Advanced Deep Learning

Sorbonne Université – Masters DAC et M2A. Gallinari, patrick.gallinari@lip6.fr, http://www-connex.lip6.fr/~gallinar/

Sylvain Lamprier, Nicolas Baskiotis, nom.prenom@lip6.fr

Année 2020-2021

Generative models

Generative Adversarial Networks

Variational Auto-Encoders

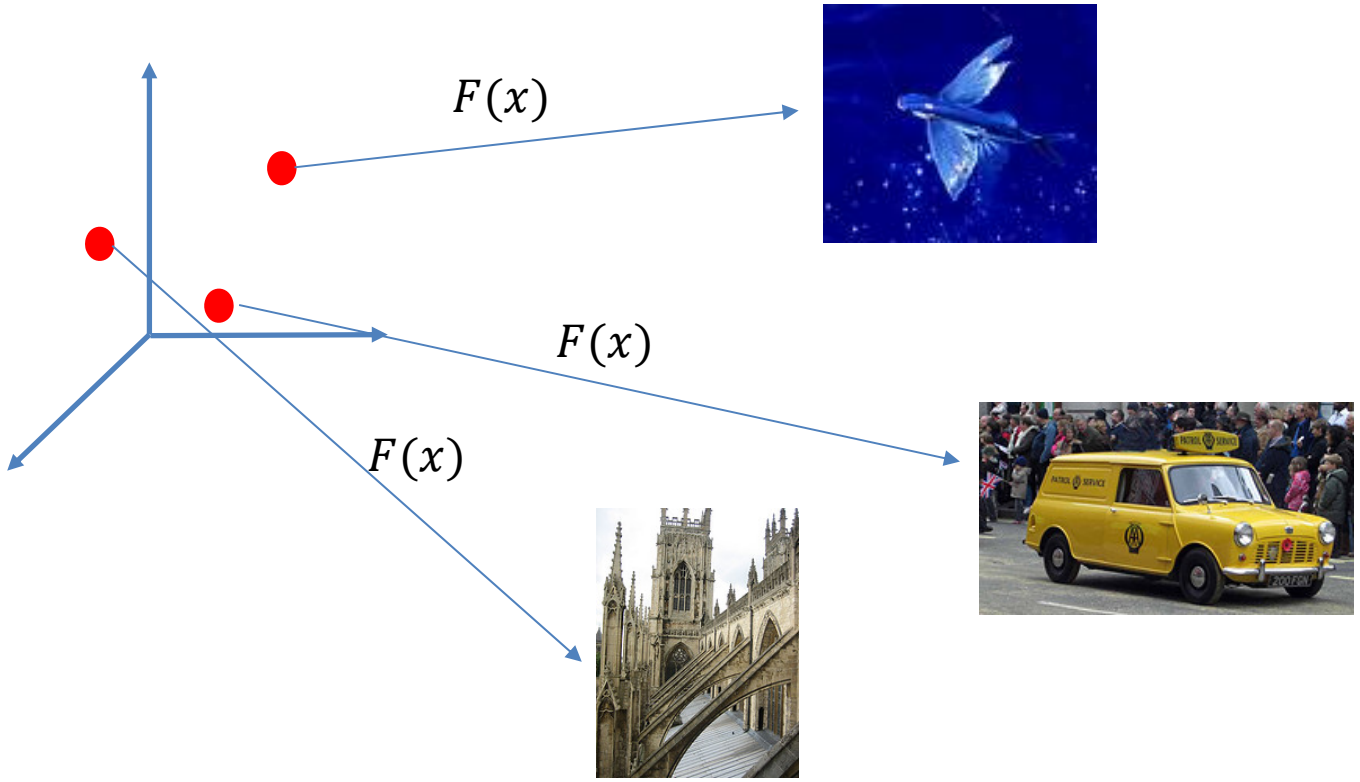
Flow models

Generative models

- Objective
  - Learn a distribution model from data samples
  - Generative means: one can generate data from the learned model
  - For complex distributions, there is no analytical form and even approximate methods (e.g. MCMC) might fail, besides requiring large amounts of data + convergence problems
  - Deep generative models recently attacked this problem
    - Objective: handle large dimensions and complex distribution
    - Here we examine three examples that propose alternatives to classical maximum likelihood approaches
      - Generative Adversial Networks
        - Learn to generate data according to some learned distribution, no density required
        - Alternative to maximum likelihood for generating samples and learning intractable distributions
      - Variational autoencoders
        - Compute an approximate likelihood of the examples
        - Hyp: there exists a density function explaining the data
      - Flow models
        - Compute the exact likelihood

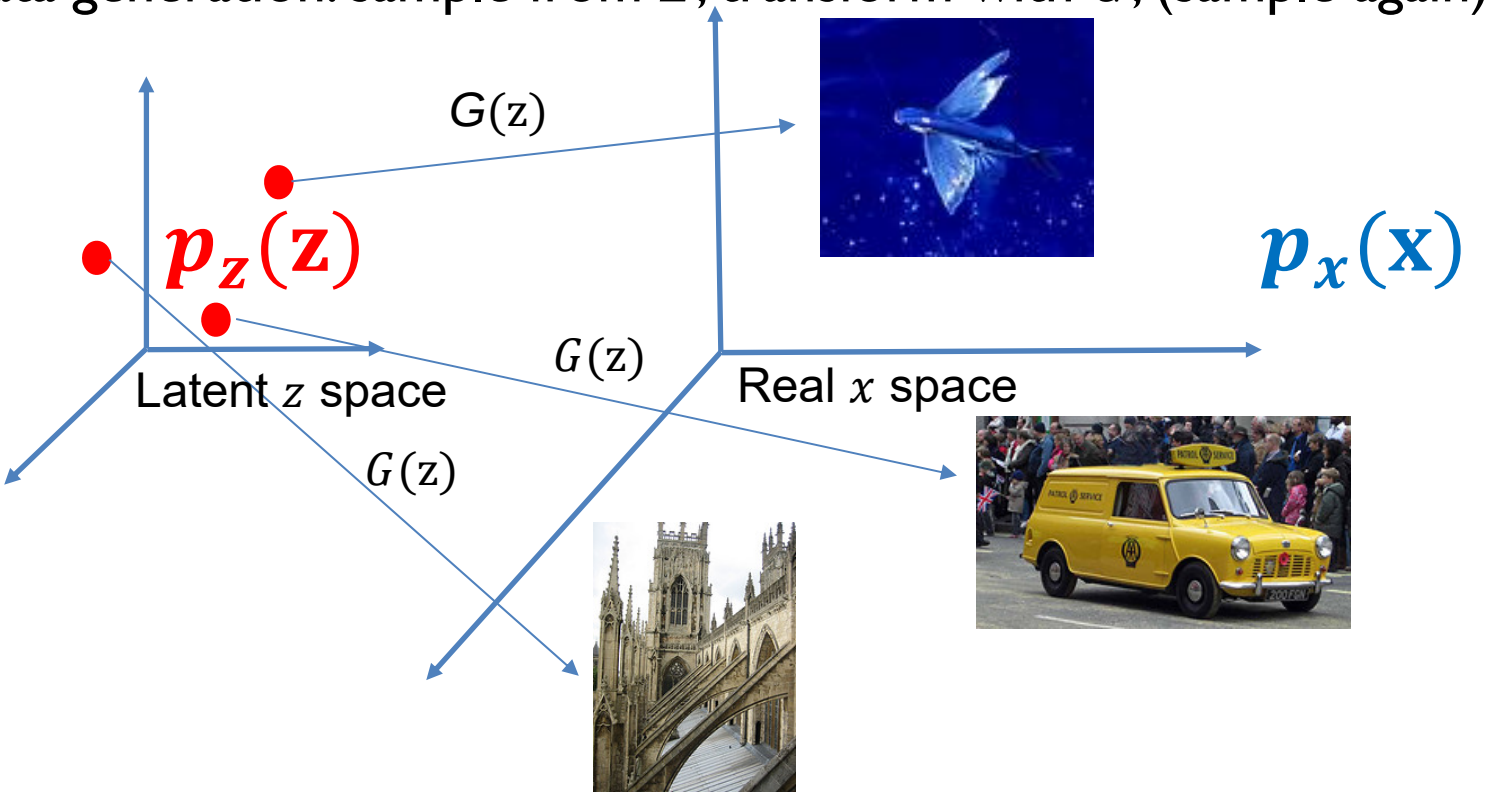
Generative models intuition

- Provided a sufficiently powerful model $F(x)$
  - It should be possible to learn complex mappings such as:



Generative models intuition

- Given a probability distribution on the latent space $p_z(\mathrm{z})$, G defines a probability distribution on the observation space
- Data generation: sample from Z, transform with G, (sample again)



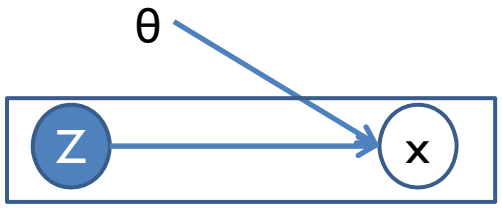
Generative Adversial Networks - GANs

Ian J. Goodfellow, et al. 2014

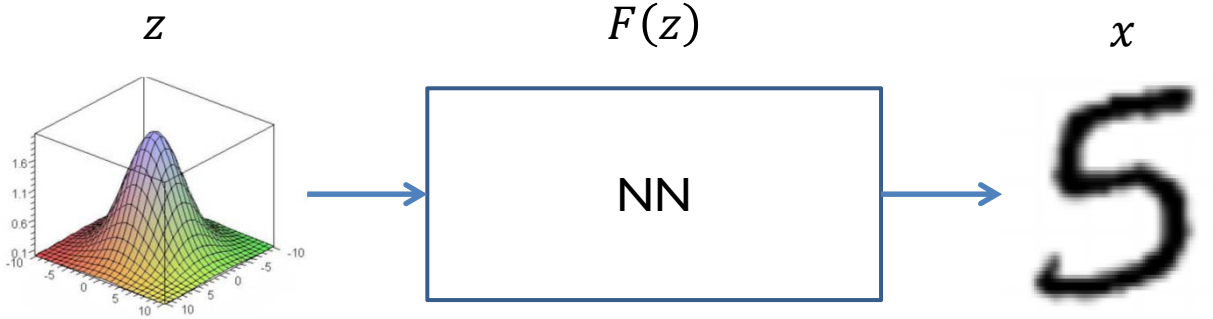
GANs have been hype for some years - O(1000) GAN papers on Arxiv

GANs

- Generative latent variable model



- Given a simple distribution $p(z)$, e.g $z \sim \mathcal{N}(0, \mathrm{I})$, use a NN to learn a possibly complex mapping $p_\theta(x|z) = F(z)$

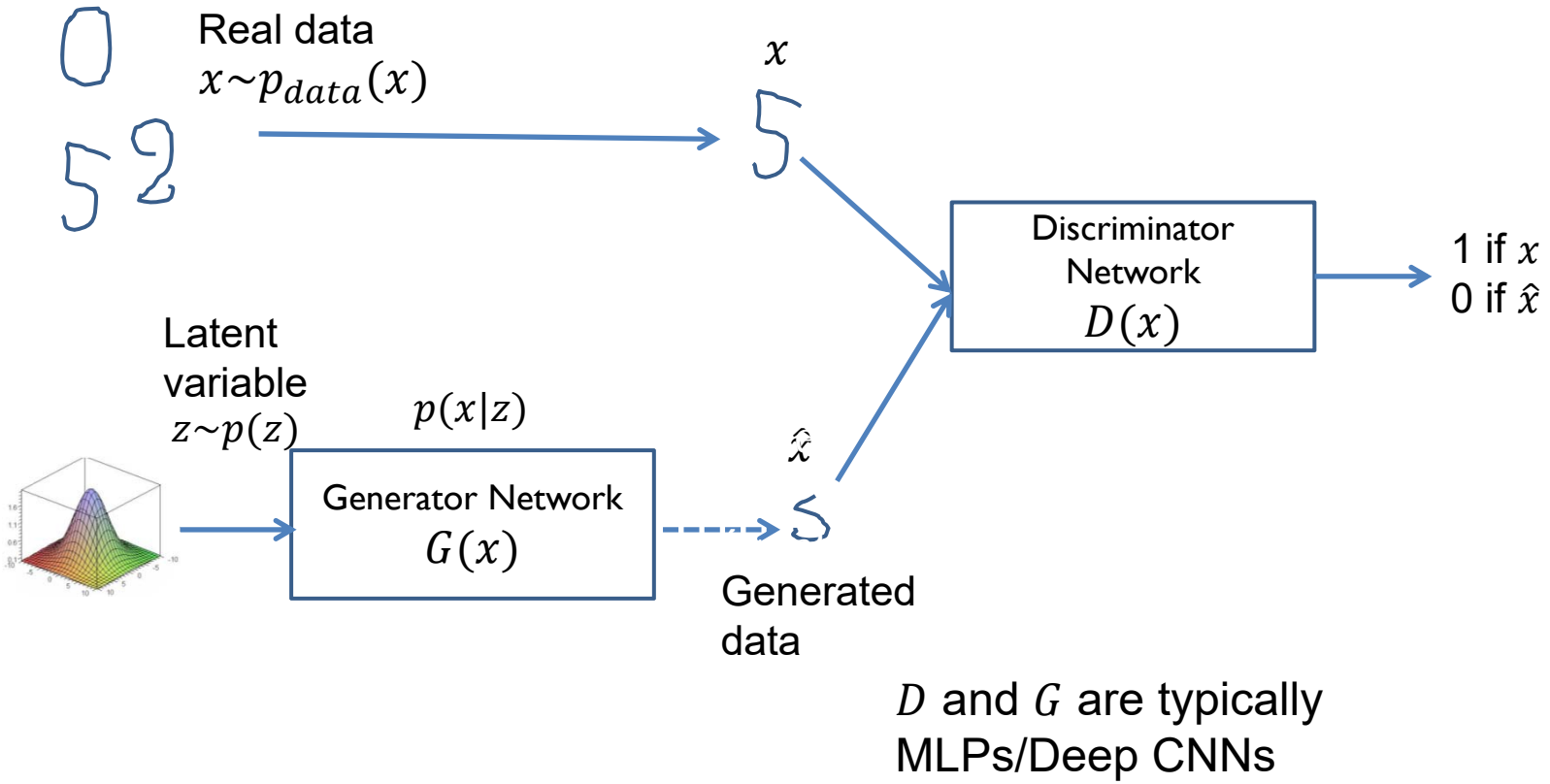


GANs (Goodfellow, et al. 2014)

- 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 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
  - Hope: the Discriminator will force the Generator to be clever and learn the data distribution
- Applications
  - Data generation, Semi-supervised learning, super resolution, domain adaptation, …and many others
- Note
  - No hypothesis on the existence of a density function
    - i.e. no max likelihood approximation
  - Maximum likelihood for density functions is replaced here by a discriminator that trains a generative network

GANs - Intuition

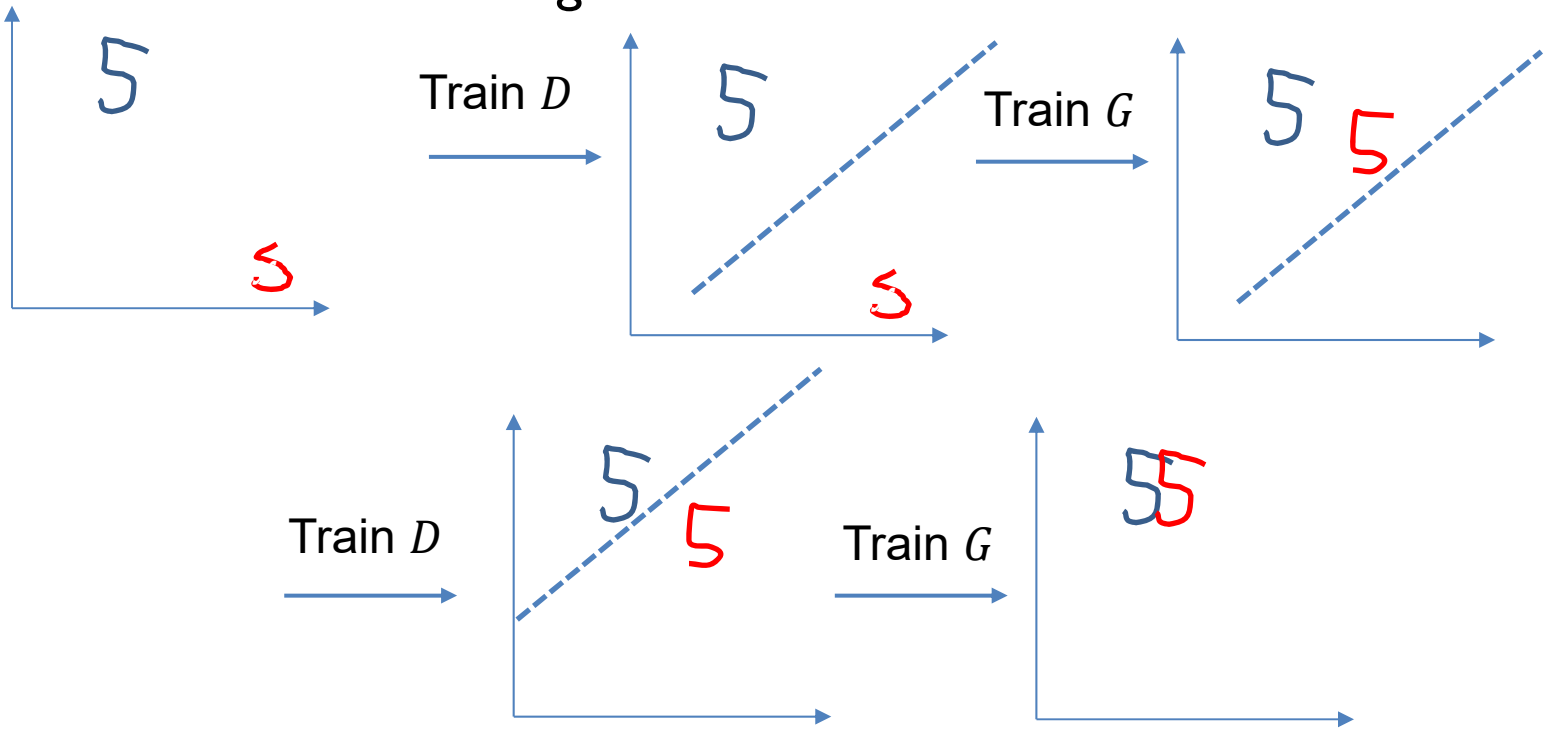
- Discriminator is presented alternatively with true (x) and fake $(\hat{x})$data



D and G are typically MLPs/Deep CNNs

GAN – Intuition - Training

- Algorithm alternates between optimizing D (separate true and generated data) and G (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 - Loss function

