean square error term, e.g.:
    - $L(g, D_Y) = E_{p_y(y)}[(D_Y(y) - 1)^2] + E_{p_X(x)}[D_Y(G(x))]$

Cycle GANs (Zhu 2017)

- Examples

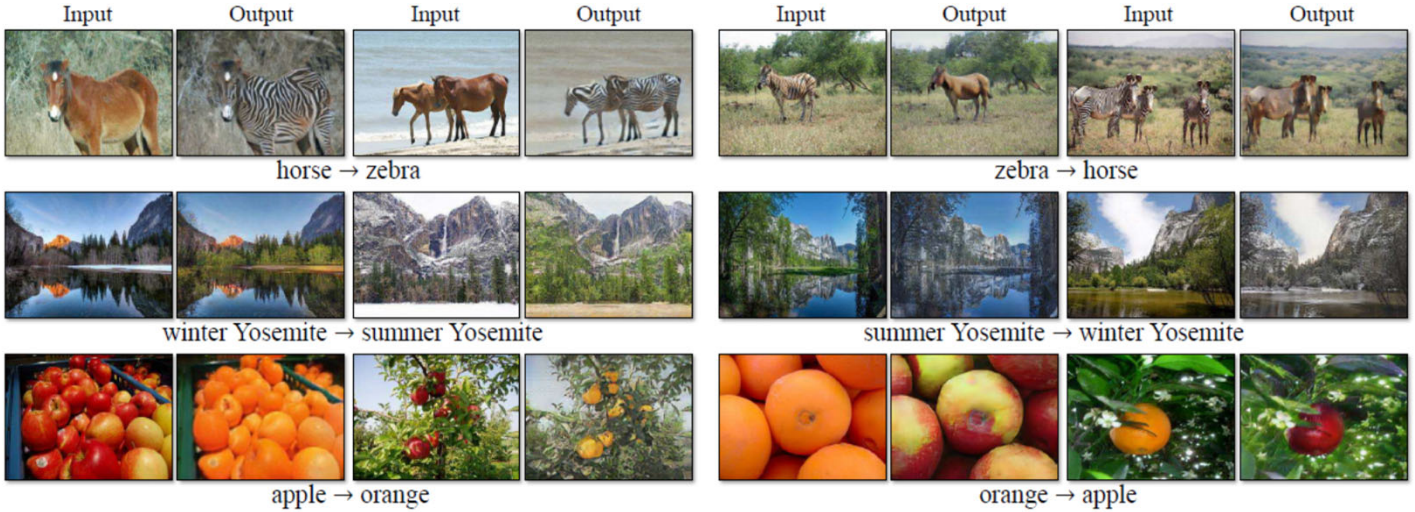
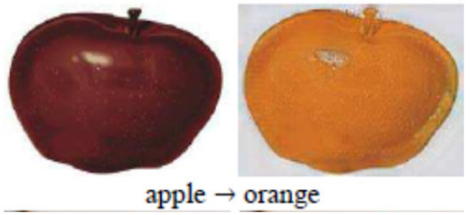


Figure 7: Results on several translation problems. These images are relatively successful results – please see our website for more comprehensive results.

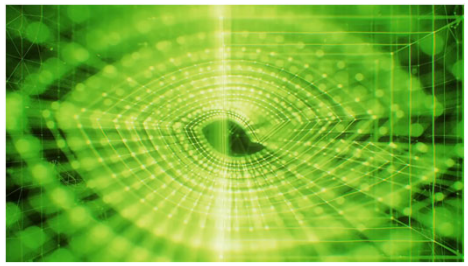
- Failures

Fig (Zhu 2017)



(Karras et al. 2019) – Style GAN

- (Karras et al. 2019) – Style GAN
- Noyte: now (2020) StyleGAN3: https://nvlabs.github.io/stylegan3/
- https://nvlabs.github.io/stylegan2/versions.html



Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- Recall batch normalization
  - $BN(x) = \gamma\left(\frac{x-\mu(x)}{\sigma(x)}\right) + \beta$, here all the quantities are vectors (or tensors) of the appropriate size
  - The mean for channel c is computed as:
    - $\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$
    - With N the number of images in the batch, H the height and W the width, i.e. x is of shape $[N, C, H, W]$
    - γ and β are trainable parameters that are different for each channel
    - BN averages over all the images in the batch
      - i.e. all the images in the batch are averaged around a single « style »

Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- Adaptive Instance Normalization (Huang 2017)
  - Idea: inject through the linear transformation defined by γ, β the feature statistics from another image (e.g. its style)
  - Let x (content) and y (style) two images or image transformations
    - $AdaIN(x, y) = \sigma(y)\left(\frac{x-\mu(x)}{\sigma(x)}\right) + \mu(y)$
    - This simply replaces the the channel-wise statistics of x by those of y
    - AdaIN can normalize the style of each individual sample to a target style

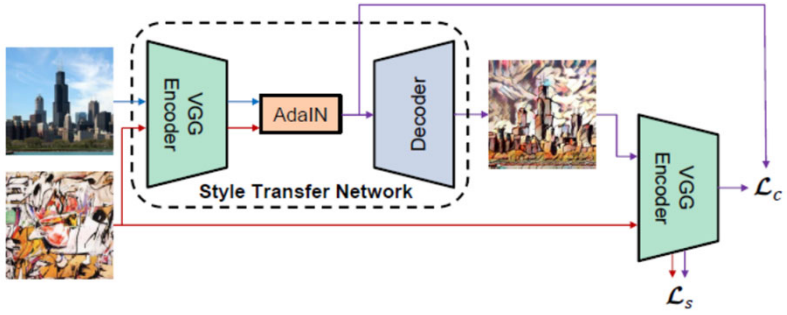


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss $\mathcal{L}_c$ (Equ. 12) and a style loss $\mathcal{L}_s$ (Equ. 13).

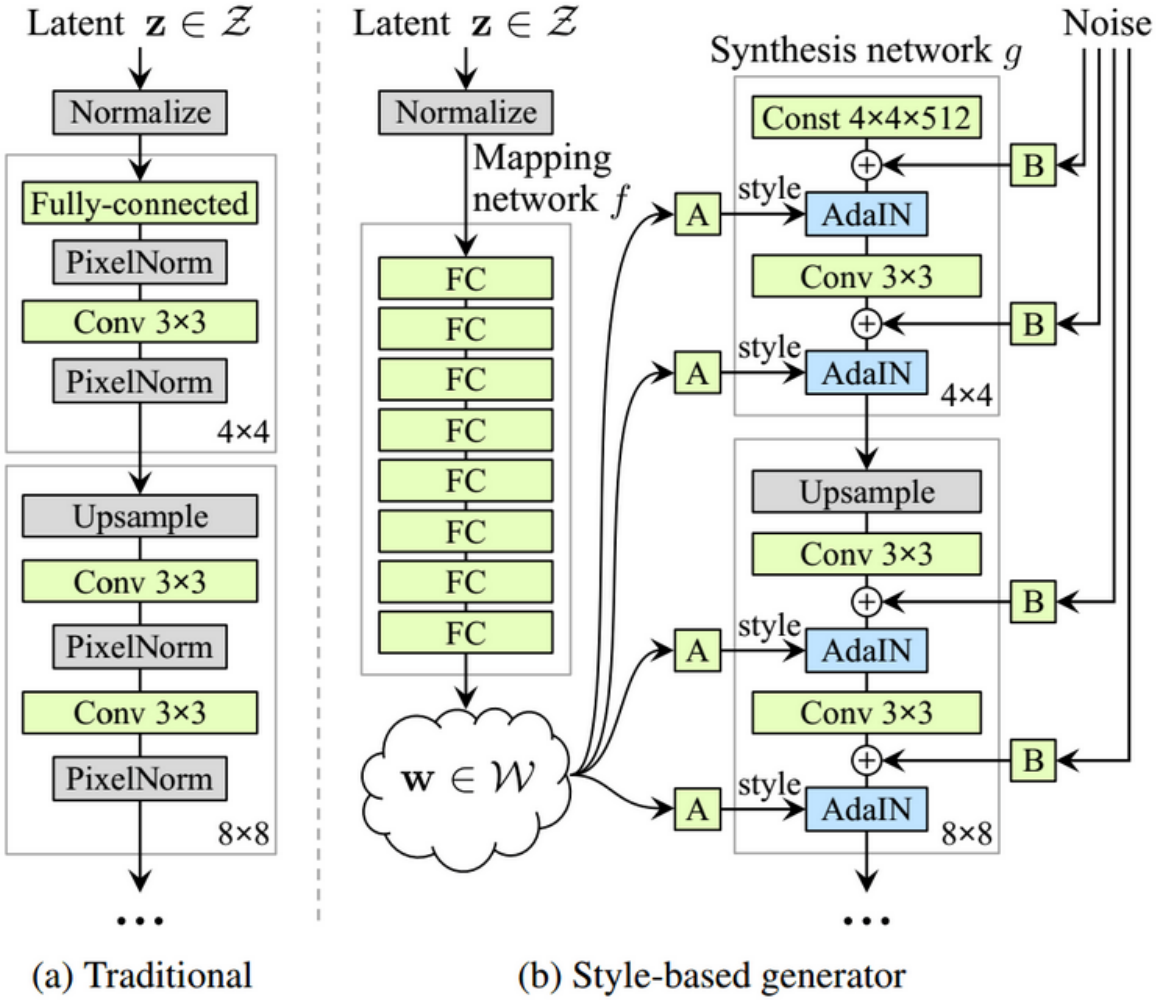
(Huang 2017)

Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- (Huang 2017) examples

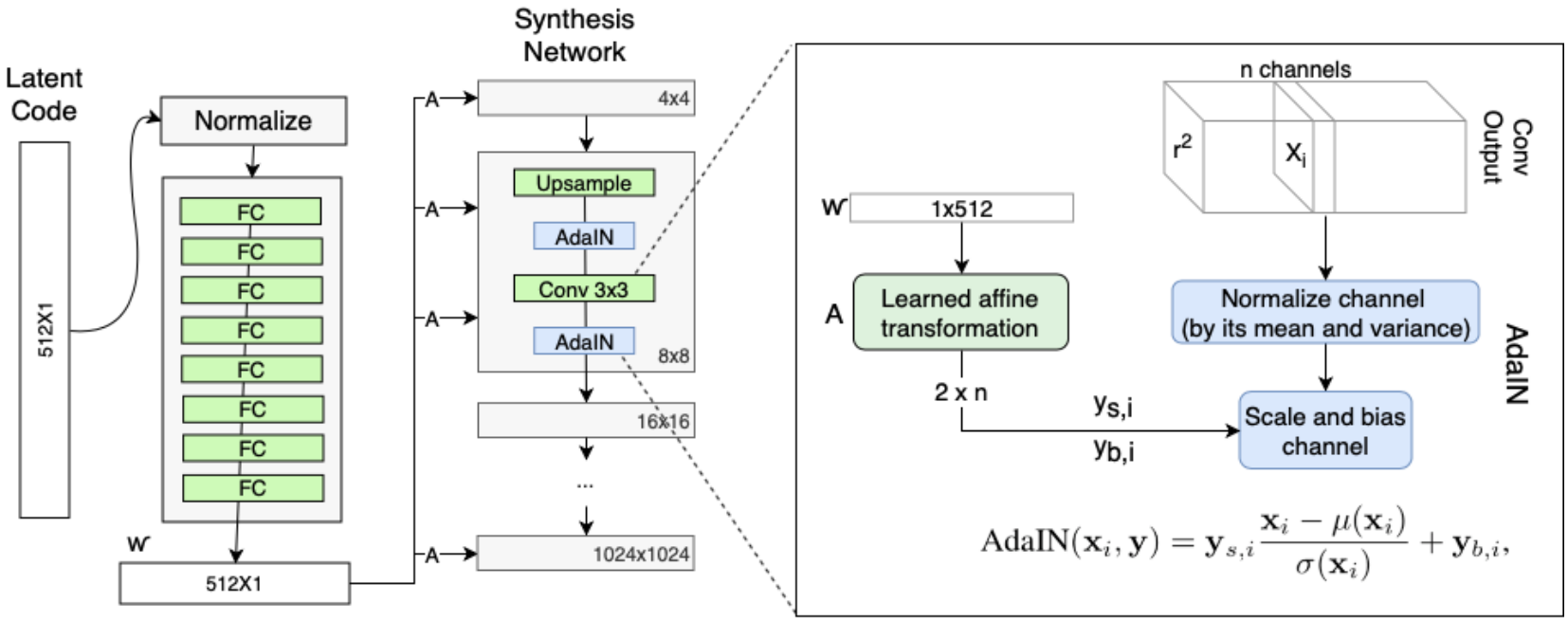
Architecture of Style Gan

Karras et al. 2019



- A mapping network generates a representation vector w
- Affine transformations (A) are trained to compute λ and β vectors for different resolution of the image generator from w – this induces different styles for each resolution
- Noise input are single channel images consisting of uncorrelated Gaussian noise – a single noise image is broadcasted to all the feature maps – this induces stochastic variations

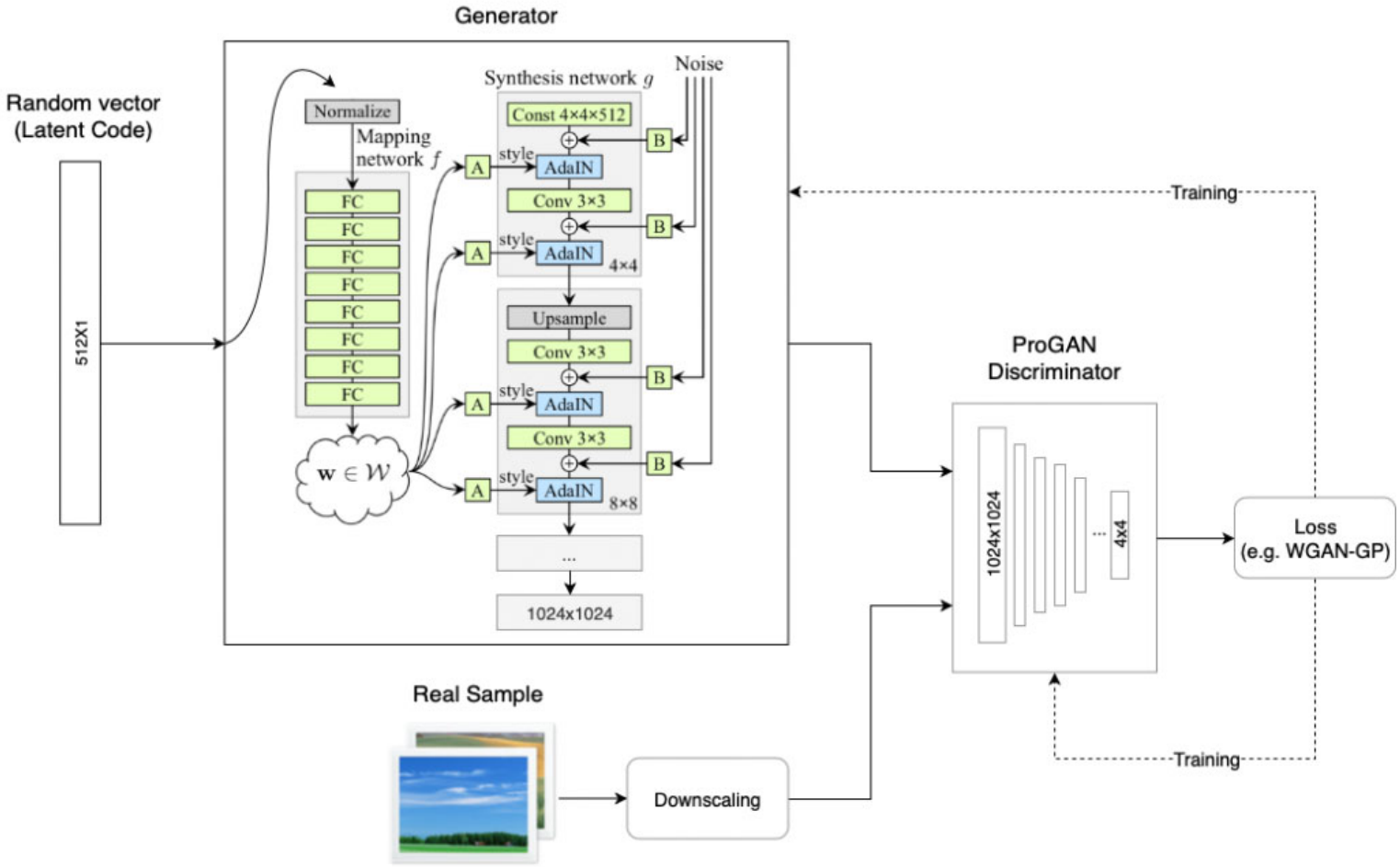
Architecture of Style Gan



- Affine transformations computed from w

https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431

Architecture of Style Gan



- Global architecture of StyleGAN

https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431

GANs

- Making GANs work is usually hard
- All papers are full of technical details, choices (architecture, optimization, etc.), tricks, not easy to reproduce.

Diffusion models

Diffusion models

- Diffusion models emerged in 2019, gained momentum in 2021
- As in 2022, diffusion models are used in several popular large scale models for text to image generation
  - e.g. Imagen https://imagen.research.google/, stable diffusion https://stablediffusionweb.com/, Dall-e-2 https://openai.com/dall-e-2/
  - Generative modeling tasks
    - Continuous space models: Image generation, super resolution, image editing, segmentation; etc.
    - Discrete space models, e.g. applications to text generation
- Several approaches including
  - Discrete time models
    - Denoising Diffusion Probabilistic Models (DDPMs)
    - Score based Generative Models (SGM)
  - Time continuous models
    - Score Based Models with Differential Equations (SGMdiffeq)

Diffusion models

- Diffusion models implement the following idea
  - Forward diffusion

    Gradually add noise to an input image until one get a fully noisy image
  - Reverse denoising
    - Generate data from the target distribution
    - Sample from the noise space and reverse the forward process

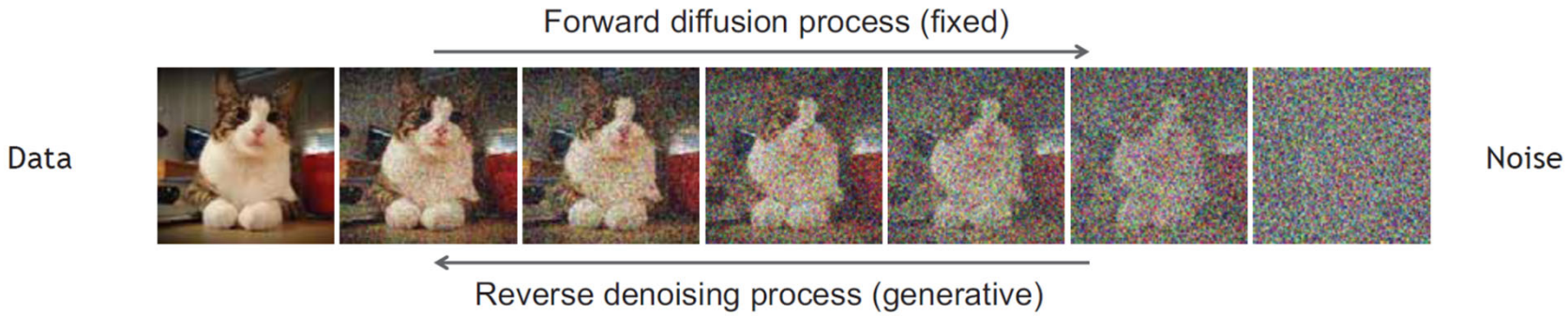


Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models - DDPM

- DDPM are based on two Markov chains
  - A forward chain that adds noise to data –> Forward process
    - Hand designed: transforms any data distribution into a simple prior distribution – here we will use a standard Gaussian for the prior
  - A reverse chain that converts noise to data –> Reverse process
    - The forward chain is reversed by learning **transition kernels** parameterized by neural networks
    - New data are generated by sampling from the simple prior, followed by ancestral sampling through the reverse Markov chain

Denoising Diffusion Probabilistic Models
Forward process

- Notations
  - data distribution $x_0 \sim q(x_0)$
  - The forward MC generates a sequence of random variables $x_1, x_2, \dots, x_T$ starting at x_0 with **transition kernel** $q(x_t|x_{t-1})$
  - $q(x_T)$ is close to a prior distribution $\pi(x)$, e.g. gaussian distribution with fixed mean and variance

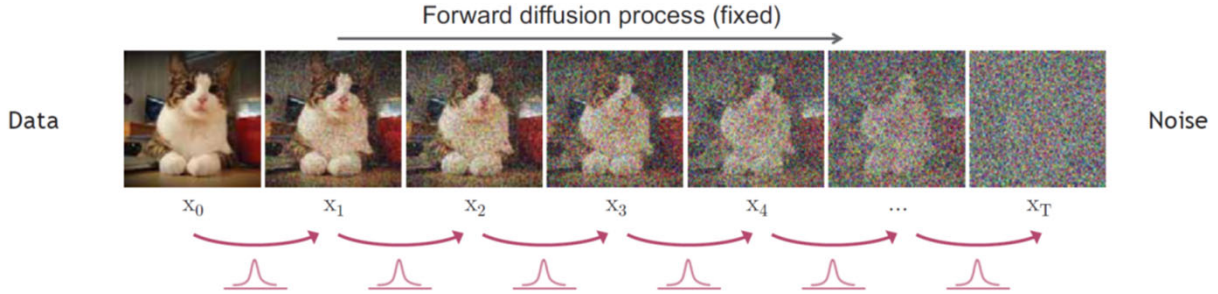


Fig. Kreis et al. 2022

  - A typical design for the kernel is a gaussian perturbation $q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}; \beta_t I\right)\ \forall t \in \{1, \dots, T\}$
    - I is the identity matrix, with the same size as image x_0, $\beta_t \in (0,1)$ is a variance parameter hand fixed or learned, we consider it hand fixed here.
      - β_t is chosen so that $\beta_t < \cdots < \beta_T$, e. g. $T = 2000, \beta_1 = 10^{-4}, \beta_T = 10^{-2}$ with a linear increase
    - Other types of kernels (than gaussians) could be used
- The forward diffusion process is then defined as

$$\mathbf{x_0} \sim \mathbf{q(x_0)},\ q(x_1, \dots, x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}),\ \mathbf{q(x_t|x_{t-1})} =$$
$$\mathcal{N}\left(\mathbf{x_t}; \sqrt{\mathbf{1-\beta_t}}\mathbf{x_{t-1}}; \mathbf{\beta_t I}\right)\ \forall t \in \{1, \dots, T\}$$
β_t is a variance hyperparemeter, $\beta_t < \cdots < \beta_T$

Denoising Diffusion Probabilistic Models
Forward process

- Property: the forward process can be sampled at any time t in closed form
  - For the gaussian transition kernel
  - $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_O, (1-\bar{\alpha}_t)I)$ – this is called the **diffusion kernel**
  - with $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
- This allows us to sample $x_t \sim p(x_t)$ using the reparametrization trick
  - Sample $x_0 \sim q(x_0)$ and then sample $x_t \sim q(x_t|x_0)$ (this is called ancestral sampling)

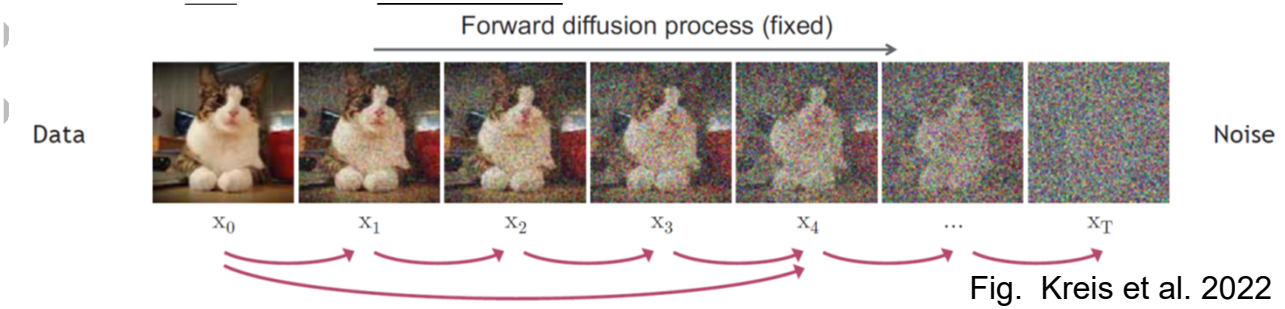


Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models
Forward process

- Illustration of the forward diffusion process – discrete trajectories in the x space

Samples
$x_0 \sim q(x_0)$

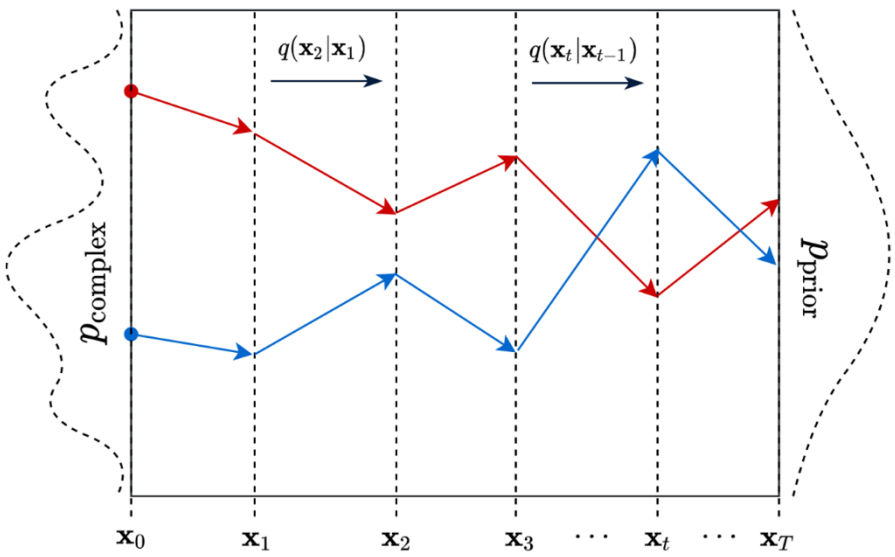


Fig. Ayan Das 2021

Samples
$x_T \sim q(x_T|x_0)$

Derivations

- Closed form for $q(x_t|x_0)$
  - $q(x_t \mid x_0) = N(x_t; \sqrt{(\bar{\alpha}_t)}x_O, (1 - \bar{\alpha}_t)I)$ with $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
    - $x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon$
      - $x_{t-1} = \sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon$
    - $x_t = \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon) + \sqrt{1-\alpha_t}\epsilon$
    - $x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon + \sqrt{1-\alpha_t}\epsilon$
    - $x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon \quad (*)$
    - ……
    - $x_t = \sqrt{\bar{\alpha}_t}x_O + \sqrt{1-\bar{\alpha}_t}\epsilon$

  - (*) Sum of two Gaussians
    - Let x and y two Gaussian random variables with the same dimensionality, $p(x) = \mathcal{N}(\mu_x, \Sigma_x)$ and $p(y) = \mathcal{N}(\mu_y, \Sigma_y)$, then their sum is also Gaussian: $\mathrm{p(x+y)} = \mathcal{N}(\mu_x + \mu_y, \Sigma_x + \Sigma_y)$

Denoising Diffusion Probabilistic Models
Reverse denoising process

- The reverse MC requires the inversion of the Markov chain
  - Sample x_T from a prior distribution $\mathrm{x_T} \sim p(x_T) = \mathcal{N}(x_T; 0, I)$
  - Iteratively sample $\mathrm{x_t} \sim q(x_{t-1}|x_t)$
- In general, $q(x_{t-1}|x_t)$ is untractable
  - One will learn $p_\theta(x_{t-1}|x_t)$ a parametric approximation of $q(x_{t-1}|x_t)$

Denoising Diffusion Probabilistic Models
Reverse denoising process

- The reverse MC is parameterized by
  - A prior distribution $p(x_T) = \mathcal{N}(x_T; 0, I)$
  - A learnable transition kernel $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$
    - $\mu_\theta(x_t, t)$ is typically implemented via a U-Net, $\mu_\theta(x_t, t)$ is the same size as x_t
    - σ_t^2 can be learned, but in (Ho et al. 2020) it is set to β_t

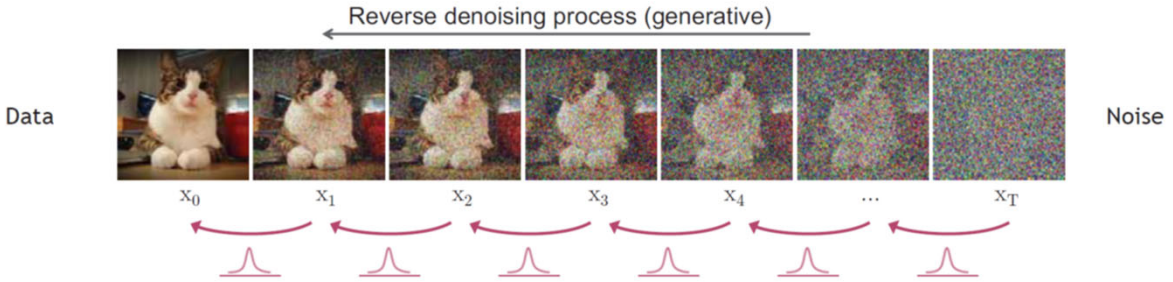


Fig. Kreis et al. 2022

    - Reverse factorization: $p_\theta(x_0, \ldots, x_T) = p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$
      - We can then generate a data sample x_0 by first sampling a noise vector $x_T \sim p(x_T)$ and then iteratively sampling from the learnable transition kernel $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ until $t = 1$

Denoising Diffusion Probabilistic Models
Training

- Training amounts at learning the θ paramaters, $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$
  - Ideally, we would like θ so that the probability assigned by the model to each training sample $p_\theta(x_0)$ is maximized, a.k.a. maximum likelihood $E_{q(x_0)}[p_\theta(x_0)]$
    - However this would require marginalizing over all possible (reverse) trajectories to compute it
    - $p_\theta(x_0) = E_{p_\theta(x_1,\ldots,x_T)}[p_\theta(x_O, x_1, \ldots, x_T)]$
- Instead, one adjusts the parameter θ so that
  - the joint distribution of the reverse MC, $p_\theta(x_0, \ldots, x_T) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$ matches the distribution of the forward process $q(x_0, \ldots, x_T) = q(x_0)\prod_{t=1}^{T} q(x_t|x_{t-1})$
  - This is achieved by minimizing the Kullback-Leibler divergence between the two distributions
  - This is similar to variational auto-encoders, i.e. this amounts at maximizing a lower bound of the log-likelihood (ELBO)
  - But here this operates on the decoder (reverse diffusion process) and not on the encoder like for VAEs

Denoising Diffusion Probabilistic Models
Training – variational lower bound

$$E_{q(x_0)}[-log p_\theta(x_0)] \leq L$$

with

$$L = E_{q(x_0)q(x_{1:T}|x_0)}[-log p_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$$

- Let us examine the three terms of the lower bound L
  - $D_{KL}\big(p(x_T|x_0) \parallel p(x_T)\big)$
    - does not depend on parameters θ and can be ignored during training
  - $p_\theta(x_0|x_1)$
    - is modeled (Ho et al. 2020) as a separate discrete decoder (not detailed here)
  - $D_{KL}\big(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\big)$ - (proofs next slides)
    - $q(x_{t-1}|x_t, x_0)$ is a tractable gaussian distribution
    - $p_\theta(x_{t-1}|x_t)$ is also a gaussian distribution
    - $D_{KL}\big(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\big)$ can then be computed in a closed form
    - It reduces to a simple form

Denoising Diffusion Probabilistic Models
Training

- Let us derive a computational form for $D_{KL}\left(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\right)$
  - $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I\right)$
    - With $\tilde{\mu}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t$ and $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$
      - Recall that $x_t = \sqrt{\bar{\alpha}_t} x_O + \sqrt{(1-\bar{\alpha}_t)}\epsilon$ for $\epsilon \sim \mathcal{N}(0, I)$
    - Then $\tilde{\mu}(x_t, x_0)$ can be rewriten in a simplified form as: $\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$
  - $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$
  - Both $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ being Gaussian, the KL divergence writes as

$$E_{q(x_0),q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)])$$
$$= E_{q(x_0),q(x_t|x_0)}\left[\frac{1}{2\sigma^2}\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|^2\right] + cte$$

  - We would like to train $\mu_\theta(x_t, t)$ to approximate $\tilde{\mu}(x_t, x_0)$

Denoising Diffusion Probabilistic Models
Training

- We would like to train $\mu_\theta(x_t, t)$ to approximate $\tilde{\mu}(x_t, x_0)$
  - i.e. $\mu_\theta(x_t, t)$ must approximate $\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$
  - x_t is available as input at training time, (Ho et al. 2020) propose the following noise prediction parametrization
  - $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$
  - i.e. parametrize the gaussian noise term $\epsilon_\theta(x_t, t)$ to make it predict ϵ from the input x_t at time t
    - Note: parametrizing $\epsilon_\theta(x_t, t)$ is just another way to parametrize $\mu_\theta(x_t, t)$, but it has been found more efficient experimentally

- With this parametrization, the loss term
  - $L_{t-1} = E_{q(x_0), q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)])$ writes
  - $L_{t-1} = E_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,1)}[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\alpha_t)} \left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_O + \sqrt{(1-\bar{\alpha}_t)}\epsilon, t\right)\right\|^2] + \text{Cte}$
- This is simplified in Ho et al. 2020 (heuristic), so that the global loss L writes as

$$L = E_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,1), t \sim \mathcal{U}(1,T)}[\left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_O + \sqrt{(1-\bar{\alpha}_t)}\epsilon, t\right)\right\|^2]$$

  - with $\mathcal{U}(1,T)$ a uniform distribution

Denoising Diffusion Probabilistic Models
Training and sampling algorithms

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Fig. Ho et al 2020

Denoising Diffusion Probabilistic Models Implementation

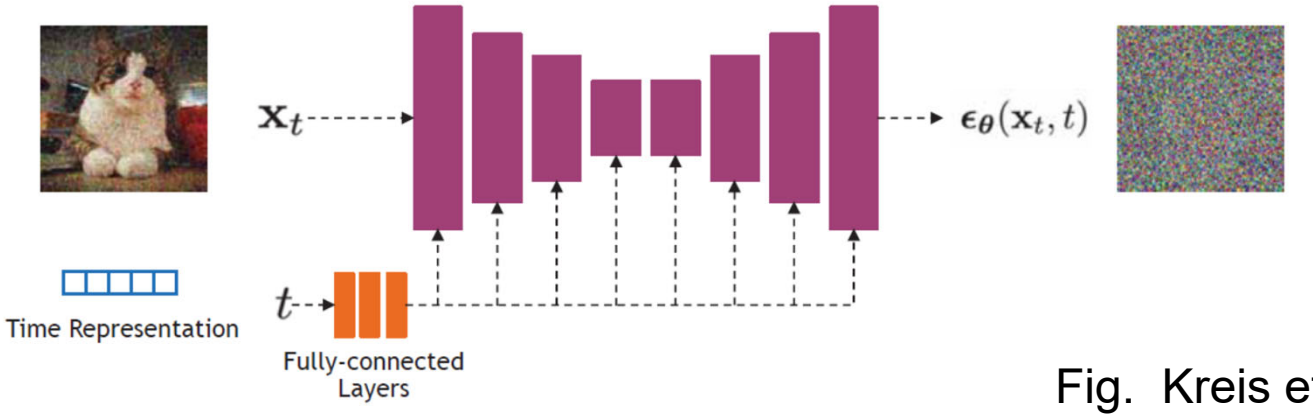


Fig. Kreis et al. 2022

- $\epsilon_\theta(x_t, t)$ is often implented with a U-Net with ResNet blocks and self attention layers (recent implementations have been proposed with transformers)
- Time features are fed to residual blocks, time encoding follows the transformers sinusoidal position embedding
- The parameters are shared for all the time steps, only the time representation makes the difference between the time steps

Denoising Diffusion Probabilistic Models
Comments

- In Ho et al. 2020
  - $T = 1000, \beta_1 = 10^{-4}, \beta_T = 0.02$ with a linear schedule
  - The pixel values are normalized in $[-1,1]$
  - As usual, lots of influential architecture/ algorithmic parameters conditioning the good behavior of the model
  - The process of generation is **extremely** slow (the original model takes up to 20 h to generate 50k images of size 32x32)
- Several variants/ improvements proposed since the Ho et al. 2020 paper
  - Conditional models allow to generate e.g. images conditionned on text
  - Latent diffusion models (Rombach et al. 2022) perform diffusion in a latent space, accelarating the generation (used e.g. in stable diffusion)
    - The image is first encoded in a smaller diemensional latent space and decoded in order to produce the generated image in the original space
    - Diffusion and denoising happen in the latent space
    - The model allows for conditioning image generation (on text, classes, …)

Denoising Diffusion Probabilistic Models
Derivations

- We first show
  - $-E_{q(x_0)} \leq E_{q(x_0:T)}[\log \frac{q(x_{1:T}\,|x_0)}{p_\theta(x_{0:T})}] \triangleq L$
- and then
  - $L = E[-logp_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$

Denoising Diffusion Probabilistic Models
Derivations

- $-E_{q(x_0)}[\log p_\theta(x_0)] \leq E_{q(x_{0:T})}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}] \triangleq L$
  - $-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0) \parallel p_\theta(x_{1:T}|x_0))$
  - $-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + E_{x_{1:T}\sim q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})/p_\theta(x_0)}]$
  - $-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + E_{x_{1:T}\sim q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} + p_\theta(x_0)]$
  - $-\log p_\theta(x_0) \leq E_{x_{1:T}\sim q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}]$
  - $-E_{q(x_0)}[\log p_\theta(x_0)] \leq E_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}]$

Denoising Diffusion Probabilistic Models
Derivations

- $L = E_{q(x_0:T)}[-\log p_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$
  - $L = E_{q(x_0:T)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=1}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)}]$
  - $L = E_{q(x_0:T)}\left[-\log p(x_T) + \sum_{t=2}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}\right]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=2}^{T} \log(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} . \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)}) + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_T|x_0)}{q(x_1|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}]$
  - $L = E_{q(x_0:T)}\left[\log \frac{q(x_T|x_0)}{p(x_T)} + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} - \log p_\theta(x_0|x_1)\right]$
  - $L = E_{q(x_0:T)}[-\log p_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$

Denoising Diffusion Probabilistic Models
Derivations

- $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I\right)$ with $\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$
  - $q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\,\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$
  - $q(x_{t-1}|x_t, x_0) \propto \exp -\frac{1}{2}\left(\frac{\left(x_t - \sqrt{\alpha_t}x_{t-1}\right)^2}{\beta_t} + \frac{\left(x_{t-1} - \sqrt{\bar{\alpha}_t}x_0\right)^2}{1-\bar{\alpha}_t} - \frac{\left(x_t - \sqrt{\bar{\alpha}_t}x_0\right)^2}{1-\bar{\alpha}_t}\right)$
  - … to be completed
  - Recall that $x_t = \sqrt{\bar{\alpha}_t}x_O + \sqrt{(1 - \bar{\alpha}_t)}\epsilon$

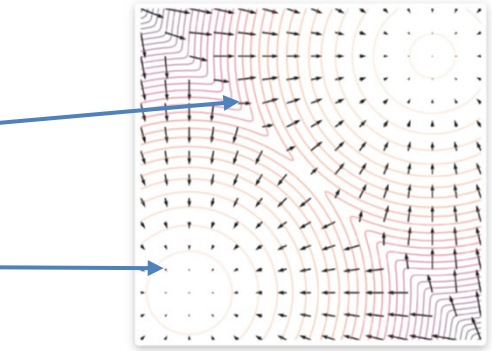
Score based models

Score based models

- **Score function** of a data distribution $q(x), x \in R^n$
  - $\nabla_x \log q(x) \in R^n$
  - Interpretation
    - Given a point x in data space, the score tells us which direction to move towards a region with higher likelihood
    - How to use this information for generating data from the distribution $q(.)$?
      - Sample x_0 from a prior (e.g. Gaussian) distribution $\pi(x)$ in R^n and iterate $x_{i+1} = x_i + \nabla_x \log q(x_i)$
      - This is similar to the formard proces in DDPMs

Low density region

High density region



Score function (the vector field) and density function (contours) of a mixture of two Gaussians.

Fig. Song 2022 illustrates the score function (arrows) and the density for a mixture of two gaussians

Score based models
Training

- **Score based model** $s_\theta(.)$ (SBM)
  - $\nabla_x \log q(x)$ is usually intractable, one will learn a score based model, i.e. a parametric model $s_\theta(x)$ to be implemented by a NN
    - $s_\theta(x) \approx \nabla_x \log q(x), s_\theta: R^n \to R^n$
    - $s_\theta(x)$ will be learned from a sample of the target distribution $q(x)$
- **Score matching**
  - SBM can be trained by minimizing the following loss between the model $s_\theta(.)$ and the data distribution $\nabla_x \log q(x)$
    - $E_{q(x)}\left[\|\nabla_x \log q(x) - s_\theta(x)\|_2^2\right] = \int \|\nabla_x \log q(x) - s_\theta(x)\|_2^2 q(x) dx$
- Summary
  - A distribution can be represented by its score function $\nabla_x \log q(x)$
  - The score function can be estimated by training a score based model $s_\theta(x)$ using samples from the target distribution with score matching

Score based models

- Generating samples from the target distribution

Langevin dynamics

- Once trained, $s_\theta(x)$ can be used by starting from a prior distribution $x_0 \sim \pi(x)$ (e.g. a Gaussian) and iterating a Markov chain for generating samples
  - $x_{i+1} = x_i + \epsilon s_\theta(x_i) + \sqrt{2\epsilon} z_i, i = 0, \ldots, K$, with $z_i \sim \mathcal{N}(0, I)$, ϵ is a small constant XX
  - This is similar to the reverse process in DDPM
  - When $\epsilon \to 0$ and K [illegible] to a sample from $q(x)$ under some regularity conditi[illegible]
    - In practice take [illegible] (100 to 1000)

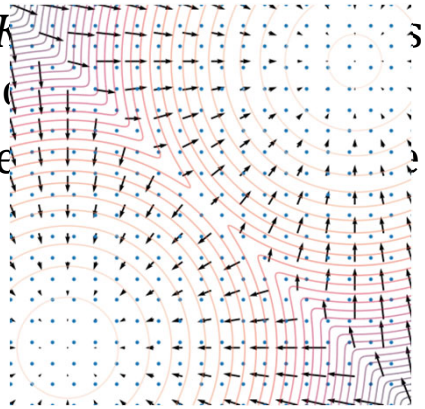


Fig. Song 2022
Langevin dynamics for sampling from a mixture of 2 gaussians, arrows indicate the score vector values, the animated Gif shows the convergence of the dynamics towards the target distribution

Score based models
Summary: training + generation

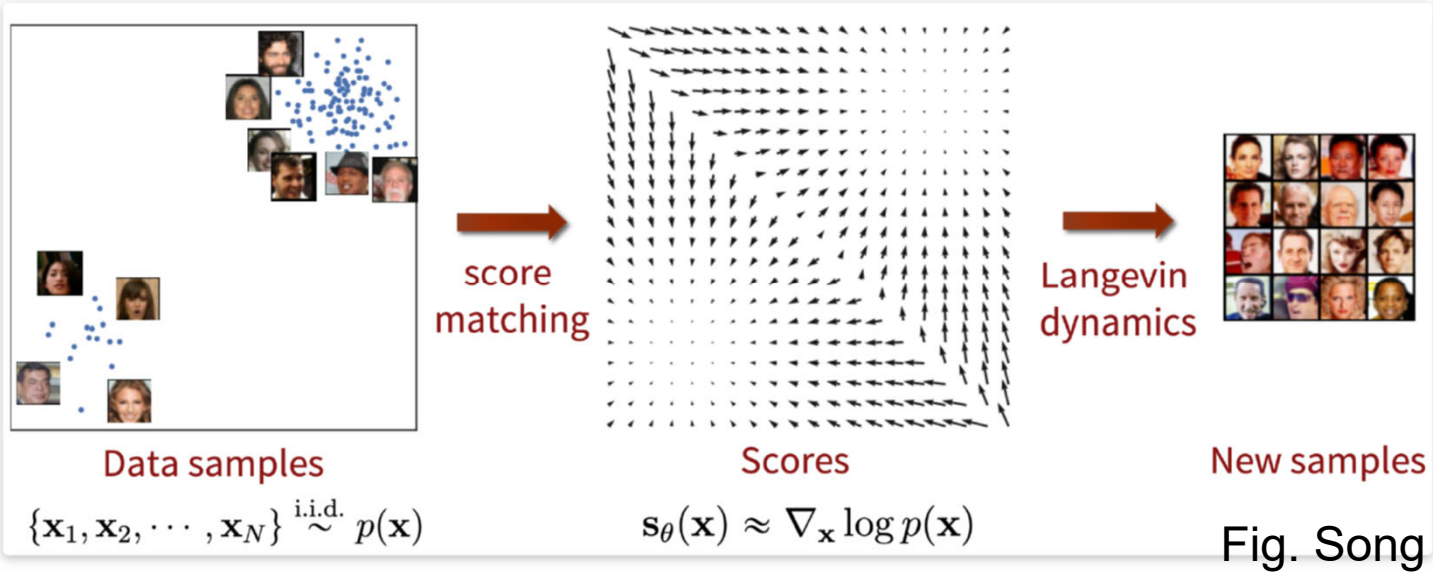


Fig. Song 2022

Score based models
Training: Noise conditionned score network (NCSN)

- let us come back to the score matching training formulation
  - $argmin_\theta E_{q(x)}[\|\nabla_x \log q(x) - \mathrm{s}_\theta(\mathrm{x})\|_2^2]$
- This formulation leaves us with 2 problems (Song et al. 2020)
  - (1) $q(x)$ is unknown
  - (2) In low density regions, there are only a few data points available so that $s_\theta(x)$ will be inaccurate.
  - (Song et al. 2020) propose different solutions to this problem, let us describe one of them:
    - Noise conditionned score network (NCSN)

Score based models
Training: Noise conditionned score network (NCSN)

- Noise conditionned score network (NCSN)
  - Intuition
    - Instead of training on the data distribution directly, train on noisy data
    - Perturb data points with noise $\mathcal{N}(0, \sigma I)$, train score based models on the noisy points using score matching.
      - If the noise magnitude is large enough this should help populating the low density regions, i.e. helps solving pb (2) ($s_\theta(x)$ innaccurate in low density regions)
      - What should be the noise scale?
        - Large noise populate the space but alters the original distribution
        - Small noise does not cover low density regions

Score based models
Training: Noise conditionned score network (NCSN)

- Noise conditionned score network (NCSN)
  - This idea is then refined as follows
    - Use multiple and increasing scales of noise $\mathcal{N}(0, \sigma_i I), i = 1 \ldots, T$ with $\sigma_1 < \sigma_2 < \cdots < \sigma_T$ to obtain T noise-perturbed distributions $q_{\sigma_i}(\tilde{x}) \triangleq \int q_{\sigma_i}(\tilde{x}|x) q(x) dx$
    - Draw samples from $q_{\sigma_i}(\tilde{x})$ by sampling $x \sim q(x)$ and computing $\tilde{x} = x + \sigma_i z$ with $z \sim \mathcal{N}(O, I)$
    - Use a **unique (θ) score function** paramaterized by $\sigma, \mathrm{s}_\theta(\mathrm{x}; \sigma)$ for all the noise scales and train it with the different noise scales using score matching so that $\mathrm{s}_\theta(\mathrm{x}; \sigma_\mathrm{i}) \approx \nabla_x \log q_{\sigma_i}(x)$
      - $\mathrm{s}_\theta(\mathrm{x}; \sigma)$ is called a **noise conditional score-based model**
    - Noise schedule: for example geometric schedule between two extreme values σ_1 to σ_T
  - Note
    - This is similar to the forward process in DDPMs

Score based models
Noise conditionned score network (NCSN)
Training formulation

- Noise conditionned score network (NCSN)
  - Let $\tilde{x}$ a perturbation of x generated according to the transition kernel $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}; \mathrm{x}, \sigma^2 I)$
    - i.e. $\tilde{x}$ is a noisy version of x
    - $\tilde{x}$ can be generated as $\tilde{x} = x + \sigma^2\epsilon, \epsilon \sim \mathcal{N}(O, I)$
    - Let us define $q_\sigma(\tilde{x}) \triangleq \int \; q_\sigma(\tilde{x}|x)q(x)dx$
  - The proposed loss function is
    - $\frac{1}{T}\sum_{i=1}^{T} \lambda(\sigma_i) E_{q_{\sigma_i}(x)} \left[\left\| \nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x}) - s_\theta(\tilde{x}, \sigma_i) \right\|_2^2 \right]$
      - This is a weighted sum of score matching losses, $\lambda(i) \in R, > 0$, often chosen as $\lambda(i) = \sigma_i^2$
  - This can be rewriten up to a constant as
    - $\frac{1}{T}\sum_{i=1}^{T} \lambda(\sigma_i) E_{x\sim q(x), \tilde{x}\sim q_{\sigma i}(\tilde{x}|x)} \left[\left\| \frac{\tilde{x}-x}{\sigma_i^2} + s_\theta(\tilde{x}, \sigma_i) \right\|_2^2 \right]$
      - $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}; \mathrm{x}, \sigma^2 I) \Rightarrow \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) = -\frac{\tilde{x}-x}{\sigma^2}$
  - $\lambda(\sigma_i)$ is set for example to σ_i^2 - so that all the components inside the summation have the same order of magnitude and do not depend on σ
    - $\frac{1}{T}\sum_{i=1}^{T} E_{x\sim q(x), \tilde{x}\sim q_{\sigma i}(\tilde{x}|x)} \left[\left\| \frac{\tilde{x}-x}{\sigma_i} + \sigma_i \, s_\theta(\tilde{x}, \sigma_i) \right\|_2^2 \right]$
  - After training $\sigma_i, s_\theta(\tilde{x}, \sigma_i)$ will return an estimate of the score $\nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x})$

Score based models
Generation

- For the generation, it is proposed to use an annealed form of the Langevin dynamics
- Initialize $x_0 \sim \mathcal{N}(O, I)$ (prior distribution)
- For $t = T$ to 1 (annealing iterations)
  - set α_t the step size e.g. $\alpha_t = \epsilon \frac{\sigma_t^2}{\sigma_1^2}$ with ϵ a small positive constant
  - For $i = 1$ to $N - 1$ (N steps of Langevin dynamics)
    - Draw $z_i \sim \mathcal{N}(0, I)$
    - $x_{i+1} = x_i + \alpha_t s_\theta(x_i, \sigma_t) +$
  - $x_0 = x_N$
- Return x_0

Remark: at each annealing iteration, one starts from the final sample of the previous iteration

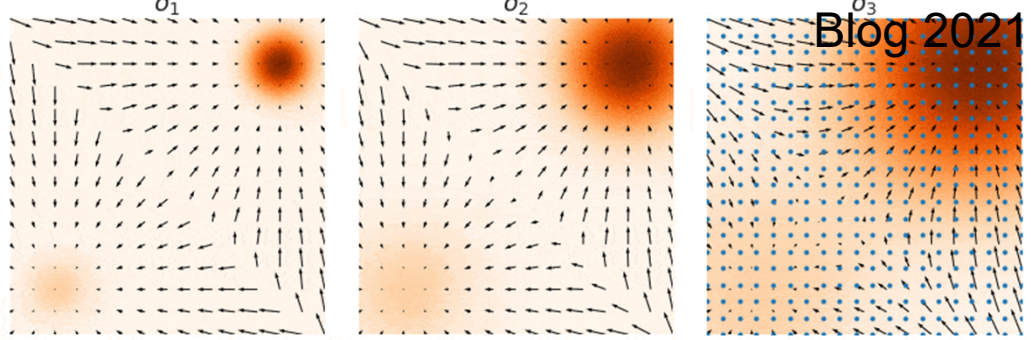


Fig. Song – Blog 2021

Score based models

- $s_\theta(x_i, t)$ is parametrized with U-Nets with residual connections as for DDPMs
- Equivalence with DDPM
  - The two training objectives (DDPM and SGM) are equivalent once we set
    - $\epsilon_\theta(x, t) = -\sigma_t(x, t)$

Score based models
Generation - example

(a) MNIST (b) CelebA (c) CIFAR-10

Figure 5: Uncurated samples on MNIST, CelebA, and CIFAR-10 datasets.

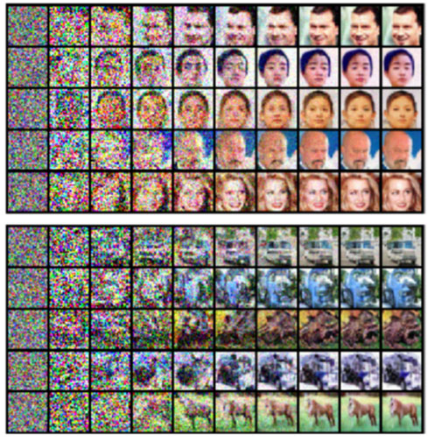


Figure 4: Intermediate samples of annealed Langevin dynamics.

Fig. Song et al 2020

Score stochastic differential equation

(Song et al. 2021)

Score stochastic differential equation

- Generalizes the discrete diffusion and score based formulations to time continuous dynamics
  - i.e. one considers the limit when the time step α_t in score based methods goes to 0
- Both DDPM and Score based approaches can be formulated as discretizations of SDE formulations

Score stochastic differential equation
Forward dynamics

- Stochastic differential equations (SDE)
  - $dx(t) = f(x,t)dt + g(t)d\omega$
    - $f(x,t)$ is a vector valued drift function, $f: R^n \rightarrow R^n$
    - $g(t)$ is a scalar valued diffusion function, $g: R \rightarrow R$
      - g is considered scalar and independent of x for s
    but could be a vector valued fonction and dependent
    - ω is a Wiener process (Brownian motion), $d\omega \sim \mathcal{N}($
    - Under some conditions, the SDE has a unique soluti
- Time discretization
  - $x_{t+\Delta t} = x_t + f(x_t,t)\Delta t + g(x_t,t)\Delta\omega$, with $\Delta\omega \sim \mathcal{N}(0,\Delta t)$
- Note
  - Langevin dynamics $x_{t+1} = x_t + \alpha_t s_\theta(x_t,t) + \sqrt{2\alpha_t}z_t$ appears as a special case of the discrete equation with:
    - $\Delta t = 1, f(x_t,t) = \alpha_t s_\theta(x_t,t), g(x_t,t) = \sqrt{2\alpha_t}, \Delta\omega = z_t$
  - As for the discrete case, the forward diffusion process does not depend on the data

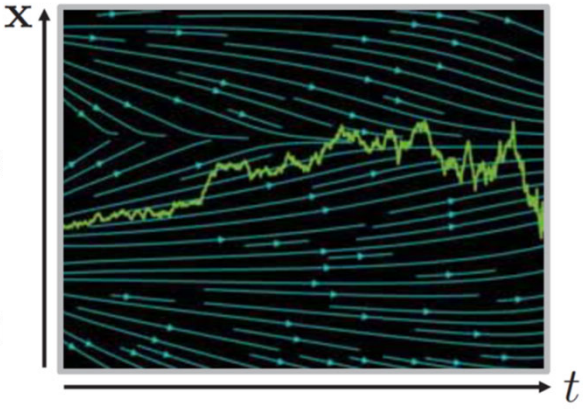


Fig. Kreis et al. 2022
Sample from a SDE
trajectory

Score stochastic differential equation
Forward dynamics

- Diffusion processes can be modeled as solutions of SDEs
  - The solution of a SDE is a continous collection of random variables $\{x(t)\}_{t\in[0,T]}$
  - These variables trace stochastic trajectories when t grows from 0 to T
- Let us denote $q_t(x)$ the probability density of $x(t)$, and $q(x(t)|x(s))$ the transition kernel from $x(s)$ to $x(t)$ with $s < t$
- The objective is to construct a forward diffusion process $\{x(t)\}_{t\in[0,T]}$, indexed by the continuous variable t so that $x(0)\sim p_0$, the data distribution and $x(T)\sim q_T$ is a tractable distribution that can be easily sampled, i.e. a prior π, e.g. a gaussian with fixed mean and variance

Score stochastic differential equation
Forward process

- Illustration: stochastic trajectories for the forward diffusion process

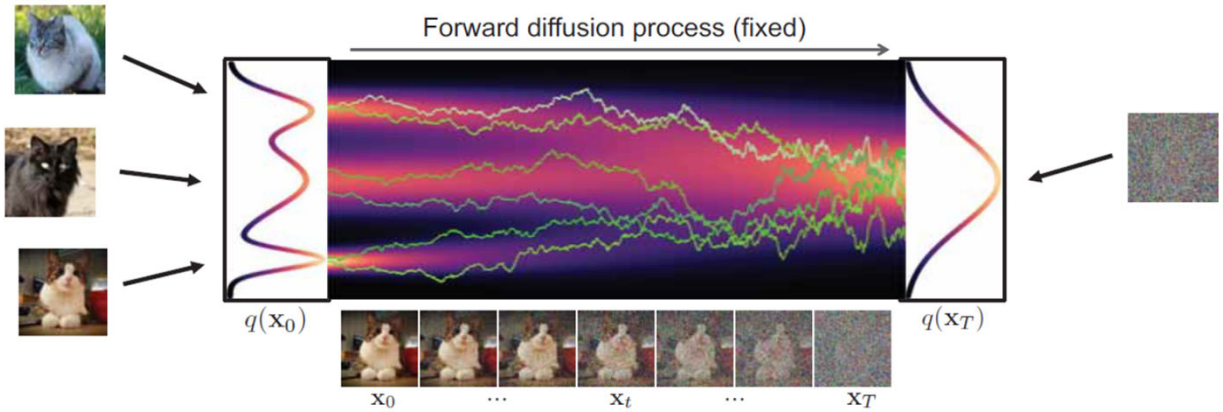


Fig. Kreis et al. 2022
Samples: SDE trajectories from different initial points

Score stochastic differential equation
Forward process

- Illustration: stochastic trajectories for the forward diffusion process

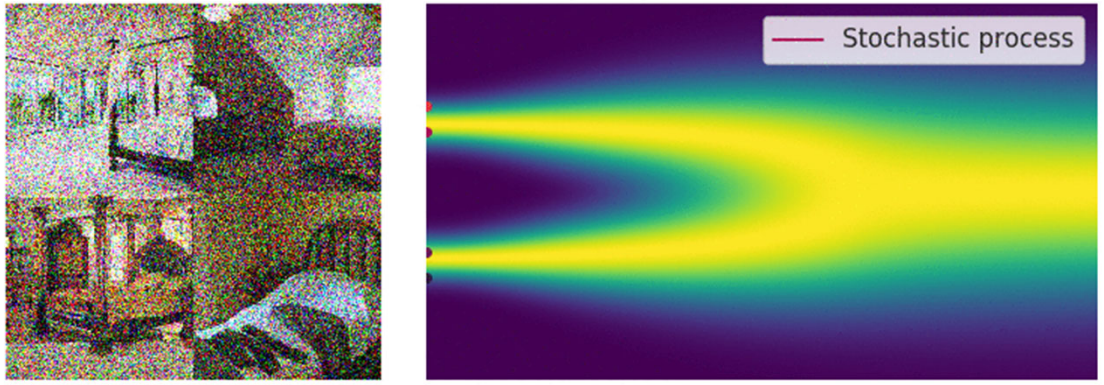


Fig. Song 2021 - https://yang-song.net/blog/2021/score/

Score stochastic differential equation

- DDPMs and SGMs are both special cases of the SDE discretization
- >>>>>>>>>>>>>>>>>>to be completed <<<<<<<<<<<<<<<<<<

Score stochastic differential equation
Reversing the SDE

- For samples generation, one needs to reverse the SDE
  - Any diffusion process modeled as a SDE can be reversed by solving the reverse SDE backward, i.e. from $t = T$ to $t = 0$
    - i.e. one starts at $x(T) \sim q_T$ and reversing the process we obtain samples $x(0) \sim q_0$
  - The reverse SDE writes as
    - $dx = \left(f(x,t) - g(t)^2 \nabla_x log q_t(x)\right)dt + g(t)dw$, with dt an infinitesimal negative time step
      - $q_t(x)$ is the distribution of x at ti
      - Once $\nabla_x log q_t(x)$ is known for a it by sampling from $q_T(x)$ to gen

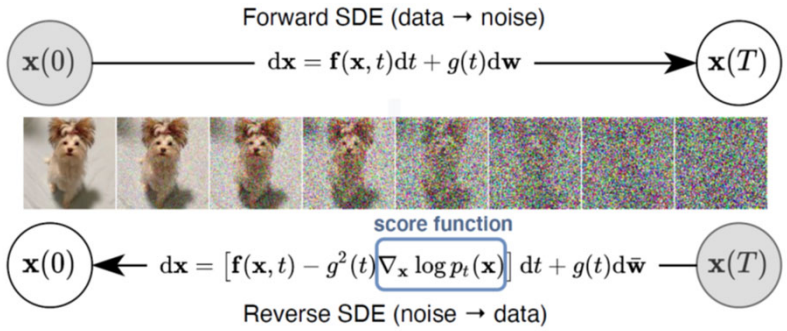


Fig. Song et al. 2021

Score stochastic differential equation
Reversing the SDE

- Reverse process illustration
  - One starts from noisy samples to generate target data samples

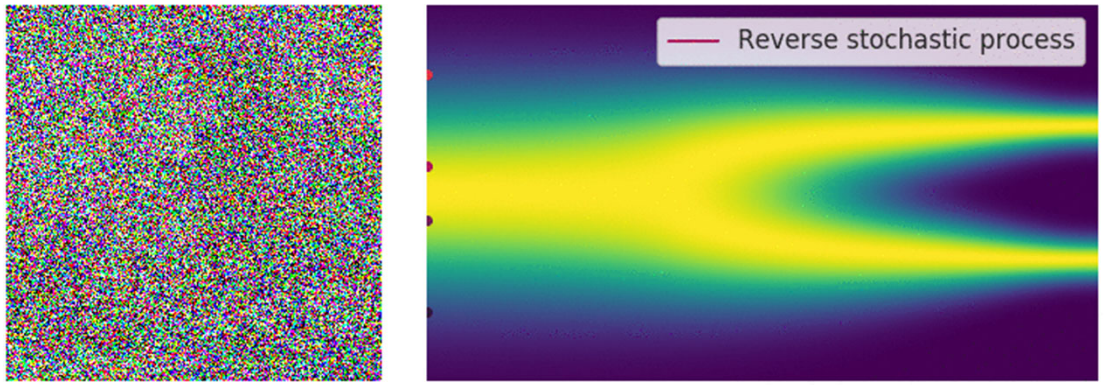


Fig. Song 2021 - https://yang-song.net/blog/2021/score/

Score stochastic differential equation

- With this formulation, we are then left with two problems
  - The training problem: how to estimate $\nabla_x log q_t(x)$ the score function of $q_t(x)$?
  - How to solve the reverse SDE?