- $x \sim p_{data}(x)$ distribution over data x
- $z \sim p(z)$ prior on z, usually a simple distribution (e.g. Normal dist.)
- Loss
  - $\min_G \max_D L(D,G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p(z)}[\log\left(1 - D\left(G(z)\right)\right)]$
  - $G(z)$ mapping from the latent (z) space to the data (x) space
  - $D(x)$ probability that x comes from the data rather than from the generator G
  - If G is fixed, $V(D,G)$ is a classical binary cross entropy for D, distinguishing real and fake examples
- Training
  - Alternates
    - k steps for optimizing (maximizing) D
    - k' steps for optimizing (minimizing) G
  - Note
    - Maximize $\log\left(D\left(G(z)\right)\right)$ instead of minimizing $\log\left(1 - D\left(G(z)\right)\right)$ provides stronger gradients and is used in practice, i.e. $\log\left(1 - D\left(G(z)\right)\right)$ is replaced by $-\log\left(D\left(G(z)\right)\right)$

GAN 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
  - $L(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{x\sim p_g(x)}[\log(1 - D(x))]$
    - with $p_{data}(x)$ the true data distribution and $p_g(x)$ the distribution of generated data
      - Note that this is equivalent to the $L(D,G)$ definition on the slide before
- If G and D have sufficient capacity
  - Computing
    - $G^* = argmin_G \max_D L(D,G)$
  - Is equivalent to compute
    - $G^* = argmin_G D_{JS}(p_{data}, p_g)$ with $D_{JS}(,)$ Jenson-Shannon dissimilarity measure between distributions
  - If the optimum is reached
    - $D(x) = \frac{1}{2}$ for all x
- In practice
  - D is not fully optimized and this ideal setting is never reached

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
  - $\mathrm{D}^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g\,(x)}$
    - 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
    - $Max_D\,L(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{\mathrm{x}\sim p_g(x)}[\log(1-D(x))]$ is then obtained for:
      - $\mathrm{D}^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g\,(x)}$

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

- Let $C(G) = \max_D L(G, D) = L(G, D^*)$
- It si easily verfied that:
  - $C(G) = -\log 4 + KL\left(p_{data}(x); \frac{p_{data}(x)+p_g(x)}{2}\right) + KL\left(p_g(x); \frac{p_{data}(x)+p_g(x)}{2}\right)$
  - Since $KL(p; q) \geq 0$ and $KL(p; q) = 0$ iff $p = q$
    - $C(G)$ is minimum for $p_g = p_{data}$ 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_g and p_{data}
- 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
  - $\frac{p_{data}(x)}{p_g(x)} = \frac{p(x|y=1)}{p(x|y=0)} = k \frac{p(y=1|x)}{p(y=0|x)} = k \frac{D^*(x)}{1-D^*(x)}$ 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
  - Tapez une équation ici.Least Square GANs
  - Wasserstein GANs

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

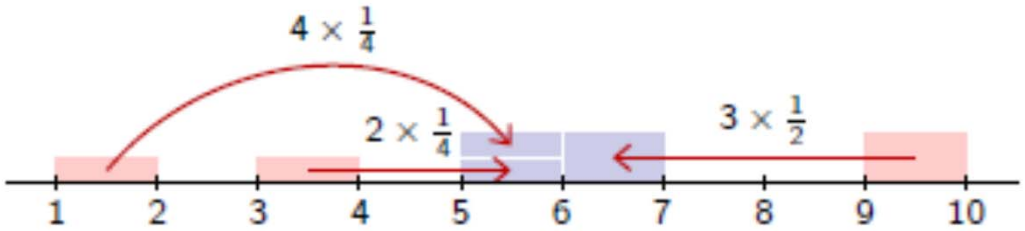
- If a generated sample is well classified but far from the real data distribution, there is no reason for the generator to be updated
- LS-GAN replaces the cross entropy loss with a LS loss which penalizes generated examples by moving them close to the real data distribution.
- The objective becomes
  - $L(D) = E_{x\sim p_{data}(x)}[(D(x) - b)^2] + E_{z\sim p_z(z)}[(D(G(z)) - a)^2]$
  - $L(G) = E_{z\sim p_z(z)}\left[\left(D\left(G(z)\right) - c\right)^2\right]$
  - 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
  - $W(p_{data}, p_g) = \inf_{\gamma \in \Pi(p_{data}, p_g)} E_{(x,x')\sim\gamma}[\| x - x' \|]$
    - where $\Pi(p_{data}, p_g)$ is the set of distributions over X^2 whose marginals are respectively $p_{data}(x)$ and $p_g(x)$, $\| x - x' \|$ is the Euclidean norm.
  - Intuitively,
    - $W(,)$ is the minimum amount of work required to transform $p_{data}(x)$ to $p_g(x)$ – see next slide
    - it makes sense to learn a generator G minimizing this metric
      - $G^* = argmin_G W(p_{data}, p_g)$

Wasserstein GANs (Arjovski et al. 2017)

- Earth Mover distance illustration
  - 2 distributions (pink (μ) and blue (μ'))
  - An elementary rectangle weights ¼
  - $W(\mu, \mu')$ is the Wasserstein distance between the two distributions, illustrated here as the Erath mover Distance to transport pink on blue.



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

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

Fig. from F. Fleuret 2018

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

- $p_{data}(x) = \int_y \gamma(x, y)dy$ is the amount of mass to move from x,
  $p_g(y) = \int_y \gamma(x, y)dx$ 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 $\inf_{\gamma \in \Pi(p_{data}, p_g)} E_{(x,x')\sim\gamma}[\| x - x' \|]$

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

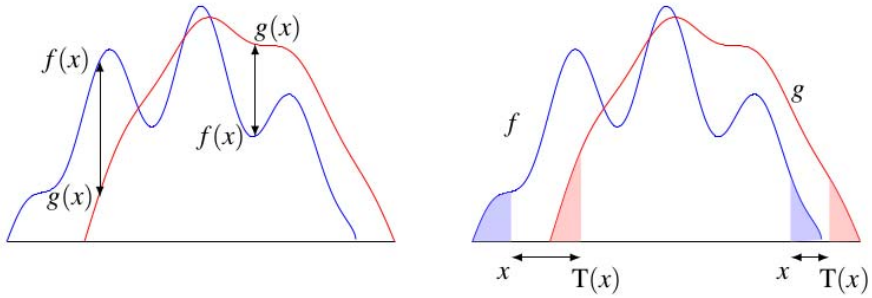


Fig. Santambrogio, 2015

- Left: standard ways to compute distance between functions (point distance)
- Right: Optimal Transport way
  - Seek the best map T which transports the blue function 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:
  - $W(p_{data}, p_g) = \sup_{\|f\|_L} E_{x\sim p_{data}}|f(x)| - E_{x\sim p_g}|f(x)|$
  - Where $f: X \to R$ is 1-Lipchitz, i.e. $|f(x) - f(y)| < 1 \parallel x - y \parallel, \forall\, x, y \in X$
- Implementation
  - Using this result, one can look for a generator G and a critic f_w:
    - $G^* = argmin_G W(p_{data}, p_g)$
    - $G^* = argmin_G \sup_{\|D\|_L} E_{x\sim p_{data}}|f_w(x)| - E_{x\sim p_g}|f_w(x)|$
    - $G^* = argmin_G \sup_{\|D\|_L} E_{x\sim p_{data}}|f_w(x)| - E_{z\sim p_z}|f_w(G(z))|$
    - 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

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

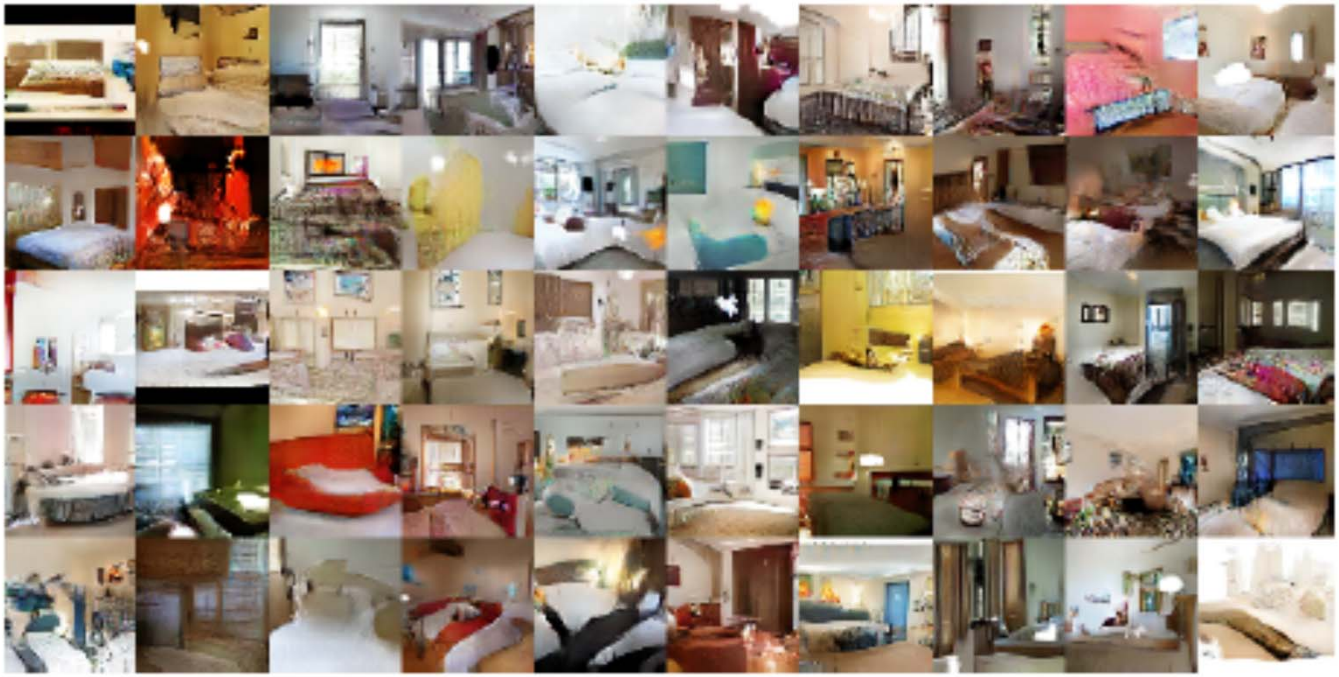
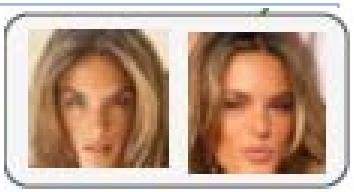


Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

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

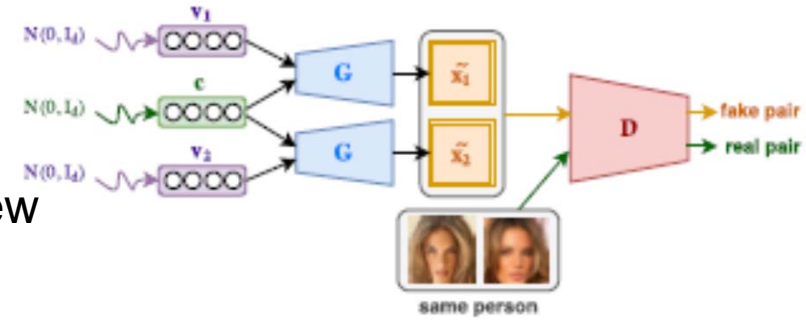
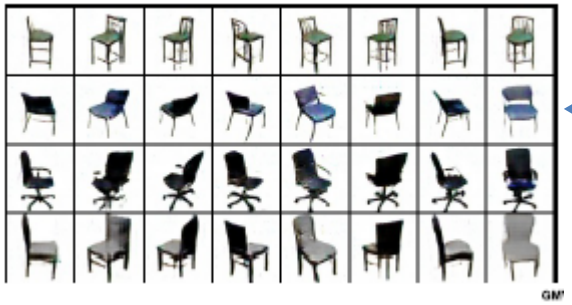


Fig. Chen 2018

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

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x|y)] + E_{z\sim p(z)}[\log\Big(1 - D\big(G(z|y)\big)\Big)]$$

Conditional GANs (Mirza 2014)

- $\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x|y)] + E_{z\sim p(z)}[\log\left(1 - D\left(G(z|y)\right)\right)]$

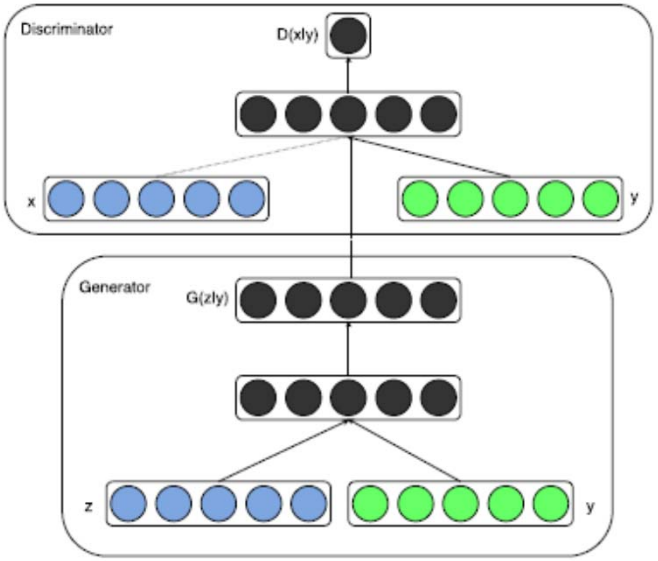


Fig. (Mirza 2014)

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

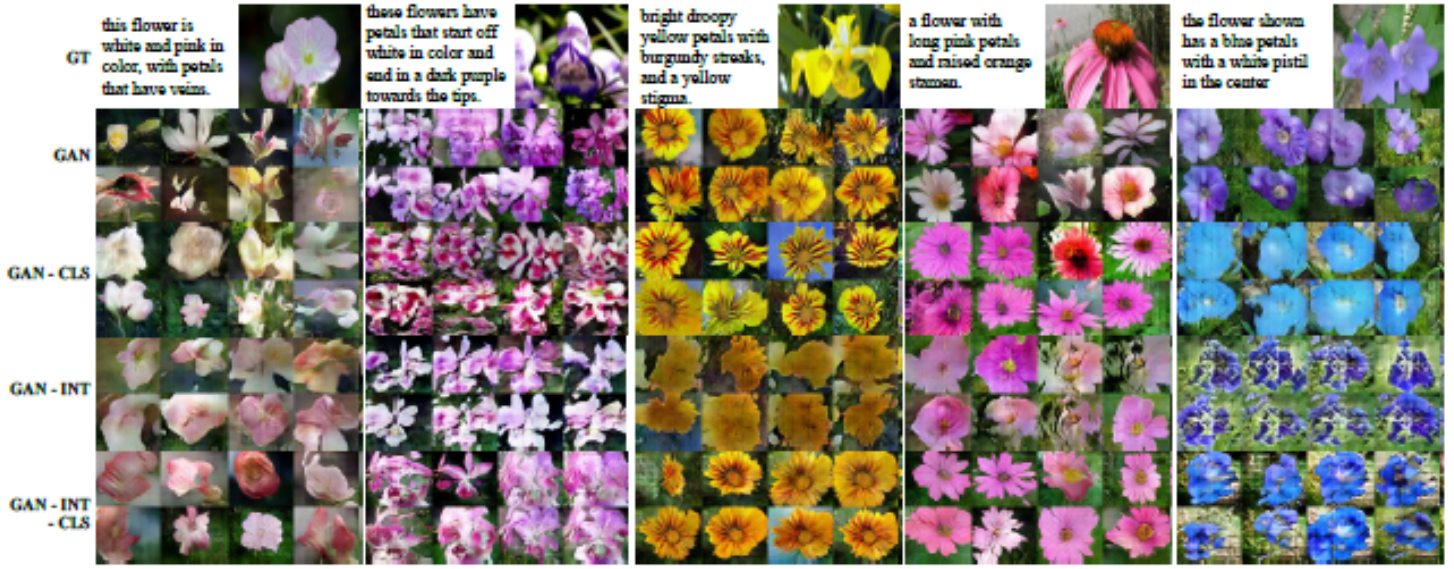
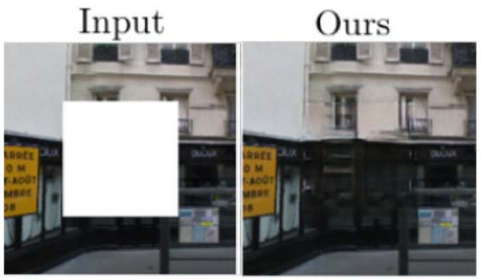


Figure 4. Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved.

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
- $V(D,G) =$

$$\min_G \max_D V(D,G) = E_{\substack{x\sim p_{data}(x) \\ y\sim p(y)}}[logD(x,y)] + E_{\substack{z\sim p(z) \\ y\sim p(y)}}[\log\big(1 - D(G(z,y),y)\big)]$$

  - 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
  - $C_{L^1}(G) = E_{x,y,z}\|x - G(y,z)\|_1$
    - Where $\|.\|_1$ is the L^1 norm
- Final loss
  - $\min_G(\max_D V(D,G) + \lambda C_{L^1}(G))$

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)

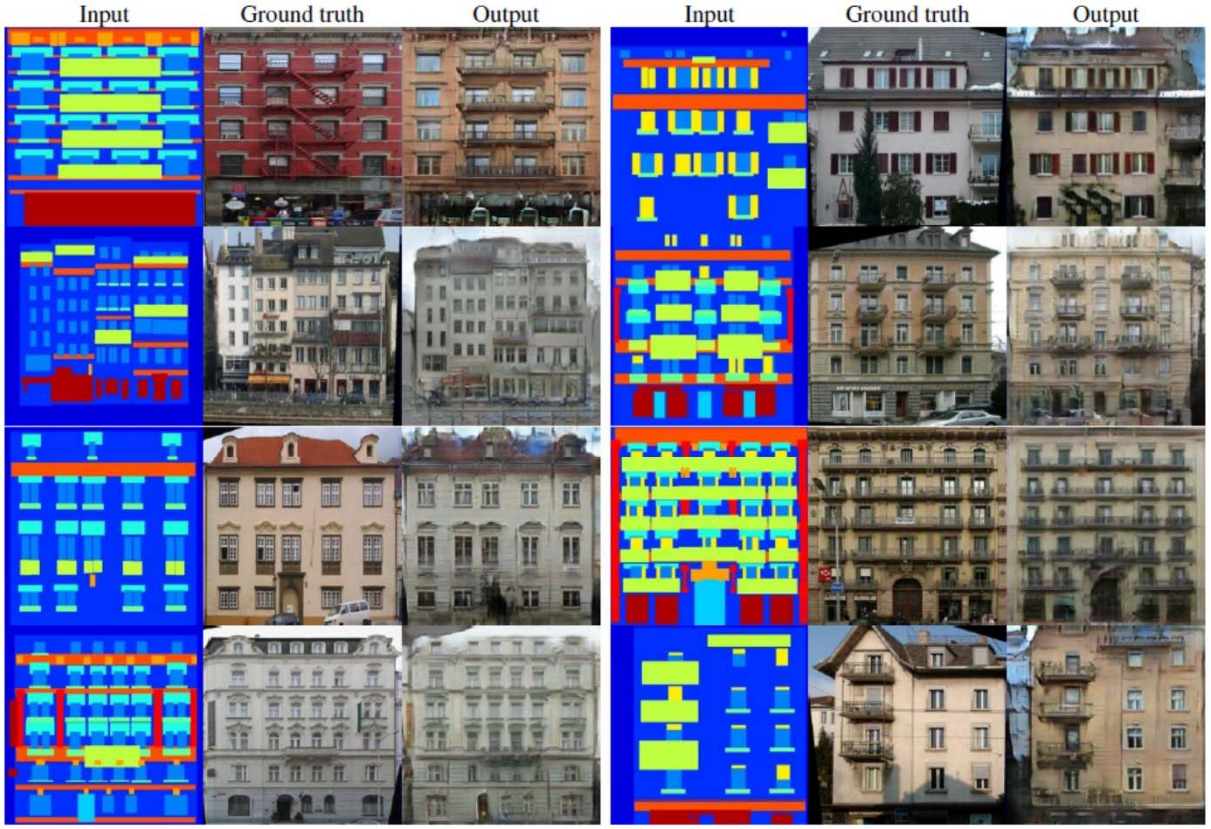


Figure 13: Example results of our method on facades labels→photo, compared to ground truth.