Score stochastic differential equation
The training problem

- Solving the reverse SDE requires to know the terminal distribution $p_T(x)$ and the score function $\nabla_x log q_t(x)$
  - For the former one uses a prior distribution $\pi(x)$, typically a gaussian
  - For the latter, one trains a time-dependent score-based model $s_\theta(x,t)$ such that $s_\theta(x,t) \approx \nabla_x log q_t(x)$
    - Note: this is analogous to the discrete case $s_\theta(x,i) \approx \nabla_x log q_{\sigma_i}(x)$
- The training objective is a continuous extension of the one used with SGMs:
  - $E_{t\sim\mathcal{U}(0,T)} E_{q_t(x)}\left[\lambda(t)\|\nabla_x log q_t(x) - s_\theta(x,t)\|_2^2\right]$
    - $\mathcal{U}(0,T)$ is a uniform distribution over $[0,T]$ and $\lambda: R \rightarrow R$ is a positive weighting function
      - As for the discrete case, $\lambda(t)$ will be set so as to balance the magintude of the different score matching losses across time
- Generation
  - Once trained, one can simulate from $dx = \left(f(x,t) - g(t)^2 s_\theta(x,t)\right)dt + g(t)dw$
- Practical training
  - Use a score matching method e.g. denoising score matching

Score stochastic differential equation
The training problem

- Denoising score matching
  - As in the discrete case, diffuse individual data points using diffusion kernels $q_t(x(t)|x(0))$
    - $Min_\theta E_{t\sim\mathcal{U}(0,T)} E_{x(0)\sim q_0(x)} E_{x(t)\sim q(x(t)|x(0)))} \left[\lambda(t) \left\| \nabla_{x_t} log q_t(x(t)|x(0)) - s_\theta(x(t),t) \right\|_2^2 \right]$
    - diffusion kernels $q(x(t)|x(0))$ are chosen Gaussian for linear SDEs (this means f is affine):

$$q(x(t)|x(0)) = \mathcal{N}\left(x(t); \gamma_t x(0), \sigma_t^2 I\right)$$

- Objective: as in the discrete case, the loss function can be derived as
  - $Min_\theta E_{t\sim\mathcal{U}(0,T)} E_{x\sim q(x)} E_{\epsilon\sim\mathcal{N}(O,I)} \left[\frac{\lambda(t)}{\sigma_t^2} \|\epsilon - \epsilon_\theta(x_t,t)\|_2^2 \right]$
- Practice
  - Different loss weightings are proposed, e.g. $\lambda(t) = \sigma_t^2$ for the simplest case
  - $s_\theta(x(t),t)$ or $\epsilon_\theta(x_t,t)$ implemented with U-Nets
  - For the time integration, one could use Fourier features on t or replace t by σ_t

Score stochastic differential equation
Solving the SDE

- Once $s_\theta(x,t)$ is learned, it can be plugged in the reverse SDE
  - $dx = \left(f(x,t) - g(t)^2 s_\theta(x,t)\right)dt + g(t)dw$
  - Starting with $x(T) \sim \pi$, one can solve this reverse SDE to obtain a sample $x(0)$ from the target distribution $q(x)$ – or at least a sample from the approximate distribution $q_\theta(x) \approx q(x)$
- How to solve the reverse SDE
  - Learning free methods
    - SDE solvers – a variety of SDE solvers is available from the numerical analysis literature
      - Discretize the SDE in time and use a SDE solver
    - ODE solvers – this is detailed in the next slides – Faster that SDE solvers
  - Learning methods
    - Take benefit from the special for of the SDE in order to optimize the reverse solver

Score stochastic differential equation
ODE solvers

- (Song et al 2021) show that it is possible to associate an ODE to any SDE without changing the marginal distribution $\{q_t(x)\}_{t\in[0,T]}$. i.e. both the ODE and the SDE share the same set of marginal distributions $\{q_t(x)\}_{t\in[0,T]}$
  - The ODE associated to the reverse SDE is:
  - $\frac{dx}{dt} = f(x,t) - \frac{1}{2}g^2(t)\nabla_x \log q_t(x)$
  - This is called the probability flow ODE associated to the SDE
- It is then possible to sample from the same distribution as the reverse SDE by solving the ODE using classical ODE solvers (e.g. Runge Kutta)
- Note
  - When $\nabla_x \log q_t(x)$ is replaced by $s_\theta(x,t)$ the ODE becomes a special case of Neural ODE (see later in the course) – more precisely it is a continuous normalizing flow

Score stochastic differential equation
ODE solvers

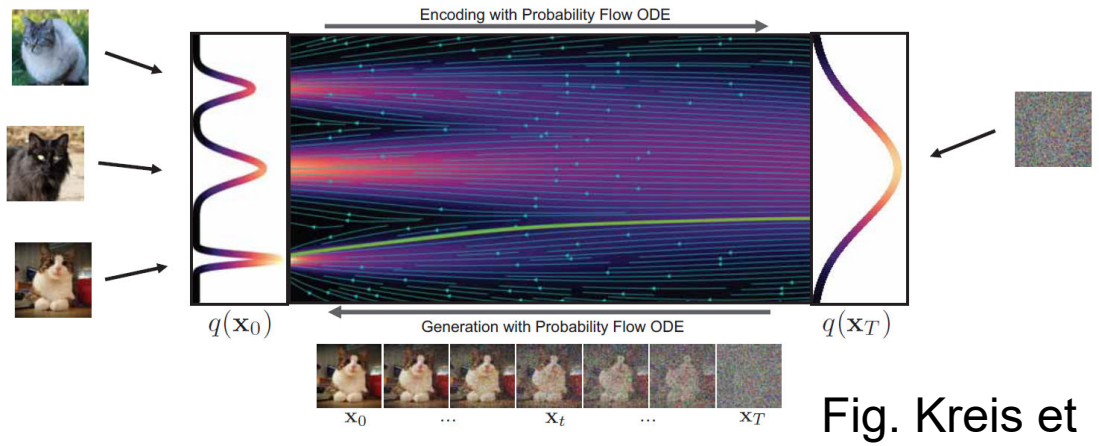


Fig. Kreis et al. 2021

- Current practice
  - Solve the forward process using the sde formulation (easy, no training)
  - Solve the reverse process using the ODE formulation
  - Note: the ODE could be used for the forward and reverse diffusion since (simply change the integration direction i.e. consider $t > 0$ for one direction and and $t < 0$ for the other direction), however the forward process is simpler with the fixed SDE formulation.

Score stochastic differential equation
ODE / SDE solvers

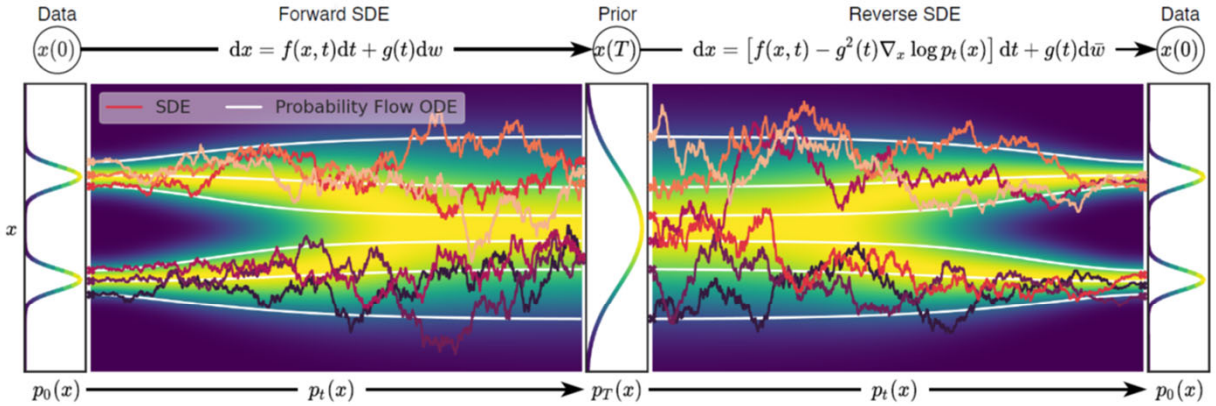


Figure 2: **Overview of score-based generative modeling through SDEs.** We can map data to a noise distribution (the prior) with an SDE (Section 3.1), and reverse this SDE for generative modeling (Section 3.2). We can also reverse the associated probability flow ODE (Section 4.3), which yields a deterministic process that samples from the same distribution as the SDE. Both the reverse-time SDE and probability flow ODE can be obtained by estimating the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ (Section 3.3).

Fig. Song et al. 2021

- ODE trajectories are smoother that SDE trajectories, however they allow to sample the same marginals $\{p_t(x)\}_{t\in[0,T]}$

Score based models
Conditional setting

- Several applications imply conditional generation
  - Text to image: DALL-E, IMAGEN
  - Class conditional generation
  - Super resolution, colorization, panorama etc (Saharia et al. 2020)

Figure 1: Image-to-image diffusion models are able to generate high-fidelity output across tasks without task-specific customization or auxiliary loss.

Figure 2: Given the central 256×256 pixels, we extrapolate to the left and right in steps of 128 pixels (2×8 applications of 50% Palette uncropping), to generate the final 256×2304 panorama. Figure D.3 in the Appendix shows more samples.

Fig from Saharia et al. 2020

Score based models
Conditional setting

- Conditional setting
  - Include the condition as input to the reverse process
  - The condition is input to the U-Net or whateverNet used for denoising
  - Class conditioning
    - Encode a scalar or class indicator as a vector embedding
  - Text conditioning
    - Vector embedding or sequence of vector embeddings, cross attetion, …
  - Image conditioning
    - Channel wise concatenation of the conditional image
- How to perform class conditioning
  - Several possibilities have been proposed
    - We detail here classifier guidance and classifier free guidance

Score based models
Conditional setting

- Classifier guidance
  - Instead of $q_t(x)$, one will attempt to compute $q_t(x|y)$ with y a conditioning variable
    - For simplification let us consider that y is a class indicator
    - $\nabla \log q_t(x|y) = \nabla \log \frac{q_t(x) q(y|x_t)}{q(y)}$
    - $\nabla \log q_t(x|y) = \nabla \log q_t(x) + \nabla \log q(y|x_t) - \log q(y)$
    - $\nabla \log q_t(x|y) = \nabla \log q_t(x) + \nabla \log q(y|x_t)$
    - To be completed

Diffusion models
Conclusion

- Pro
  - performance competitive with the best generative models
- Cons
  - extremely slow – due to the large number of sampling steps
- Several improvements – more to come
  - Sampling process
  - Training dynamics
  - Noise level parametrization

Diffusion models

- References
  - Blogs
    - Ayan Das 2021, https://ayandas.me/blog-tut/2021/12/04/diffusion-prob-models.html, https://ayandas.me/blog-tut/2021/07/14/generative-model-score-function.html
    - Lilian Weng, What are diffusion models: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
    - Song, Y. (2021). Generative modeling by estimating gradients of the data distribution. https://yang-song.net/blog/2021/score/
  - Slides and video
    - K. Kreis, R. Gao, A. Vahdat, CVPR tutorial, https://cvpr2022-tutorial-diffusion-models.github.io/
  - Papers
    - Karras, T., Aittala, M., Aila, T., & Laine, S. (2022). *Elucidating the Design Space of Diffusion-Based Generative Models*. *NeurIPS*. http://arxiv.org/abs/2206.00364
    - Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Neurips*, *2020-Decem*(NeurIPS 2020), 1–25.
    - Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. *CVPR*, 10674–10685. https://doi.org/10.1109/cvpr52688.2022.01042
    - Saharia, C., Chan, W., Chang, H., Lee, C., Ho, J., Salimans, T., Fleet, D., & Norouzi, M. (2022). Palette: Image-to-Image Diffusion Models. In *Proceedings of ACM SIGGRAPH* (Vol. 1, Issue 1). Association for Computing Machinery. https://doi.org/10.1145/3528233.3530757
    - Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *ICML*, *3*, 2246–2255.
    - Song, Y., & Ermon, S. (2020). Generative modeling by estimating gradients of the data distribution. *Neurips*.
    - Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-Based Generative Modeling through Stochastic Differential Equations. *ICLR*, 1–36. http://arxiv.org/abs/2011.13456
  - Tutorial / survey papers
    - Luo, C. (2022). Understanding Diffusion Models: A Unified Perspective. *Http://Arxiv.Org/Abs/2208.11970*, 1–23
    - Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Shao, Y., Zhang, W., Cui, B., & Yang, M.-H. (2022). *Diffusion Models: A Comprehensive Survey of Methods and Applications*. *1*(1). http://arxiv.org/abs/2209.00796

Normalizing Flows

Normalizing Flows
Obectives

- Normalizing Flows are a family of generative models which produces tractable distributions where both sampling and density evaluation can be **exact** (this is not the case for GANs and VAEs)
  - They can be trained by maximum likelihood
- They operate by **pushing** a simple **base density** through a **series of invertible and differentiable transformations** to produce a richer and **more complex** distribution
  - This is achieved through the change of variable theorem for densities

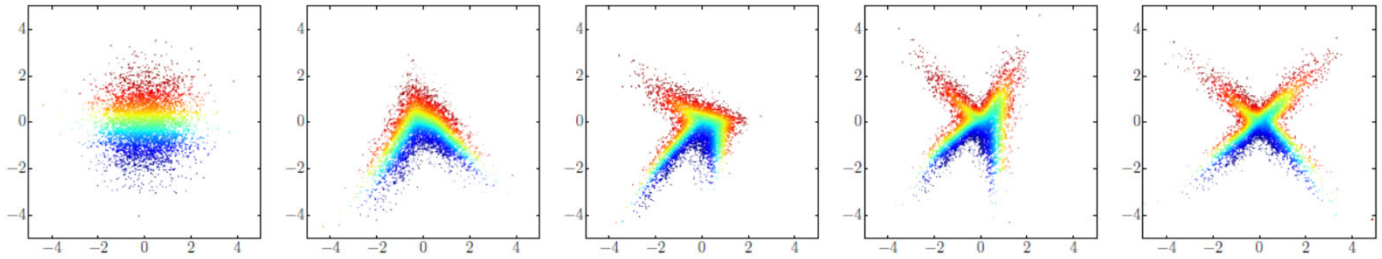


Figure 1: Example of a 4-step flow transforming samples from a standard-normal base density to a cross-shaped target density.

Fig. Papamakarios et al. 2019

Normalizing Flows
Prerequisite - Change of variable formula

- Notations
  - $x \in X$ observed variable, $x \in R^n$, $x \sim p_x$
  - $z \in Z$ latent variable, $z \in R^n$, $z \sim p_z$
  - $f: Z \to X$ invertible continuous function (bijection), $g = f^{-1}$ is the inverse of f
  - Both f and g are differentiable, i.e. f is a diffeomorphism
    - Diffeomorphism
      - $f: U \to V$ with U and V open sets of R^n is a **diffeomorphism** if $1)$ f is bijective, $2)$ f is differentiable on U, $3)$ its inverse is differentiable on V
  - Under these conditions, the density of x is well defined and can be obtained by a change of variables

Normalizing Flows
Prerequisite - Change of variable formula

- Change of variable formula
  - $p_x(x) = p_z(z) \left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|^{-1}$
    - We denote $J_f(z) = \frac{\partial f(z)}{\partial z}$ the Jacobian of f at z
    - $p_x(x)$ is called the **pushforward of the density p_z by f**, and denoted $f_{\#}p_z$
  - Equivalently
  - $p_x(x) = p_z(g(x)) \left|\det\left(\frac{\partial g(x)}{\partial x}\right)\right|$ since $g = f^{-1}$
    - We denote $J_g(x) = \frac{\partial g(x)}{\partial x}$ the Jacobian of g at x
    - x and z need to be continuous and have the same dimensions

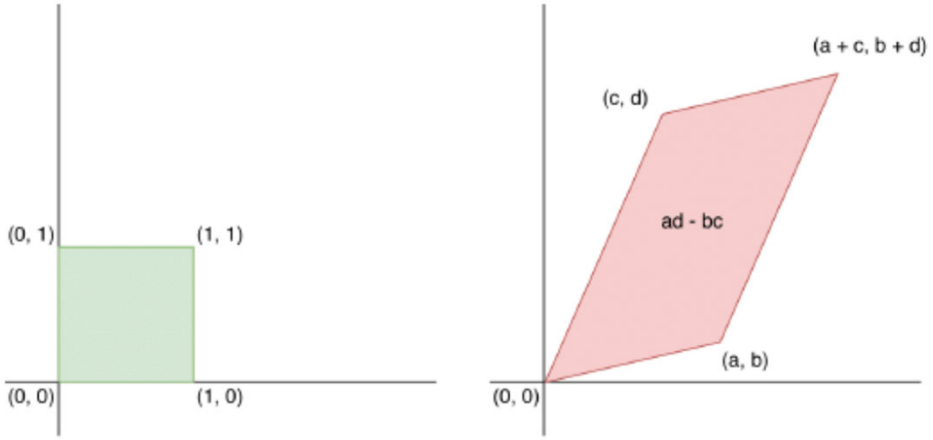
- Constructing a Flow based model, is made by implementing f or g via a neural network, taking p_z to be a simple density (e.g. multivariate Gaussian)

Normalizing Flows
Prerequisite - Change of variable formula

- 1 dimensional case
- $p_x(x) = p_z(g(x))|g'(x)|$
- Proof
  - Assume f is monotone, increasing
  - Let us consider the cumulative distribution functions
    - $F_x(x) = p_x(X \leq x) = p_x(f(Z) \leq x) = p_z(Z \leq g(x)) = F_z(g(x))$
  - Taking the derivatives w.r.t. x
    - $p_x(x) = \frac{dF_x(x)}{dx} = \frac{dF_z(g(x))}{dx} = \frac{dF_z(g(x))}{dz}\frac{dz}{dx} = p_z\big(g(x)\big)g'(x)$
  - If f is decreasing, we get $p_x(x) = p_z\big(g(x)\big)(-g'(x))$
  - To summarize:
    - $p_x(x) = p_z(g(x))|g'(x)|$

Normalizing Flows
Prerequisite - Change of variable formula

- 2 dimensions
  - Let z be a uniform random vector in $[0,1]^2$
  - Let $x = Az$ for a square invertible matrix A, with inverse W,
    - $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$
  - What is the distribution of x?
- Geometrically: A maps $[0,1]^2$ to a parallelogram with volume $\det(A) = ad - bc$



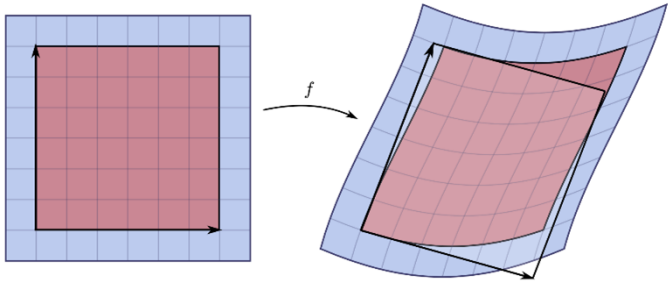
Normalizing Flows
Prerequisite - Change of variable formula

- Moe generally
  - Let z be a uniform random vector in $[0,1]^n$
  - Let $x = Az$ for a square invertible matrix A, with inverse W
  - Then
    - $p_x(x) = p_z\left(A^{-1}x\right)\left|\det(A^{-1})\right|$
    - $p_x(x) = p_z\left(A^{-1}x\right)/|\det(A)|$

    - Note $\det\left(A^{-1}\right) = \frac{1}{\det(A)}$

Normalizing Flows
Prerequisite - Change of variable formula – general case

- Determinant of the jacobian for a function $f: Z \to X$
  - $\left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|$ quantifies the relative change of volume of a small neighborhood around z due to f
  - If a small neighborhood dz around z is mapped to a small neighborhood dx around x, $\left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|$ is equal to the volume of dx divided by the volume of dz.



Source Wikipedia

A nonlinear map $f: R^2 \to R^2$ sends a small square (left, in red) to a distorted parallelogram (right, in red). The Jacobian at a point gives the best linear approximation of the distorted parallelogram near that point (right, in translucent white), and the Jacobian determinant gives the ratio of the area of the approximating parallelogram to that of the original square.

Normalizing Flows
Prerequisite – Jacobian – Determinants

- Jacobian
  - $J_f(z) = \frac{\partial f}{\partial z} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$
  - The determinant of a triangular matrix is the product of the diagonal terms

- Compositionality
  - Diffeomorphisms are composable
  - Let f_1 and f_2 be two diffeomophisms, and $f_2 \circ f_1$ their composition, then:
    - $(f_2 \circ f_1)^{-1} = f_1^{-1} \circ f_2^{-1}$
    - $\det\left(\frac{\partial f_2 \circ f_1(z)}{\partial z}\right) = \det\left(\frac{\partial f_2 \circ f_1(z)}{\partial f_1(z)}\right).\det\left(\frac{\partial f_1(z)}{\partial z}\right)$
    - Note
      - If A and B are twi square matrices of equal size $\det(AB) = \det(A)\det(B)$

Normalizing Flows
Prerequisite – Jacobian – Determinants

- Compositionality properties mean that complex transformations can be built from simple ones without loosing the properties of inversibility and differentiability
  - In practice, complex transformations f, can be built by composing simpler transformations $f = f_K \circ \cdots \circ f_1$, with each f_k transforming z_{k-1} to z_k, assuming $z_0 = z$ and $z_K = x$
  - The term **flow** refers to the trajectory that a collection of samples from $p_z(z)$ follows as they are transformed by the sequence of transformations $f_1, \ldots, f_K$
  - The term **normalizing** refers to the fact that the inverse flow $g = g_1 \circ \cdots \circ g_K$ takes a collection of samples from $p_x(x)$ and transforms them into a collection from a prescribed density $p_z(z)$, in a sense normalizing them
  - Change of variable formula for the composition of functions
    - $p_x(x) = p_z(z) \prod_{k=1}^{K} \left| \det \left(\frac{\partial f_k(z)}{\partial z_k} \right) \right|^{-1}$

Normalizing Flows
Density estimation and sampling

- A flow based model provides two operations:
  - 1. sampling via sample $z \sim p_z(z)$, and then $x = f(z)$
  - 2. density estimation via $p_x(x) = p_z(g(x)) \left| \det \left(\frac{\partial g(x)}{\partial x} \right) \right|$

f: sampling direction

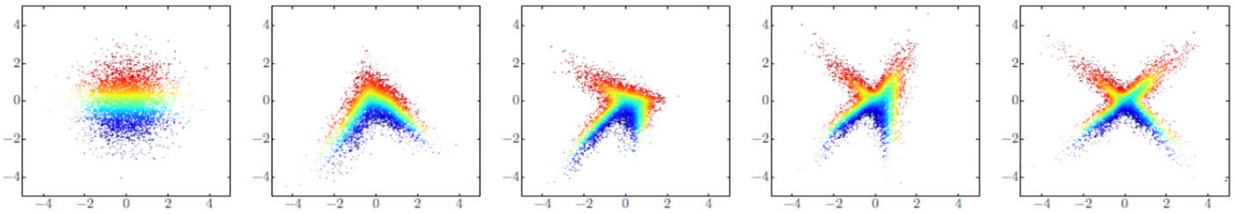


Figure 1: Example of a 4-step flow transforming samples from a standard-normal base density to a cross-shaped target density.

g: density estimation direction

Fig. Papamakarios et al. 2019

Normalizing Flows
Density estimation and sampling

- A flow based model provides two operations:
  - 1. sampling via $z \sim p_z(z)$, and then $x = f(z)$
  - This requires the ability to sample from $p_z(z)$ and to compute the forward transformation $x = f(z)$
    - The cost of sampling is dominated by the cost of the forward mapping f
  - 2. density estimation via $p_x(x) = p_z(g(x)) \left| \det \left(\frac{\partial g(x)}{\partial x} \right) \right|$
    - This requires the ability to
      - Compute the inverse transformation $g = f^{-1}$
      - Compute its jacobian determinant $\det \left(\frac{\partial g(x)}{\partial x} \right)$
      - Evaluate the density $p_z(z)$
    - The cost of density estimation is dominated by the cost of computing the inverse mapping and the log determinant

Normalizing Flows
Density estimation and sampling

- These two applications have different computational requirements
  - The definition of flow based model are dictated by these different requirements
  - Depending on what the model is used for, different computational trade-offs will be used
- Note (important)
  - For density estimation it is common to model a flow in the normalizing direction, i.e. by specifying g instead of f
    - This is because computation of the inverse of f may be difficult

Normalizing Flows
Expressive power of flow models

- Under some conditions, flow based models can express any distribution $p_x(x)$
  - Proof sketch in the course

Normalizing Flows
Density estimation and sampling – Training the flow model

- Let θ and ϕ be respectively the parameters of flow $f: Z \to X$ and base density p_Z, and $\Theta = (\theta, \phi)$
- Given observed data $D = \{x^i\}_{i=1\ldots N}$ the parameters can be estimated by maximum likelihood
- The data likelihood is
  - $\mathrm{L}(\Theta) = \log p(D;\Theta) = \sum_{i=1}^{N} \log p_x(x^i;\Theta)$
  - $\mathrm{L}(\Theta) = \log p(D;\Theta) = \sum_{i=1}^{N} \log p_z(g(x^i;\theta);\phi) + \log \left| \det \left(\frac{\partial g(x^i;\theta)}{\partial x} \right) \right|$
    - The first term is the likelihood of the sample under the base measure $p_z(z)$
    - The second term accounts for the change of volume induced by the transformation

Normalizing Flows
Density estimation and sampling - Training the flow model

- The parameters can be estimated iteratively with a stochastic gradient algorithm. The gradients of $L(\Theta)$ w.r.t. parameters θ and ϕ are as follows:
  - $\nabla_\theta L(\Theta) = \sum_{i=1}^{N} \nabla_\theta \log p_z(g(x^i;\theta);\phi) + \nabla_\theta \log \left| \det \left(\frac{\partial g(x^i;\theta)}{\partial x} \right) \right|$
  - $\nabla_\phi L(\Theta) = \sum_{i=1}^{N} \nabla_\phi \log p_z(g(x^i;\theta);\phi)$
    - i.e. one needs to compute the inverse transformation $g = f^{-1}$, its jacobian determinant $\det \left(\frac{\partial g(x)}{\partial x} \right)$ and evaluate the density $p_z(z)$, as well as differentiate through them
    - Efficiency of these computations determines the efficiency of the training
- Note
  - Other criteria (than max. likelihood) can be used for training (e.g. adversarial flows)
- Once trained, the model can be used for:
  - Sampling from p_z through the forward mapping f:
    - $z \sim p_z(z)$ and then $x = f_\theta(z)$
  - Evaluating the likelihood, performed through the inverse mapping g
    - Change of variable formula
  - Infering a latent representation
    - $z = g_\theta(x)$

Normalizing Flows
Density estimation and sampling

- Note
  - The empirical max likelihood estimate is an approximation of $D_{KL}(p^*(x)||p_x(x;\Theta))$ with $p^*(x)$ the target distribution and $p_x(x;\Theta)$ the model distribution
    - $D_{KL}(p_x^*(x)||p_x(x;\Theta)) = -E_{p_x^*(x)}[logp_x(x;\Theta)] + const$
    - $D_{KL}(p_x^*(x)||p_x(x;\Theta)) = -E_{p_x^*(x)}\left[\log p_z(g(x);\theta);\phi) + \log\left|\det\left(\frac{\partial g(x;\theta)}{\partial x}\right)\right|\right] + const$
    - Maximizing the empirical log-likelihood is then equivalent to minimizing an approximation of the KL divergence between the target and the model distribution
  - Other measures of difference between distributions can be used for training flows
    - e.g. we could choose the generator of a GAN as a normalizing flow and train the flow parameters using adversarial training, Wasserstein losses, Maximum Mean Discrepancy, etc.

Normalizing Flows
Constructing flows

- As seen before, Normalizing Flows should satisfy several constraints in order to be practical
  - These constraints differ depending on the application (e.g. sampling or density estimation)
  - Flows should be:
    - Invertible
    - Expressive enough to model complex real distributions
    - Computationally efficient: Jacobian calculation, sampling from the base distribution, calculation of the forward and inverse functions
  - Several flows have been proposed in the litterature, we describe some representative models in this course
- Finite composition
  - The flow is described as a finite composition of transformations $f = f_K \circ \cdots \circ f_1$
  - Examples described in the course:
    - Coupling layers, Auto-regressive flows, Residual Flows
- Continuous time composition
  - The flow is described as a continuous time function, i.e. its dynamics is described by an Ordinary Differential Equation (ODE)
  - Examples described in the course: To be completed

Normalizing Flows
Constructing flows – finite composition

- The flow is described as a finite composition of transformations $f = f_K \circ \cdots \circ f_1$
  - Each f_i implements a simple transformation $f_i: R^n \rightarrow R^n$, with a tractable inverse and Jacobian determinant. Let us denote $z_0 = z$ and $z_K = x$
- The forward evaluation is:
  - $z_k = f_k(z_{k-1})$
- The inverse evaluation is:
  - $z_{k-1} = g_k(z_k)$
- The jacobian determinant computation is:
  - $\log \left| \frac{\partial f(z)}{\partial z} \right| = \sum_{k=1}^{K} \log \left| \frac{\partial f_k(z_{k-1})}{\partial z_{k-1}} \right| = \sum_{k=1}^{K} \log \left| \frac{\partial z_k}{\partial z_{k-1}} \right|$, idem with the inverse g:
  - $\log \left| \frac{\partial g(x)}{\partial x} \right| = \sum_{k=1}^{K} \log \left| \frac{\partial g_k(z_k)}{\partial z_k} \right| = \sum_{k=1}^{K} \log \left| \frac{\partial z_{k\_1}}{\partial z_k} \right|$
- $f_k$or g_k will be implemented with a NN, the corresponding parameters are denoted θ_k (in the following, θ_k may represent the parameters of either$f_k$or g_k, depending on the setting)
  - In each case, one must ensure that the NN is invertible and has a tractable determinant
  - For a nxn matrix (n is the size of the data), the matrix determinant computation complexity is $O(n^3)$, most proposed method try to keep this complexity $O(n)$ by choosing transformations so that the Jacobian has a special structure
  - Depending on the application, one will implement either $f_k$or g_k

Normalizing Flows
Constructing flows – finite composition – **Coupling layers**

- We describe different possibilities for implementing the f_ks
  - For notation simplification, we consider a single tranformation f, and the inputs/ outputs are respectively denoted z and x. However this should be understood for a component transformation f_k for which the inputs/ outputs are respectively z_{k-1} and z_k
- **Coupling layers** make use of transformations which are easily invertible and for which the Jacobian determinant is easily computable. The general scheme is as follows:
  - Partition the variables z into two disjoint subsets, e.g. $z_{1:d}$ and $z_{d+1:n}$, and let $\varphi: R^{n-d} \to R^{n-d}$ be a bijection
  - Forward transformation $f: R^n \to R^n$ can be defined as:
    - $x_{1:d} = z_{1:d}$ identity
    - $x_{d+1:n} = \varphi(z_{d+1:n};\ \mathrm{h}(z_{1:d}))$ with $h: R^d \to R^{n-d}$ an arbitrary function which depends only on $z_{1:d}$ and which is called the **conditioner** or **coupling function** and φ an easily invertible transformation
  - Inverse transformation g is then
    - $z_{1:d} = x_{1:d}$ identity
    - $z_{d+1:n} = \varphi^{-1}(x_{d+1:n}; \mathrm{h}(z_{1:d}))$

Normalizing Flows
Constructing flows – finite composition – Coupling layers

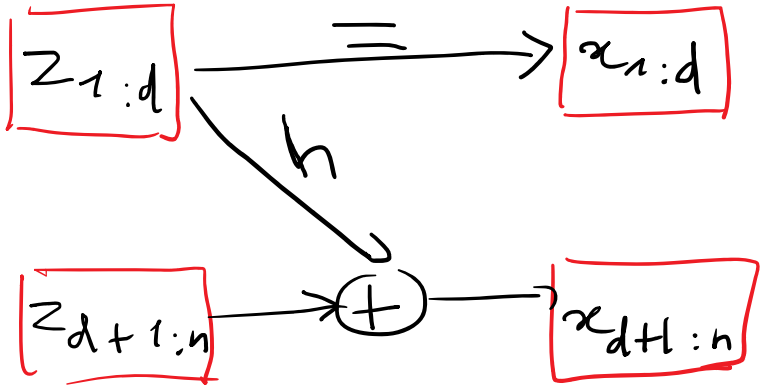
- With these definitions, the Jacobian is bloc triangular with the following form
  - $J = \frac{\partial x}{\partial z} = \begin{pmatrix} I_d & O \\ \frac{\partial x_{d+1:n}}{\partial z_1 : d} & D \end{pmatrix}$
  - where
    - I_d is a dxd identity matrix
    - $\frac{\partial x_{d+1:n}}{\partial z_{1:d}}$ is a $(\mathrm{n} - d)\mathrm{x}d$ full matrix
    - $D = \frac{\partial x_{d+1:n}}{\partial z_{d+1:n}}$ is a $(\mathrm{n} - d)\mathrm{x}(\mathrm{n} - d)$ matrix
    - The determinant of the Jacobian $\det(J_f)$ is simply the determinant $\det(D) = \det(J_\varphi)$
- Partitioning
  - Different strategies depending on the application and on the model
- Combining coupling layers
  - Several coupling layers are usually combined to obtain more complex transformations
  - A different partioning strategy of the input variables is used in the successive layers in order to modify every dimension
    - A simple one is to exchange the partitions between the modified and the unmodified variables

Normalizing Flows
Constructing flows – finite composition - Coupling layers – NICE (Dinh 2015)

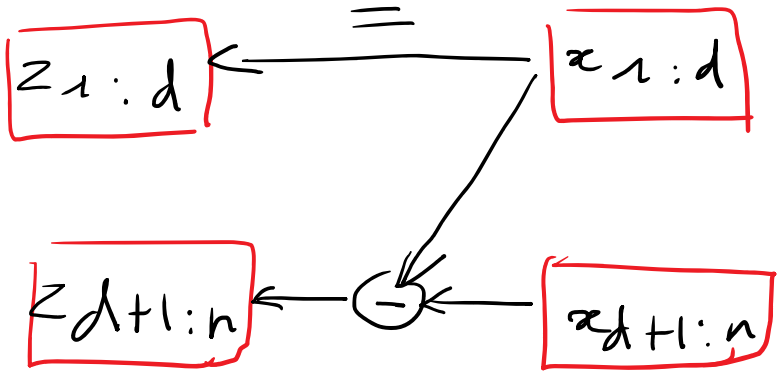
- Coupling layers were introduced in (Dinh 2015)
  - Forward transformation are implemented as **additive** transformations
    - $x_{1:d} = z_{1:d}$ (identity)
    - $x_{d+1:n} = z_{d+1:n} + \mathrm{h}(z_{1:d})$ with h a NN with d input units and $n - d$ output units
  - Inverse transformation can be easily computed
    - $z_{1:d} = x_{1:d}$ (identity)
    - $z_{d+1:n} = x_{d+1:n} - \mathrm{h}(x_{1:d})$
    - Computing the inverse is as simple as computing the forward transformation
    - There is no restriction on h, besides having the proper domain of definition and co-domain
  - Jacobian of the forward mapping
    - $J = \frac{\partial x}{\partial z} = \begin{pmatrix} I_d & 0 \\ \frac{\partial x_{d+1:n}}{\partial z_1 : d} & I_{n-d} \end{pmatrix}$
    - $\det(J) = 1$, i.e. this is a volume preserving transformation

Normalizing Flows
Constructing flows – finite composition - Coupling layers – NICE (Dinh 2015)