Fig. (Isola 2016)

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

- Failure examples

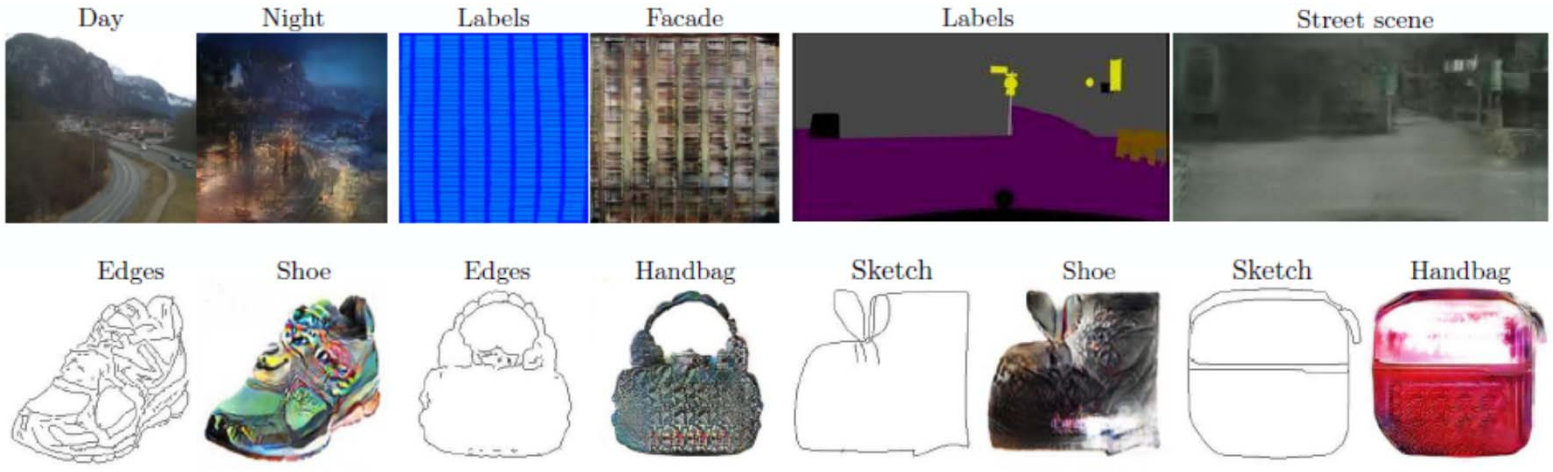


Figure 20: Example failure cases. Each pair of images shows input on the left and output on the right. These examples are selected as some of the worst results on our tasks. Common failures include artifacts in regions where the input image is sparse, and difficulty in handling unusual inputs. Please 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 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)

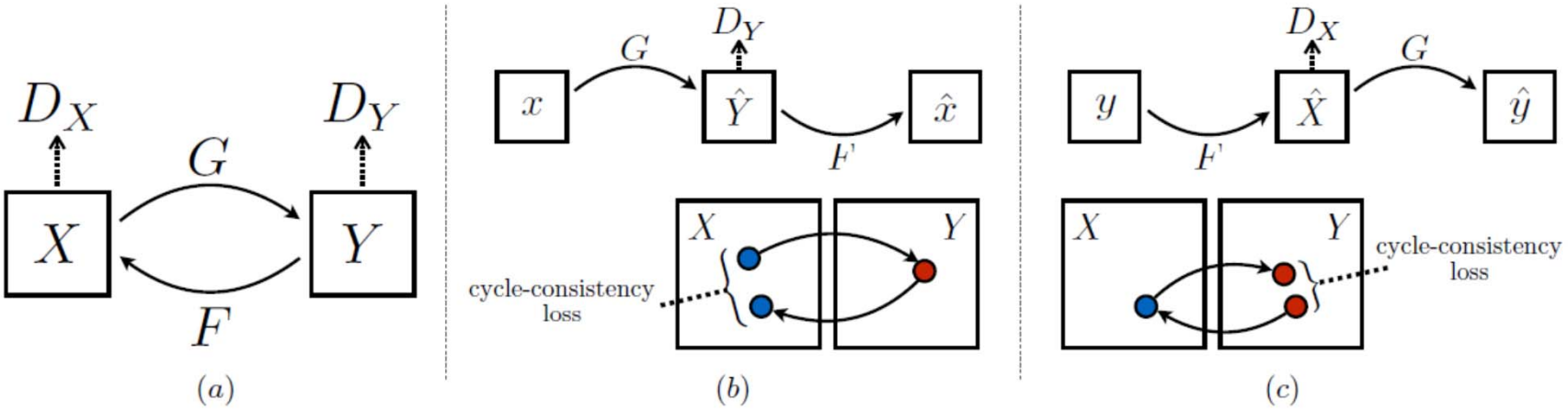


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, 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
  - $C_{cycle}(F, G) = E_{p_{data}(x)}[\|F(G(x)) - x\|_1] + E_{p_{data}(y)}[\|G(F(y)) - y\|_1]$
  - $C = L(G, D_Y) + L(F, D_X) + C_{cycle}(F, G)$
  - Note: they replaced the usual $V(G, D_Y)$ and $V(F, D_X)$ term by a mean square error term, e.g.:
    - $L(G, D_Y) = E_{pdata(y)}[(D_Y(y) - 1)^2] + E_{data(x)}[D_Y(G(x))]$

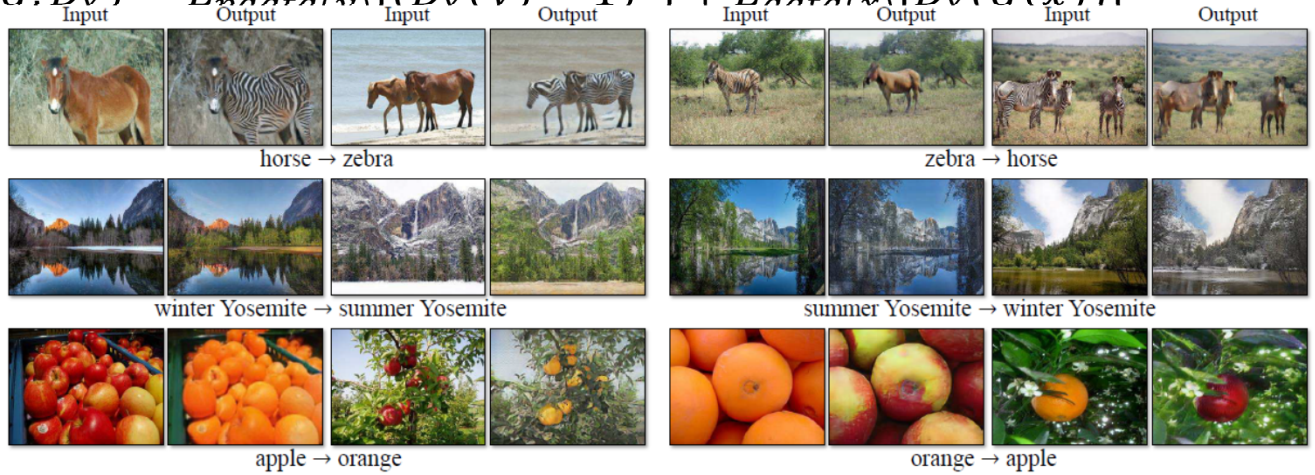


Figure 7: Results on several translation problems. These images are relatively successful results – please see our website for more comprehensive results.

Cycle GANs (Zhu 2017)

- Examples

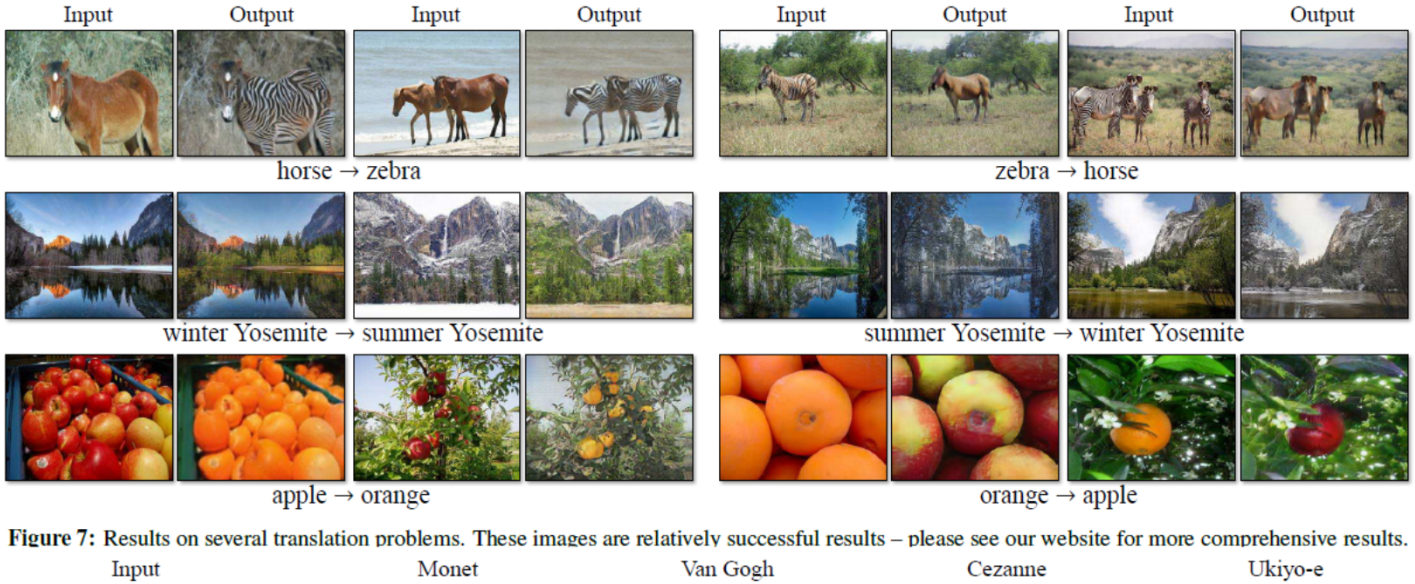
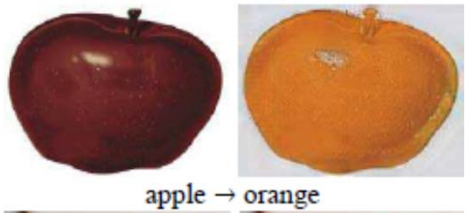


Figure 7: Results on several translation problems. These images are relatively successful results – please see our website for more comprehensive results.

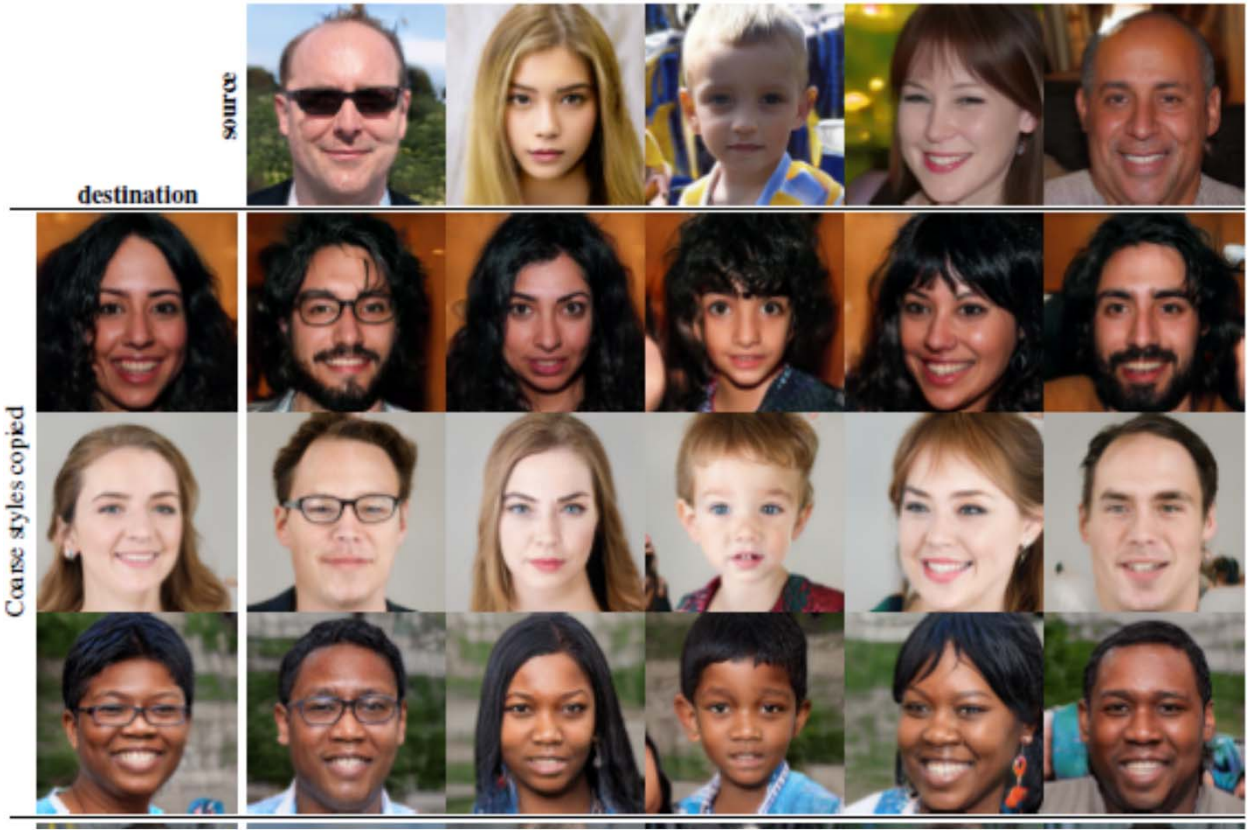
- Failures

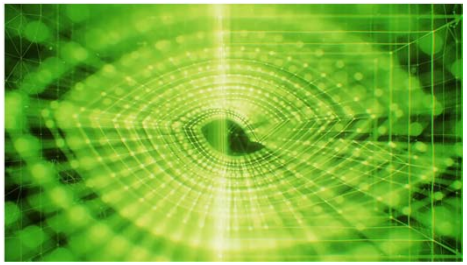
Fig (Zhu 2017)



(Karras et al. 2019) – Style GAN

- (Karras et al. 2019) – Style GAN





GANs

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

Variational Auto-Encoders

After Kingma D., Welling M., Auto-Encoding Variational Bayes, ICLR 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 $p = q$
    - $D_{KL}(p(y)||q(y)) = -E_{p(y)}\left[log\frac{q(y)}{p(y)}\right] \geq -\log E_{p(y)}\left[\frac{q(y)}{p(y)}\right] = 0$
      - 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

- Let us suppose available
  - A set of observed and latent variables, respectively $X = \{\mathrm{x}^1, \ldots, \mathrm{x}^\mathrm{N}\}$ and $Z = \{z^1, \ldots, z^N\}$
  - A central task is the evaluation of the posterior distribution $p(\mathrm{Z}|\mathrm{X})$ and the evaluation of expectations w.rt. this distribution
    - e.g. E.M (Expectation Maximization) requires expectation of the complete data likelihood w.rt. $p(z|x)$
  - For many problems, evaluating this posterior and then the expectations is intractable
  - In this case one ressorts to approximation schemes
    - Variational methods are deterministic approximations based on analytical approximations of $p(\mathrm{Z}|\mathrm{X})$

Preliminaries – Variational methods

- Objective
  - Approximate $p(Z|X)$ as well as $\ln p(X)$
- Variational methods
  - We introduce a distribution $q(Z)$ defined over the latent variables
  - Instead of maximizing the **evidence** $log\, p(X)$, we will maximize an **ELBO** (Evidence Lower BOund) defined by a functional of $q(Z)$
  - We will make hypotheses on $q(Z)$ allowing for tractable computations
  - $q(Z)$ will then provide an approximation to $p(Z|X)$

Preliminaries – Variational methods
Evidence Lower Bound

- Log likelihood **for** X decomposes as (proof next slide):
  - $\ln p\,(X) = D_{KL}(q(Z)||p(Z|X)) + V_L(q)$
  - With
    - $V_L(q) = E_{q(Z)}[log\left(\frac{p(X,Z)}{q(Z)}\right)]$
    - $D_{KL}(q(Z)||p(Z|X)) = -E_{q(Z)}[log\left(\frac{p(Z|X)}{q(Z)}\right)]$
    - Rq: $V_L(q)$ is a functional of q, i.e. it takes a function as input (here q) and returns a value
- How does it work
  - $D_{KL}(q||p) \geq 0$, then $V_L(q)$ is a lower bound of $log\,p\,(X)$
  - Maximizing $V_L(q)$ w.r.t. q is equivalent to minimizing $D_{KL}(q||p)$ w.r.t. q
  - The max is obtained when $q(Z) = p(Z|X)$
  - We are interested in cases when $p(Z|X)$ is intractable
  - We consider a restricted family of distributions q that comprises only tractable dstributions

Preliminaries – Variational methods
Evidence Lower Bound

- Proof: $\ln p\,(X) = D_{KL}(q(Z)||p(Z|X)) + V_L(q)$
  - $V_L(q) = \int q(Z) \log\left(\frac{p(X,Z)}{q(Z)}\right) dZ$
  - $V_L(q) = \int q(Z) \log\left(\frac{p(Z|X)p(X)}{q(Z)}\right) dZ$
  - $V_L(q) = \int q(Z)(\log\left(\frac{p(Z|X)}{q(Z)}\right) + \log(p(X)))\, dZ$
  - $V_L(q) = -\mathrm{D_{KL}}(\mathrm{q(Z)||p(Z|X)}) + \log(\mathrm{p(X)})$

- Then $log\, p\,(X) = D_{KL}(q(Z)||p(Z|X)) + V_L(q)$

Preliminaries – Variational methods
Example: Mean Field Approximation

- MF approximation is the most common variational approximation
- We partition the elements of Z in disjoint groups Z_i , $i = 1 \ldots M$
  - e.g. a partition of the features of individual patterns $z = (z_1, \ldots, z_M)$
- Hypothesis: q factorizes as $q(Z) = \prod_{i=1}^{M} q_i(Z_i)$
- Among all distributions of this form, we seek the distribution for which $V_L(q)$ is the largest
- For that we optimize $V_L(q)$ w.r.t. each factor $q_i(Z_i)$ in turn

Preliminaries – Variational methods
Example: Mean Field Approximation (followed)

- Optimization of $V_L(q)$ w.r.t. factor $q_j(Z_j)$
  - Denote $q_j = q_j(Z_j)$
- Substituting $q(Z) = \prod_{i=1}^{M} q_i(Z_i)$ in $V_L(q)$ we get:
  - $V_L(q) = \int \prod_i \; q_i(\log p(X,Z) - \sum_i \; \log q_i) dZ$
  - $V_L(q) = \int q_j(\int \log p(X,Z) \prod_{i \neq j} q_i dZ_i) dZ_j - \int q_j \log q_j \, dZ_j + const$
  - $V_L(q) = \int q_j E_{i \neq j}[\log p(X,Z)] dZ_j - \int q_j \log q_j \, dZ_j + const$
    - With $E_{i \neq j}[\log p(X,Z)] = \int \log p(X,Z) \prod_{i \neq j} q_i dZ_i$

- Maximizing $V_L(q)$ w.r.t. $q_j(Z_j)$ is obtained when
  - $\log q_j = E_{i \neq j}[\log p(X,Z)] + const, \qquad j = 1 \ldots M$

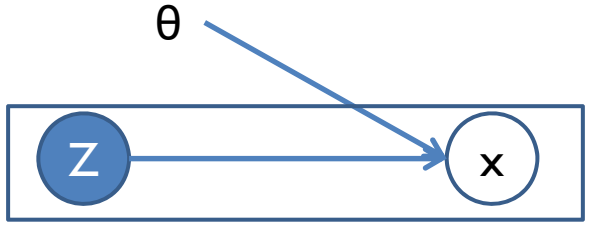
  - This equation provides the basis for the application of variational methods
  - $const$ is a normalizing constant w.r.t. Z_j

Preliminaries – Variational methods
Example: Mean Field Approximation (followed)

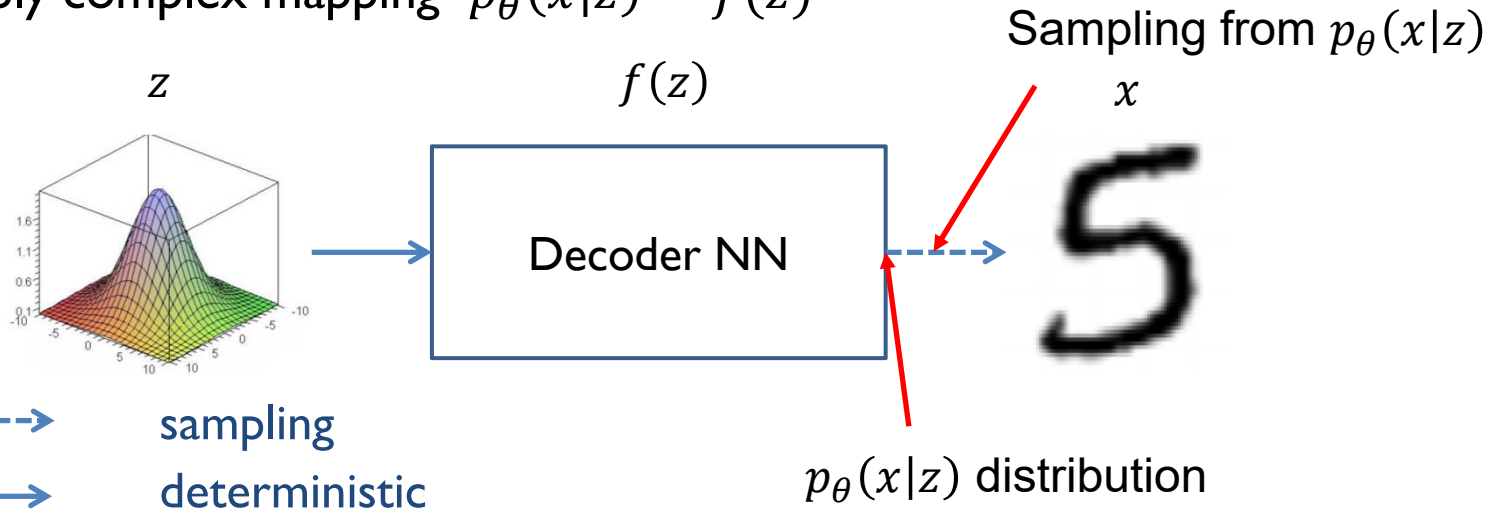
- The set of equations
  - $\log q_j = E_{i \neq j}[\log p(X, Z)] + const, \qquad j = 1 \ldots M \qquad (*)$
  - Represents consistency conditions for the maximization of the ELBO
  - They do not however represent explicit solutions, because q_j depends on expectations of other variables
  - Most solutions require iterative optimizations
    - Initialize the $q_j(\mathrm{Z_j})$ appropriately (e.g. priors)
    - Cycle through the factors using (*) optimizing 1 factor at a time, keeping the others fixed

VAEs - Intuition

- Generative latent variable model



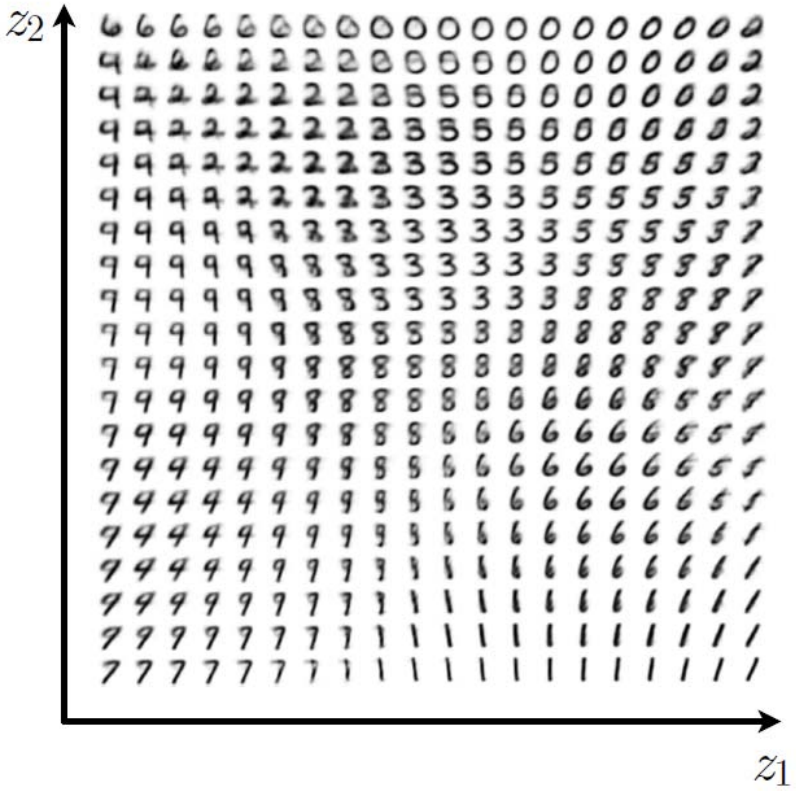
- Sample from $z \sim p(z)$ and generate $p_\theta(x|z)$
- Given a simple distribution $p(z)$, e.g $z \sim \mathcal{N}(0, \mathrm{I})$, use a NN to learn a possibly complex mapping $p_\theta(x|z) = f(z)$



VAEs - Intuition

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

MNIST:



Frey Face dataset:

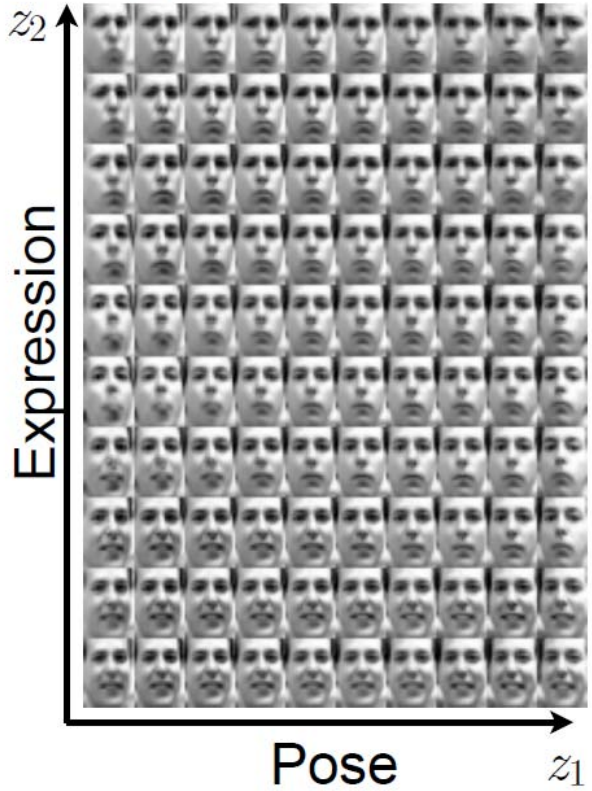
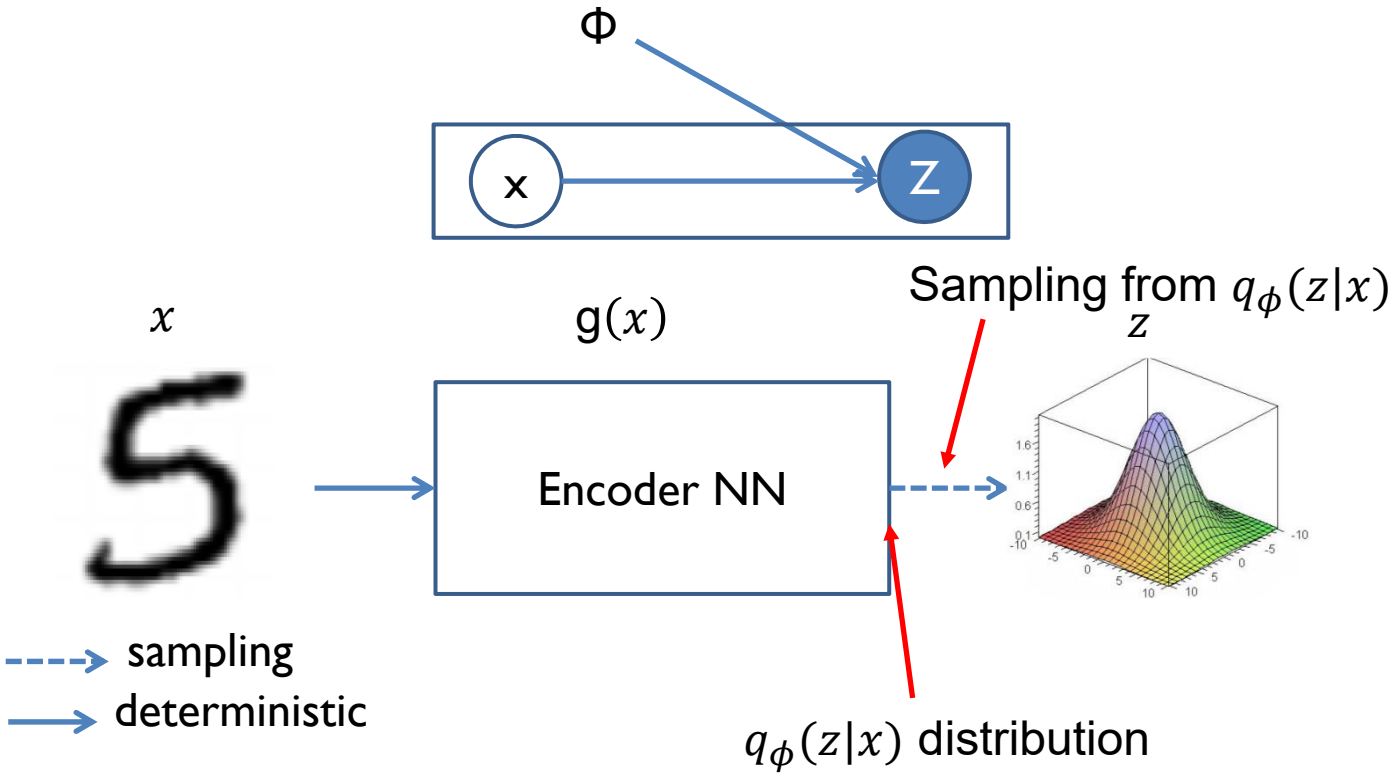


Fig. (Kingma 2015)

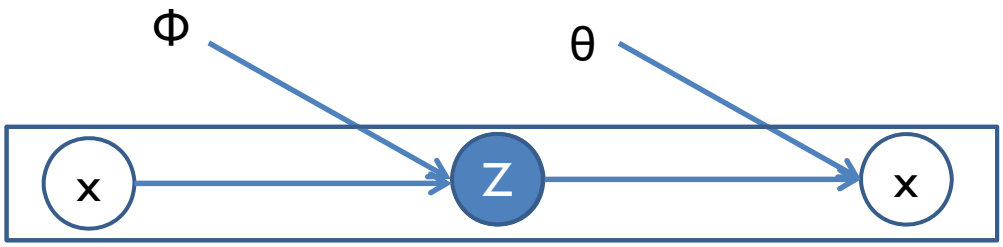
VAEs - Intuition

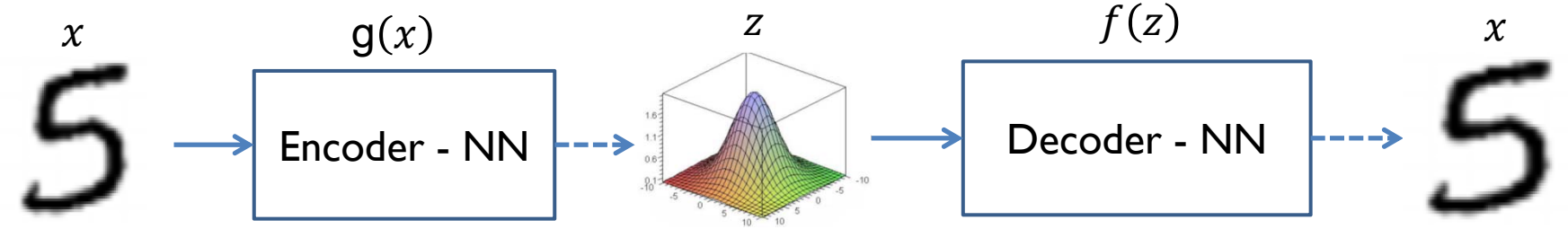
- How to choose z?
  - Inference model
  - VAEs compute z using a parametric function $g(x) = q_\phi(z|x)$



VAEs - Intuition

- Putting it all together





- ⇢ sampling
- → deterministic

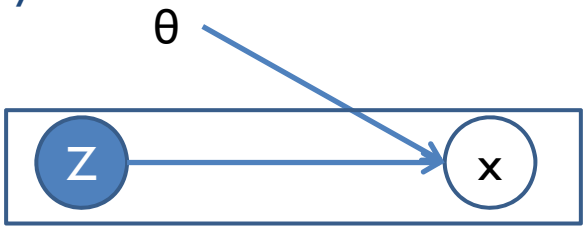
Note: both the encoder and the decoder are stochastics!