- Forward



- Reverse



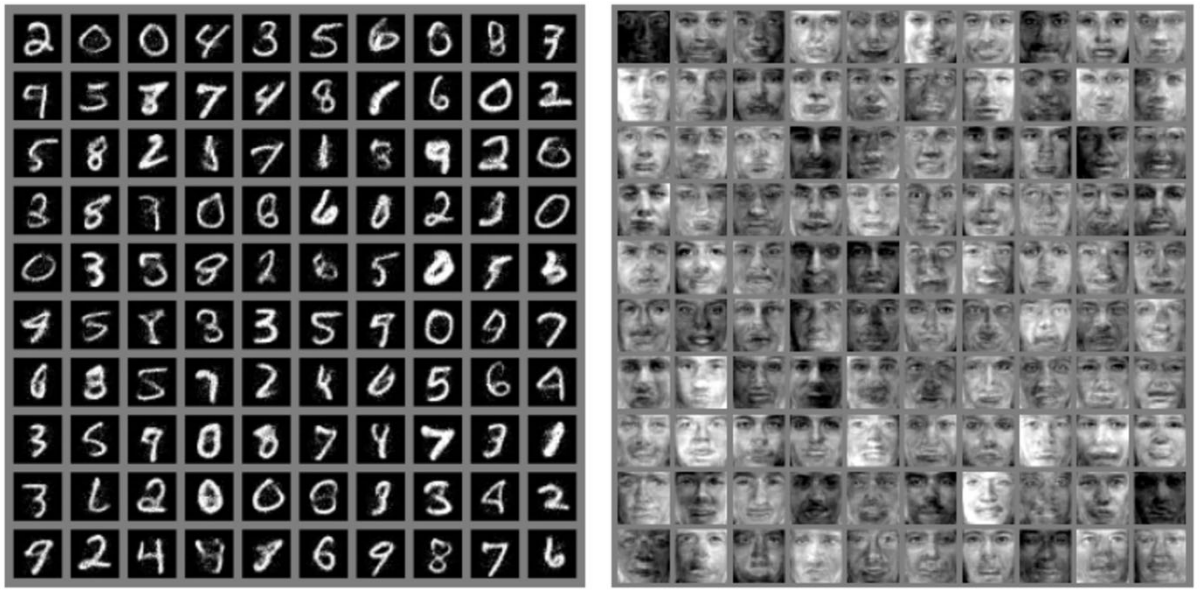
Normalizing Flows
Constructing flows – finite composition - Coupling layers – NICE (Dinh 2015)

- The coupling layers are composed together with an arbitrary partition of the variable on each layer
  - The overall tranformation is still volume preserving with the composition
  - In order to allow for more flexible transformations (non volume preserving) there is an additional rescaling of the variables at the last layer
    - $x_i = s_i z_i$ with $s_i > 0$
    - The inverse mapping then computes $z_i = \frac{x_i}{s_i}$
    - Jacobian of forward mapping $J = diag(s), \det(J) = \prod_{i=1}^{n} s_i$

- Base distributions
  - In the experiments they choose as priors simple factored distributions (their components are independent) such as gaussians or logistics

Normalizing Flows
Constructing flows – finite composition - Coupling layers – NICE (Dinh 2015)

- Samples generated via NICE
  - $z \sim p_z(z)$ and then $x = f_\theta(z)$
  - $p_z(z)$ is a logistic distribution for MNIST and Gaussian for TFD



(a) Model trained on MNIST (b) Model trained on TFD

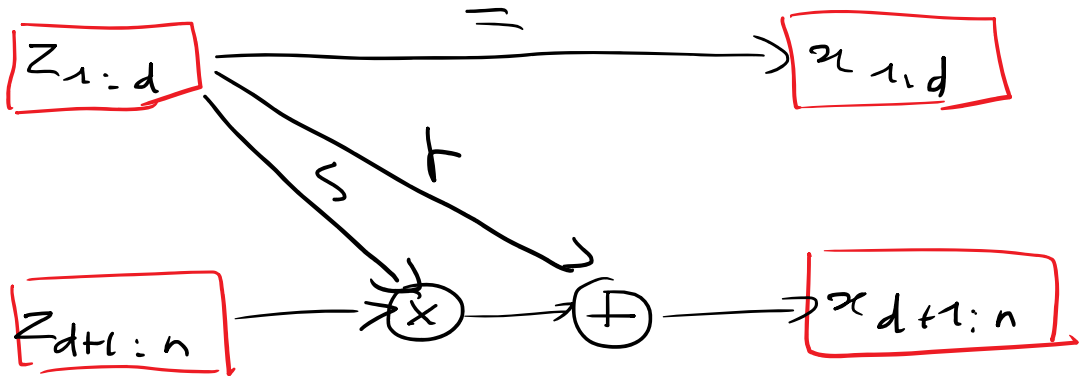
- They also evaluate the quality of the model likelihood on the data

Normalizing Flows
Coupling layers – Real NVP (Dinh 2017)

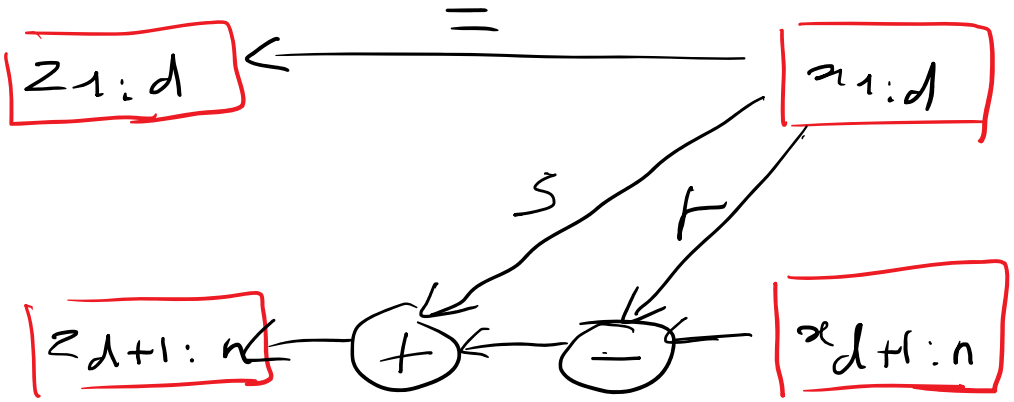
- Extension of NICE to Non Volume Preserving (NVP) Flows
- Forward transformation
  - $x_{1:d} = z_{1:d}$ (identity)
  - $x_{d+1:n} = z_{d+1:n} \odot \exp\left(\mathrm{s}(z_{1:d})\right) + t(z_{1:d})$ with s and t both $R^d \rightarrow R^{n-d}$ implemented as NNs , $\odot$ the Hadamard product, $s()$ is a **scaling** operator and $t()$ a **translation** operator, both implemented with NNs, $\exp\left(\mathrm{s}(z_{1:d})\right)$ is a vector, i.e. $\exp()$ is applied to each term of the vector
- Inverse transformation
  - $z_{1:d} = x_{1:d}$ (identity)
  - $z_{d+1:n} = \left(x_{d+1:n} - t(z_{1:d})\right) \odot \exp(-\mathrm{s}(z_{1:d}))$
  - Again computing the inverse is as simple as computing the forward mapping
  - Computing the inverse does not require computing the inverse of s or t so that these transformations can be arbitrarily complex

Normalizing Flows
Coupling layers – Real NVP (Dinh 2017)

- Forward



- Reverse



Normalizing Flows
Coupling layers – Real NVP (Dinh 2017)

- Jacobian of the forward mapping
  - $J = \frac{\partial x}{\partial z} = \begin{pmatrix} I_d & O \\ \frac{\partial x_{d+1:n}}{\partial z_1:d} & diag(\exp(s(z_{1:d}))) \end{pmatrix}$
  - $diag(\exp(s(z_{1:d})))$ is a diagonal matrix whose diagonal elements correspond to the vector $\exp(s(z_{1:d}))$
  - $\det(J) = \prod_{i=d+1}^{n} \exp(s(z_{1:d})_i) = \exp(\sum_{i=d+1}^{n} s(z_{1:d})_i)$
  - Determinant can be greater or smaller than 1, i.e. Non Volume Preserving (NVP)
  - Well adapted to large dimensions
- Additional tricks
  - Partitioning is performed at each layer by taking alternating pixels or blocks of chanels in the case of images
  - They use different normalizations

Normalizing Flows
Coupling layers – Real NVP (Dinh 2017)

- Samples

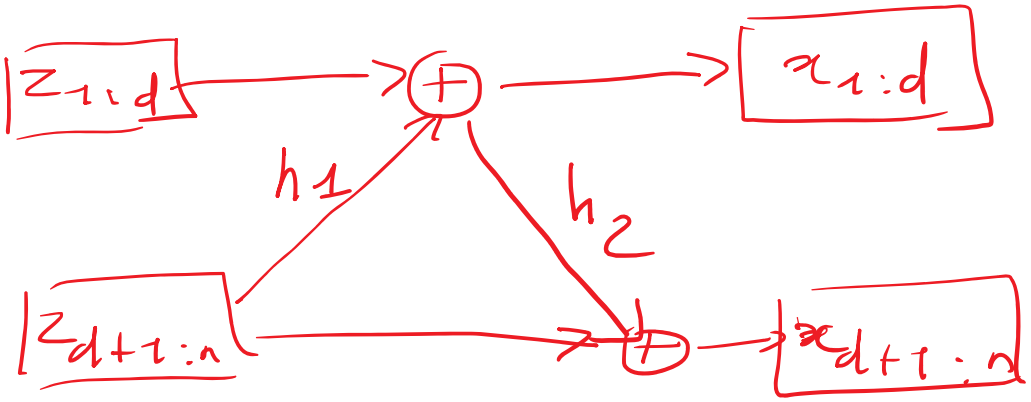
Figure 5: On the left column, examples from the dataset. On the right column, samples from the model trained on the dataset. The datasets shown in this figure are in order: CIFAR-10, Imagenet (32×32), Imagenet (64×64), CelebA, LSUN (bedroom).

Normalizing Flows
Coupling layers – RevNet (Gomez 2017)

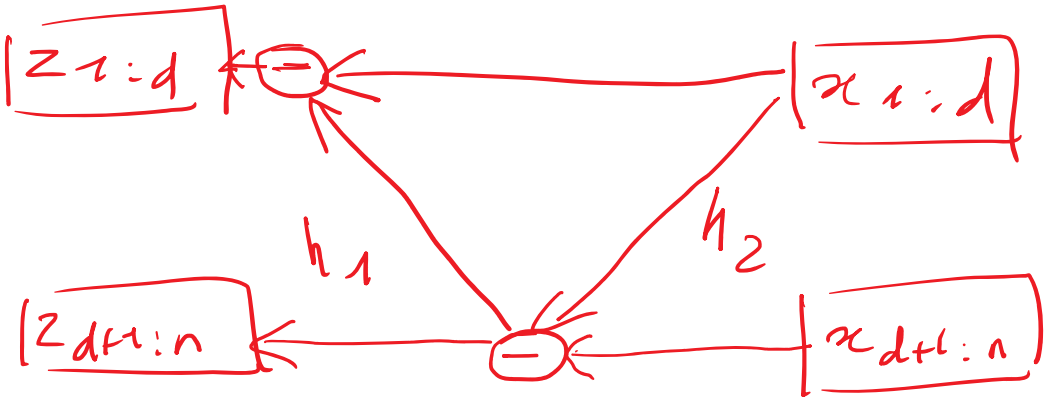
- RevNet is a reversible variant of ResNets
  - It builds on earlier coupling layers ideas (like NICE) in order to propose a reversible net where each layer activation can be computed from next layer activation thus eliminating the need for storing these activations during back-propagation
  - The main objective is then to save memory. However the computation of the Jacobian is not efficient.
  - It is shown here for because it has initiated a series of work on reversible ResNets and on models that do not require storing the activations for backpropagating
- Forward transformation
  - $x_{1:d} = z_{1:d} + \mathrm{h}_1(z_{d+1:n})$
  - $x_{d+1:n} = z_{d+1:n} + \mathrm{h}_2(x_{1:d})$
  - Note: this is similar to residual connections
- Backward transformation
  - $z_{d+1:n} = x_{d+1:n} - \mathrm{h}_2(x_{1:d})$
  - $z_{1:d} = x_{1:d} - \mathrm{h}_1(z_{d+1:n})$
- Property
  - Given outputs $(x_{1:d}, x_{d+1:n})$ and their total derivative $(\frac{dC}{dx_{1:d}}, \frac{dC}{dx_{d+1:n}})$, with C a loss function, backprop can be performed without storing the activations in the forward path.

Normalizing Flows
Coupling layers – RevNet (Gomez 2017)

- Forward



- Reverse

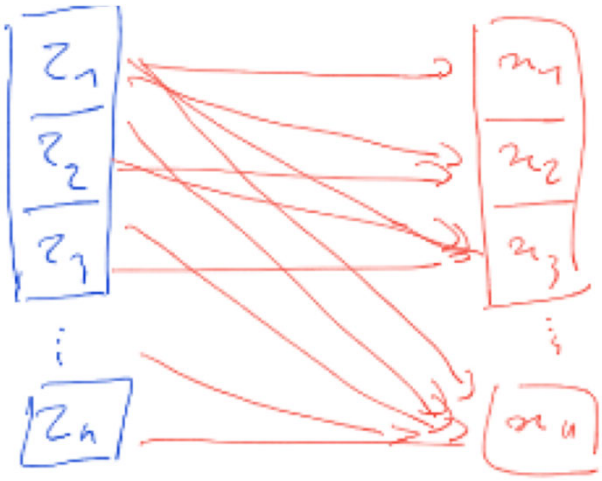


Normalizing Flows
Coupling layers – RevNet (Gomez 2017)

- RevNet is a forward Euler implementation of coupled ODEs
  - $\frac{dx}{dt} = h_1(y)$
  - $\frac{dy}{dt} = h_2(x)$
- Discretization
  - $x_{n+1} = x_n + h_1(y_n)$
  - $y_{n+1} = y_n + h_2(x_{n+1})$

Normalizing Flows
Auto-regressive models

- Let us suppose as before that we have defined an ordering on the variables
  - $z = (z_1, \dots, z_K)$ and the corresponding $x = (x_1, \dots, x_K)$
- A mapping $f: z \to x$, is said autoregressive if x_t can only depends on $z_{\leq t} = z_{1:t}$
- Using notations as before
  - $x_t = \varphi\left(z_t; h_t(z_{1:t-1})\right)$ where:
    - $\varphi: R \to R$
    - $h_t: R^{t-1} \to set\ of\ parameters$ (see examples)
    - h_0 is a constant
- Jacobian is triangular



- $J = \frac{\partial x}{\partial z} = \begin{pmatrix} \frac{\partial \varphi(z_1;h_1)}{\partial z_1} & \cdots & 0 \\ & \ddots & \vdots \\ L & \cdots & \frac{\partial \varphi(z_K;h_K)}{\partial z_K} \end{pmatrix}$
- $\log(\det J) = \sum_{t=1}^{k} \log(|\frac{\partial \varphi(z_t;h_t)}{\partial z_t}|)$

Normalizing Flows
Auto-regressive models

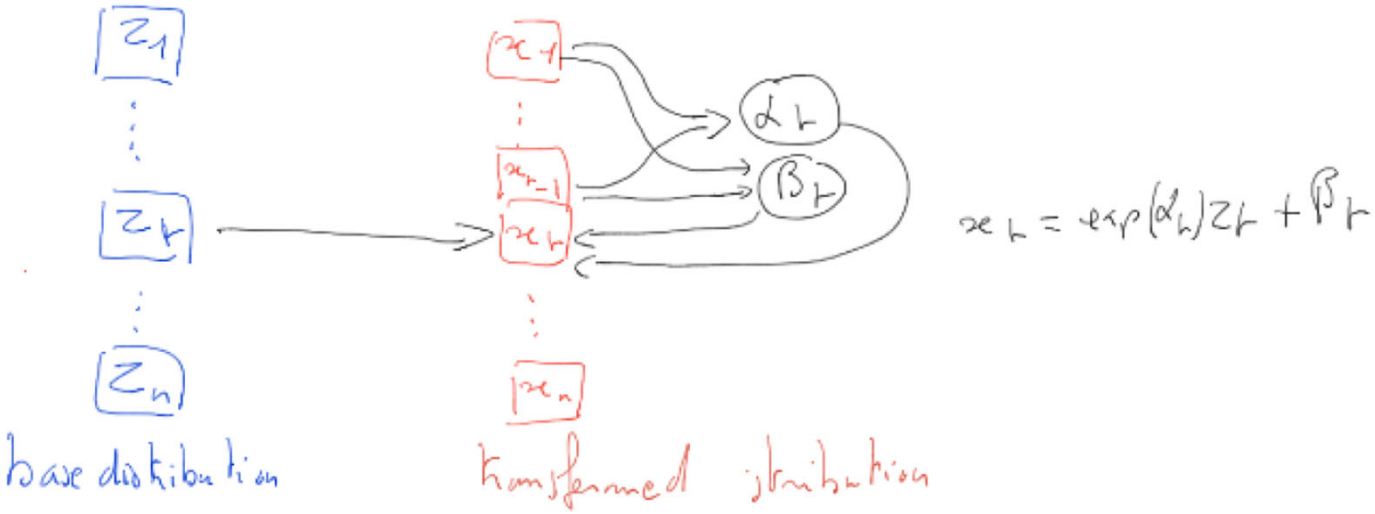
- Given the inverse of f' the inverse of f can be found by recursion
  - $z_1 = \varphi^{-1}(x_1; h_1)$
  - …
  - $z_t = \varphi^{-1}(x_t; h_t(x_{1:t-1}))$
- Complex flows
  - As with the coupling models, AR flow modules can be stacked to implement complex transformations
- Auto-regressive and Coupling models share similarities
  - Both make use of a bijection which takes as input one part of the space and is parameterized by an other part
  - However, computation with auto-regressive models is inherently sequential, and cannot be parallelized on GPUs
- Universality of auto-regressive models (Huang 2018, Jaini 2019)
  - Informally: an AR flow can learn any target density to any required precision given sufficient capacity and data

Normalizing Flows
Auto-regressive models – MAF (Papamakarios 2017)

- Masked Auto-regressive Flows
  - is a conditional Gaussian model
  - has been proposed for density estimation (normalizing way – i.e. inverse mapping)
- Inspiration
  - Comes from gaussian autoregressive model
    - Let $x = (x_1, \dots, x_n) \in R^n$
    - $p(x_i|x_{1:i-1}) = N(x_i|\mu_i, \exp(\alpha_i))$ where $\mu_i = f_{\mu_i}(x_{1:i-1})$ and $\alpha_i = f_{\alpha_i}(x_{1:i-1})$
    - Data can be generated via
      - Sample $z_i \sim N(0,1)$, compute $x_i = z_i exp\ \alpha_i + \mu_i$ with α_i and μ_i as above
      - This provides a characterization of an auto-regressive process as a transformation from the space of random numbers z (here gaussians) to the space of the data x.
      - The inverse can be easily computed as
        - $z_i = (x_i - \mu_i)\exp(-\alpha_i)$ with α_i and μ_i as above

Normalizing Flows
Auto-regressive models – MAF (Papamakarios 2017)

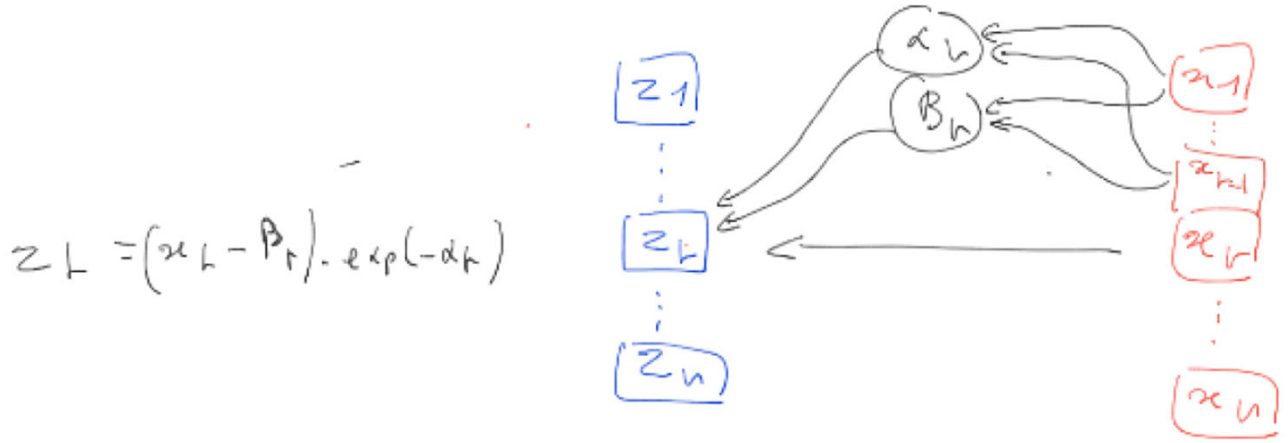
- Forward mapping



- $x_1 = \exp(\alpha_1)\, z_1 + \beta_1, \ldots, x_t = \exp(\alpha_t)\, z_t + \beta_t, \ldots, x_n = \exp(\alpha_n)\, z_n + \beta_n$
  - With α_t and β_t implemented with NNs are functions of $(x_1, \ldots, x_{t-1})$
    - $\alpha_t = f_{\alpha_t}(x_1, \ldots, x_{t-1})$ and $\beta_t = f_{\beta_t}(x_1, \ldots, x_{t-1})$
  - Sampling sequential and slow

Normalizing Flows
Auto-regressive models – MAF (Papamakarios 2017)

- Inverse mapping



- $z_1 = \frac{x_1 - \beta_1}{\exp(\alpha_1)}, \ldots, z_t = \frac{x_t - \beta_t}{\exp(\alpha_t)}, \ldots, z_n = \frac{x_n - \beta_n}{\exp(\alpha_n)}$
  - Jacobian is lower diagonal, determinant can be computed efficiently
    - $\left|\det\left(\frac{\partial g(x;\theta)}{\partial x}\right)\right| = \exp(-\sum_i \alpha_i)$
  - likelihood evaluation is parallelizable and fast using masks (MADE – Germain 2015)

Normalizing Flows
Residual Flows

- Coupling and auto-regressive implementations of flows force the invertibility through the algebraic form of the flow
- A different option is to impose constraints on the functional that make them invertible
- Residual flows are such models inspired by background on Ordinary Differential Equations and dynamical systems
  - Consider an ODE
    - $\frac{d}{dt}x(t) = F\big(x(t), \theta(t)\big)$
    - Where $F: R^n \mathrm{x} \Theta \rightarrow R^n$ determines the dynamics of x and $\theta: R \rightarrow \Theta$ is a parametrization
  - The forward Euler discretization scheme for this ODE is
    - $x_{k+1} = x_k + \epsilon F(x_k, \theta_k)$
  - This is similar to a ResNet module:
  - $x_{k+1} = x_k + F(x_k, \theta_k)$
- Residual flows are Resnet constrained to be invertible

Normalizing Flows
Residual Flows - prerequisite

- Contractive map
  - $F: R^n \to R^n$ is said to be contractive (or a contraction mapping) for a distance function d if there exists a constant $k \in [0,1[$ such that $\forall\, x, y \in R^n$ we have:
    - $d\big(F(x), F(y)\big) \leq k\, d(x, y)$
    - The smallest k value is called the Lipschitz constant
    - let $Lip(f)$ be this constant, $Lip(f) = \sup_{x \neq y}\left(\frac{d\big(F(x),F(y)\big)}{d(x,y)}\right)$
    - This definition generalizes to any metric space (E, d)
  - A contractive map F is uniformly continuous
    - $\forall\, \epsilon > 0, \exists\, \delta > 0$ such that $\forall\, x, y \in E$ with $d(x, y) < \delta$, we have $d\big(F(x), F(y)\big) \leq \epsilon$
      - Note: This is more constraining than simple continuity at y where δ is depends on y

Normalizing Flows
Residual Flows - prerequisite

- Banach fixed point theorem
  - Let (E, d) a non empty complete metric space with a contraction mapping $F: E \to E$. Then F admits a unique fixed point $x^* \in E$.
  - x^* can be found by the following iterative process
    - Start from an arbitrary $x_0 \in E$, iterate $x_t = F(x_{t-1})$, then $x_t \to x^*$
    - This is known as the **successive approximation** method
  - Note:
    - A metric space E is **complete** (or a **Cauchy space**) if every Cauchy sequence of points in E has a limit that is also in E.
    - A Cauchy sequence (f_n) is a sequence whose elements become progressively arbitray close to each other $\lim_{m>n, n\to\infty} ||f_n - f_m|| = 0$
- Property
  - The rate of convergence can be characterized as, e.g. $d(x^*, \mathrm{x_t}) \leq \frac{k^t}{1-k} d(x_0, x_1)$
- Note
  - The theorem provides non only the existence of a fixed point, but also an algorithm (successive approximation method) and a bound on the error

- Banach fixed point
  - $d(x_{n+1}, x_n) \leq k^n d(x_1, x_0)$ (by induction)
  - Let $m > n \in \mathbb{N}$
  - $d(x_m, x_n) \leq d(x_m, x_{m-1}) + \cdots + d(x_{n+1}, x_n)$
  - $d(x_m, x_n) \leq k^{m-1} d(x_1, x_0) + \cdots + k^n d(x_1, x_0)$
  - $= k^n d(x_1, x_0) \sum_{i=0}^{m-n-1} k^i$
  - $\leq k^n d(x_1, x_0) \sum_{i=0}^{\infty} k^i$
  - $= k^n d(x_1, x_0) \frac{1}{1-k}$
  - Let $\epsilon > 0$, since $k \in [0,1[$, $\exists N \in \mathbb{N}$ such that $k^N < \frac{\epsilon(1-k)}{d(x_1,x_0)}$
  - Let $m, n > N$:
  - $d(x_m, x_n) \leq k^n d(x_1, x_0) \frac{1}{1-k} < \frac{\epsilon(1-k)}{d(x_1,x_0)} d(x_1, x_0) \frac{1}{1-k} = \epsilon$
  - This proves that $\{x_n\}$ is Cauchy. By completenss of (E, d), the sequence has a limit x^*
  - Hence $d(x^*, x_n) \leq k^n d(x_1, x_0) \frac{1}{1-k}$
  - x^* is a fixed point of F
    - $x^* = \lim_{n\to\infty} x_n = \lim_{n\to\infty} F(x_{n-1}) = F(\lim_{n\to\infty} x_{n-1}) = F(x^*)$

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

- A ResNet can be described as
  - z_0 initial state (input)
  - $z_{t+1} = z_t + \mathrm{F_t}(z_t)$ dynamics, with f_t a residual bloc, and t representing a layer index
- A ResNet can be made invertible if F_t is constrained appropriately
  - The reverse dynamics writes
  - $z_t = z_{t+1} - \mathrm{F_t}(z_t)$
  - Solving the dynamics backward would implement an inverse of the corresponding ResNet
- iResNet provides a mechanism for designing invertible ResNets

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

- Sufficient condition for invertible ResNets
  - Let $f: R^n \to R^n$ with $f = f_K \circ \cdots \circ f_1$ denote a ResNet with blocks $f_t = I + F_t$.The ResNet f is invertible if $Lip(F_t) < 1$ for all $t = 1 \ldots K$
- Proof
  - f is invertible if each block f_t is invertible
  - Let us consider bloc t: $\mathrm{f_t(z_t)} = z_{t+1} = z_t + \mathrm{F_t(z_t)}$
    - Invertibility of f_t directly follows from F_t being contractive:
    - Let $y_0 = z_{t+1}$ and consider the iteration $y_{k+1} = z_{t+1} - F_t(y_k)$
    - Since F_t is contractive, $G_t: y \to z_{t+1} - F_t(y)$ is also contractive with the same Lipschitz constant
    - Iteration $y_{k+1} = z_{t+1} - F_t(y_k)$ converges to a fixed point $y^* = G_t(y^*)$
    - $y^* = G(y^*) \Leftrightarrow z_{t+1} = y^* + F_t(y^*)$, since the fixed point is unique, $y^* = z_t$
    - This shows that f_t is invertible and then f is invertible

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

- Enforcing $Lip(F_t) < 1$ makes the ResNet module invertible but we have no analytic form for the inverse of f_t
- The successive approximation method provides an **algorithm** for computing the inverse of f_t
  - Algorithm 1
    - Start from any point, e.g. $z_{t+1}$and iterate the fixed point equation $y_{k+1} = z_{t+1} - F_t(y_k)$ for a finite number of iterations
  - The rate of convergence is exponential in the number of iterations t: $d(y^*, y_t) \leq \frac{k^t}{1-k} d(y_0, y_1)$
    - The smaller the constant k, the faster the convergence for inverting the flow, but the residual transformation becomes less flexible – limit case $k = 0$, the iteration converges after 1 iteration, but the transformation is limited to adding a constant to the starting point.
- Invertibility of the whole network
  - Let $f: R^n \to R^n$ with $f = f_K \circ \cdots \circ f_1$ denote the whole ResNet
  - The Lipschitz constant of f, is $Lip(f) = \prod_{t=1}^{K} Lip(F_t)$
  - If F_t is a NN, it is sufficient to make each F_t Lpischitz with $Lip(F_t) \leq 1$ and one of them $Lip(F_t) < 1$

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

- How to make F_t Lipschitz?
  - Many elementwise non linearities (sigmoid, tanh, ReLU) are already Lipshitz with a constant ≤ 1
  - Linear layers can be made contractive w.r.t. a norm by dividing them with a constant strictly superior to their induced operator norm.
  - In (Behrman 2019), spectral normalization (operator norm) is used to make the linear layers contractive w.r.t. the Euclidian norm (Behrman 2019, Chen 2019)
    - Spectral norm
      - For a matrix with real coefficients
      - $\|A\|_2 = \sup\{\|Ax\|, x \in R^n, \|x\| = 1\} = \sigma_{max}(A) = \sqrt{\lambda_{max}(A^T A)}$
      - $\|x\|$ is the euclidean vector norm, $\|A\|_2$ is the corresponding operator norm (spectral norm).
  - The specific NN implementation of ResNet in iResNet, makes use of a convolutional network for the residual block. The spectral radius of each convolutional layer is then constrained to be less than 1.

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

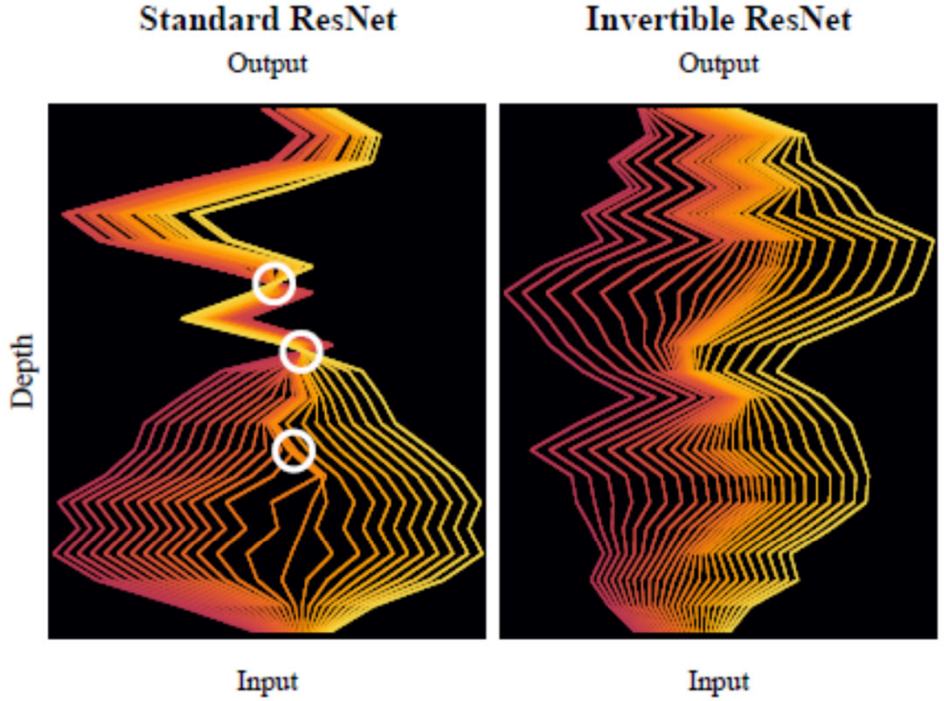


Figure 1. Dynamics of a standard residual network (left) and invertible residual network (right). Both networks map the interval $[-2, 2]$ to: 1) noisy x^3-function at half depth and 2) noisy identity function at full depth. Invertible ResNets describe a bijective continuous dynamics while regular ResNets result in crossing and collapsing paths (circled in white) which correspond to non-bijective continuous dynamics. Due to collapsing paths, standard ResNets are not a valid density model.

(Fig. Behrman 2019)

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

Figure 7. Original images (top), i-ResNets with $c = 0.9$ (middle) and reconstructions from vanilla (bottom). Surprisingly, MNIST reconstructions are close to exact for both models, even without explicitly enforcing the Lipschitz constant. On CIFAR10 however reconstructions completely fail for the vanilla ResNet, but are qualitatively and quantitatively exact for our proposed network.

(Fig. Behrman 2019)

Normalizing Flows
Continuous transformations

- All the methods considered up to now consider transformations through a successiuon of discrete steps, i.e. $z_k = f_k(z_{k-1})$. An alternative is to consider flows in continuous time. The flow is parameterized by its infinitesimal dynamics and then iontegrating to find the corresponding transformation.

- The flow is defined by an ODE:
  - $\frac{dz(t)}{dt} = f(z(t), t; \theta)$
- The generative process is defined as follows
  - Sample from a base distribution $z_0 \sim p_{z_0}(z_0)$
  - Solve the initial value problem: $z(t_0) = z_0, \frac{dz(t)}{dt} = f(z(t), t; \theta)$ in order to obtain $z(t_1)$

Normalizing Flows
Continuous transformations

- The change in log-density follows a second ODE: instantaneous change of variables formula:
- $\frac{\partial logp(z(t))}{\partial t} = -Tr(\frac{\partial \mathrm{f}}{\partial z(t)})$
- The change in log density can be calculated by integrating across time
- $logp(z(t_1)) = logp(z(t_0)) - \int_{t_0}^{t_1} Tr(\frac{\partial \mathrm{f}}{\partial z(t)})\, dt$

Supplementary material - iResnet

Normalizing Flows
Residual Flows - prerequisite

- Let A be a nxn non singular real square matrix
  - iResNet makes use of the following identity
    - $\ln(\det A) = Tr(\ln A)$
    - What is then $\ln A$ and how can it be evaluated?
- Matrix exponential and logarithm
  - Exponential
    - Let A be a nxn real or complex matrix
    - $\exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k, A^0 = I$
    - This series always converges, exp(A) is then well defined
    - Computation
      - If A is diagonal $\exp(A) = diag(\exp(a_{ii}))$, i.e. the diagonal matrix formed by the exponential of the diagonal elements of A
      - If A is diagonalizable (invertible), $A = UDU^{-1}$ with D diagonal and $\exp(A) = Uexp(D)U^{-1}$

Normalizing Flows
Residual Flows - prerequisite

- Matrix exponential and logarithm
  - Logarithm
    - B is a logarithm of A if $A = \exp(B)$
    - The matrix logarithm does not always exists
    - Computation
      - If A is diagonal $\ln(A) = diag(\ln(a_{ii}))$, i.e. the diagonal matrix formed by the logarithm of the diagonal elements of A
      - If A is diagonalizable (invertible), $A = UDU^{-1}$ with D diagonal and $\ln(A) = Uln(D)U^{-1}$
      - If A is sufficiently close to the identity matrix, then a logarithm of A may be computed by means of the following power series:
        - $\sum_{k=1}^{\infty}(-1)^{k+1}\frac{(A-I)^k}{k}$

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

- ResNet as generative models (Flows)
- Recall the change of variable formula
  - $f: Z \to X$ invertible continuous function (bijection), $g = f^{-1}$
  - $p_x(x) = p_z(z) \left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|^{-1}$ (1)
  - $p_x(x) = p_z(g(x)) \left|\det\left(\frac{\partial g(x)}{\partial x}\right)\right|$ (2)

  - Since i-ResNets are invertible, they can be used to parametrize the inverse mapping g in eq (2)
- Sampling application
  - Draw $z \sim p_z(z)$
  - Compute $x = g^{-1}(z)$ with algorithm 1 (fixed point iteration)

  - Example Fig 2 from Berman
- Density estimation
  - One needs to compute $\det\left(\frac{\partial g(x)}{\partial x}\right)$ in order to estimate the density
  - Let us denote the Jacobian of g as $J(g)$

Normalizing Flows
Residual Flows – iResNet (Behrman 2019, Chen 2019)

- The Jacobian determinant of the iResNet cannot be computed directly
  - (Behrman 2019) proposes a biased stochastic estimation, which is imprpoved in (Chen 2019)
- General ideas of this construction
  - Because F is contractive with $Lip(F) < 1$, one has
    - $|\det(J(f))| = \det(J(f))$, i.e. $\det\big(J(f)\big) > 0$
  - Then
    - $\ln|\det(J(f))| = \ln\det\big(J(f)\big) = Tr(\ln\big(J(f)\big) = Tr(\ln(I + J(f))$
  - Let us consider a power series for the trace of the matrix log.
    - $Tr(\ln\big(I + J(f)\big) = \sum_{t=1}^{\infty}(-1)^{t+1}\frac{Tr(J(F)^t}{t}$
  - By truncating the series, one can calculate an approximation of the log Jacobian determinant
  - (Behrman 2019, Chen 2019) propose ways to compute estimators of this series using stochastic algorithms + different tricks

References: papers used as illustrations for the presentation

- GANs
  - Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein Generative Adversarial Networks. In Proceedings of The 34th International Conference on Machine Learning (pp. 1–32). Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In ICCV (pp. 2223–2232).
  - Chen M. Denoyer L., Artieres T. Multi-view Generative Adversarial Networks without supervision, 2017 , https://arxiv.org/abs/1711.00305.
  - Goodfellow I, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio , Generative adversarial nets, NIPS 2014, 2672-2680
  - Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. In arxiv.org/abs/1411.1784.
  - Radford, Luke Metz, Soumith Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2016, http://arxiv.org/abs/1511.06434
  - Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. and Lee, H. 2016. Generative Adversarial Text to Image Synthesis. Icml (2016), 1060–1069.
  - Huang X., Belongie S. J. : Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. CoRR abs/1703.06868 (2017)

References: papers used as illustrations for the presentation

- Normalizing Flows

  - KOBYZEV, I., PRINCE, S., AND BRUBAKER, M.A. 2019. Normalizing Flows: Introduction and Ideas. 1–35.
  - PAPAMAKARIOS, G., NALISNICK, E., REZENDE, D.J., MOHAMED, S., AND LAKSHMINARAYANAN, B. 2019. Normalizing Flows for Probabilistic Modeling and Inference. 1–60.
  - BEHRMANN, J., GRATHWOHL, W., CHEN, R.T.Q., DUVENAUD, D., AND JACOBSEN, J.-H. 2019. Invertible Residual Networks. *ICML*.
  - CHEN, R.T.Q., BEHRMANN, J., DUVENAUD, D., AND JACOBSEN, J.-H. 2019. Residual Flows for Invertible Generative Modeling. 1–21.
  - DINH, L., SOHM-DICKSTEIN, J., AND BENGIO, S. 2017. Density Estimation using Real NVP. *ICLR*.
  - DINH, L., KRUEGER, D., AND BENGIO, Y. 2015. NICE: Non-linear Independent Components Estimation. *ICLR Wshop*.

Physics Based Deep Learning

Outline

- Introduce two complementary topics with in common the notion of dynamics and differential equations

- The dynamics of Neural Networks – explained by ODE
  - NNs with an infinite number of layers modeled as ordinary differential equations (ODE)
  - NNs interpretation as numerical schemes for solving ODEs
  - Opens the way to the
    - Use of numerical ODE solvers for a variety of ML problems
    - Use of ODE solvers theory for analyzing NNs dynamics – e.g. stability – convergence

- Modeling Spatio-temporal dynamics with Neural Networks
  - Four problems
    - Discovering dynamics from data
    - NNs as surogate models for solving PDEs
    - Incorporating physical knowledge in dynamics models
    - Generalization for agnostic ML models for dynamics modeling

Motivations

- Machine learning successes in the numerical world
  - Deep Learning is SOTA in several domains: vision, speech, language, games, . etc
  - Success relies on the availability/ exploitation of data and on computer ressources
  - Mainly restricted to the virtual world
- What about applications in the real world?
  - Much less advanced AI applications
  - The dominant paradigm is still the classical physics based one
  - Classical science and engineering rely on a scientific paradigm involving a deep understanding of the laws of nature in physics, biology, etc
    - Scientific background knowledge accumulated throughout human history
- Challenge: Interaction between the Physical Model Based and the Statistical paradigms
  - How to incorporate prior physical knowledge in Deep Neural Networks?
  - Can statistical models learn physical principles?
    - e.g. conservation laws

Motivations - AI for Science - Challenges

- AI as an enabler for science
  - Integrating scientific knowledge in ML algorithms
  - Discover new scientific knowledge and undertanding
  - Physical plausibility/ interpretability of the solutions provided by ML
  - Robustness guaranties/ uncertainty
  - Generalization: how can agnostic methods be biased to generalize to different conditions/ environments

NN meets ODE

Numerical modeling for designing/ analyzing NNs

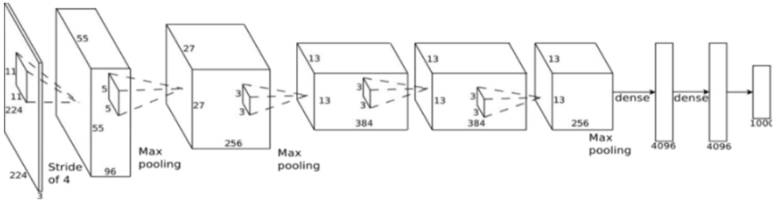
Neural Networks as dynamical systems

- Message
  - Neural networks can be considered as the discretization of dynamical systems
  - Inference and training can be formulated as solving ODEs
- Relevance
  - This establishes a link between large size NNs, their optimization and numerical analysis procedures for solving ODEs
  - This helped popularize the use of differentiable numerical solvers in the ML community
    - Opens the way to integrating physics and ML:
      - Physics based deep learning
      - Differentiable ML based physics

NNs meet ODE

- Deep NN as function composition

AlexNet, Krizhevsky et al. 2012



VGGNet, Simonian et al. 2014

- General form of the NN
  - $F(x,\theta) = f_T \circ f_{T-1} \circ \ldots \circ f_1(x)$

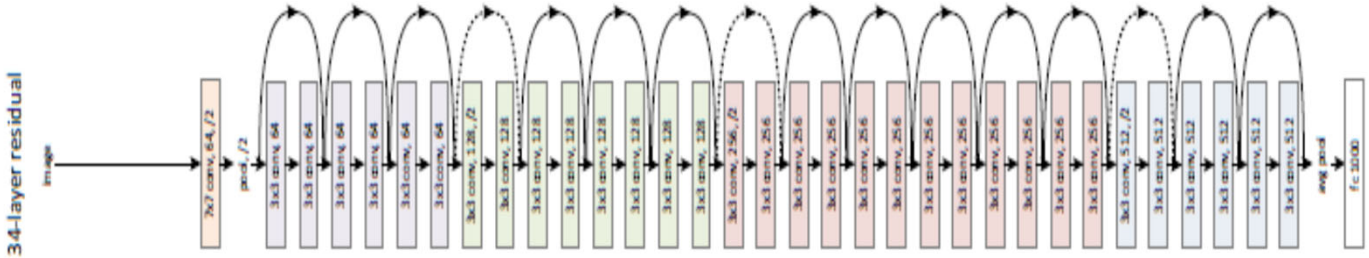
- Learning problem
  - $Min_\theta \quad L(F(x,\theta), y)$
    - $s.t. \quad x_l = f_l(x_{l-1})\ , l = 1 \ldots T$ with initial value $x_0 = x$

← Forward graph of the NN

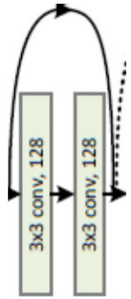
  - x input, y target, θ parameters, x_l layer l activation
  - For simplification we consider here only one couple of input-output data (x, y)
    - Extension to multiple examples is trivial
  - Solving this pb directly leads to back-propagation

NNs meet ODE

- Several NN use skip connections
- e.g. ResNet



- Resnet Module



x_t $\quad x_{t+1} = x_t + f(x_t, \theta_t)$

- Changes the function composition perspective
  - Input x is progressively modified by a residual $f(x, \theta)$
  - Residual f is akin to a derivative function, in continuous time:
    - $\frac{x(t+h)-x(t)}{h} = f(x(t))$

Neural Networks as dynamical systems - Forward pass

- Resnet Module
  - $x_{t+1} = x_t + f(x_t; \theta_t)$, input x_0
    - t is a discrete variable, it indexes the Resnet layers, θ_t are the parameters (weights) of the corresponding layer
    - This is a forward Euler Scheme for an ODE initial value problem (E 2017, Haber 2017, Chang 2018, Lu 2018, …)
- ODE for initial value problem
  - $\frac{dx}{dt} = f(x(t); \theta(t))$ for $t \in [0, T], x(0) = x_0$
    - t is a continuous variable, $\theta(t)$ are the parameters of function f (weights if implemented via a NN)
    - Euler scheme
      - $x_{t+1} = x_t + hf(x_t, \theta_t)$: Euler scheme with step size h
- The forward pass can be modeled as solving an ODE

NNs meet ODE – Training

- NN training relies on gradient based methods
  - During training, the parameters follow an evolution equation
  - Consider steepest gradient descent
    - Initialise θ_0
    - Iterate $\theta_{k+1} = \theta_k - \epsilon \frac{\partial L(\theta_k)}{\partial \theta_k}$
    - This is similar to the Euler scheme for solving the following IVP
      - $\theta(0) = \theta_0$
      - $\frac{d\theta}{dt} = -\epsilon \frac{\partial L(\theta)}{\partial \theta}$ (gradient flow)
      - i.e. gradient descent has a simple numerical interpretation: integrating the gradient flow equation
- Training too can be modeled as solving an ODE
  - More on that e.g. (Scieur et al. 2017)

NNs meet ODEs – Learning problem

- Learning problem with Resnets
  - $Min_\theta \quad L(F(x,\theta),y)$
    $s.t. \quad x_l = x_{l-1} + hf_l(x_{l-1}), l = 1 \ldots T, x_0 = x$

    ← The constraint describes the Forward graph of the Resnet
  - x input, y target, θ parameters, x_l layer l activation, T layers

  - Solving this pb leads to back-propagation and requires alternating
    - Forward pass – numerical scheme for solving
      - $\frac{dx}{dt} = f(x(t),\theta(t))$ for $t \in [0,T], x(0) = x_0$
    - Backward pass – numerical scheme for solving
      - $\frac{d\theta}{dt} = -\epsilon \frac{\partial L(\theta(t))}{\partial \theta}, \theta(0) = \theta_0$

NNs meet ODE
Continuous limit

- Continuous limit
  - If we let $h \to 0$, the ResNet learning problem becomes
  - $Min_\theta L(F(x,\theta),y)$
    - $s.t. \quad \frac{\partial x}{\partial t} = F\left(x(t),\theta(t)\right), t \in [0,T]\ , x_0 = x$
- Two different families of methods for solving the learning problem:
  - Discretize then Optimize
  - Optimize then Discretize

NNs meet ODE
Continuous limit

- learning problem:
  - Discretize then Optimize
    - Discretize the forward equation in time (i.e. discretize the $x(t), \theta(t)$ into a series(x_1, θ_1), ... (x_N, θ_N)), then differentiate the objective w.r.t. the parameters (automatic differentiation – a.k.a autograd) to get the gradient (backward step of BP)
    - This is the classical NN setting leading for example to Back Propagation
    - For the Resnet
      - forward pass corresponds to Euler forward
      - Backward pass corresponds to a differentiation through the Euler scheme
    - Other numerical schemes can be used e.g. RK4, in this case
      - Forward pass corresponds to RK4
      - Backward pass corresponds to a differentiation through the RK4 scheme
      - Note: as for the ResNet, this could be implemented through a specific NN architecture
  - Key idea
    - **One could use a numerical solver to train the NN**
    - The numerical solver implements both the forward and backward pass
    - Opens the way to the use of differentiable solvers
    - **This is useful in particular for modeling dynamical systems**

NNs meet ODE
Continuous limit

- learning problem:
  - Optimize then Discretize
    - Keep the (x, θ) continuous in time
    - Solve the forward and backward equations using e.g. adjoint method
    - The forward and the backward passes are modeled by two different ODEs
    - Forward and backward steps are performed via a Black Box differentiable ODE Solver
    - In the NN literjature this has been popularized by NeuralODE

NNs meet ODE
Continuous limit

- Discretize – optimize vs optimize-discretize
- In practice, discretize – optimize (DTO) should be prefered to optimize-discretize (OTD)
  - The gradient of the backward pass correspond to the function computed during the forward pass (not the case for OTD)
  - Well known in numerical analysis
  - By default auto-differentiation performs DTO

  - OTD opens the way to the use of adaptive solvers and is less demanding on memory usage – no need to store the state of the system during the forward pass, they are recomputed during the backward pass (for the backward equation)

Interlude

Short crash notes on ODEs

Short crash notes on ODEs

- Initial value problem
  - $\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(0) = x_0 \end{cases}$ (1)
    - With $f: [0, T] \times R^n \rightarrow R^n$ differentiable and $x_0 \in R^n$ an initial value
  - What is the value of $x(T)$?
- Integral formulation: solution of (1)
  - $x(T) = x(0) + \int_O^T f\ \big(t, x(t)\big) dt$
  - Property: the integral formulation is equivalent to formulation (1)

Example

$$\frac{\partial x}{\partial t} = 2t; x(0) = 1; x(1)?$$

$$x(1) = x(0) + \int_0^1 2t dt$$

$$x(1) = 1 + 1^2 - 0^2 = 2$$

Short crash notes on ODEs

- Property (Cauchy- Lipschitz)
  - If f is uniformly Lipschitz w.r.t. t and globally w.r.t. variable x, ($\|f((t,x) - f(t,x')\| \leq L\|x - x'\|$) in a neigborhood of $(0, x_0)$, then a solution exists and is unique
- Corollary
  - If f is continuously differentiable w.r.t. t, x, the solution to the initial value problem is unique
- Geometrical interpretation
  - Solution curves for different solutions (initial values) do not intersect

Short crash notes on ODEs

- Trajectories (solution curves)
- Flow of an ODE
  - $\phi : R \times R^n \rightarrow R^n$ of f is defined by $\phi(t, x_0) = x(t)$.

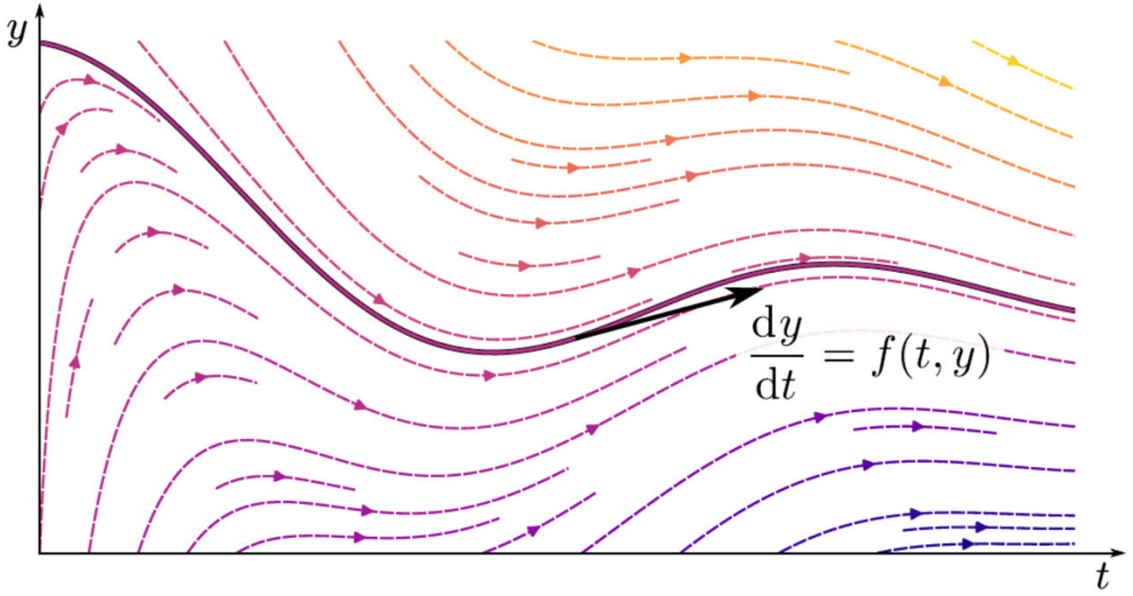
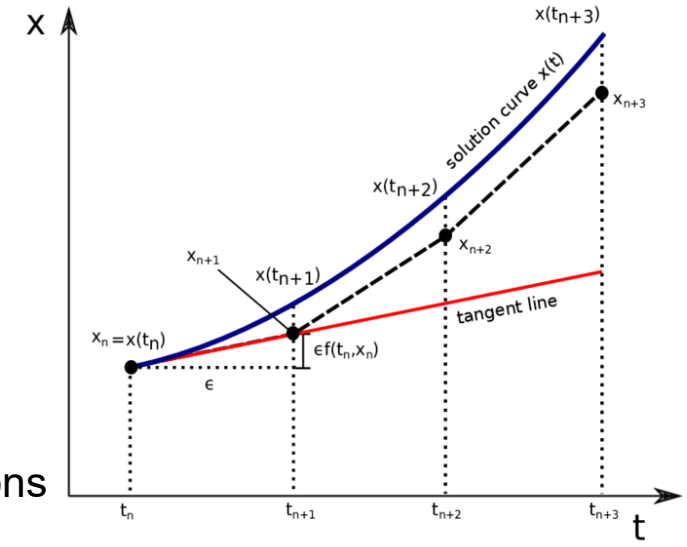


Figure 2.2.: Illustration (in dashed lines) of the continuous flow of an ODE, with a particular solution that is plotted with a solid thicker line. The black arrow represents the tangent to the highlighted solution, which is fully determined by its derivative and initial condition according to variants of the Cauchy-Lipschitz theorem (Demailly, 2006, Chapter V, Section 3.4). In this example, f is defined as $f\colon (t, y) \mapsto \frac{1}{t}(\cos t - y)$, the ODE admitting as solutions over $I = (0, +\infty)$ functions $y_C\colon t \mapsto \frac{C}{t} + \operatorname{sinc} t$ for all $C \in \mathbb{R}$.

Short crash notes on ODEs

- Numerical solvers
  - $x(T) = x(0) + \int_0^T f\left(t, x(t)\right)dt$
    - What if the integral cannot be analytically integrated?
  - $\int_0^T f\left(t, x(t)\right)dt$ is approximated via numerical integration
    - Objective: build a sequence of values $x_0, x_1, \ldots x_N$ that approximate the solution at the discretization points $x(t_0), x(t_1), \ldots, x(t_N)$
- Exemple: Euler forward
  - Step size h
    - $t_{n+1} = t_n + h$
  - Update using the gradient at $f(t_n)$
    - $x_{n+1} = x_n + hf(x_n, \theta_n)$

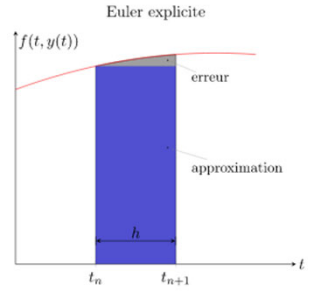


Note: the same solver can be revovered also via the differential formulation through derivative approximations e.g.

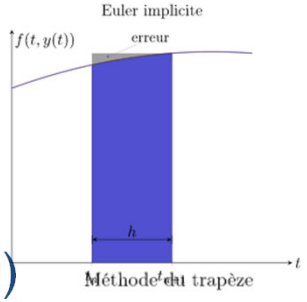
$\frac{\partial x}{\partial t} \approx \frac{x(t+h)-x(t)}{h}$ leads to $x_{n+1} = x_n + hf(t_n, x_n)$

Short crash notes on ODEs

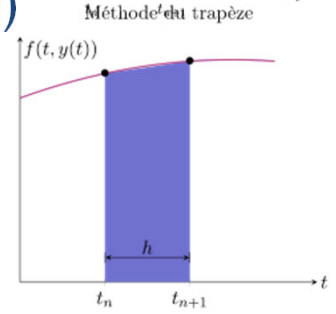
- One step methods - exemples
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ with ϕ a function depending on f
  - Euler forward (explicit)
    - $x_{n+1} = x_n + hf(t_n, x_n)$



  - Euler backward (implicit)
    - $x_{n+1} = x_n + hf(t_{n+1}, x_{n+1})$



  - Runge Kutta e.g. RK2 (RK4 often used as a default option)
  - $$\begin{cases} x_{n,1} = x_n \\ x_{n,2} = x_n + hf\left(t_n, x_{n,1}\right) \\ x_{n+1} = x_n + \frac{h}{2} f\left(t_n, x_{n,1}\right) + \frac{h}{2} f\left(t_{n+1}, x_{n,2}\right) \end{cases}$$



Short crash notes on ODEs

- Summary: using numerical solvers
  - $$\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(t_0) = x_0 \end{cases} \qquad (1)$$
  - What is the value of $x(t_1)$?
  - Note: we introduced here t_0 and t_1 when without loss of generality we used $t_0 = 0$ and $t_1 = T$ before

$$x(t_1) = ODESolve(f(x(t), t, \theta), x(t_0), t_0, t_1)$$

Numerical solver | Differential | Initial value | Initial time | Final time

Short crash notes on ODEs

- Properties
- For simplicity we consider one step methods of the form
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ (2)
  - Stability
    - Intuition: a perturbation of the initial value and of the the ϕ term does not lead to a divergence of the schema
    - Property
      - If there exists $L > 0$ such that $\forall x, x' \in R^m, \forall h \in [0,1], \forall t \in [0, T]$, $\|\phi(t, x, h) - \phi(t, x', h)\| \leq L\|x - x'\|$ then the numerical scheme is stable
      - i.e. ϕ is Lipschitz w.r.t. x, uniformly w.r.t. t and h

Short crash notes on ODEs

- Properties
- For simplicity we consider one step methods of the form
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ (2)
  - Consistency
    - Measures how well the sequence $x_0, x_1, \ldots x_N$ approximates $x(t_0), x(t_1), \ldots, x(t_N)$
    - Truncation error
      - $\epsilon_n = x(t_{n+1}) - x(t_n) - h\phi(t_n, x(t_n), h)$, with $x(t)$ a solution of the ODE (1)
    - A numerical scheme is consistent if the summation of all the truncation errors, for all discretization steps goes to 0 with h
  - Convergence
    - What are the conditions on ϕ for schema (2) to be convergent, i.e. for x_n to converge to $x(t_n)$ when $h \to 0$
    - If the scheme (2) is stable and consistent then it is convergent, meaning that
    - $\lim_{\substack{h \to 0 \\ x_0 \to x(0)}} \sup_{0 \le n \le N} \left|\left| x_n - x(t_n) \right|\right|^2 = 0$

- End of the interlude on ODEs

NNs meet ODE
Discretize then Optimize

- The different numerical schemes do have different specificities, like:
  - Complexity
  - Precision (wr.t. the true solution … which is unknown)
  - Well posedness, stability (forward pass)
  - Reversibility
- The link between ODE numerical schemes and NN allows us to:
  - exploit numerical schemes properties
  - Define new network structures e.g. via constraints on the network parameters
- Objectives
  - For a NN implementation, one is interested in complexity/ stability
  - Improve performance
    - Accuracy, model size (# parameters)

NNs meet ODE
Discretize then Optimize

- NN architectures motivated by ODE numerical schemes
  - Different discretisation methods used in place of Forward Euler
  - Linear multi-step (Lu et al. 2018)
    - $x_{t+1} = (1 - k_t)x_t + k_t x_{t-1} + f(x_t; \theta_t)$, θ_t are the parameters of f
  - Leapfrog Network (Chang et al. 2018)
    - $x_{t+1} = 2x_t - x_{t-1} - h^2 f(x_t, \theta_t)$
  - Runge Kutta order k (example order 2)
    - $\hat{x}_{t+1} = x_t + hf(x_t; \theta_t)$
    - $x_{t+1} = x_t + \frac{h}{2} f(x_t; \theta_t) + \frac{h}{2} f(\hat{x}_{t+1},; \theta_{t+1})$
    - RK order 2 refines the forward Euler scheme
  - Implicit schemes
    - e.g. backward Euler scheme
      - $x_{t+1} = x_t + hf(x_{t+1}; \theta_{t+1})$
      - Note: requires solving a non linear equation at each step
  - Each numerical scheme leads to a specific NN architecture (a la ResNet)

NNs meet ODEs
Stability of ResNet like architectures (Haber 2017, Chang 2018)

- Analysis of the forward stability of a simplified ResNet
  - $x_{t+1} = x_t + hf(x_t; \theta_t)$ for transformations of the form $f(x_t; \theta_t) = \sigma(W_t x_t + b_t)$
  - The corresponding ODE is
    - $$\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ \quad x(0) = x_0 \end{cases}$$
  - Intuition
    - Stability of the ODE ensures that the solution will remain in a bounded set
    - For the numerical scheme, this ensures that small deviations in the input data will not be amplified (this is required e.g. generalization, robustness to adversarial examples, …)
    - (Haber 2017, Chang 2018) propose to control the stability via the NN architectures and constraints on weights

NNs meet ODEs
Stability of ResNet like architectures (Haber 2017, Chang 2018)

- Informal description of the ideas
- Stability of the ODE
  - The ODE is stable if θ_t is changing sufficiently slowly and the Jacobian $J_t \triangleq \nabla_x f(x_t, \theta_t)$ satisfies (sufficient condition):
    - $\max_i Re(\lambda_i(J_t)) \leq 0\,, \forall t \in [0, T]$ with $\lambda_i(J_t)$ the ith eigenvalue of J_t and $Re()$ the real component
- Stability of the forward Euler scheme
  - Forward Euler is stable if:
    - $\max_i |1 + h\lambda_i(J_k)| \leq 1 \; \forall k = 0, \ldots, l-1$ (k indexes the layers) with $J_k \triangleq \nabla_x f(x_k, \theta_k)$

NNs meet ODE
Stability of ResNet like architectures (Haber 2017, Chang 2018)

- Informal description of the ideas
- Intuition
  - The stability conditions of the ODE and of the numerical scheme should be considered in the training optimization problem
    - $\max_i Re\left(\lambda_i(J_t)\right) > 0$ amplifies the signal, and may lead to divergence
    - $\max_i Re\left(\lambda_i(J_t)\right) \ll 0$ may imply signal loss
    - They propose architectures for which $Re\left(\lambda_i(J_t)\right), i = 1..l$ is close to 0
  - For example (Chang 2018) introduces reversible architectures that are stable (analogous to e.g. Revnet)

Interlude

Introduction to the adjoint method via the back propagation algorithm

Notations – matrix derivatives

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \mathrm{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \alpha \in R, W: \ p \times q$$

Vector by scalar

$$\frac{\partial x}{\partial \alpha} = \begin{pmatrix} \frac{\partial x_1}{\partial \alpha} \\ \vdots \\ \frac{\partial x_n}{\partial \alpha} \end{pmatrix}$$

Scalar by vector

$$\frac{\partial \alpha}{\partial x} = \left(\frac{\partial \alpha}{\partial x_1}, \cdots, \frac{\partial \alpha}{\partial x_n} \right)$$

Vector by vector

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

Matrix cookbooks
http://www.cs.toronto.edu/~roweis/notes/matrixid.pdf –
http://www.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/ imm3274.pdf

Back Propagation and Adjoint

- BP is an instance of a more general technique: the Adjoint method
- Adjoint method
  - has been designed for computing **efficiently** the sensitivity of a loss to the parameters of a function (e.g. weights, inputs or any cell value in a NN).
  - It is used to compute the derivative f the loss w.r.t. initial values when the number of parameters is large
  - Can be used to solve different constrained optimization problems (including BP)
  - Is used in many fields like control, geosciences
  - Interesting to consider the link with the adjoint formulation since this opens the way to generalization of the BP technique to more general problems
    - e.g. continuous NNs (Neural ODE)

Back Propagation and Adjoint

- Learning problem
  - $Min_W c = \frac{1}{N}\sum_{k=1}^{N} c(F(x^k), y^k)$
  - With $F(x) = F_l \circ \cdots \circ F_1(x)$
- Rewritten as a constrained optimisation problem
  - $Min_W c = \frac{1}{N}\sum_{k=1}^{N} c(z^k(l), y^k)$
    - Subject to the recurrence equations:

$$\begin{cases} z^k(l) = F_l\left(z^k(l-1), W(l)\right) & \text{Last layer} \\ z^k(l-1) = F_{l-1}(z^k(l-2), W(l-1)) & \\ \cdots & \\ z^k(1) = F_1\left(x^k, W(1)\right) & \text{First layer} \end{cases}$$

  - Note: dimension of the derivatives
    - z and W are **vectors** of the appropriate size
    - e.g. $z(i)$ is $n_z(i) \times 1$ and $W(i)$ is $n_W(i) \times 1$