VAEs - Intuition

- What are they used for?
  - Data reconstruction as classical auto-encoders
  - Data generation (top part of the VAE), by sampling from the latent space
  - Data encoding (low part of the VAE), by learning latent space probabilistic representations
- Learning
  - Optimize empirical data log likelihood $\log p_\theta(x)$
    - $p_\theta(x)$ has no explicit form in the VAE scheme: one can sample from z, compute $f(z)$ and sample again, that is all
    - VAE propose a tractable approximation of this density
    - Training proceeds by **learning the θ and ϕ parameters**
    - **Using stochastic gradient descent**
- Note: VAEs rely on the hypothesis that there exists a density function for modeling the data

VAEs - Intuition

- Difficulties
  - $p(x)$, and $p(z|x)$ may be intractable



  - $p(x) = \int p(x,z)p(z)dz$
    - Computing the integral requires evaluating over all the configurations of latent variables, often intractable
  - $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$
    - Same problem as with the $p(x)$
  - note: to learn the parameters in a latent model using max. likelihood, one needs to compute $p(z|x)$ – remember Expectation Maximization
  - Solutions
    - Markov Chain Monte Carlo
    - Variational methods → a special form of variational inference is used in VAE

VAE description

- Let us suppose available
  - A set of observed variables x
  - Continuous latent variables z
  - Joint pdf $p_\theta(x, z)$, θ model parameters, p_θ continuous and differentiable
- Objective
  - Learn the θ parameters using a proxy of the maximum likelihood on observed data $\{x^1, \dots, x^N\}$
- VAE
  - Trained as an auto-encoder (end to end)
  - Using stochastic gradient
  - Problem
    - $p_\theta(z|x)$ intractable and is needed for max. likelihood estimation of the parameters
  - Solution
    - Approximate $p_\theta(z|x)$ with $q_\phi(z|x)$
      - Using a variational approximation

VAE
Loss criterion – summary

- Log likelihood **for data point** x:
  - $\log p_\theta(x) = D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + V_L(\theta, \phi; x)$
    - $D_{KL}(.||.) \geq 0$, 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 $V_L(\theta, \phi; x)$
- Variational lower bound (ELBO: Evidence Lower Bound):
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$

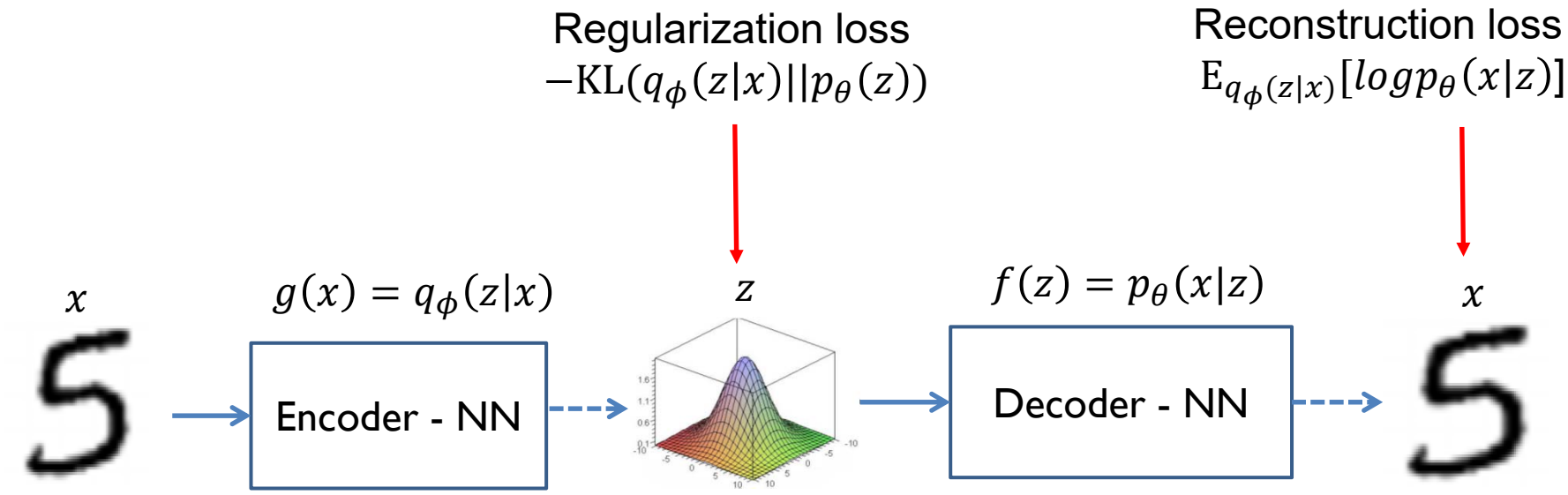
- Because each representation z is associated to a unique x, the loss likelihood can be decomposed for each point – this is what we do here
  - The global log likelihood is then the summation of these individual losses

VAE
Loss criterion – summary

- 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 $p(z)$
      - $p(z)$ has usually a simple form e.g. $\mathcal{N}(0, I)$, then $q_\phi(z|x)$ is also forced to remain simple
  - This form provides a link with a NN implementation
    - The generative $p_\theta(x|z)$ and inference $q_\phi(z|x)$ modules are im^lemented 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



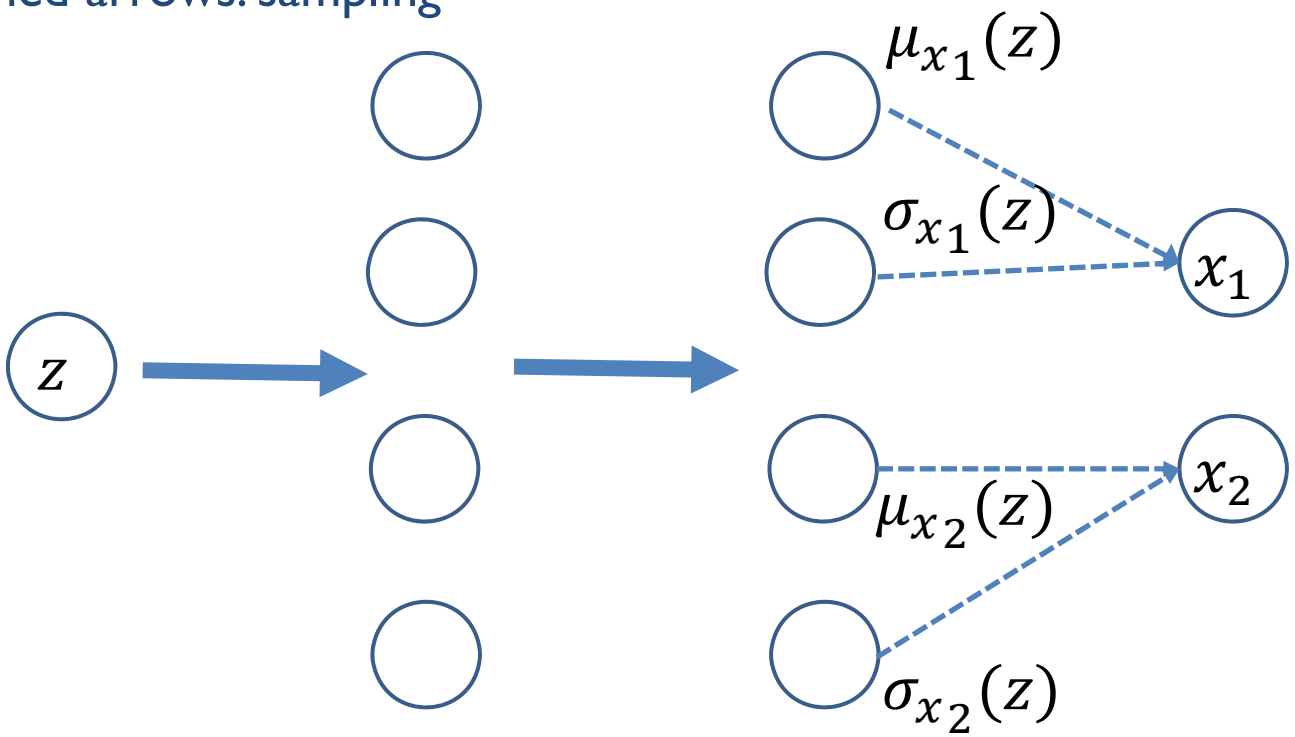
- Training performed via Stochastic gradient
  - This requires an analytical expression for the loss functions and for gradient computations
    - ⇢ Sampling
    - → deterministic

VAE
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, $z \in R^J$

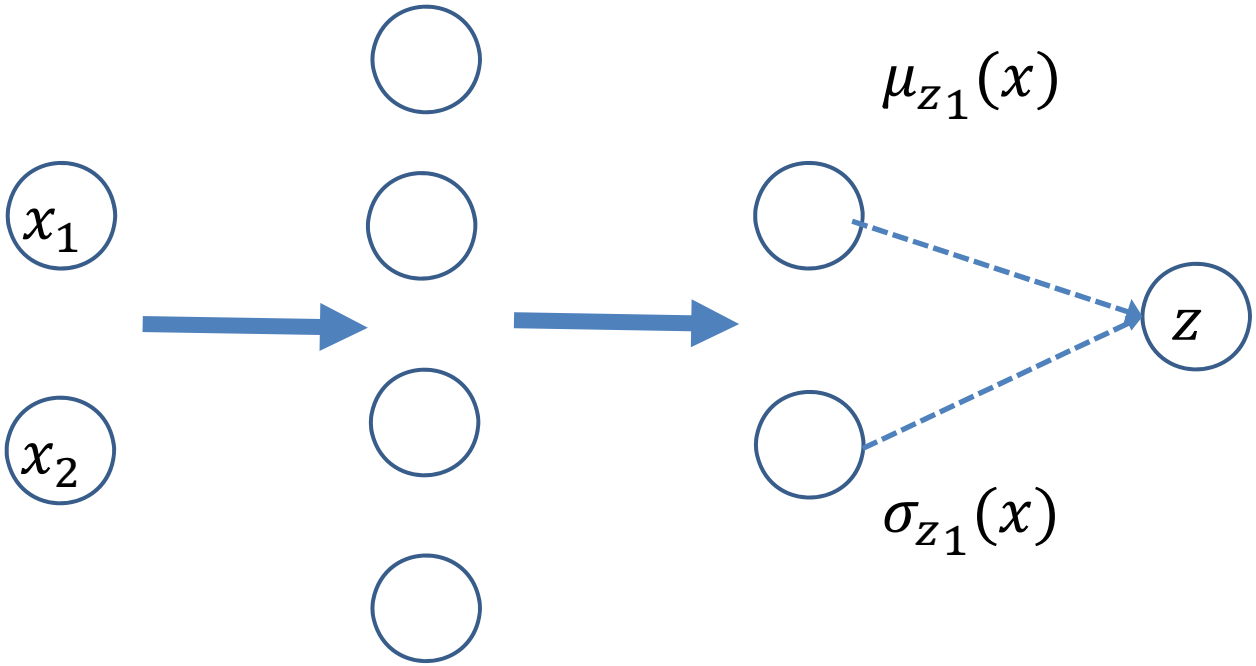
VAE
Exemple: Gaussian priors and posteriors - illustration

- 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



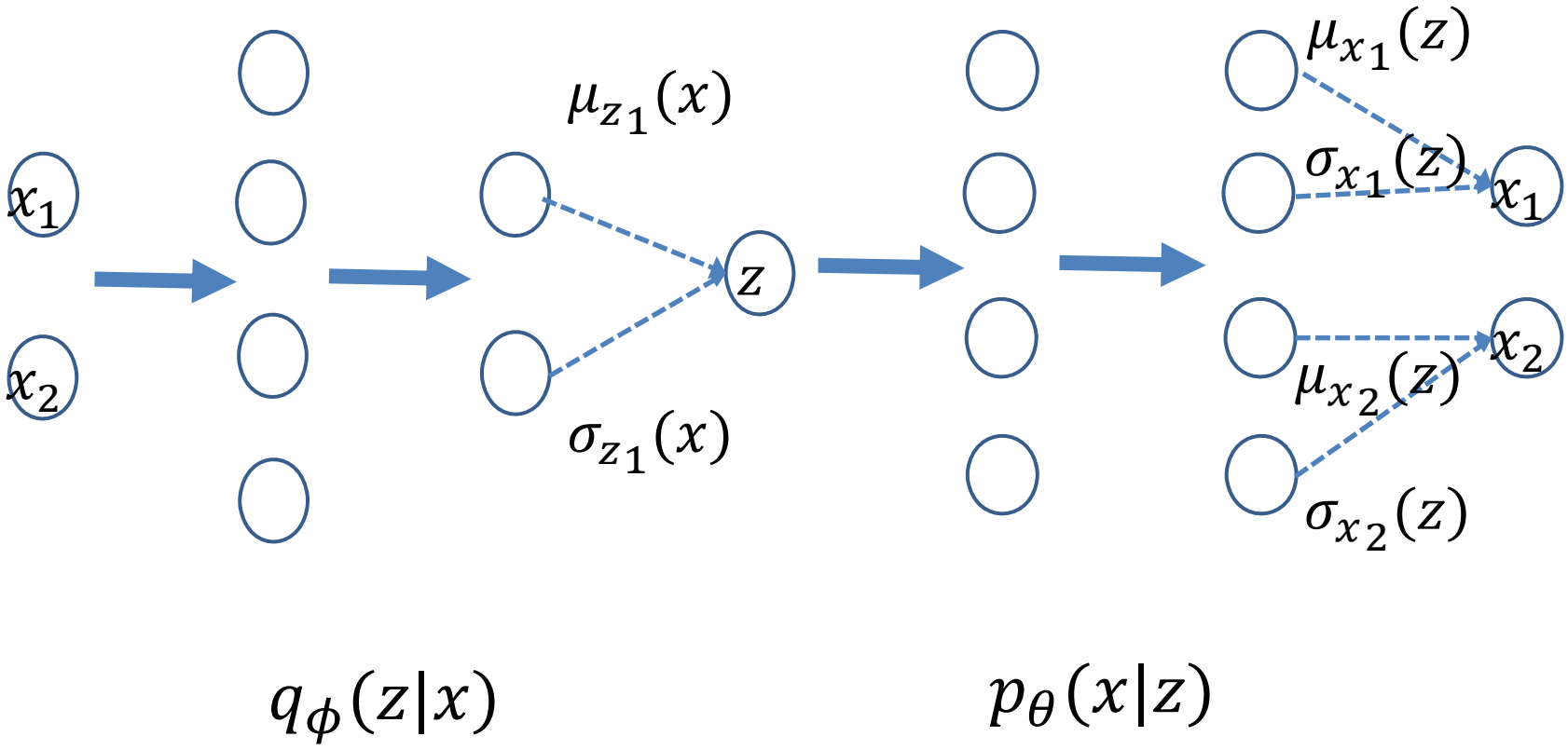
VAE
Gaussian priors and posteriors - illustration

- 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

VAE details
Derivation of the loss function

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

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

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

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

- This form provides a link with a NN implementation
  - The generative $p_\theta(x|z)$ and inference $q_\phi(z|x)$ modules will be trained to maximize the reconstruction error for each (z, x): $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - 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- 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)

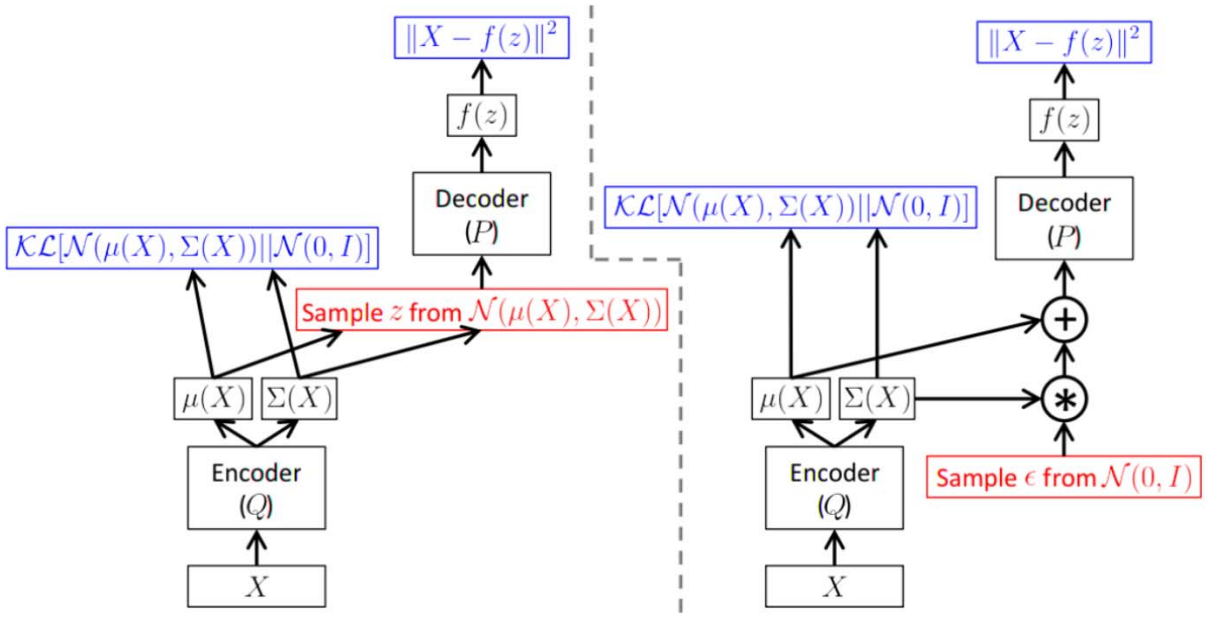


Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior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

VAE – 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
    - $E_{q_\phi(z|x)}[\log p_\theta(x|z)] \simeq \frac{1}{L}\sum_{l=1}^{L}\log(p_\theta(x|z^{(l)})$
    - $\log(p_\theta(x|z^{(l)}) = -\sum_{j=1}^{D}\frac{1}{2}\log(\sigma_{x_j}^2(z^{(l)})) + \frac{(x_j - \mu_{x_j}(z^{(l)}))^2}{2\sigma_{x_j}^2(z^{(l)})}$
    - i.e. L samples with $z^{(l)} = g_\phi(x, \epsilon^{(l)})$

VAE - instanciation example
Gaussian priors and posteriors

- If $z \in R^J$: $-KL(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left((\sigma_j)^2\right) - (\mu_j)^2 - (\sigma_j)^2)$
  - $KL(q_\phi(z)||p(z)) = \int \; q_\phi(z) \log \frac{q_\phi(z)}{p(z)} dz$
- 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 $E_q[z^2]$?
  - $E_{q_\Phi}[(z-\mu)^2] = \sigma^2$
  - $E_{q_\Phi}[z^2] - 2E_{q_\Phi}[z\mu] - \mu^2 = \sigma^2$
  - $E_{q_\Phi}[z^2] = \ \mu^2 + \sigma^2$
- 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
  - $\log\big(p(x|z)\big) = \sum_{j=1}^{D}\frac{1}{2}\log(\sigma_j^2(z)) + \frac{(x_j - \mu_j(z))^2}{2\sigma_j^2(z)}$
- 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

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

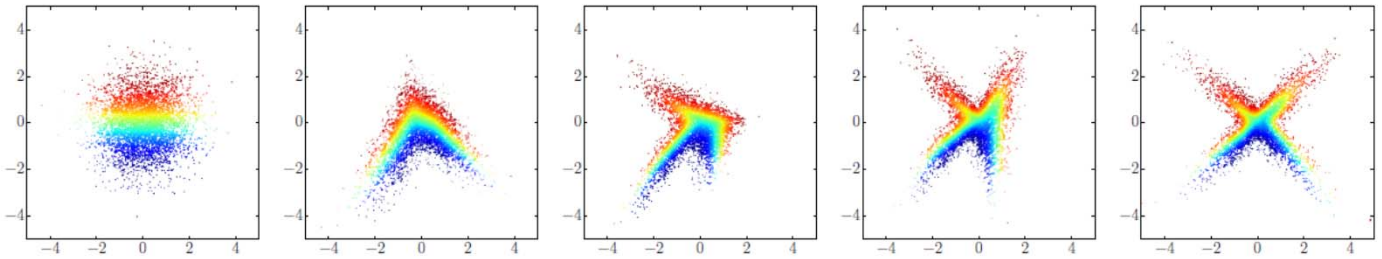


Figure 1: Example of a 4-step flow transforming samples from a standard-normal base density to a cross-shaped target density.