Back Propagation and Adjoint

- For simplifying, one considers SGD, i.e. $N = 1$
- The Lagrangian associated to the optimization problem is
  - $\mathcal{L}(x, W) = c(z(l), y) - \sum_{i=1}^{l} \lambda_i^T (z(i) - F_i(z(i-1), W(i))$
- The partial derivatives of the Lagrangian are
  - $\frac{\partial \mathcal{L}}{\partial z(l)} = -\lambda_l^T + \frac{\partial c(z(l),y)}{\partial z(l)}$
  - $\frac{\partial \mathcal{L}}{\partial z(i)} = -\lambda_i^T + \lambda_{i+1}^T \frac{\partial F_{i+1}(z(i),W(i+1))}{\partial z(i)}$, $\mathrm{i} = 1 \ldots \mathrm{l} - 1$
  - $\frac{\partial \mathcal{L}}{\partial W_i} = \lambda_i^T \frac{\partial F_i(z(i-1),W(i))}{\partial W(i)}$, $\mathrm{i} = 1 \ldots \mathrm{l}$
  - $\frac{\partial \mathcal{L}}{\partial \lambda_i} = z(i) - F_i(z(i-1), W(i))$, $\mathrm{i} = 1 \ldots \mathrm{l}$
  - Note
    - $\frac{\partial \mathcal{L}}{\partial z(i)}$ is $1 \times n_z(i)$, $\frac{\partial \mathcal{L}}{\partial W_i}$ is $1 \times n_W(i)$, $\frac{\partial \mathcal{L}}{\partial \lambda_i}$ is $1 \times n_\lambda(i)$, λ_i is $n_z(i) \times 1$, $\frac{\partial F_{i+1}(z(i),W(i+1))}{\partial z(i)}$ is $n_z(i+1) \times n_z(i)$, $\frac{\partial c(z(l),y)}{\partial z(l)}$ is $1 \times n_z(l)$, $\frac{\partial F_i(z(i-1),W(i))}{\partial W(i)}$ is $n_z(i) \times n_W(i)$

Back Propagation and Adjoint

- Forward equation
  - $\frac{\partial \mathcal{L}}{\partial \lambda_i} = z(i) - F_i(z(i-1), W(i))\ ,\ \ i = 1 \ldots l$, represent the constraints
  - One wants $\frac{\partial \mathcal{L}}{\partial \lambda_i} = 0,\ \ \ i = 1 \ldots l$
  - Starting from $i = 1$ up to $i = l$, this is exactly the forward pass of BP

Back Propagation and Adjoint

- Backward equation
  - Remember the Lagrangian
    - $\mathcal{L}(x, W) = c(z(l), y) - \sum_{i=1}^{l} \lambda_i^T (z(i) - F_i(z(i-1), W(i))$
  - Since one imposes $(z(i) - F_i\big(z(i-1), W(i)\big) = 0$ (forward pass), one can choose λ_i^T as we want
  - Let us choose the $\lambda_i s$ such that $\frac{\partial \mathcal{L}}{\partial z(i)} = 0, \forall\, i$
  - The $\lambda_i s$ can be computed backward starting at $i = l$ down to to $i = 1$
    - $\lambda_l^T = \frac{\partial c(z(l), y)}{\partial z(l)}$
    - …
    - $\lambda_i^T = \lambda_{i+1}^T \frac{\partial F_{i+1}(z(i), w(i+1))}{\partial z(i)} = \lambda_{i+1}^T \frac{\partial z(i+1)}{\partial z(i)}$
    - …
  - Note
    - $\frac{\partial c(z(l), y)}{\partial z(i)} = \frac{\partial c(z(l), y)}{\partial z(l)} \frac{z(l)}{\partial z(l-1)} \ldots \frac{\partial z(i+1))}{\partial z(i)} = \lambda_i^T$

$$\lambda_i^T = \frac{\partial c(z(l), y)}{\partial z(i)}$$

Back Propagation and Adjoint

- Derivatives
  - All that remains is to compute the derivatives of $\mathcal{L}$ wrt the W_i
    - $\frac{\partial \mathcal{L}}{\partial W(i)} = \lambda_{i+1}^T \frac{\partial F_i(z(i-1),W(i))}{\partial W(i)}$, $\forall\, i$

      - $\frac{\partial F_i(z(i-1),W(i))}{\partial W(i)}$ are easy to compute, this is the classical derivation of state variables in layer i w.r.t. the weights of this layer

Back Propagation and Adjoint – Algorithm Recap

- Recap, BP algorithm with Adjoint
- Forward
  - Solve forward $\frac{\partial \mathcal{L}}{\partial \lambda_i} = 0$
    - $z(1) = F_1(z(0), W(1))$
    - …
    - $z(i) = F_i(z(i-1), W(i))$
- Backward
  - Solve backward $\frac{\partial \mathcal{L}}{\partial z(i)} = 0$
    - $\lambda_l^T = \frac{\partial c(z(l), y)}{\partial z(l)}$
    - …
    - $\lambda_i^T = \lambda_{i+1}^T \frac{\partial F_{i+1}(z(i), w(i+1))}{\partial z(i)} = \lambda_{i+1}^T \frac{\partial z(i+1)}{\partial z(i)}$
- Derivatives
  - $\frac{\partial \mathcal{L}}{\partial W(i)} = \lambda_{i+1}^T \frac{\partial F_i(z(i-1), W(i))}{\partial W(i)}$, $\forall\, i$

Adjoint method – Adjoint equation

- Let us consider the Lagrangian written in a simplified form
  - $\mathcal{L}(x, W) = c(z(l), y) - \lambda^T g(z, W)$
    - z, W represent respectively all the variables of the NN and all the weights
      - Here all the $z(i),\ i = 1 \ldots l$ and the ws for all weigh layers
    - z is a $1 \times n_z$ vector, and W is a $1 \times n_W$ vector
    - $g(z, W) = 0$ represents the constraints written in an implicit form
      - here the system $z(i) - F_{l-1}\big(z(i-1), W(i)\big) = 0, i = 1 \ldots l, \lambda$ is a vetor with dimension the number of (scalar) constraints

The derivative of $\mathcal{L}(x, W)$ wrt W is

- $\frac{d\mathcal{L}(x,W)}{dW} = \frac{\partial c}{\partial z}\frac{\partial z}{\partial W} - \lambda^T\left(\frac{\partial g}{\partial z}\frac{\partial z}{\partial W} + \frac{\partial g}{\partial W}\right)$
- $= \left(\frac{\partial c}{\partial z} - \lambda^T \frac{\partial g}{\partial z}\right)\frac{\partial z}{\partial W} + \lambda^T \frac{\partial g}{\partial W}$
- In order to avoid computing $\frac{\partial z}{\partial W}$, choose λ such that (remember that since the constraint imposes $g(z, W) = 0, \lambda$ can be chosen as we want):
  - $\frac{\partial c}{\partial z} - \lambda^T \frac{\partial g}{\partial z} = 0,$
    - rewriten as

$$\boldsymbol{\frac{\partial g^T}{\partial z}\,\lambda = -\frac{\partial c}{\partial z}}$$

<<<<<<<<< Adjoint Equation, specifies λ,

λ is called the adjoint vector

Adjoint method

- λ is determined from the Adjoint equation
  - Different options for solving λ, depending on the problem
  - For MLPs, the hierarchical structure leads to the backward scheme

- End of the interlude

NNs meet ODE
Optimize then Discretize

- Neural ODE - NODE (Chen et al. 2018)
  - Take the continuous limit of the constrained formulation of BackProp
  - Learning problem
    - $Min_\theta L(F(x,\theta),y)$
      - $s.t.$ $\frac{\partial x}{\partial t} = F\ (x(t);\theta)$
  - (Chen et al. 2018) solve the problem using the adjoint method for continuous time dynamics
    - Generalizes back- propagation

NNs meet ODE
Optimize then Discretize

- Note on the adjoint mehod
  - Learning problem
    - $Min_\theta L(F(x,\theta),y)$
      - $s.t.$ $\frac{\partial x}{\partial t} = F\ (x(t);\theta), \mathrm{t} \in [0,T]$
    - With F a Neural Network, $x \in R^n$
    - This equation plays the same role as the constraint equations in the MLP/ ResNet formulation
    - In a classical ODE, F is defined, **here we want to learn the parameters θ**
  - The problem is rewriten in an unconstrained form using a Lagrangian formalism
    - $L(F(x;\theta),y) + \int_0^T a(t)^T \left(\frac{\partial x}{\partial t} - F\ (x(t);\theta)\right) dt$
    - $a(t)$ is a function of $t, a(t) = \frac{\partial L}{\partial x(t)}$
    - $a(t)^T$ are the adjoint vectors, they play the same role as the λ^T in the adjoint derivation of the B.P. i.e. in the discrete constrained optimization formulation

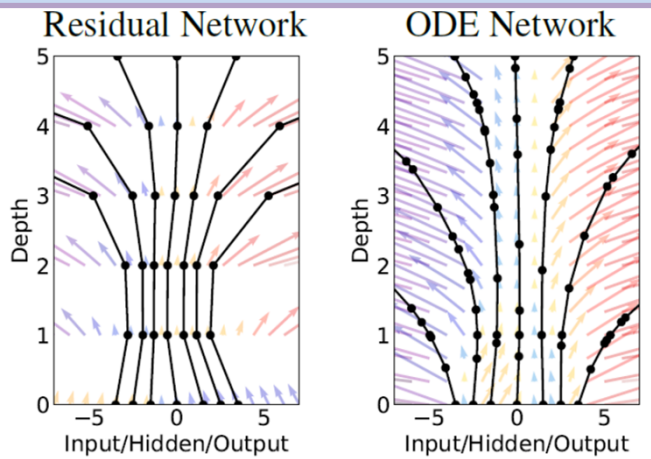
NNs meet ODE
Optimize then Discretize

- Note on the adjoint mehod
  - Solving the optimization problem will proceed in tha same way as for the MLP, in 3 steps:
    - Forward propagation
    - Backward propagation
    - Gradient computation
  - Both the Forward and the Backward passes amount to solve an ODE
  - Each will be solved through a a call to a solver for their specific ODE: ODESolve(.)

NNs meet ODE
Optimize then Discretize

- Forward pass
  - This amounts to solve
    - $\frac{\partial x}{\partial t} = F\ (x(t), \theta(), \mathrm{t} \in [0, T]$
    - $x(0) = x_0$
  - Solver call

$$x(T) = ODESolve(F(x(t), t, \theta), x(t_0), t_0 = 0, t_1 = T)$$



NNs meet ODE
Optimize then Discretize

- Backward pass
  - Lagrangian
    - $L(F(x,\theta),y) + \int_0^T a(t)^T \left(\frac{\partial x}{\partial t} - F\left(x(t),\theta(t)\right)\right) dt$
    - Our objective is to compute the derivative w.r.t. the parameters θ in order to apply a gradient algorithm
  - The derivative w.r.t. the parameters θ is:
    - $\frac{dL}{d\theta} = -\int_{t=T}^{0} a(t)^T \frac{\partial F\left(x(t),\theta(t)\right)}{\partial \theta} dt$
  - Where
    - $a(t) = \frac{\partial L}{\partial x(t)}$ is the adjoint vector, function of t
    - $a(t)$ is itself characterized by an ODE:
      - $\frac{da(t)}{dt} = -a(t)^T \frac{\partial F(x(t),\theta)}{\partial x(t)}$ (instantaneous analog of the chain rule)

NNs meet ODE
Optimize then Discretize

- Backward pass
  - This ODE will be solved backward, starting at $a(T) = \frac{\partial L}{\partial x(T)}$ which is directly computable from the expression of L
    - $\mathrm{a}(t_0 = 0) = ODESolve(-a(t)^T \frac{\partial F(x(t),\theta(t))}{\partial x(t)}, a(T), t_0 = T, t_1 = 0)$
    - Note: this is a backward integration from $t_0 = T$ to $t_1 = 0$
  - We need to compute $\frac{\partial F(x(t),\theta(t))}{\partial x(t)}$, and for that we need $x(t)$
    - But $x(t)$ has not been saved in the forward pass.
    - One compute them starting from $x(T)$, by ODE reversibility
    - $x(0) = ODESolve(F(x(t), t, \theta), x(t_1), t_0 = T, t_1 = 0)$
- Summary of the backward pass

$$x(0) = ODESolve(F(x(t), t, \theta), x(t_1), t_0 = T, t_1 = 0)$$
$$\mathrm{a}(t_0 = 0) = ODESolve(-a(t)^T \frac{\partial F(x(t),\theta(t))}{\partial x(t)}, a(T), t_0 = T, t_1 = 0)$$

NNs meet ODE
Optimize then Discretize

- Gradients
  - $\frac{dL}{d\theta} = \int_{t=T}^{0} a(t)^T \frac{\partial F(x(t),\theta(t))}{\partial \theta} dt$
  - $\frac{dL}{d\theta}$ is itself the solution of an ODE for a vector $b(t)$ such that
    - $b(T) = 0$
    - $\frac{db(t)}{dt} = -a(t) \frac{\partial F(x(t),\theta(t))}{\partial \theta}$
    - $\frac{dL}{d\theta} = ODESolve(-a(t)^T \frac{\partial F(x(t),\theta(t))}{\partial \theta}, 0\_\theta, t_0 = T, t_1 = 0)$
- Summary for gradients computations

$$x(0) = ODESolve(F(x(t), t, \theta), x(t_1), t_0 = T, t_1 = 0)$$
$$\mathrm{a}(t_0 = 0) = ODESolve\left(-a(t)^T \frac{\partial F(x(t),\theta(t))}{\partial x(t)}, a(T), t_0 = T, t_1 = 0\right)$$
$$\frac{dL}{d\theta} = ODESolve(-a(t)^T \frac{\partial F(x(t),\theta(t))}{\partial \theta}, 0\_\theta, t_0 = T, t_1 = 0)$$

NNs meet ODE
Optimize then Discretize

- Summary of the algorithm
- Forward pass

$$x(T) = ODESolve(F(x(t), t, \theta), x(t_0), t_0 = 0, t_1 = T)$$

- Gradient « back propagation »

$$x(0) = ODESolve(F(x(t), t, \theta), x(t_1), t_0 = T, t_1 = 0)$$

$$\mathrm{a}(t_0 = 0) = ODESolve\left(-a(t)^T \frac{\partial F(x(t), \theta(t))}{\partial x(t)}, a(T), t_0 = T, t_1 = 0\right)$$

$$\frac{dL}{d\theta} = ODESolve(-a(t)^T \frac{\partial F(x(t), \theta(t))}{\partial \theta}, 0\_\theta, t_0 = T, t_1 = 0)$$

Neural Networks as dynamical systems - Continuous limit – Neural ODE (Chen et al. 2018)

- The forward pass can be modeled as solving an ODE
- The backward pass (training) can also be modeled as solving an ODE
  - Neural ODE (Chen 2018) solves a continuous optimization problem
    - Both the Forward and the Backward passes amount to solve an ODE
    - Each will be solved trough a a call to a solver for their specific ODE: ODESolve(.)
- Neural ODE
  - Highlighted the relations between training Deep NNs and solving ODEs
  - Initiated the development of libraries usable in Deep NN platforms, eg. TorchDiffEq, JuliaDiffEq, etc.
  - This leads to differentiable physics – and the combination of solvers and NNs

Neural Networks for modeling dynamical systems

Four problems

Neural Networks for modeling dynamical systems
Outline

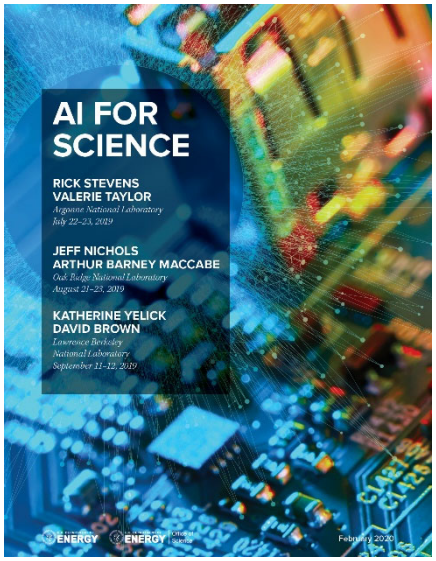
- Context
  - AI4Science
- Four problems
  - Discovering dynamics from data
  - NNs as surogate models for solving PDEs
  - Incorporating physical knowledge in statistical dynamics models
  - Generalization for agnostic ML models for dynamics modeling

Context: AI for Science

- What about applicationsof AI in the real world for Science and Engineering?
  - The dominant paradigm is still the classical physics based one
  - Classical science and engineering rely on a scientific paradigm involving a deep understanding of the laws of nature in physics, biology, etc
    - Scientific background knowledge accumulated throughout human history
  - Recently data has become abundant in many scientific domains and contain lot of information underexploited
  - Question:
    - How to make use of AI in scientific domains?
    - Is it possible to leverage the classical scientific paradigm together with the more recent paradigm of data science?

Context - AI for science

- AI hype has motivated investigations in several scientific/ engineering domains
- AI for science as a new scientific paradigm
- Example: DOE report 2020



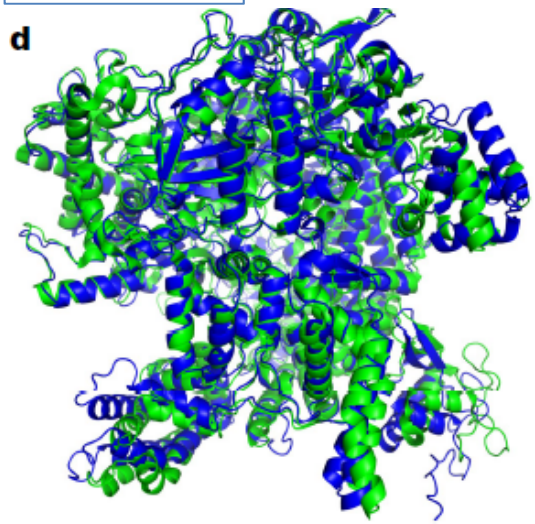
  - Examines the potential of AI for several application domains
    - Materials, environment, life sciences, high energy physics, smart energy infrastructures, etc
  - Highlights commonalities for different domains
    - Models: Integrating domain knowledge in AI systems, Integration with large scale HPC models
    - Data: Creation of FAIR (Findable, Accessible, Reusable) data
    - Infrastructures

Context - AI for Science – Domain examples

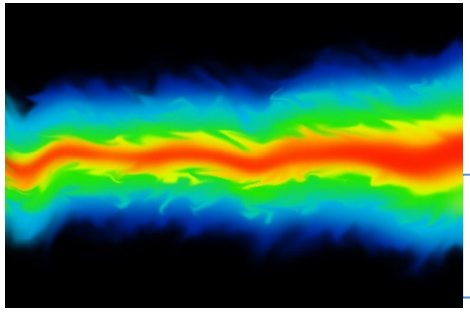
Climate

CLIMATE MACHINE: HTTPS://CLIMA.CALTECH.EDU/
We are developing the first Earth system model that automatically learns from diverse data sources: machine learning and data assimilation ….

Biology



AlphaFold, Jumper et al. 2021



Simulation - computational Fluid Dynamics
https://blog.spatial.com/cfd-analysis

Context - AI for Science - Challenges

- AI as an enabler for science
  - Integrating scientific knowledge in ML algorithms
  - Discover new scientific knowledge and undertanding
  - Physical plausibility/ interpretability of the solutions provided by ML
  - Robustness guaranties/ uncertainty
  - Generalization: how can agnostic methods be biased to generalize to different conditions/ environments

NN for dynamical processes

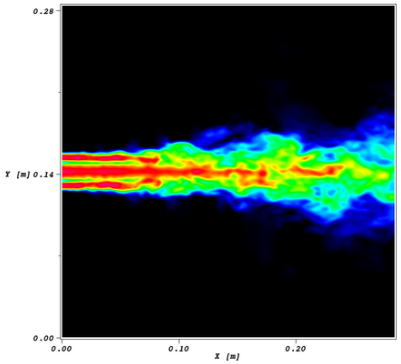
Neural Networks for modeling dynamical systems
Motivation

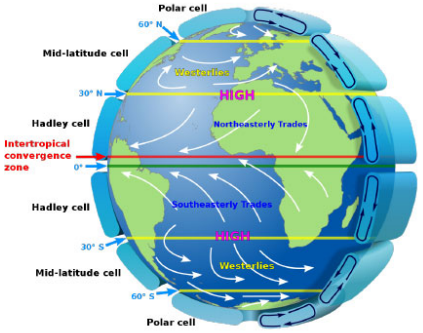
- Modeling the complex dynamics arising in natural/ physical processes
  - Objective: understanding, predicting, controling
- Physical models
  - Mathematical equations of dynamical systems
  - Often take the form of PDEs and associated numerical models
  - Stem from a deep understanding of the underlying physics
- Data driven modeling
  - In many cases data are plentiful (climate, simulations, etc)
  - Can we leverage ML for modeling these complex systems?
    - Way more complex than current ML successes (vision, language)
- Challenge: Interaction between the physical model based and the statistical paradigms

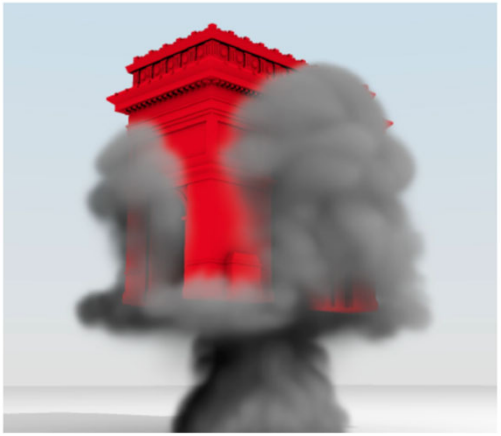
Neural Networks for modeling dynamical systems
Motivation

- Applications domains - examples

Computational Fluid Dynamics	Earth System Science - Climate	Graphical design



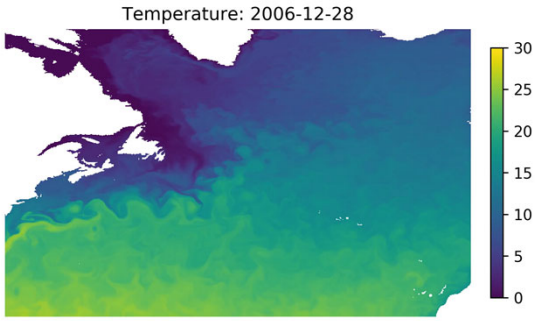


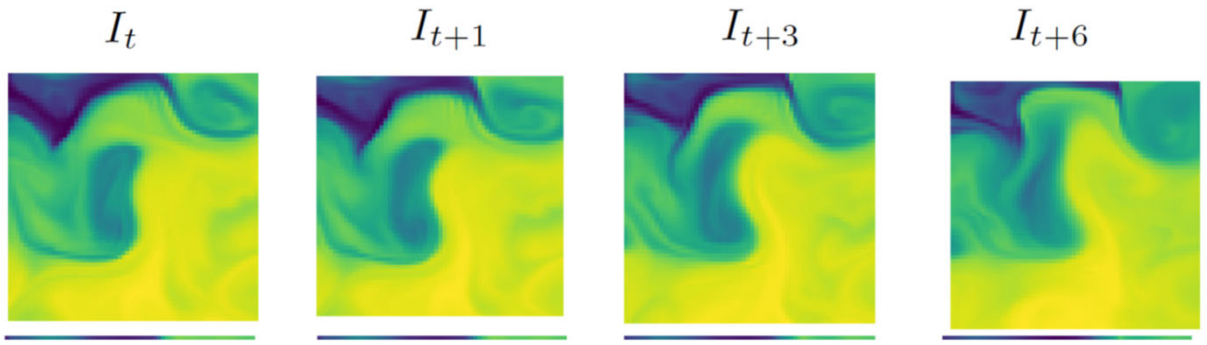


Tompson et al. 2017

- Objectives
  - Reduce computational cost – e.g. CFD
  - Complement physical models
  - Replace physical models

Neural Networks for modeling dynamical systems
Example of a physical process: temperature at the surface of north Atlantic





Discovering dynamics from data

- Question
  - Is it possible to learn from observation the underlying physics, here the form of the underlying PDE?
- Illustration
  - The SINDy framework (Rudy et al 2017)
    - Given a dictionary of differential terms, learn a sparse regression
  - PDE-Net
    - More modern attempt to the problem (Long et al. 2018)
      - Limited hypothesis + Neural Networks + automatic differentiation

Neural Networks for modeling dynamical systems
Objectives

- Is it possible to learn the laws of nature from data?
- Arguments
  - Data are ubiquitous in many domains
    - Climate, Finance, Epidemiology, ecology, etc
  - Variables or governing dynamics may be partially known or unknown
- Objective
  - Consider non linear equations of the form
    - $u_t = F(u, u_x, u_{xx}, \dots, x, \theta), \mathrm{x} \in \Omega \subset \mathbb{R}^2, t \in [0, T], u(t, x) \in \mathbb{R}^d, u_t \in \mathbb{R}^d$
    - with $u_t = \frac{\partial u}{\partial t}, u_x = \frac{\partial u}{\partial x}, u_{xx} = \frac{\partial^2 u}{\partial x^2}$, etc
  - Relevant variables and F may be unknown
  - Is it possible to learn the explicit form of the underlying PDE and perform accurate predictions?
    - Reaching both goals should allow
      - to generalize to new initial conditions – see generalization discussion
      - If long term forecasting is accurate, one could be confident on the learned underlying model

Data driven discovery of PDEs– (Rudy et al. 2017)
Sparse regression

- Collect observations
- Consider a library of terms, e.g. system state, derivatives, cross term $\frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, u\frac{\partial u}{\partial x} \dots$
- Use sparse linear regression i.e. additive model, to fit observations on library terms
  - i.e. solve the inverse problem – learn the regression coefficients $a_i, i = 0, 1, \ldots$
  - $\frac{\partial u}{\partial t} = a_0 \frac{\partial u}{\partial x} a_1 u \frac{\partial u}{\partial x} + a_2 \frac{\partial^2 u}{\partial x^2} + \dots$

Neural Networks for modeling dynamical systems
Data driven discovery of PDEs– (Rudy et al. 2017)
Sparse regression

- Collect observations
- Consider a library of terms, e.g. system state, derivatives, …
- Use sparse linear regression to fit observations on library terms

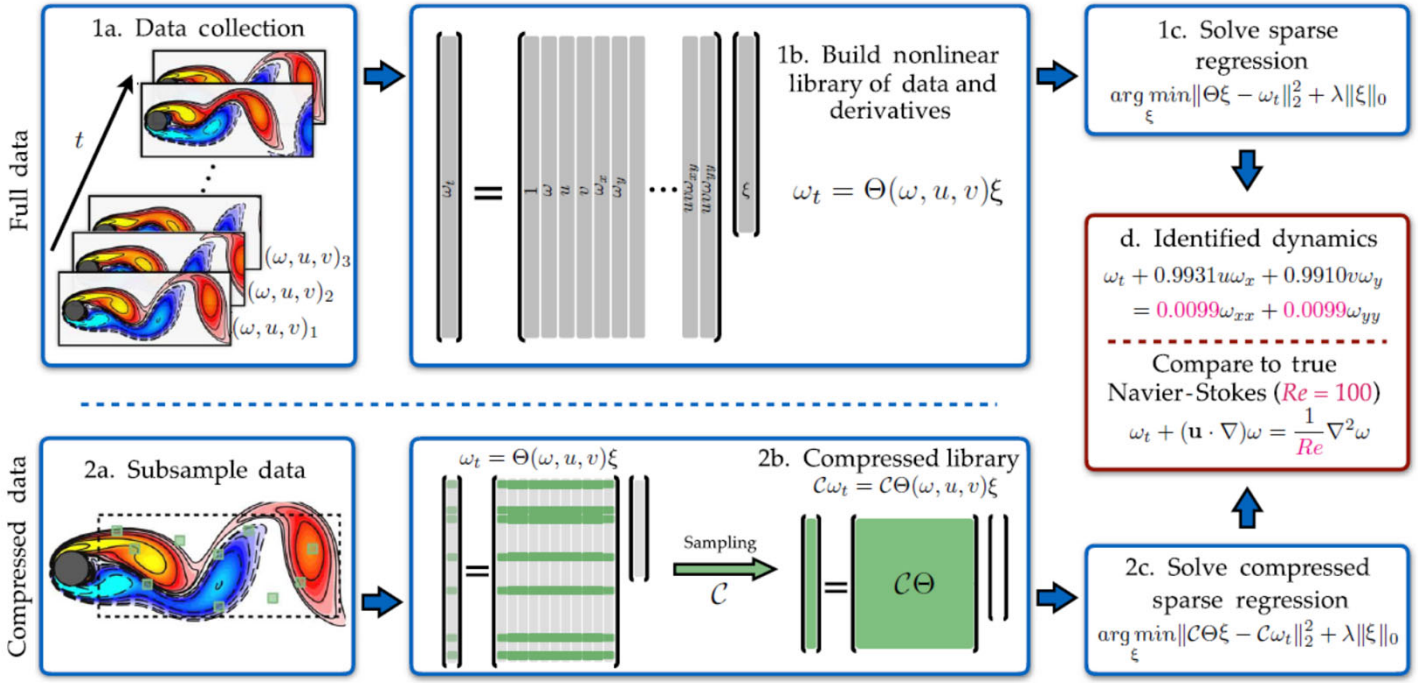
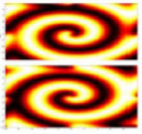
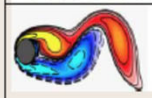


Fig. 1. Steps in the PDE functional identification of nonlinear dynamics (PDE-FIND) algorithm, applied to infer the Navier-Stokes equations from data. (1a) Data are collected as snapshots of a solution to a PDE. (1b) Numerical derivatives are taken, and data are compiled into a large matrix Θ, incorporating candidate terms for the PDE. (1c) Sparse regressions are used to identify active terms in the PDE. (2a) For large data sets, sparse sampling may be used to reduce the size of the problem. (2b) Subsampling the data set is equivalent to taking a subset of rows from the linear system in Eq. 2. (2c) An identical sparse regression problem is formed but with fewer rows. (d) Active terms in ξ are synthesized into a PDE.

Neural Networks for modeling dynamical systems
Data driven discovery of PDEs– (Rudy et al. 2017)
Examples

Table 1. Summary of regression results for a wide range of canonical models of mathematical physics. In each example, the correct model structure is identified using PDE-FIND. The spatial and temporal sampling of the numerical simulation data used for the regression is given along with the error produced in the parameters of the model for both no noise and 1% noise. In the reaction-diffusion system, 0.5% noise is used. For Navier-Stokes and reaction-diffusion, the percent of data used in subsampling is also given. NLS, nonlinear Schrödinger; KS, Kuramoto-Sivashinsky.