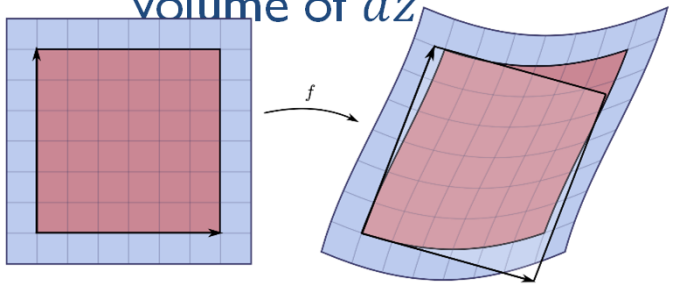
Fig. Papamakarios et al. 2019

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:
- 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
  - $p_x(x) = p_z(g(x)) \left| \det \left(\frac{\partial g(x)}{\partial x} \right) \right|$
    - We denote $J_g(x) = \frac{\partial g(x)}{\partial x}$ the Jacobian of g at x

Normalizing Flows
Prerequisite - Volumes

- 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)
- Determinant of the jacobian
  - $\left|\det\left(\frac{\partial f(z)}{\partial z}\right)\right|$ 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 $f: R^2 \rightarrow R^2$ 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:
    - $(f_2 \circ f_1)^{-1} = f_1^{-1} \circ f_2^{-1}$
    - $\det\left(\frac{\partial f_2 \circ f_1(z)}{\partial z}\right) = \det\left(\frac{\partial f_2 \circ f_1(z)}{\partial f_1(z)}\right) . \det\left(\frac{\partial f_1(z)}{\partial z}\right)$

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 $p_z(z)$ follow 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

Normalizing Flows
Density estimation and sampling

- A flow based model provides two operations:
  - 1. sampling via $x = f(z)$, where $z \sim p_z(z)$
  - 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

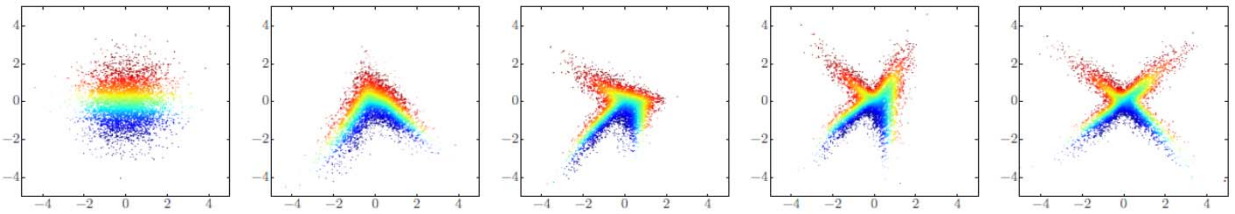


Figure 1: Example of a 4-step flow transforming samples from a standard-normal base density to a cross-shaped target density.

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 $x = f(z)$, where $z \sim p_z(z)$
    - 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 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
    - 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 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 funcion, 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 $f': R^d \rightarrow R^d$ be a bijection
  - Forward transformation $f: R^n \rightarrow R^n$ can be defined as:
    - $x_{1:d} = z_{1:d}$ identity
    - $x_{d+1:n} = f'(z_{d+1:n};\, \mathrm{h}(z_{1:d}))$ with $h: R^d \rightarrow R^{n-d}$ an arbitrary function which depends only on $z_{1:d}$ and which is called the **conditioner** or **coupling function** and f' an easily invertible transformation
  - Inverse transformation g is then
    - $z_{1:d} = x_{1:d}$ identity
    - $z_{d+1:n} = f'^{-1}(x_{d+1:n}; \mathrm{h}(z_{1:d}))$

Normalizing Flows
Constructing flows – finite composition – Coupling layers

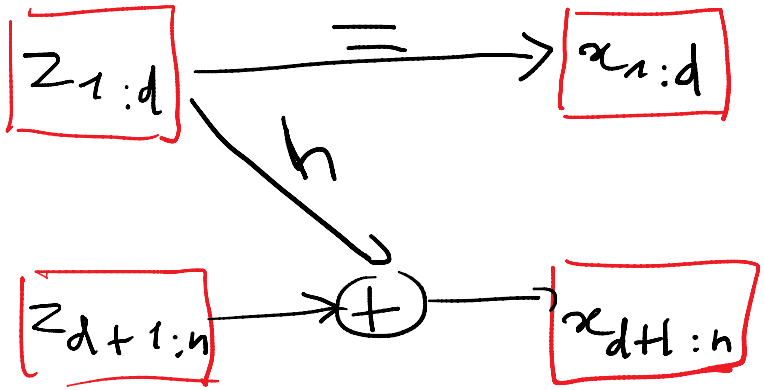
- With these definitions, the Jacobian is bloc triangular with the following form
  - $J = \frac{\partial x}{\partial z} = \begin{pmatrix} I_d & O \\ \frac{\partial x_{d+1:n}}{\partial z_1 : d} & D \end{pmatrix}$
  - where
    - I_d is a dxd identity matrix
    - $\frac{\partial x_{d+1:n}}{\partial z_{1:d}}$ 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 $\det(D) = \det(J_{f'})$
- 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)

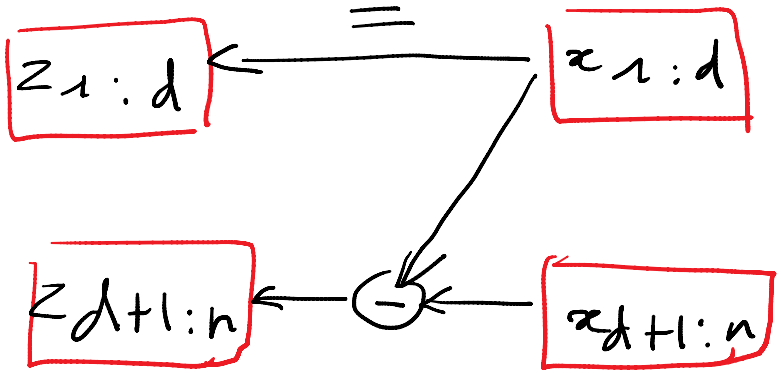
- Coupling layers were introduced in (Dinh 2015)
  - Forward transformation are implemented as **additive** transformations
    - $x_{1:d} = z_{1:d}$ (identity)
    - $x_{d+1:n} = z_{d+1:n} + \mathrm{h}(z_{1:d})$ with h a NN with d input units and $n - d$ output units
  - Inverse transformation can be 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



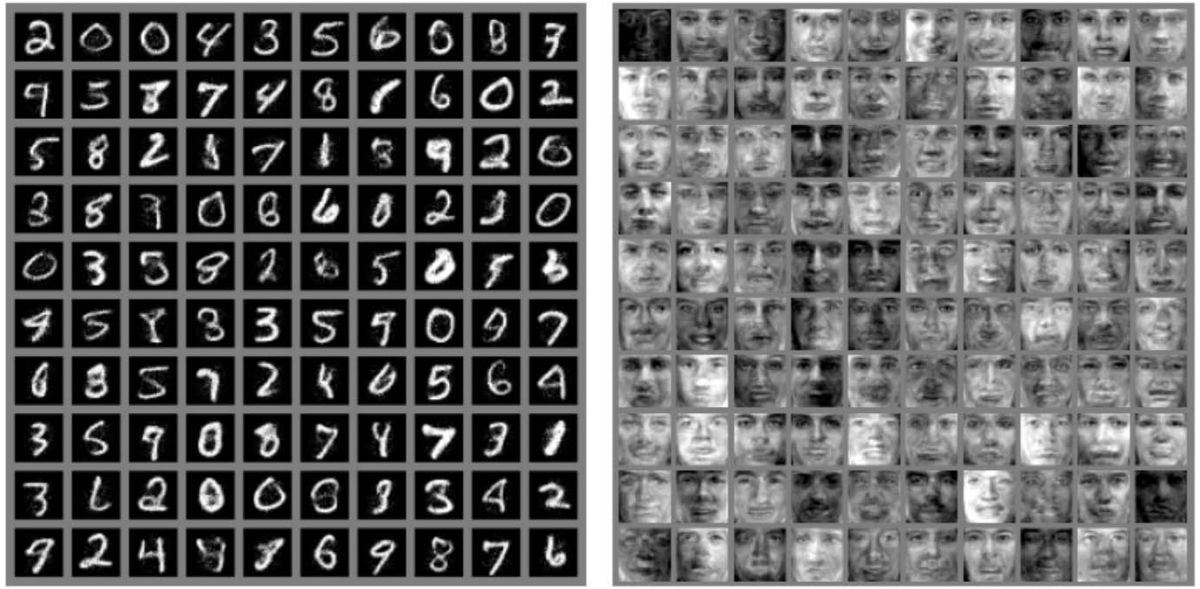
Normalizing Flows
Constructing flows – finite composition - Coupling layers – NICE (Dinh 2015)

- The coupling layers are composed together with an arbitrary partition of the variable on each layer
  - The overall tranformation is still volume preserving
  - 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

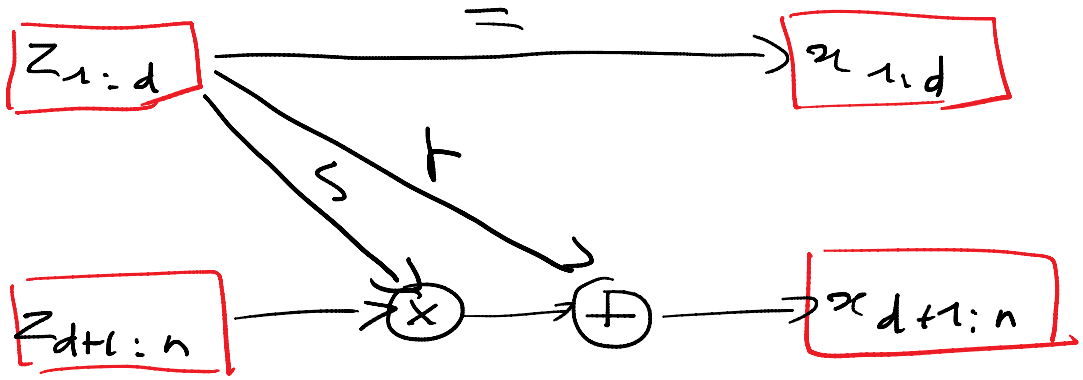
- They also evaluate the quality of the model likelihood on the data

Normalizing Flows
Coupling layers – Real NVP (Dinh 2017)

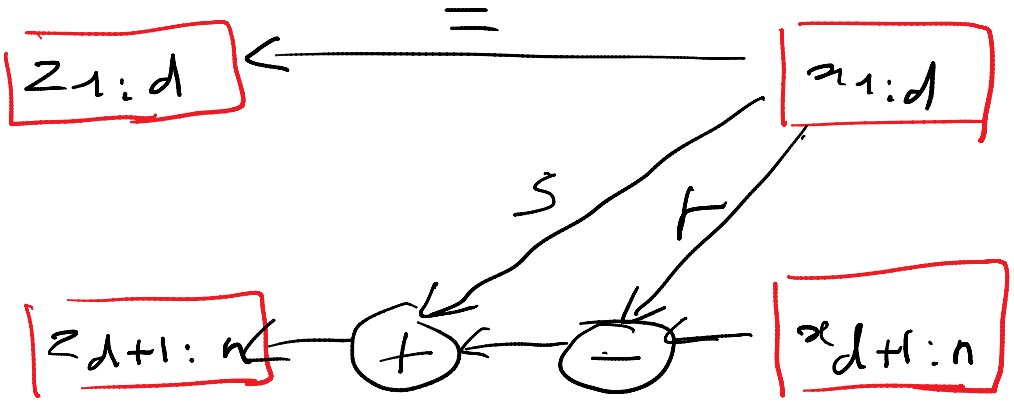
- Extension of NICE to Non Volume Preserving (NVP) Flows
- Forward transformation
  - $x_{1:d} = z_{1:d}$ (identity)
  - $x_{d+1:n} = z_{d+1:n} \odot \exp\big(\mathrm{s}(z_{1:d})\big) + t(z_{1:d})$ with s and t both $R^d \rightarrow R^{n-d}$ implemented as NNs , $\odot$ the Hadamard product, $s()$ is a **scaling** operator and $t()$ a **translation** operator, both implemented with NNs
- Inverse transformation
  - $z_{1:d} = x_{1:d}$ (identity)
  - $z_{d+1:n} = (x_{d+1:n} - t(z_{1:d})) \odot \exp(-\mathrm{s}(z_{1:d}))$
  - 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
  - $J = \frac{\partial x}{\partial z} = \begin{pmatrix} I_d & O \\ \frac{\partial x_{d+1:n}}{\partial z_1{:}d} & diag(\exp(\mathrm{s}(z_{1:d}))) \end{pmatrix}$
  - $diag(\exp(\mathrm{s}(z_{1:d})))$ is a diagonal matrix whose diagonal elements correspond to the vector $\exp(\mathrm{s}(z_{1:d}))$
  - $\det(J) = \prod_{i=d+1}^{n} \exp(\mathrm{s}(z_{1:d})_i) = \exp(\sum_{i=d+1}^{n} \mathrm{s}(z_{1:d})_i)$
  - 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

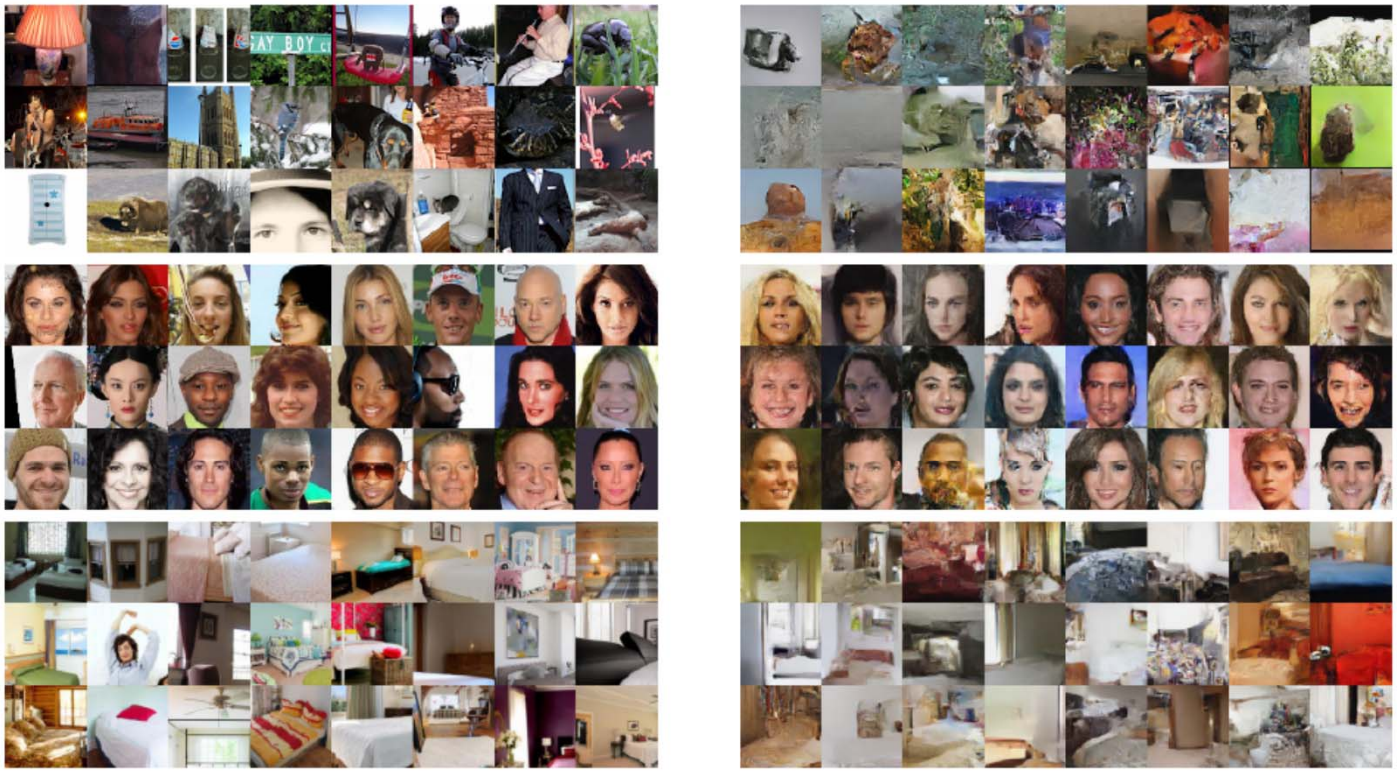


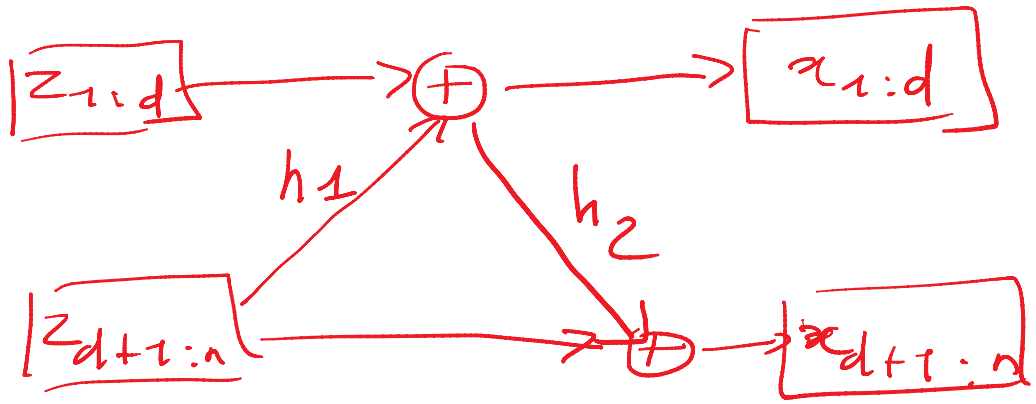
Figure 5: On the left column, examples from the dataset. On the right column, samples from the model trained on the dataset. The datasets shown in this figure are in order: CIFAR-10, Imagenet (32×32), Imagenet (64×64), CelebA, LSUN (bedroom).

Normalizing Flows
Coupling layers – RevNet (Gomez 2017)

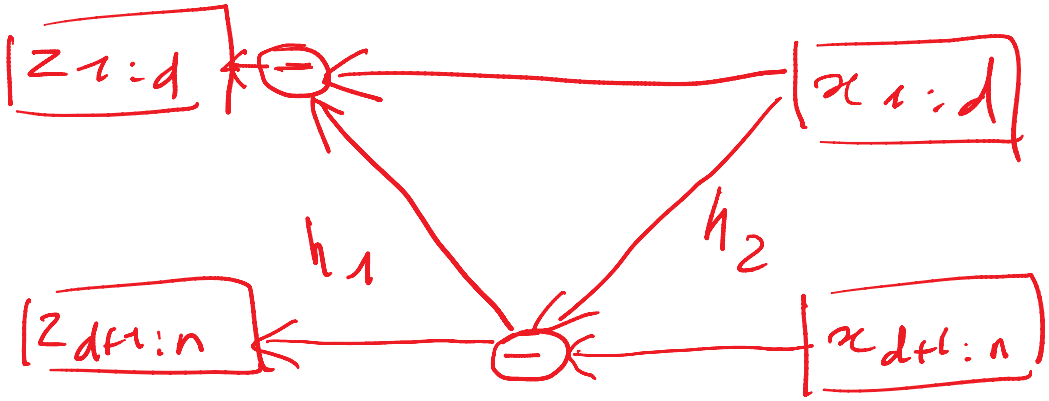
- RevNet is a reversible variant of ResNets
  - It builds on earlier coupling layers ideas (like NICE) in order to propose a reversible net where each layer activation can be computed from next layer activation thus eliminating the need for storing these activations during back-propagation
  - The main 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 $(\frac{dC}{dx_{1:d}}, \frac{dC}{dx_{d+1:n}})$, backprop can be performed without storing the activations in the forward path.

Normalizing Flows
Coupling layers – RevNet (Gomez 2017)

- Forward



- Reverse

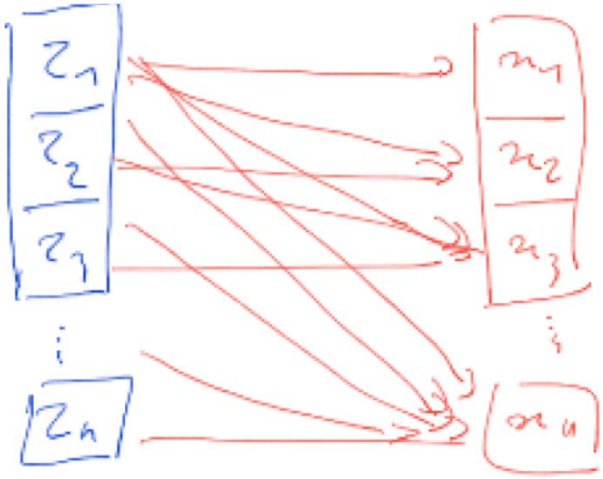


Normalizing Flows
Coupling layers – RevNet (Gomez 2017)

- 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
  - $x_t = f'(z_t; h_t(z_{1:t-1}))$ where:
    - $f': R \to R$
    - $h_t: R^{t-1} \to set\ of\ parameters$ (see examples)
    - h_0 is a constant
- Jacobian is triangular
- $J = \frac{\partial x}{\partial z} = \begin{pmatrix} \frac{\partial f'(z_1;h_1)}{\partial z_1} & \dots & 0 \\ & \ddots & \vdots \\ L & \dots & \frac{\partial f'(z_K;h_K)}{\partial z_K} \end{pmatrix}$
- $\log(\det J) = \sum_{t=1}^{k} \log(|\frac{\partial f'(z_t;h_t)}{\partial z_t}|)$



Normalizing Flows
Auto-regressive models

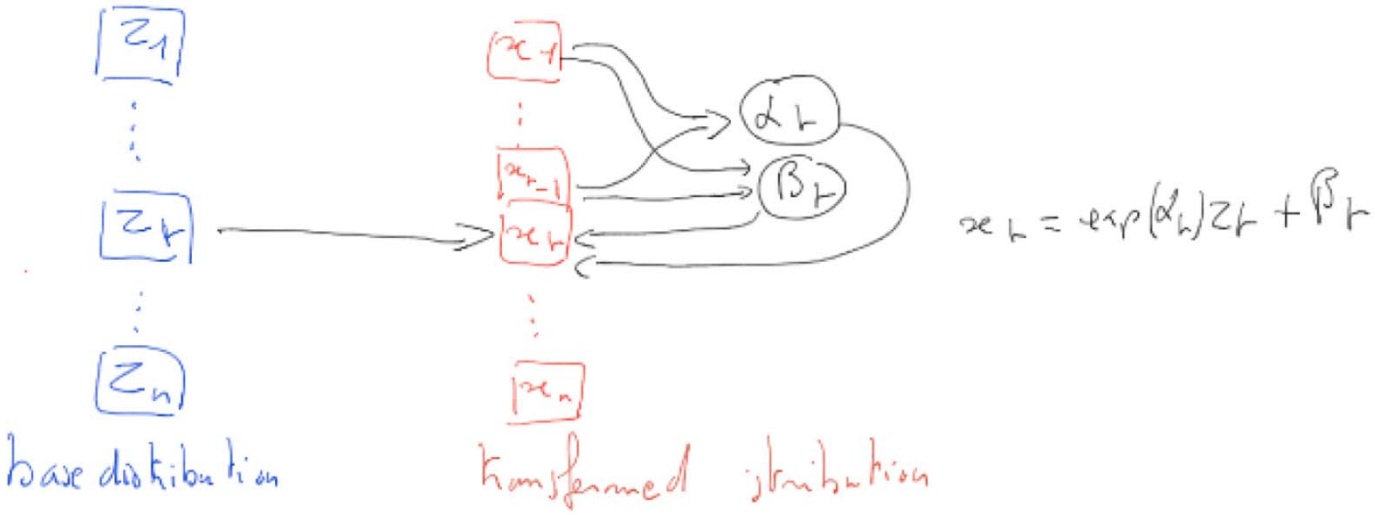
- Given the inverse of f' the inverse of f can be found by recursion
  - $z_1 = f'^{-1}(x_1; h_1)$
  - …
  - $z_t = f'^{-1}(x_t; h_t(x_{1:t-1}))$
- Complex flows
  - As with the coupling models, AR flow modules can be stacked to implement complex transformations
- Auto-regressive and Coupling models 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)

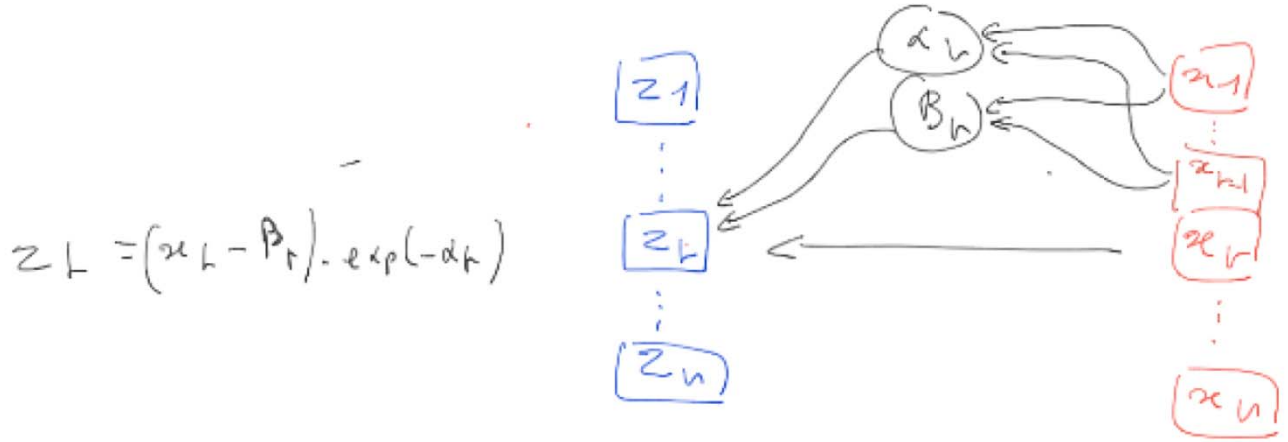
- Forward mapping



- $x_1 = \exp(\alpha_1)\, z_1 + \beta_1, \ldots, x_t = \exp(\alpha_t)\, z_t + \beta_t, \ldots, x_n = \exp(\alpha_n)\, z_n + \beta_n$
  - 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



- $z_1 = \frac{x_1 - \beta_1}{\exp(\alpha_1)}, \dots, z_t = \frac{x_t - \beta_t}{\exp(\alpha_t)}, \dots, z_n = \frac{x_n - \beta_n}{\exp(\alpha_n)}$
  - 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 $F: R^n \mathrm{x} \Theta \rightarrow R^d$ 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 $Lip(f)$ 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
- 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 ethod) and a bound on the error

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)

- 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: $\mathrm{f_t}(\mathrm{z_t}) = z_{t+1} = z_t + \mathrm{F_t}(\mathrm{z_t})$
    - 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 $f: R^n \rightarrow R^n$ 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)

- 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

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

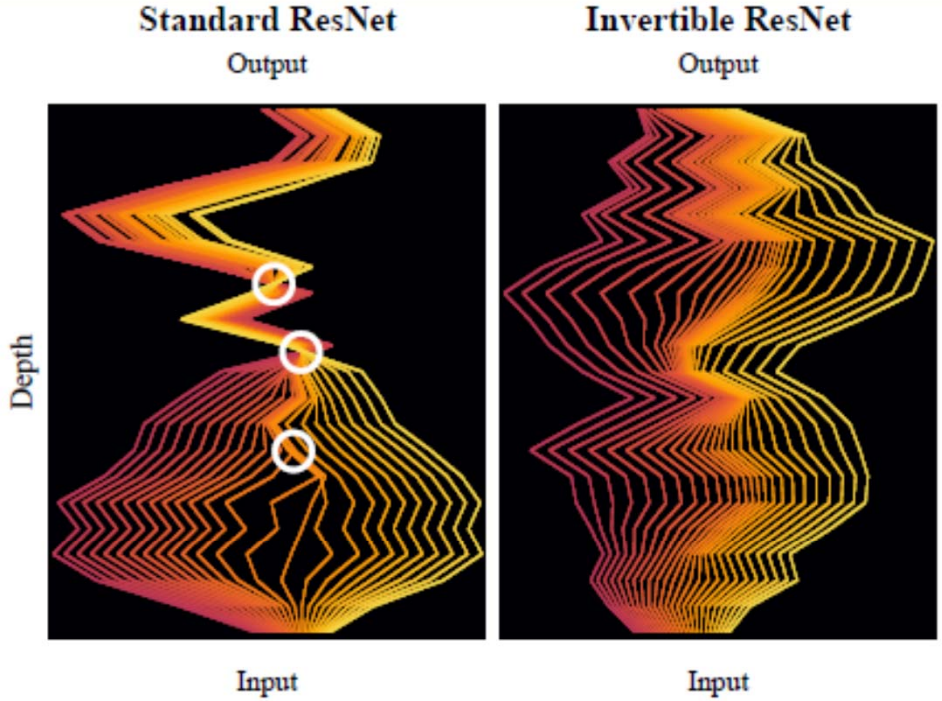


Figure 1. Dynamics of a standard residual network (left) and invertible residual network (right). Both networks map the interval $[-2, 2]$ to: 1) noisy x^3-function at half depth and 2) noisy identity function at full depth. Invertible ResNets describe a bijective continuous dynamics while regular ResNets 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)

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.

References: papers used as illustrations for the presentation

- VAEs

- 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*.

Deep Learning meets Numerical Modeling

Outline

- Temporal dynamics
  - NNs meet Ordinary Differential Equations
    - ODE numerical schemes inspired NN design
      - Stability
    - ODE solvers for solving learning problems
- Spatio-temporal dynamics
  - NNs for learning the form of Partial Differential equations
    - Could PDE explicit form be learned from data?
  - NNs for solving Partial Differential Equations
  - NN models for learning dynamical systems from data
    - Incorporating prior knowledge
    - Learning from incomplete observations
    - Hybrid models: combining solvers and Neural Networks

Motivation

- Machine learning at the heart of the AI revolution
  - Deep Learning SOTA in several domains:Vision, speech, language, Games, . etc
  - Success relies on the availability/ exploitation of data and on computer ressources
  - Agnostic approach: no/ weak prior knowledge
- Numerical modeling of physical phenomena
  - Prior physical knowledge - analytic modeling – numerical models
  - Dynamical systems often modeled via differential equations
- 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
  - How to incorporate data in physical models?

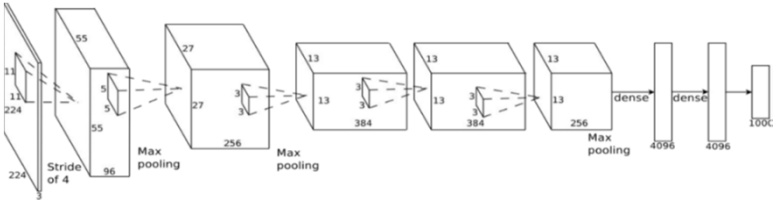
NN meets ODE

Numerical modeling for designing/ analyzing NNs

NNs meet ODE

- Deep NN as function composition

AlexNet, Krizhevsky et al. 2012



VGGNet, Simonian et al. 2014

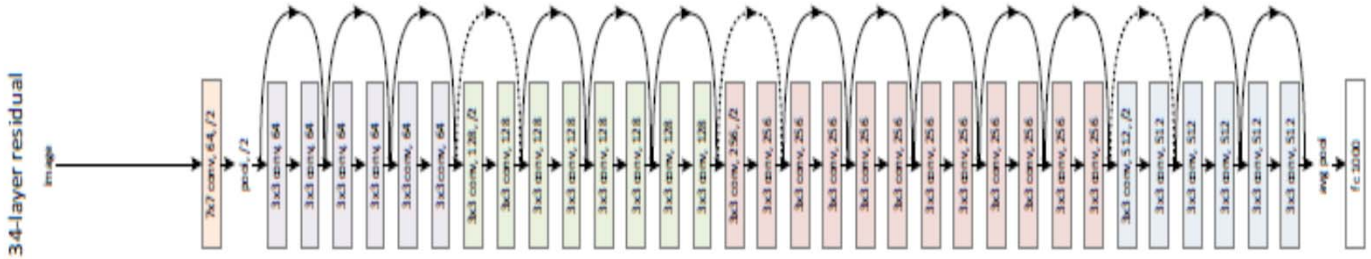
- General form of the NN
  - $F(x,\theta) = f_T \circ f_{T-1} \circ \ldots \circ f_1(x)$

- Learning problem
  - $Min_\theta \quad L(F(x,\theta), y)$
    - $s.t. \quad x_l = f_l(x_{l-1}), l = 1 \ldots T$
  - x input, y target, θ parameters, x_l layer l activation
  - Solving this pb directly leads to back-propagation
  - See the Adjoint method slides in the AMAL course

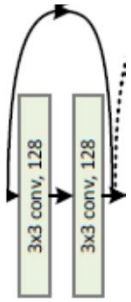
Forward graph of the NN

NNs meet ODE

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



- Resnet Module



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

- Changes the function composition perspective
  - Input x is progressively modified by a residual $f(x, \theta)$
  - x information is somewhat preserved in the forward propagation

NNs meet ODE
Training pb

- Resnet Module
  - $x_{t+1} = x_t + f(x_t, \theta_t)$ $\longrightarrow$ $x_{t+1} = x_t + hf(x_t, \theta_t)$
    - t indexes the Resnet layers, θ_t are the parameters (weights) of the corresponding layer
    - For small h, this is the forward Euler Scheme for an ODE (E 2017, Haber 2017, Chang 2018, Lu 2018, …):
    - $\frac{dx}{dt} = F(x(t), \theta(t))$
      - t is a continuous variable, $\theta(t)$ are the parameters (weights)
- ResNet - Learning Problem

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

NNs meet ODE
Continuous limit

- Continuous limit
  - If we let $h \to 0$, the ResNet learning problem becomes
  - $Min_\theta L(F(x,\theta),y)$
    - $s.t. \quad \frac{\partial x}{\partial t} = F\left(x(t),\theta(t)\right), t \in [0,T]$
- Two different families of methods for solving the 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 differetiation) to get the gradient (backward step of BP)
    - This is the classical NN setting leading to Back Propagation
  - Optimize then Discretize
    - Keep the (x,θ) continuous in time
    - Solve the forward and backward equations (e.g. adjoint method)
    - Forward and backward steps are performed via a Black Box differentiable ODE Solver

Interlude

Crash notes on ODEs

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)
  - What is the value of $x(T)$?
    - With $f: [0, T] \times R^n \rightarrow R^n$ differentiable

- Integral formulation: solution of (1)
  - $x(T) = x(0) + \int_0^T f\ \big(t, x(t)\big) dt$
  - Property: the integral formulation is equivalent to formulation (1)

Example

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

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

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

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 contiuously differentiable w.r.t. t, x, the the solution to the initial value problem is unique
- Geometrical interpretation
  - Solution curves for different solutions (initial values) do not intersect

Crash notes on ODEs

- Numerical solvers
  - $x(T) = x(0) + \int_O^T f\left(t, x(t)\right)dt$
  - What if the integral cannot be analytically integrated?
  - $\int_O^T f\left(t, x(t)\right)dt$ is approximated via numerical integration
    - Euler
    - Leapfrog
    - Runge Kutta, etc

Crash notes on ODEs

- Numerical solvers
  - Objective: build a sequence of values $x_0, x_1, \dots x_N$ that approximate the solution at the discretization points $x(t_0), x(t_1), \dots, x(t_N)$
  - There are different ways to do that:
- Derivative approximations
  - e.g. $\frac{\partial x}{\partial t} \simeq \frac{x(t+h)-x(t)}{h}$ leads to $x_{n+1} = x_n + hf(t_n, x_n)$ known as the forward Euler scheme
  - Note: we define $\mathrm{h} = \frac{T}{N}$
- Numerical integration
  - $x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} f\left(s, x(s)\right) ds$
    - Forward Euler can be recovered from this perspective too by considering approximations by left rectangles

Crash notes on ODEs

- One step methods - exemples
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$
  - 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
  - $$\begin{cases} x_{n,1} = x_n \\ x_{n,2} = x_n + hf(t_n, y_{n,1}) \\ x_{n+1} = x_n + \frac{h}{2} f(t_n, x_{n,1}) + \frac{h}{2} f(t_{n+1}, x_{n,2}) \end{cases}$$

Crash notes on ODEs

- Summary: using numerical solvers
  - $\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(t_0) = x_0 \end{cases}$ (I)
  - 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

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

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
      - $\epsilon_n = x(t_{n+1}) - x(t_n) - h\phi(t_n, y(t_n), h)$, with x 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 y_n toconverge to $y(t_n)$ when $h \to 0$
    - If the scheme (2) is stable and consistent then it is convergent, meaning that
    - $\lim_{\substack{h \to 0 \\ y_0 \to y(0)}} \sup_{0 \le n \le N} \left|\left| y_n - y(t_n) \right|\right|^2 = 0$
    - **End of the interlude**

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)
    - $x_{t+1} = (1 - k_t)x_t + k_t x_{t-1} + f(x_t, \theta_t)$
  - 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
    - $\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(0) = x_0 \end{cases}$
  - 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
    - $\max_i Re(\lambda_i(J_t)) > 0$ amplifies the signal, and may lead to divergence
    - $\max_i Re(\lambda_i(J_t)) \ll 0$ may imply signal loss
    - They propose architectures for which $Re(\lambda_i(J_t)), i = 1..l$
  - 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

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:
  - $$\begin{cases} z^k(l) = F_l(z^k(l-1), W(l)) \\ z^k(l-1) = F_{l-1}(z^k(l-2), W(l-1)) \\ \cdots \\ z^k(1) = F_1(x^k, W(1)) \end{cases}$$
  - Note
    - 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
  - $\frac{\partial \mathcal{L}}{\partial \lambda_i} = z(i) - F_i(z(i-1), W(i))$, $i = 1 \dots l$, represent the constraints
  - One wants $\frac{\partial \mathcal{L}}{\partial \lambda_i} = 0, \quad i = 1 \dots l$
  - Starting from $i = 1$ up to $i = l$, this is exactly the forward pass of BP
- 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 λs such that $\frac{\partial \mathcal{L}}{\partial z(i)} = 0, \forall\, i$
  - The λs can be computed backward Starting at $i = l$ down to to $i = 1$
    - $\lambda_l^T = \frac{\partial c(z(l), y)}{\partial z(l)}$
    - …
    - $\lambda_i^T = \lambda_{i+1}^T \frac{\partial F_{i+1}(z(i), w(i+1))}{\partial z(i)} = \lambda_{i+1}^T \frac{\partial z(i+1)}{\partial z(i)} \dots$

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 λs such that $\frac{\partial \mathcal{L}}{\partial z(i)} = 0, \forall\, i$
  - The λs 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
    - $\frac{\partial c(z(l), y)}{\partial z(i)} = \frac{\partial c(z(l), y)}{\partial z(l)} \frac{z(l)}{\partial z(l-1)} \dots \frac{\partial z(i+1))}{\partial z(i)} = \lambda_i^T$

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

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

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
    - 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 $z(i) - F_{l-1}\big(z(i-1), W(i)\big) = 0, i = 1 \dots l$

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 $g(z, W) = 0$, λ 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, 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
  - Learning problem
    - $Min_\theta L(F(x,\theta),y)$
      - $s.t. \quad \frac{\partial x}{\partial t} = F\,(x(t),\theta(t))$

Fig. Chen 2018

  - (Chen et al. 2018) solve the problem using the adjoint method for continuous time dynamics
    - Generalizes back- propagation
  - Lagrangian
    - $L(F(x,\theta),y) + \int_0^T a(t)^T \left(\frac{\partial x}{\partial t} - F\,\left(x(t),\theta(t)\right)\right) dt$

NNs meet ODE
Optimize then Discretize

- Note on the adjoint mehod
  - Learning problem
    - $Min_\theta L(F(x,\theta),y)$
      - $s.t.$ $\frac{\partial x}{\partial t} = F\left(x(t),\theta(t)\right), \mathrm{t} \in [0,T]$
    - This equation plays the same role as the constraint equations in the MLP/ ResNet formulation
  - Lagrangian
    - $L(F(x,\theta),y) + \int_0^T a(t)^T \left(\frac{\partial x}{\partial t} - F\left(x(t),\theta(t)\right)\right) dt$
    - $a(t)$ is a function of 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.