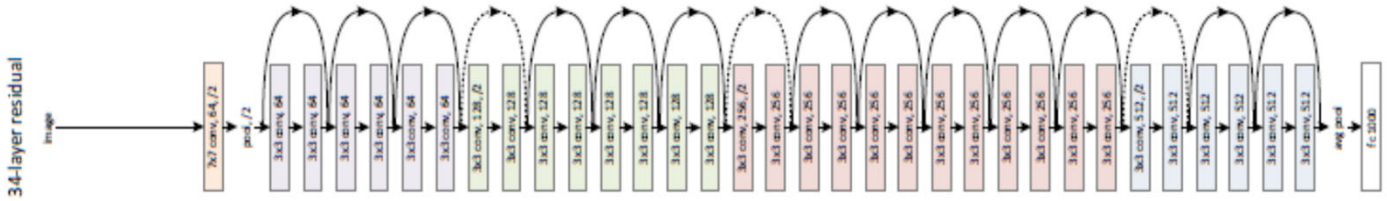
PDE	Form	Error (no noise, noise)	Discretization
 KdV	$u_t + 6uu_x + u_{xxx} = 0$	$1 \pm 0.2\%, 7 \pm 5\%$	$x \in [-30, 30], n = 512,\ t \in [0, 20], m = 201$
 Burgers	$u_t + uu_x - \epsilon u_{xx} = 0$	$0.15 \pm 0.06\%, 0.8 \pm 0.6\%$	$x \in [-8, 8], n = 256,\ t \in [0, 10], m = 101$
 Schrödinger	$iu_t + \frac{1}{2}u_{xx} - \frac{x^2}{2}u = 0$	$0.25 \pm 0.01\%, 10 \pm 7\%$	$x \in [-7.5, 7.5], n = 512,\ t \in [0, 10], m = 401$
 NLS	$iu_t + \frac{1}{2}u_{xx} + \lvert u\rvert^2 u = 0$	$0.05 \pm 0.01\%, 3 \pm 1\%$	$x \in [-5, 5], n = 512,\ t \in [0, \pi], m = 501$
 KS	$u_t + uu_x + u_{xx} + u_{xxxx} = 0$	$1.3 \pm 1.3\%, 52 \pm 1.4\%$	$x \in [0, 100], n = 1024,\ t \in [0, 100], m = 251$
u  v Reaction Diffusion	$u_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v$ $v_t = 0.1\nabla^2 v + \omega(A)u + \lambda(A)v$ $A^2 = u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2$	$0.02 \pm 0.01\%,\ 3.8 \pm 2.4\%$	$x, y \in [-10, 10], n = 256,\ t \in [0, 10], m = 201$ subsample 1.14%
 Navier-Stokes	$\omega_t + (\mathbf{u} \cdot \nabla)\omega = \frac{1}{Re}\nabla^2 \omega$	$1 \pm 0.2\%\ ,\ 7 \pm 6\%$	$x \in [0, 9], n_x = 449,\ y \in [0, 4], n_y = 199,$ $t \in [0, 30], m = 151,$ subsample 2.22%

Neural Networks for modeling dynamical systems
Data driven discovery of PDEs– (Rudy et al. 2017)

- Assumptions
  - The full state of the system is observable <<<<<< strong assumption
  - An overcomplete dictionary is available
    - All required terms are available
    - Non linearity in the derivatives (e.g. multiplication of two derivatives) requires cross product terms in the dictionary, i.e. works only for linear combinations of inputs
  - Only a few terms will be involved in the dynamics
  - These are the conditions for sparse regression to work
- Comments
  - Allows interpretability
  - Requires numerical differentiation around the sample points in order to estimate the derivatives – less flexible than automatic differentiation
    - Pb for large datasets and large dimensional spaces
  - In their experiments noise level does not exceed 1%, i.e. not robust to noise, and works only for clean data originating from PDE simulation
  - Requires « enough » data points

Neural Networks for modeling dynamical systems
PDE-Net (Long et al. 2018) PDE-Net 2.0 (Long et al. 2019)

- Inspired from classical NN architectures, e.g. ResNet



- Each block in PDE-Net implements
  - Spatial differentiation via convolution filters
    - Exploits local spatial properties with learned convolutional filters
    - Used to approximate spatial differential operators
    - Constraints on the filters forces the approximation of differential operators of given order
  - Temporal differentiation via skip connections
    - $x_{t+1} = x_t + f(x_t)$, implements a numerical ODE scheme, here 1 step Euler
- Implement the F dynamics with a specific NN module
  - This module is able to represent all the polynomials of its inputs (differential operators) with a limited number of operations

Neural Networks for modeling dynamical systems
PDE-NET (Long et al. 2018) PDE-NET 2.0 (Long et al. 2019)

- Considers equations of the form
  - $u_t = F(u, u_x, u_{xx}, \dots, \mathrm{x}, \theta), \mathrm{x} \in \Omega \subset \mathbb{R}^2, t \in [0, T], u(t, x) \in \mathbb{R}^d, u_t \in \mathbb{R}^d$
- PDE-Net 2.0 combines elementary NN modules/ blocks
  - Each block discretizes the PDE
    - In time via e.g. forward Euler
    - In space via finite differences
  - There are 2 main components (i) automatic determination of the differential operator dictionary, (ii) approximation of F via numerical integration
- Each elementary module (δt −block) implements 1 step of forward Euler (time discretization)
  - $u(t + \delta t, .) \approx u(t, .) + \delta t. Net(D_{00}u, D_{01}u, D_{10}u, \dots)$
  - With
    - D_{ij} convolution operator associated to a filter approximating a spatial differential operator and learned
      - $D_{ij}u \approx \frac{\partial^{i+j}u}{\partial^i x \partial^j y}$
    - *Net* is a specific « neural network » approximating F

Neural Networks for modeling dynamical systems
PDE-NET 2.0 (Long et al. 2019) – General architecture

- δt –block – one step ahead prediction i.e. δt

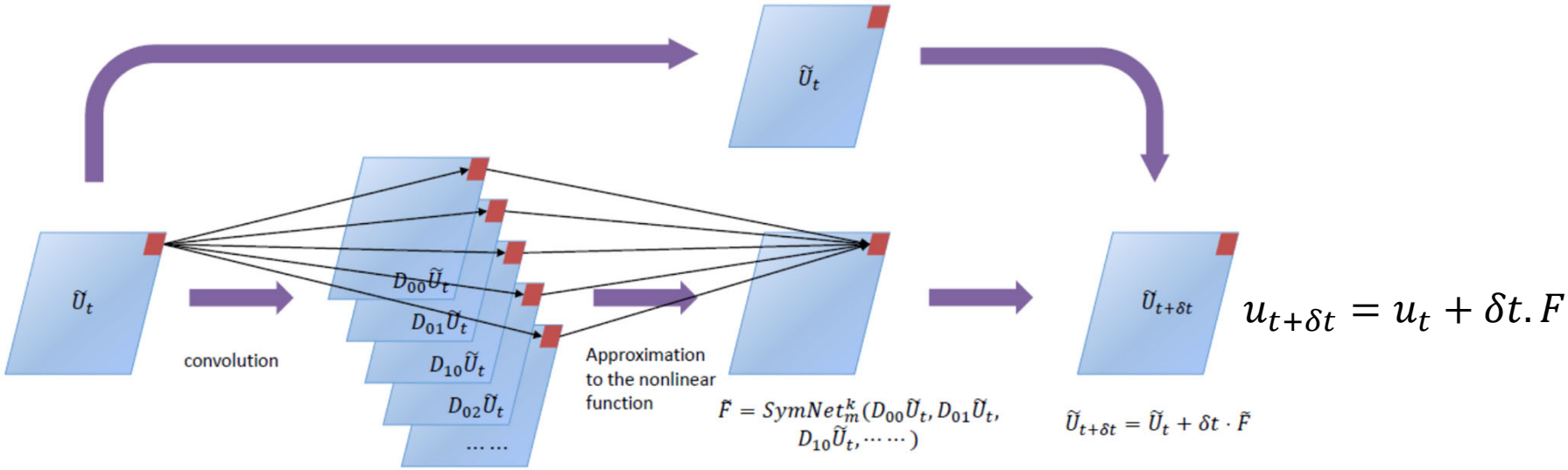


Figure 1: The schematic diagram of a δt-block.

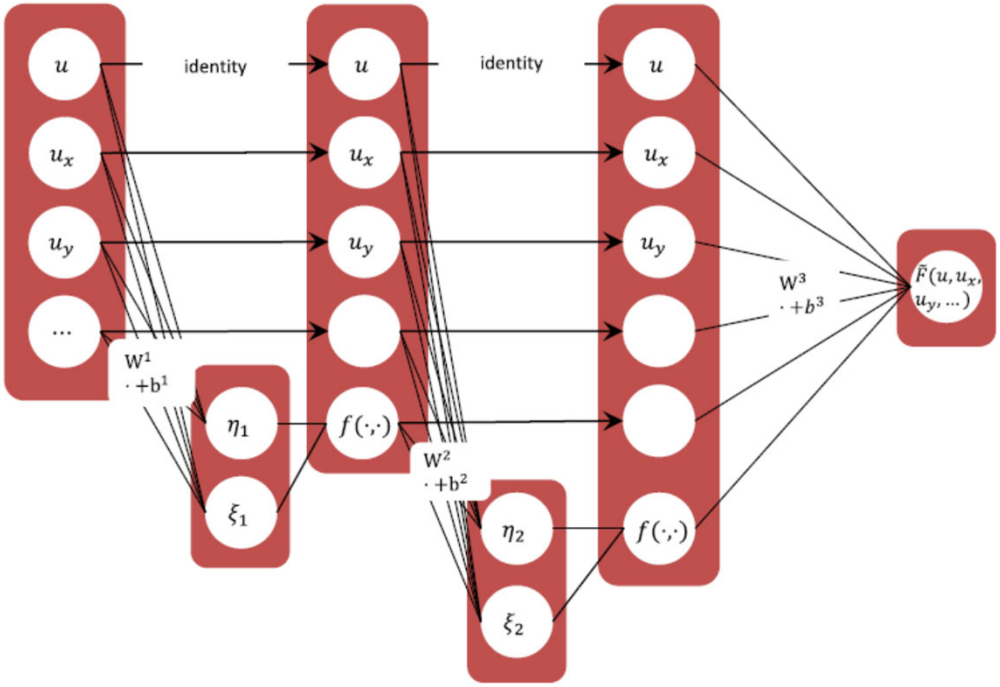
Spatial filters: learned convolutions

Net_m^k: combines its arguments to compute F

Time integration

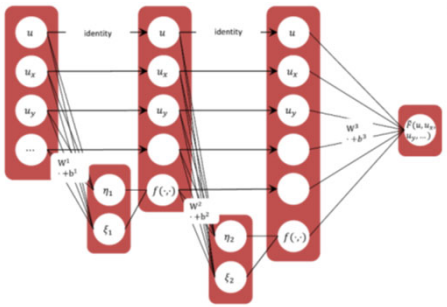
Neural Networks for modeling dynamical systems
PDE-NET 2.0 (Long et al. 2019) – learning the differential operator

- Objective
  - Given a dictionary of elementary terms learn combinations
  - Module Net_m^k, m inputs, k layers.
  - Example on the figure Net_m^2 a 2 hidden layer



Neural Networks for modeling dynamical systems
PDE-NET 2.0 (Long et al. 2019) – learning the differential operator

- Example Net_m^2 a 2 hidden layer, m inputs

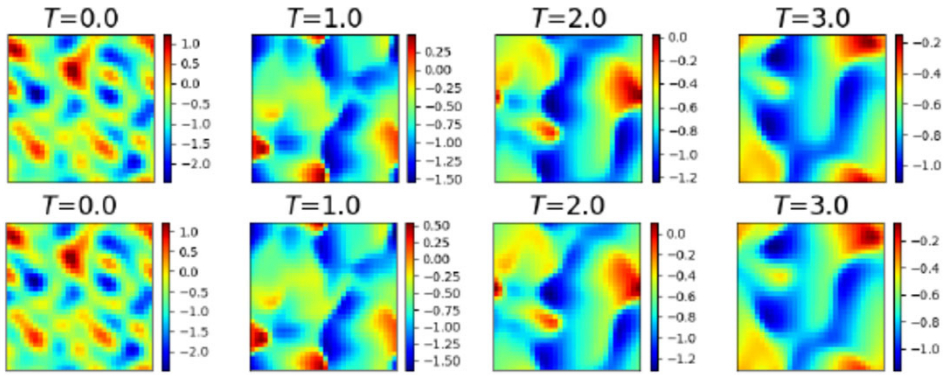


- One layer to the next: performs identity (copy the m inputs) and adds a new value $f(\eta, \xi) = \eta \times \xi \; or \; \eta/\xi$ (either multiplication or division)
- We consider in the following multiplication as an example $f(\eta, \xi) = \eta \times \xi$
  - For a given layer, let $x = (x_1, \dots, x_m) \in \mathbb{R}^m$ the layer inputs
  - Let $\eta = w_\eta . x, \xi = w_\xi . x$, with $w_\eta = (w_{\eta,1}, \dots, w_{\eta,m}) \in \mathbb{R}^m$
  - $f(\eta, \xi) = \eta \times \xi = \sum_{j,l} w_{\eta,j} w_{\xi,l} x_j x_l$, i.e. a 2nd order polynom of its inputs
- Property
  - Net_m^k represents all the polynomial of input variables $(x_1, \dots, x_m)$, up to order k
- Training proceeds layerwise

Neural Networks for modeling dynamical systems
PDE-NET (Long et al. 2018)

- More flexible than Sindy (Rudy et al 2017)
  - learned filters
  - Efficient implementation of polynomials
  - Still requires access to the true state variables
  - Provides an explicit form of the underlying PDE
- e.g. 2 D Burgers
  - Only 1st component (u) shown here
    - Correct: $u_t = -uu_x - vu_y + 0.05(u_{xx} + u_{yy})$
    - Identified: $u_t = -0.98uu_x - 0.97vu_y + 0.054u_{xx} + 0.054u_{yy}$

u component
- Top: true dynamics
- Bottom: predicted, $\delta t = 0.01$



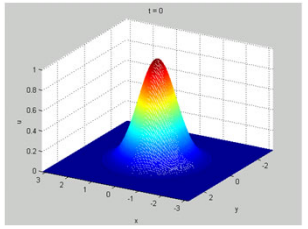
NN for solving Partial Differential Equations

Learning surrogate models for solving PDEs (Lagaris 1998, Sirignano 2018, Raissi 2019)

- Classical numerical schemes for solving PDEs
  - May be extremely expensive
  - Not applicable in large dimensions
- Objective
  - Build a reduced order (parametric) model, implemented by a NN, to offer a cheap approximate solution of a PDE
  - Assumption: the form of the PDE is known as for classical solvers
- Results
  - The algorithm solves the PDE using a single parametric function, for all space and time conditions
  - The algorithm solves a unique IVP – and shall be re-trained for a new IVP
    - i.e. no generalization when the initial conditions or the dynamics change
    - Known for not extrapolating in time

Learning surrogate models for solving PDEs - Known PDE (Lagaris 1998 , Sirignano 2018, Raissi 2019)

- Problem
  - Parabolic PDE with d spatial dimensions
  - $$\begin{cases} \frac{\partial u(t,x)}{\partial t} + \mathcal{L}u(t,x) = 0, (t,x) \in [0,T] \times \Omega, \Omega \subset R^d & \text{PDE} \\ u(t=0,x) = u_0(x), x \in \Omega & \text{Initial conditions} \end{cases}$$
  - $u(t,x)$ is the spatio-temporal quantity of interest
  - $\mathcal{L}u(t,x)$ is the differential term of the PDE
    - e.g. Burgers $1D$: $\frac{\partial u(t,x)}{\partial t} = -\mathrm{u}\frac{\partial u(t,x)}{\partial x} + v\frac{\partial^2 u(t,x)}{\partial x^2}$



- Objective
  - Approximate $u(t,x)$ with a NN $f(t,x;\theta), \theta \in R^K$ are the network parameters

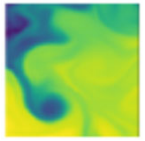
Learning surrogate models for solving PDEs - Known PDE (Lagaris 1998 , Sirignano 2018, Raissi 2019)

- Formulate the problem as minimizing an objective function

Initial condition loss

PDE loss: constraint

- $J(f) = \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_1} + \left\|\frac{\partial f(t,x;\theta)}{\partial t} - \mathcal{L}f(t,x;\theta)\right\|^2_{[0,T]\times\Omega,\nu_2}$

$u_0()$: 

- Learn $f(O,x;\theta)$ by sampling from Ω, the initial condition
- This is a regression problem
- This provides a parametric approximation of target $u(x, t = 0)$

- Constrains $f(t,x;\theta)$ to follow the PDE expression by sampling uniformly from $[0,T] \times \Omega$
- $\frac{\partial f}{\partial t}$ and $\mathcal{L}u(t,x)$ computed by automatic differentiation

- Solved using stochastic gradient descent
- Several extensions, e.g. sampling from data from the PDE loss, etc

Learning surrogate models for solving PDEs - Known PDE (Lagaris 1998 , Sirignano 2018, Raissi 2019)

Algorithm 1: The PINN algorithm.

Construct a neural network (NN) $u(x, t; \theta)$ with θ the set of trainable weights w and biases b, and σ denotes a nonlinear activation function. Specify the measurement data $\{x_i, t_i, u_i\}$ for u and the residual points $\{x_j, t_j\}$ for the PDE. Specify the loss $\mathcal{L}$ in Eq. (3) by summing the weighted losses of the data and PDE. Train the NN to find the best parameters θ^* by minimizing the loss $\mathcal{L}$.

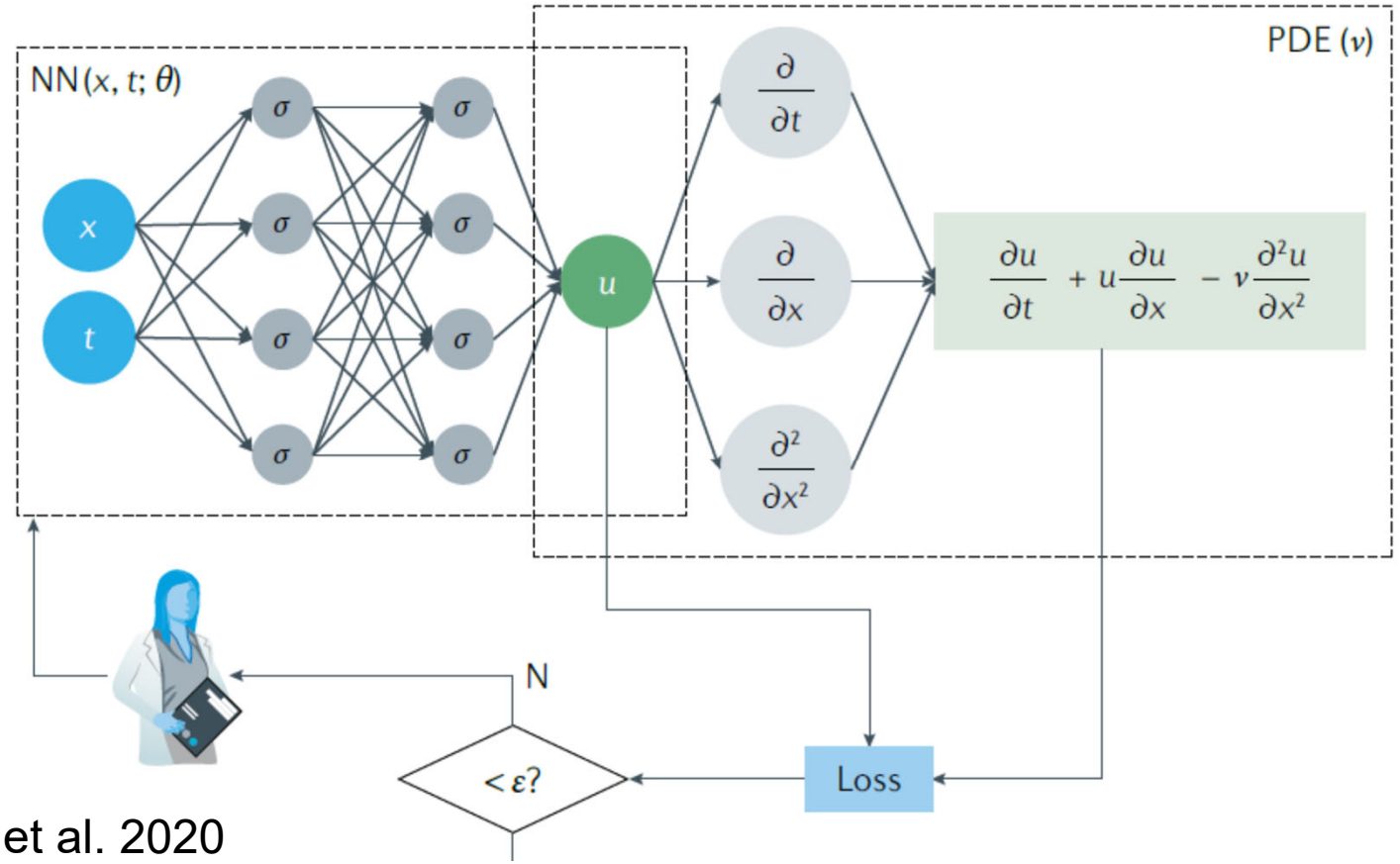


Fig. Karniadakis et al. 2020

NN for solving Partial Differential Equations
Sirignano 2018, Raissi 2019

- Algorithm
  - Iterate
    - Sample (t_n, x_n) from $[0, T] \times \Omega, \nu_1$; sample the initial condition point z_n from Ω, ν_2
    - Calculate the squared error $G(\theta_n, s_n)$ at the sampled points $s_n = \{(t_n, x_n), (\tau_n, y_n), z_n\}$ with:
      - $G(\theta_n, s_n) = (\frac{\partial f(t_n, x_n; \theta_n)}{\partial t} - \mathcal{L} f(t_n, x_n; \theta_n))^2 + \left(f(0, z_n; \theta_n) - u_0(z_n)\right)^2$
    - Take a gradient step
      - $\theta_{n+1} = \theta_n - \epsilon_n \nabla_\theta G(\theta_n, s_n)$
- Comment
  - Still much slower than classical solvers
  - Limited applicability

Learning surrogate models for solving PDEs - Known PDE (Sirignano 2018, Raissi 2019)

- Implicit representation with periodic activation functions (Sitzmann 2020) – example: Wave equation

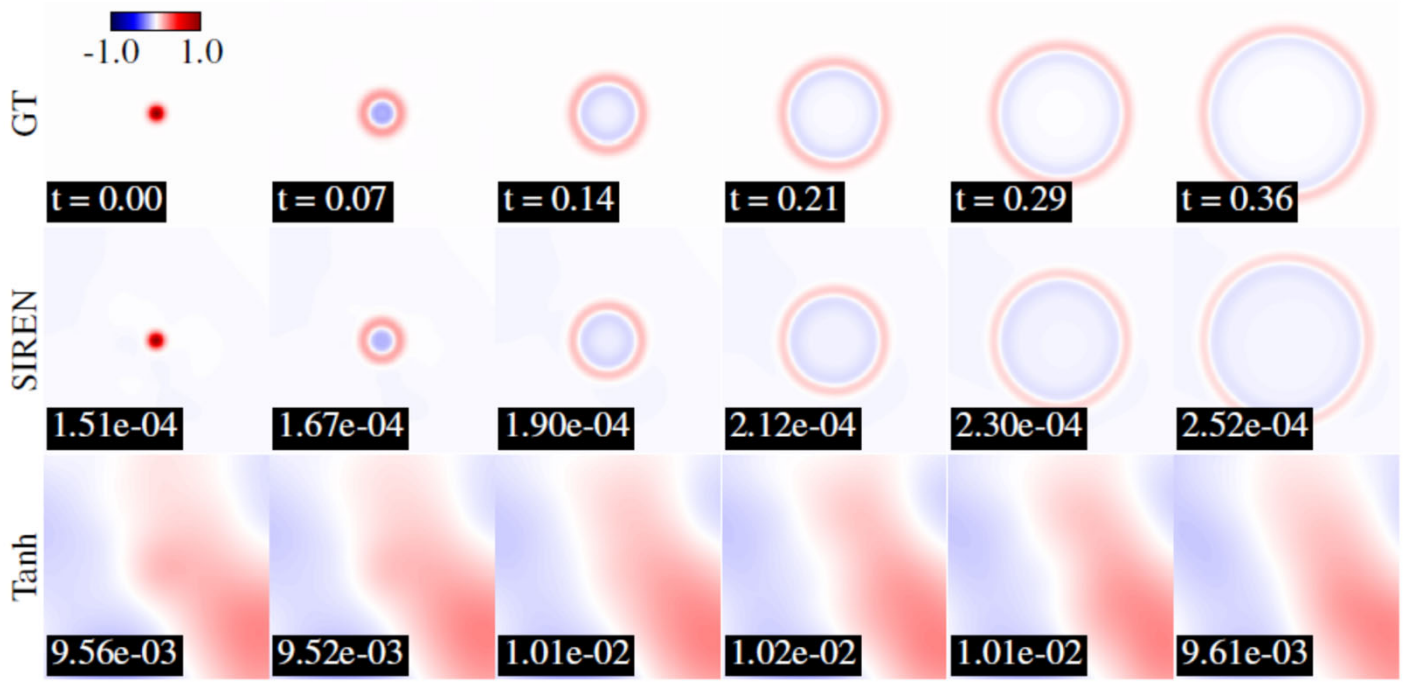


Figure 5: Solving the wave equation initial value problem. For an initial condition corresponding to a Gaussian pulse, SIREN recovers a wavefield that corresponds closely to a ground truth wavefield computed using a principled wave solver [40]. A similar network using tanh activations fails to converge to a good solution. MSE values are shown for each frame, where the time value is indicated in the top row.

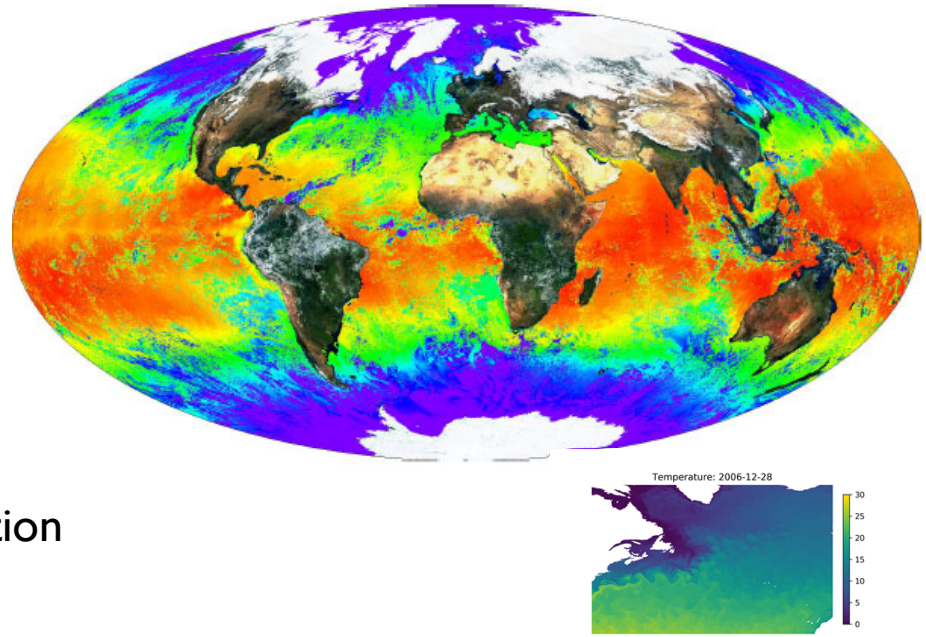
Incorporating physical knowledge in dynamics models

Incorporating physical knowledge in dynamics models

- Question
  - How to build dynamics models that incorporate physical knowledge and ML so that they can be trained from data
- Illustration
  - de Bezenac, E., Pajot, A., & Gallinari, P. (2018). Deep Learning For Physical Processes: Incorporating Prior Scientific Knowledge. ICLR.
    - Learning when one has some knowledge on the analytic form of the solution
  - Yin, Y., Le Guen, V., Dona, J., de Bezenac, E., Ayed, I., Thome, N., & Gallinari, P. (2021). Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting. *ICLR*.
    - Learning using partially known PDE models
    - Leveraging differentiable solvers with ML modules
  - Kashtanova, V., Ayed, I., Arrieula, A., Potse, M., Gallinari, P., & Sermesant, M. (2022). Deep Learning for Model Correction in Cardiac Electrophysiological Imaging. MIDL
    - Application to electrophysiology

Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge (de Bezenac 2018)
Example: Sea Surface Temperature Prediction - SST

- Problem: predicting SST (< 1 meter deep) on Atlantic ocean
- Data: satellite imagery (IR)
- Use cases:
  - Weather prediction, anomaly detection, component of climate models
- Classical approach
  - Data assimilation
    - Differential equations
    - Discretization
      - Finite difference
    - Model
    - Assimilation
      - Coupling with SST data
        - Adjust to initial conditions
      - Forward integration for prediction



Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge (de Bezenac 2018)
Physical model: Advection – Diffusion

- Describes transport of I through **advection** and **diffusion**

$$\frac{\partial I}{\partial t} + (w.\nabla)I = D\nabla^2 I$$

  - I: quantity of interest (Temperature Image)
  - $w = \frac{\Delta x}{\Delta t}$ motion vector, D diffusion coefficient

- There exists a closed form solution
  - $I_{t+\Delta t}(x) = (k * I_t)(x - w(x))$
  - $I_{t+\Delta t}(x)$can be obtained from I_t through a convolution with kernel k (pdf of a Normal distribution: $k(x - w, y) = N(y|x - w, 2D\Delta t)$)

- If we knew the motion vector w and the diffusion coefficient D we could calculate $I_{t+\Delta t}(x)$ from I_t
  - ***w* and *D* unknown**
  - **−> Learn *w* and *D***

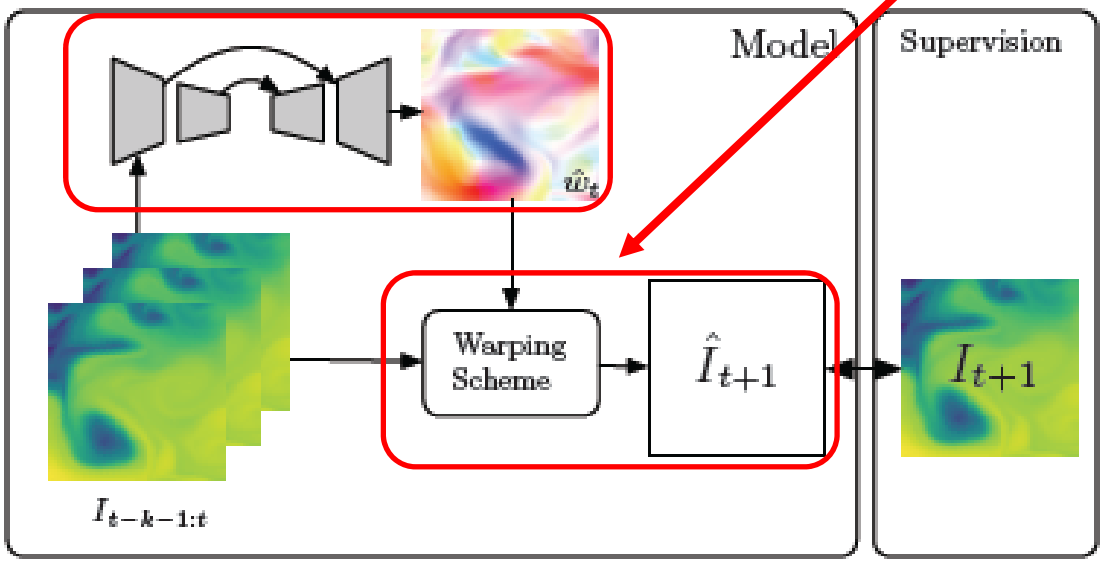
Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge (de Bezenac 2018)

Prediction Model: Objective: predict I_{t+1} from past I_t, I_{t-1}, …

- 2 components:

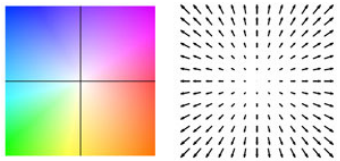
Convolution- Deconvolution NN for estimating motion vector w_t

Warping Scheme Implements discretized A-D solution



Past Images

Target image



Color: orientation
Intensity: flow intensity

- End to End learning using only I_{t+1} supervision
- Stochastic gradient optimization

Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge (de Bezenac 2018)
Experiments

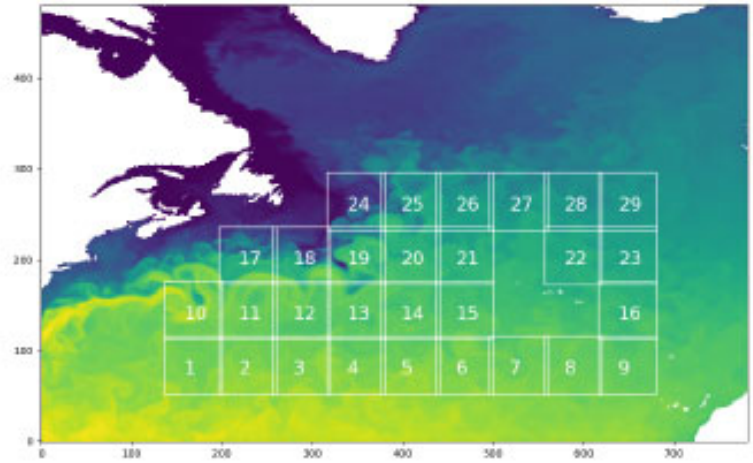


Figure 4: Sub regions extracted for the dataset. Test regions are regions 17 to 20.

- Data
  - Synthetic data from the NEMO simulator (Nucleus for European Modeling of the Ocean)
  - The generated data is based on real SST data (using reanalysis).
  - 3734 daily SST images of 481 x 781 pixels from 2006 to 2017
  - we concentrate of 64 x 64 pixel sub regions
  - we use 2006-2015 for training and validation, and the rest as test

Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge (de Bezenac 2018)
Experiments

- Quantitative results

SOTA Numerical model

Model	Average Score (MSE)	Average Time
Numerical model Béréziat & Herlin (2015)	1.99	4.8 s
ConvLSTM Shi et al. (2015)	5.76	0.018 s
ACNN	15.84	0.54 s
GAN Video Generation (Mathieu et al. (2015))	4.73	0.096 s
Proposed model with regularization	**1.42**	0.040 s
Proposed model without regularization	2.01	0.040 s

Table 1: Average score and average time on test data. Average score is calculated using the *mean square error* metric (MSE), time is in seconds. The regularization coefficients for our model have been set using a validation set with $\lambda_{\text{div}} = 1$, $\lambda_{\text{magn}} = -0.03$ and $\lambda_{\text{grad}} = 0.4$.

SOTA Deep NN models

Proposed model

APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Context
  - Incomplete background knowledge is available, e.g. PDE that partially explains the phenomenon
  - Complement the physical model with a statistical component
  - Provide a principled framework to make model based and data based framework cooperate
    - Identify correctly the physical parameters (inverse problem)
    - The NN component should learn to describe the information that cannot be captured by the physics (direct problem)

APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Illustration: damped pendulum

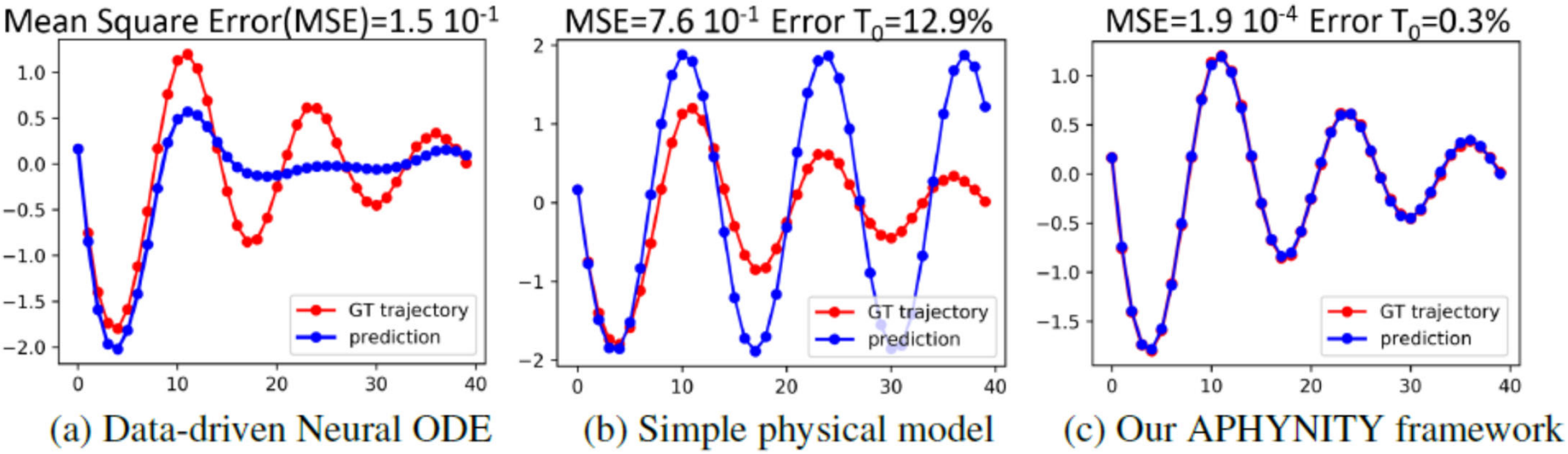


Figure 1: Predicted dynamics for the damped pendulum vs. ground truth (GT) trajectories $\mathrm{d}^2\theta/\mathrm{d}t^2 + \omega_0^2 \sin\theta + \alpha \mathrm{d}\theta/\mathrm{d}t = 0$. We show that in (a) the data-driven approach (Chen et al., 2018) fails to properly learn the dynamics due to the lack of training data, while in (b) an ideal pendulum cannot take friction into account. The proposed APHYNITY shown in (c) augments the over-simplified physical model in (b) with a data-driven component. APHYNITY improves both forecasting (MSE) and parameter identification (Error T_0) compared to (b).

Combining NN and differential solvers (Yin et al. 2021)

- We consider
  - dynamics of the form $\frac{dX_t}{dt} = F(X_t)$
  - two families of functions
    - $\mathcal{F}_p$: parametric set of functions for prior knowledge (physics)
    - $\mathcal{F}_a$: parametric set of functions with high approximation power (NNs)
- We study decompositions of the form
  - $\frac{dX_t}{dt} = F(X_t) = F_p(X_t) + F_a(X_t)$, with $F_p \in \mathcal{F}_p$ and $F_a \in \mathcal{F}_a$
- Problem
  - We want to solve both the forward and inverse problem
  - The decomposition $F_p(X_t) + F_a(X_t)$ is usually not unique
    - Ill posed problem

Combining NN and differential solvers (Yin et al. 2021)

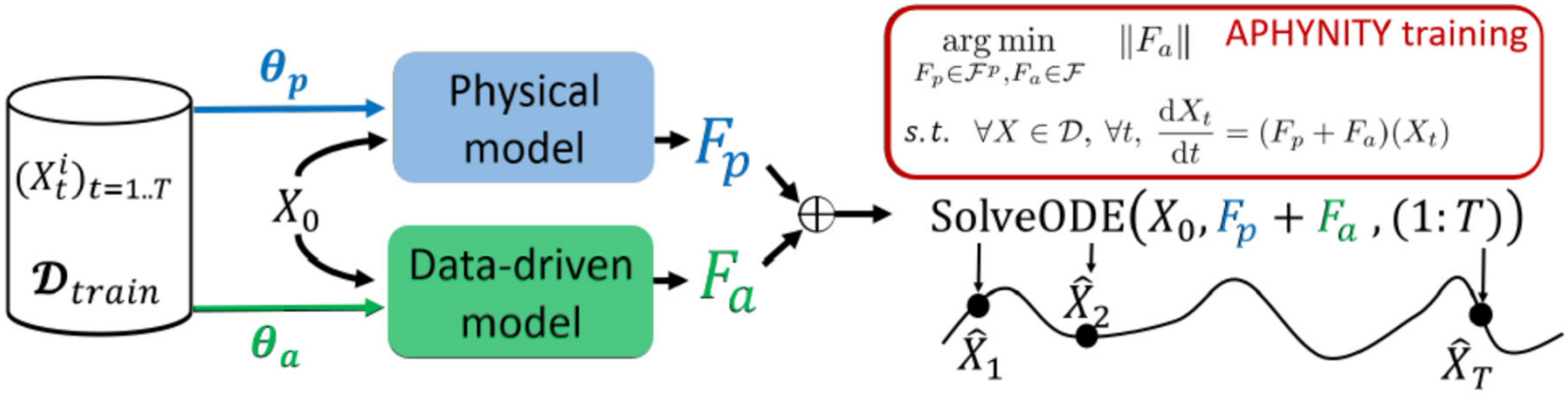
- Turning the learning problem to a well posed problem
- Intuition
  - By hypothesis F_p is a good approximation of reality, but incomplete
  - F_p should explain as much of the dynamics as possible
  - $F_p + F_a$ should explain perfectly the dynamics
  - Learn F_a and F_p so that F_a explains only the residual unexplained by F_p
- Formalization: training objective
  - Given a normed vector space $(\mathcal{F}, \|.\|)$
  - $Min_{F_p \in \mathcal{F}_p, F_a \in \mathcal{F}_a} \|F_a\|$, s.t. $\forall X \in D, \frac{dX_t}{dt} = F_p(X_t) + F_a(X_t)$
- Theoretical insights
  - If $\mathcal{F}_p$ is a proximinal set, there exists a minimizing decomposition.
  - If $\mathcal{F}_p$ is a Chebyshev set, the optimization problem admits a unique minimizer, hence identifiability is guaranteed.

Combining NN and differential solvers (Yin et al. 2021)

- Practical instantiation
  - $\mathcal{F}_p$ is parameterized with a differential equation $F_p^{\theta_p}$
  - $\mathcal{F}_a$ is parameterized by a NN $F_a^{\theta_a}$
  - We fit trajectories instead of derivatives $\frac{dX_t}{dt}$
  - The problem is solved in the trajectory space:
    - Suppose available sequences of size n: $(\mathrm{X}^{\mathrm{i}}_{\mathrm{t_0}}, \ldots, \mathrm{X}^{\mathrm{i}}_{\mathrm{t_n}})$
    - Solve:
      - $Min_{\{\theta_a,\theta_p\}} \left\| F_a^{\theta_a} \right\| + \lambda \sum_i \sum_{\{k=1\}}^{n} ||\hat{X}^i_{t_k} - X^{\mathrm{i}}_{\mathrm{t_k}}||$
      - with prediction $\hat{X}^i_{t_k}$ obtained by a differentiable solver:
        - $\hat{X}^i_{t_k} = \mathrm{X}^{\mathrm{i}}_{\mathrm{t_{k-1}}} + \int_{t_{k-1}}^{t_k} \left(F_p^{\theta_p} + F_a^{\theta_a} \right) \left(\hat{X}^i\ (\tau) \right) d\tau$

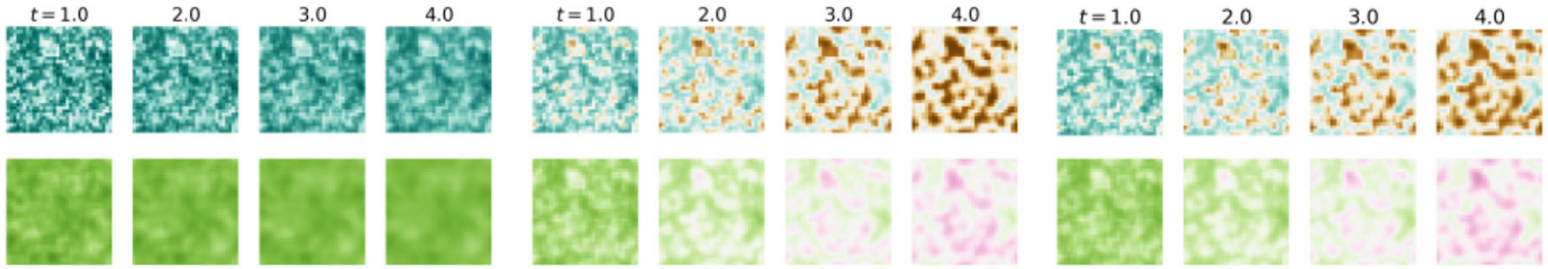
Combining NN and differential solvers (Yin et al. 2021)

- Summary



Combining NN and differential solvers (Yin et al. 2021)

- Example: reaction diffusion equation
- $\frac{\partial u}{\partial t} = a\Delta u + R_u(u, v, k), \frac{\partial v}{\partial t} = \mathrm{b}\Delta v + R_v(u, v)$
  - a,b, diffusion coefficients; R_u, R_v reaction terms; Δ Laplace operator
  - Background physical knowledge
    - Diffusion with coefficients to be estimated
    - Reaction terms are ignored and shall be estimated by F_a



(a) Param PDE (a, b), diffusion-only (b) APHYNITY Param PDE (a, b) (c) Ground truth simulation

Figure 2: Comparison of predictions of two components u (top) and v (bottom) of the reaction-diffusion system. Note that $t = 4$ is largely beyond the dataset horizon ($t = 2.5$).

Combining NN and differential solvers
Cardiac electrophysiology (Kashtanova et al. 2022)

- Objective
  - Modeling the dynamics of cardiac electrical activity
- Models
  - Complex models
    - e.g. Ten Tusscher-Panfilov 2004
    - # hidden variables and parameters, computationally expensive (43 variables)
  - Surrogate low fidelity models
    - e.g. Mitchell Schaeffer 2003 (2 variables)
      - Rapid prototyping, less precise
      - Reaction-diffusion model
  - Objective
    - Learn to simulate real data (here TenTusscher) using a combination of low fidelity model and residual neural network – similar to the APHYNITY framework
      - Idendify from examples the low fidelity parameters and provide good forecasts of the dynamics
      - Limited to the polarization phase

Cardiac electrophysiology (Kashtanova et al. 2022)

- Example: polarization phase
  - Slab of 2D cardiac tissue of 24x24 elements

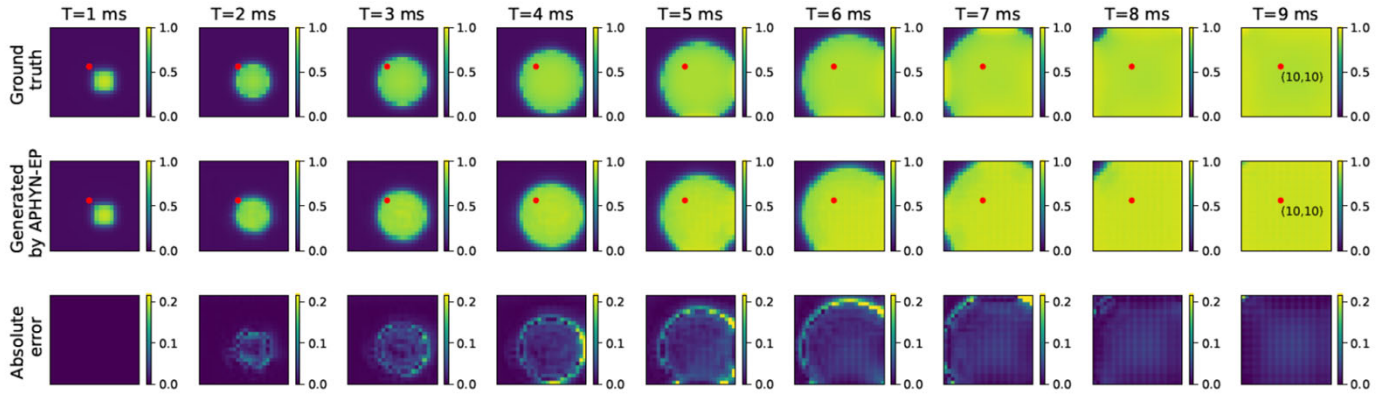


Figure 2: APHYN-EP predicted dynamics for the transmembrane potential diffusion. The figure shows a 9 ms of forecast).

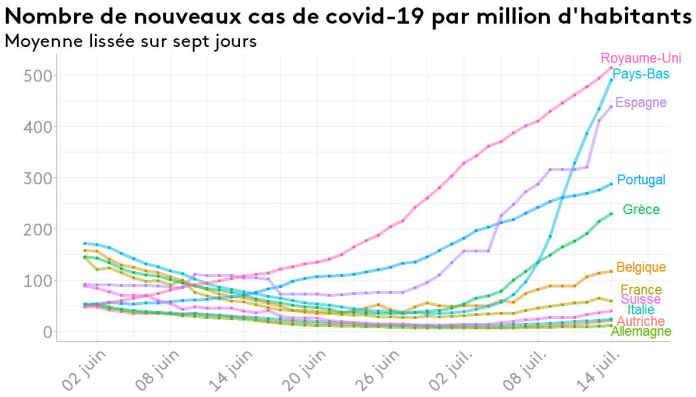
Table 1: Mean-squared error (MSE) of normalised transmembrane potential forecasting per time-step for different forecasting horizons.

	MSE (6 ms)	**MSE (12 ms)**	**MSE (24 ms)**	**MSE (50 ms)**
APHYN-EP	0.0057	0.0037	0.0029	0.002
Physical model only	0.0093	0.0111	0.0096	0.0085
Resnet model only	0.0195	0.0220	0.1593	9.9212

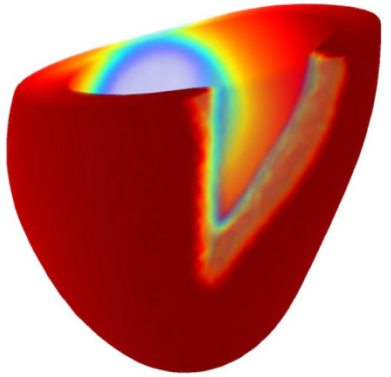
Tackling the generalization problem iwith statistical dynamical models

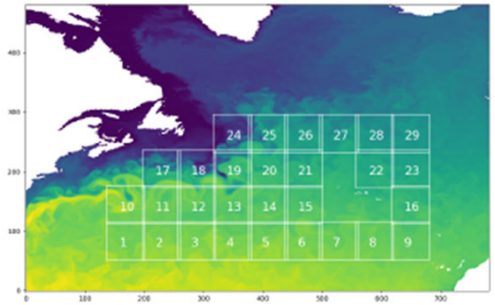
Tackling the generalization problem for dynamical systems Illustrative examples

Modelling epidemics in different countries



Modeling heart electrical diffusion from different patients





Sub regions extracted for the dataset. Test regions are regions 17 to 20.

Predictions of sea surface temperature from satellite data

Tackling the generalization problem for dynamical systems Domain Generalization

- Usual practice in ML
  - Data gathered in different contexts are shufled
- Classical i.i.d. assumption in machine learning - ERM
  - Erroneous in many cases – e.g. dynamical systems
- Domain generalization
  - Assumption: data come from several domains
    - Sharing commonalities e.g. general form of the dynamics (parametric PDE)
    - With specificities, e.g. parameters of the dynamics, initial conditions
  - Objectives
    - Learn on a sample of the domains' distribution, in order to get:
    - Better performance on training domains: interpolation
      - Yin, Y., Ayed, I., de Bézenac, E., Baskiotis, N., & Gallinari, P. (2021). LEADS: Learning Dynamical Systems that Generalize Across Environments. Neurips.
    - Generalize on new domains: extrapolation
      - Kirchmeyer, M., Yin, Y., Dona, J., Baskiotis, N., Rakotomamonjy, A., & Gallinari, P. (2021). Generalizing to New Physical Systems via Context-Informed Dynamics Model. ArXiv:2202.01889v1.
- Idea
  - Leverage our knowledge on the existence of different environments
  - Assumption: there exists a set of environments $E = \{e^i\}$, each governed by a differential equation $\frac{dx_t^e}{dt} = f_e(x_t^e)$

Tackling the generalization problem for dynamical systems LEADS framework (Yin et al. 2021)

- Problem 1: learn to **interpolate** among different environments
  - Intuition: model the evolution as the combination of two terms
    - $f_e = f + g_e$
    - f common across environments, g_e environment specific
- Well-posed optimization problem
  - $\min_{f,\{g_e\}} \sum_{e \in E} \Omega(g_e)$ subject to $\forall x^{e,i}, \forall t, \frac{dx_t^{e,i}}{dt} = (f + g_e)\left(x_t^{e,i}\right)$
  - $\Omega(g_e)$ functional norm controlling the hypothesis space of the increments g_e
- Theoretical outcome
  - Parameters learning
    - If Ω convex, the problem admits at least one decomposition
    - If Ω strictly convex, the problem admits a unique decomposition
  - Generalization bounds
    - The more the environments, the lower the generalization error
    - Sharing parameters (f function), improves generalization in new environments

Tackling the generalization problem for dynamical systems LEADS framework (Yin et al. 2021)

- Lotka Voltera - Prey – predator trajectories – Phase portraits

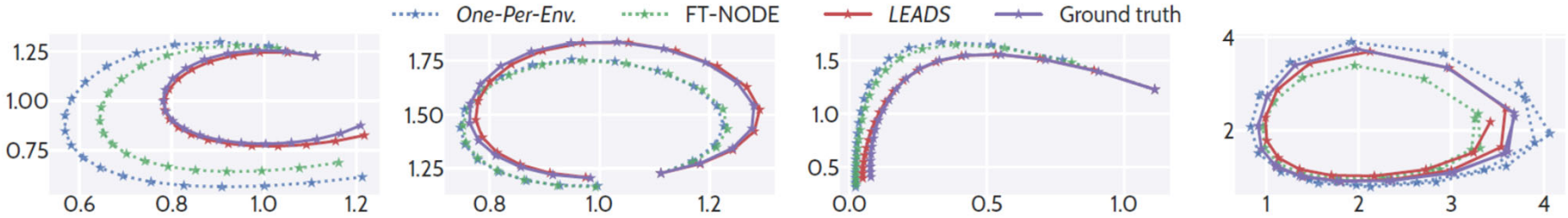


Figure 3: Test prediction obtained with two baselines (*One-Per-Env.* and FT-ODE) and *LEADS* compared with ground truth for LV in phase space for 4 envs., one per figure from left to right.

Gray-Scott (Reaction-Diffusion) and Navier Stokes - final state forecast

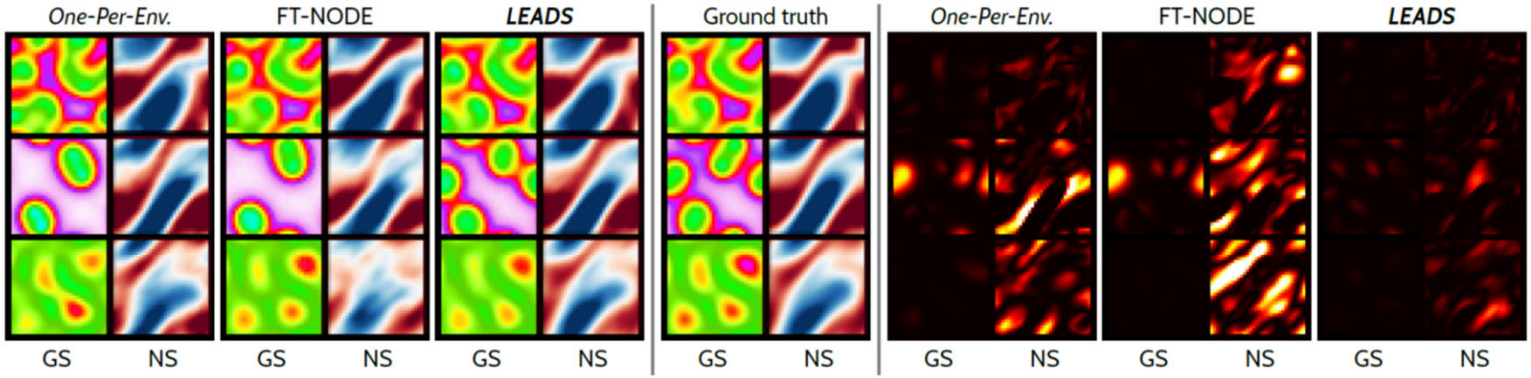
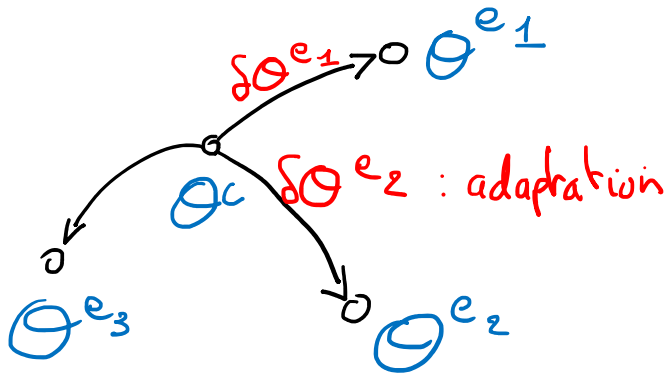


Figure 2: Left: final states for GS and NS predicted by the two best baselines (*One-Per-Env.* and FT-NODE) and *LEADS* compared with ground truth. Different environment are arranged by row (3 in total). Right: the corresponding MSE error maps; darker is smaller. (See Sup. D for full sequences)

Tackling the generalization problem for dynamical systems CODA framework (Kirchmeyer et al. 2022)

- Problem 2 : extrapolation
  - Learn on a sample of the domain distribution
  - Generalize on new domains never seen during training
  - How to: Intuition
    - learn to condition the learned function f_{θ^e} on the environment e
    - So that it could adapt fast and with a few samples to a new environment
    - Adaptation rule :$\theta^e = \theta^c + \delta\theta^e$
      - θ^e shared parameters, $\delta\theta^e$ environment specific parameters infered for each new environment



Tackling the generalization problem for dynamical systems CODA framework (Kirchmeyer et al. 2022)

- Dynamical function for environment e
  - $f_{\theta^e}, \theta^e = \theta^c + \delta\theta^e$
- Training objective
  - $\min_{\theta^c, \delta\theta^e; e \in E} \sum_{e \in E} \|\delta\theta^e\|^2$ s.t. $\forall x^e \in D^e, \forall t, \frac{dx^e(t)}{dt} = f_{\theta^c + \delta\theta^e}\left(x^e(t)\right)$
    - θ^c learned over a training set sampled from a family of environments E
    - $\delta\theta^e$ conditionned on environment e
    - Locality constraint $\min_e \|\delta\theta^e\|^2$: θ^e should lie in the neighborhood of θ^c: $\delta\theta^e$ lies on a low dimensional manifold -> fast adaptation to environments

- Lotka-Voltera ODE
- Loss landscape for 3 environments
- Centered on the shared θ^c
- Local min θ^e indicated by arrow

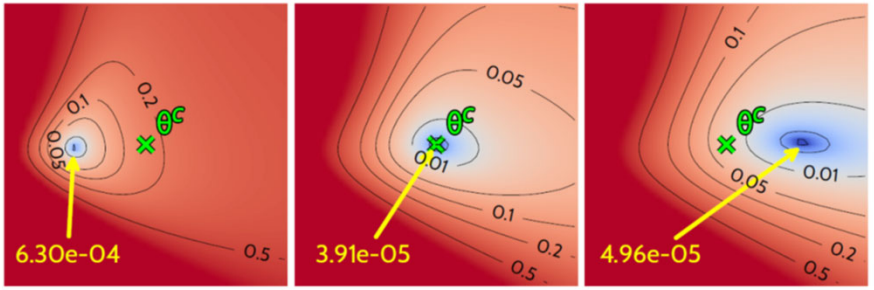
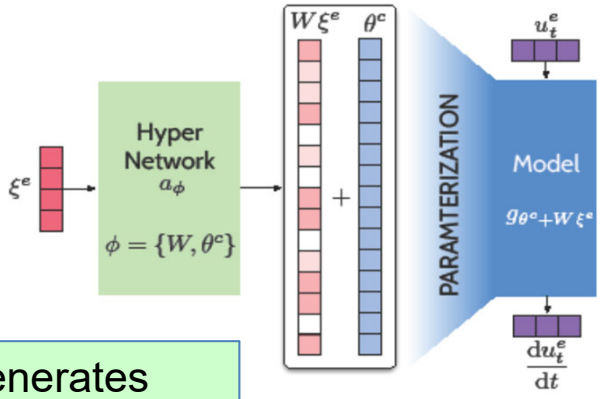


Figure 1. CoDA's loss landscape centered in θ^c, marked with ×, for 3 environments on the *Lotka-Volterra* ODE. Loss values are projected onto subspace $\mathcal{W}$, with $d_\xi = 2$. $\forall e$, $\rightarrow$ points to the local optimum $\theta^{e\star}$ with loss value reported in yellow.

Tackling the generalization problem for dynamical systems CODA framework (Kirchmeyer et al. 2022)

- Conditioning to an environment
  - $\theta^e = \theta^c + \delta\theta^e$
  - Environment specific parameters: Implementation via a hypernetwork
    - $\delta\theta^e = W\xi^e$ with W weight matrix and ξ^e learned code
    - Code ξ^e is infered from a few observations for each new environment
  - θ^c and W are shared parameters learned on the training set
- Inference: for a new environment, learn code ξ^e from a few observations and infer θ^e



(1) Environment encoding ξ^e (learned)

Neural network predictor: forecast next frames via numerical integration

(2) Hypernetwork: generates the parameters of the neural network predictor, conditionned on the environment ξ^e

Tackling the generalization problem for dynamical systems
CODA framework (Kirchmeyer et al. 2022)

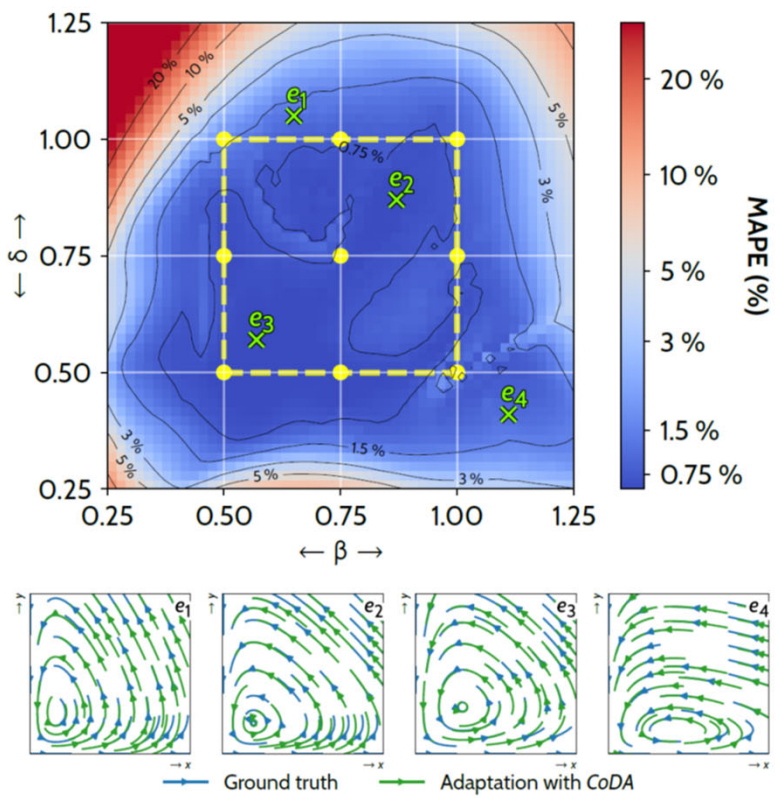


Figure 2. Adaptation results with CoDA-ℓ_1 on LV. Parameters (β, δ) are sampled in $[0.25, 1.25]^2$ on a 51×51 uniform grid, leading to 2601 adaptation environments $\mathcal{E}_{\text{ad}}$. • are training environments $\mathcal{E}_{\text{tr}}$. We report MAPE (↓) across $\mathcal{E}_{\text{ad}}$ (Top). On the bottom, we choose four of them (×, e_1–e_4), to show the ground-truth (blue) and predicted (green) phase space portraits. x, y are respectively the quantity of prey and predator in the system in Eq. (15).

Lotka-Volterra (LV, Lotka, 1925) The system describes the interaction between a prey-predator pair in an ecosystem, formalized into the following ODE:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \alpha x - \beta xy \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \delta xy - \gamma y \end{aligned} \tag{15}$$

where x, y are respectively the quantity of the prey and the predator, $\alpha, \beta, \delta, \gamma$ define how two species interact.

- Four parameters, two fixed (α, γ) and two (β, δ) change accross environments
- Training on 9 environments (yellow)
- Top: Evaluation on 2600 new environments
- Bottom: phase portraits for 4 new environments e_1 to e_4
  - **Blue trajectories: ground truth**
  - **Green trajectories: predicted**

Conclusion

- ++
  - Physics-based deep learning is a rapidly growing area of AI for science
  - Started only a few years ago
  - Mobilizes several communities from different engineering domains
  - Lots of promises in many domains
- --
  - Important gap between academic developments/ POCs and practical implementations
  - Further developments require strong R&D efforts multidisciplinary consortiums and industrial involvement

References used in the presentation

- General references
  - Stevens R., V. Taylor, J. Nichols, A.B. Maccabe, K. Yelick, D. Brown - **AI for Science: Report on the Department of Energy (DOE) Town Halls on Artificial Intelligence (AI) for Science.** Report from the U.S. Department of Energy, February 2, 2020. https://doi.org/10.2172/1604756 (2020)
- Deep Learning for Dynamical Processes
  - A blog on Neural ODE: https://www.depthfirstlearning.com/2019/NeuralODEs
  - Ayed I., de Bezenac E., Pajot A., Brajard J. , Gallinari P. , (2019), Learning Dynamical Systems from Partial Observations, arXiv:1902.11136
  - Chen, R.T.Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. 2018. Neural Ordinary Differential Equations. *NIPS*.
  - de Bezenac, E., Pajot, A., & Gallinari, P. (2018). Deep Learning For Physical Processes: Incorporating Prior Scientific Knowledge. In *ICLR, also in Journal of Statistical Mechanics: Theory and Experiment, 2019.*
  - Dona, J., Dechelle, M., Gallinari, P., & Levy, M. (2022). Constrained Physical-Statistical Models for Dynamical System Identification and Prediction. *ICLR*.
  - Haber, E. and Ruthotto, L. 2018. Stable architectures for deep neural networks. *Inverse Problems*. 34, 1 (2018).
  - Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422–440.
  - Kashtanova, V., Ayed, I., Arrieula, A., Potse, M., Gallinari, P., & Sermesant, M. (2022). Deep Learning for Model Correction in Cardiac Electrophysiological Imaging. *MIDL*, 1–11.
  - Kirchmeyer, M., Yin, Y., Dona, J., Baskiotis, N., Rakotomamonjy, A. and Gallinari, P. 2021. Generalizing to New Physical Systems via Context-Informed Dynamics Model. *arXiv:2202.01889v1* (2021).
  - Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000. https://doi.org/10.1109/72.712178
  - Long, Z., Lu, Y., & Dong, B. (2019). PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, *399*, 108925. https://doi.org/10.1016/j.jcp.2019.108925
  - Raissi, M. 2018. Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations. *Journal of Machine Learning Research 19*, 1–24.
  - Rudy, S. H., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2017). Data-driven discovery of partial differential equations. *SCIENCE ADVANCES*, *3* no. *4*(April).
  - Sirignano, J. and Spiliopoulos, K. 2018. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*. 375, Dms 1550918 (2018), 1339–1364.
  - Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., Wetzstein, G., & University, S. (2020). Implicit Neural Representations with Periodic Activation Functions. *Neurips*.
  - Yin, Y., Le Guen, V., Dona, J., de Bezenac, E., Ayed, I., Thome, N., & Gallinari, P. (2021). Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting. *ICLR*.
  - Yin, Y., Ayed, I., de Bézenac, E., Baskiotis, N., & Gallinari, P. (2021). LEADS: Learning Dynamical Systems that Generalize Across Environments. *Neurips*.