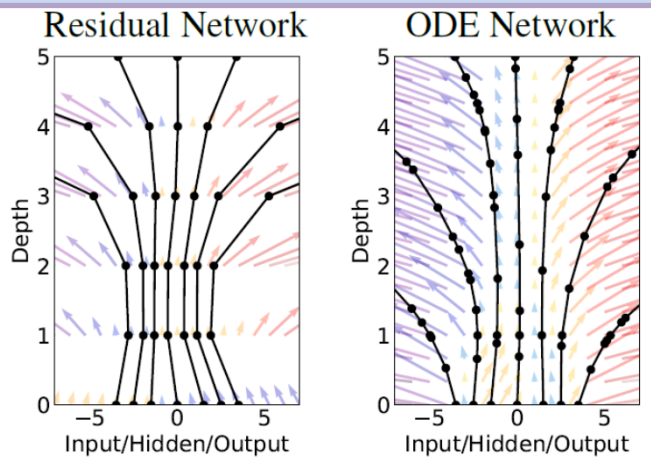
NNs meet ODE
Optimize then Discretize

- 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 trough a a call to a solver for their specific ODE: ODESolve(.)

NNs meet ODE
Optimize then Discretize

- Forward pass
  - This amounts to solve
    - $\frac{\partial x}{\partial t} = F\ (x(t), \theta(t)), \mathrm{t} \in [0, T]$
    - $x(0) = x_0$
  - 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:
      - $\frac{da(t)}{dt} = -a(t)^T \frac{\partial F\left(x(t),\theta(t)\right)}{\partial x(t)}$ (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

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

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 »

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

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

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

NN for solving Partial Differential Equations

NN for solving Partial Differential Equations
Sirignano 2018, Raissi 2019

- Objective: solve PDE in potentially large dimensional spaces
- Hypothesis
  - The form of the PDE is known, the objective is then to find an alternative method to classical solvers
- Method
  - Build a reduced order (parametric) model, implemented by a NN, to approximate the solution of a family of PDEs
- Results
  - The algorithm solves the PDE using a single parametric function, for all space and time conditions
- Note
  - The presentation follows (Sirignano 2018)

NN for solving Partial Differential Equations
Sirignano 2018, Raissi 2019

- Problem
  - Parabolic PDE with d spatial dimensions
  - $$\begin{cases} \frac{\partial u}{\partial t}(t,x) + \mathcal{L}u(t,x) = 0, (t,x) \in [0,T] \times \Omega, \Omega \subset R^d \ \text{PDE} \\ u(t=0,x) = u_0(x) \qquad \text{initial conditions} \\ u(t,x) = g(t,x), x \in \partial\Omega \qquad \text{boundary conditions} \end{cases}$$
  - $\mathcal{L}u(t,x)$ is the dufferential term of yhe PDE
  - DGM approximates u with a NN $f(t,x;\theta), \theta \in R^K$ are the network parameters
  - Note
    - For a given NN $f(t,x;\theta)$, $\frac{\partial f}{\partial t}$ and $\mathcal{L}u(t,x)$ can be expressed analytically

NN for solving Partial Differential Equations
Sirignano 2018, Raissi 2019

- Objective function
  - $J(f) = \left\| \frac{\partial f(t,x;\theta)}{\partial t} - \mathcal{L} f(t,x;\theta) \right\|^2_{[0,T]\times\Omega,\nu_1} + \|f(t,x;\theta) - \mathrm{g(t,x)}\|^2_{[0,T]\times\partial\Omega,\nu_2} + \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_3}$
  - $J(f)$ measures how well the function $f(t,x;\theta)$ satisfies (1) the PDE differential operator, (2) boundary conditions, (3) initial conditions
  - Here $\|f(y)\|^2_{y,\nu} = \int_y |f(y)|^2 \nu(y) dy$ with $\nu(y)$ a probability density function on $\mathcal{Y}$
  - The objective is to find parameters θ that minimize $J(f)$
    - The solution function $f(t,x;\theta)$ will then be a reduced order model of $u(t,x)$
  - Method
    - Minimize $J(f)$ using SGD, on a series of points sampled from Ω and $\partial\Omega$

NN for solving Partial Differential Equations
Sirignano 2018, Raissi 2019

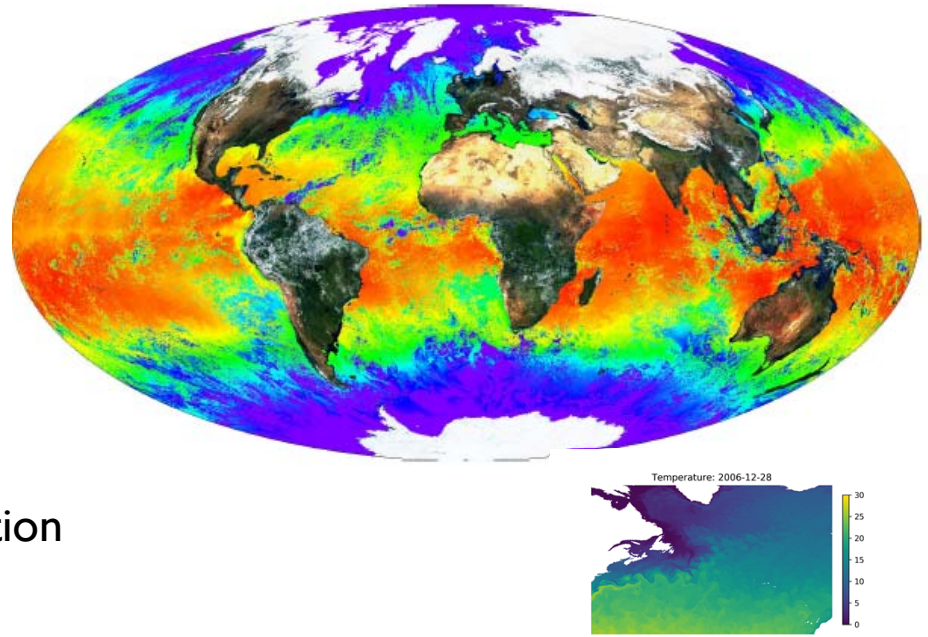
- Algorithm
- Iterate
  - Sample (t_n, x_n) from $[0,T] \times \Omega, \nu_1$; sample (τ_n, y_n) from $[0,T] \times \partial\Omega, \nu_2$, sample the initial condition point z_n from Ω, ν_3
  - Calculate the squared error $G(\theta_n, s_n)$ at the sampled points $s_n = \{(t_n, x_n), (\tau_n, y_n), z_n\}$ with:
    - $G(\theta_n, s_n) = (\frac{\partial f(t_n, x_n; \theta_n)}{\partial t} - \mathcal{L} f(t_n, x_n; \theta_n))^2 + (f(t_n, x_n; \theta_n) - (g(\tau_n, y_n))^2 + \big(f(0, z_n; \theta_n) - u_0(z_n)\big)^2$
  - Take a gradient step
    - $\theta_{n+1} = \theta_n - \epsilon_n \nabla_\theta G(\theta_n, s_n)$

Learning from observations and hybrid PDE-NN systems

- Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge, (de Bezenac et al. 2018)
- Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting. (Le Guen et al. 2021)
- Learning Dynamical Systems from Partial Observations, (Ayed et al. 2019)

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

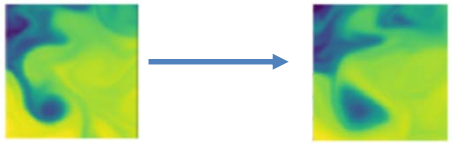
- Problem: predicting SST (< 1 meter deep) on Atlantic ocean
- Data: satellite imagery (IR)
- Use cases:
  - Weather prediction, anomaly detection, component of climate models
- Classical approach
  - Data assimilation
    - Differential equations
    - Discretization
      - Finite difference
    - 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: $k(x - w, y) = N(y|x - w, 2D\Delta t)$)

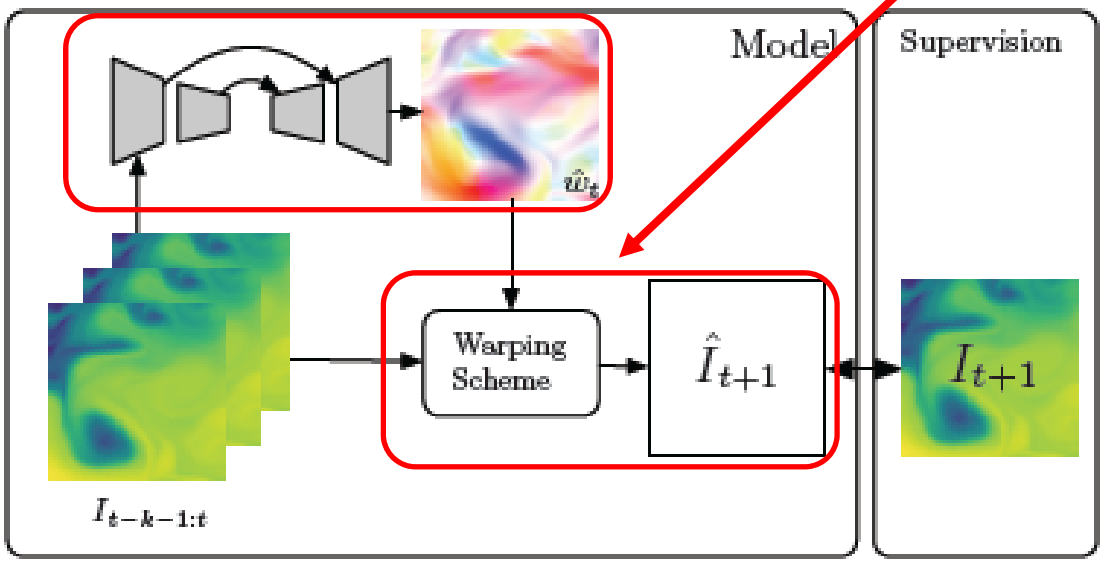
- If we knew the motion vector w and the diffusion coefficient D we could calculate $I_{t+\Delta t}(x)$ from I_t
  - **w and D unknown**
  - **-> Learn w and D**

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:

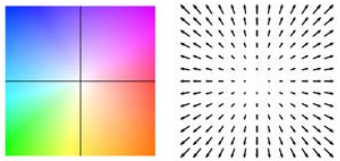
Convolution- Deconvolution NN for estimating motion vector w_t

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)
Loss function

- Prediction loss + penalty terms incorporating prior physical knowledge
- $L_t = \sum_{x \in \Omega} \rho\left(\hat{I}_{t+1}(x) - I_{t+1}(x)\right) + penalty\ terms$
  - $\rho(x) = (x + \epsilon)^{1/\alpha}$: Charbonier loss – reduces the influence of outliers compared to L_2 loss – equivalent to L_2 for $\epsilon = 0, \alpha = \frac{1}{2}$
- Penalty
  - $\lambda_{div}\left(\nabla . \widehat{w}_t(x)\right)^2 + \lambda_{magn}\|\widehat{w}_t(x)\|^2 + \lambda_{grad}\|\nabla \widehat{w}_t(x)\|^2$

divergence magnitude gradient

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

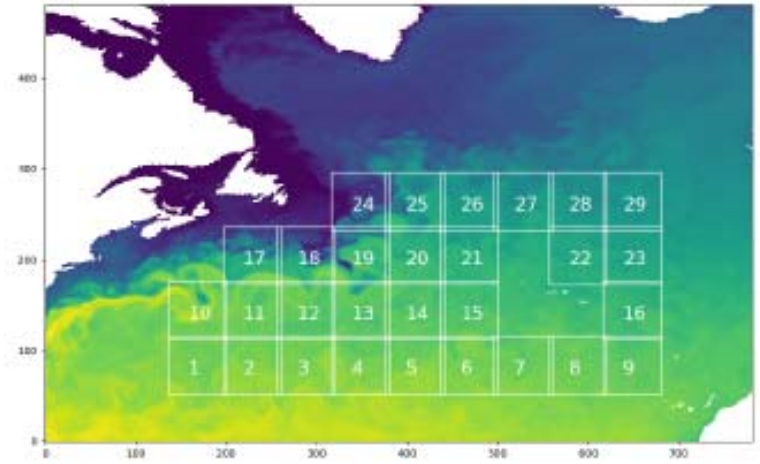


Figure 4: Sub regions extracted for the dataset. Test regions are regions 17 to 20.

- Data
  - Synthetic data fr[illegible]ng 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

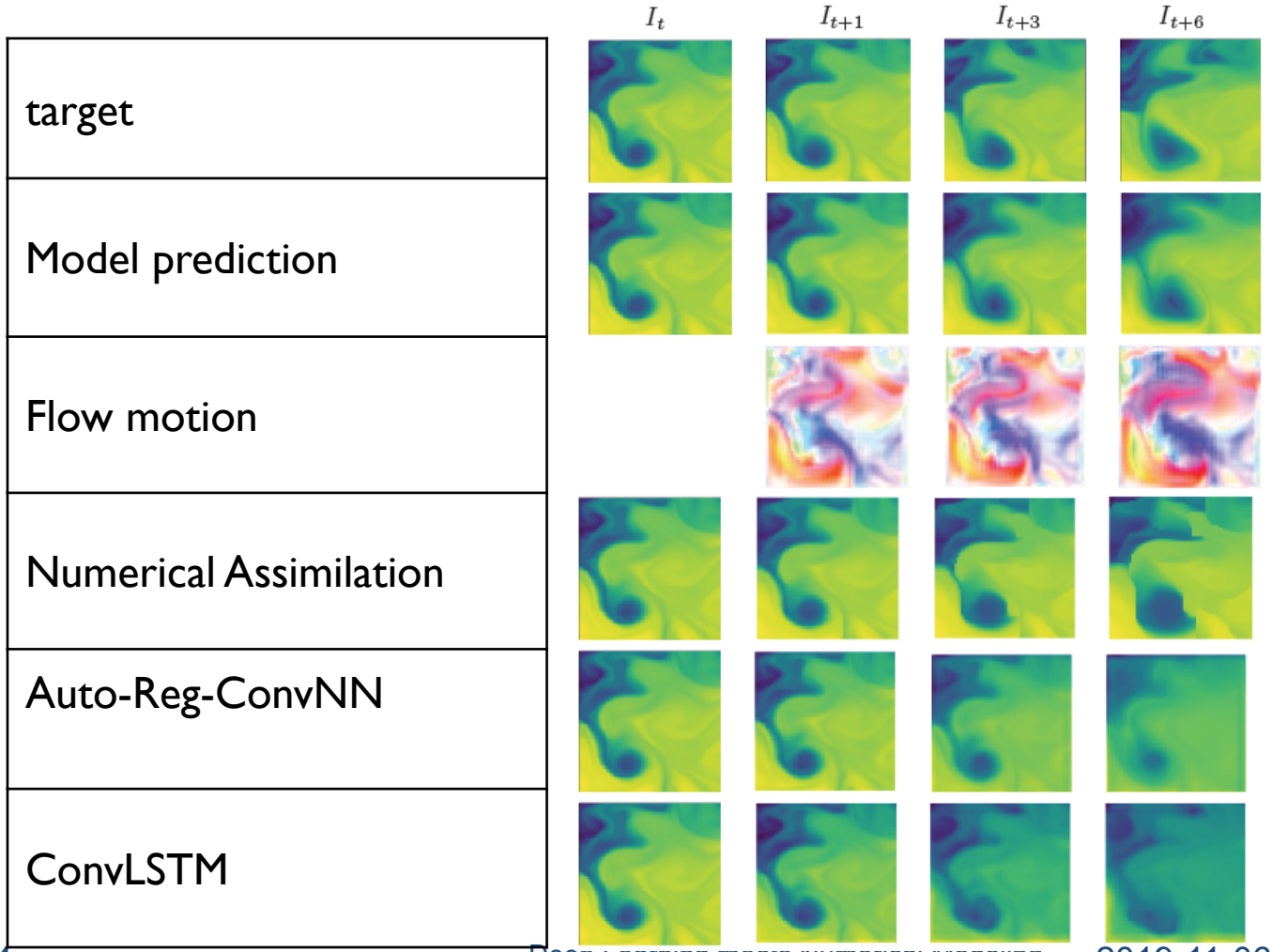
Model	Average Score (MSE)	Average Time
Numerical model Bérériat & Herlin (2015)	1.99	4.8 s
ConvLSTM Shi et al. (2015)	5.76	0.018 s
ACNN	15.84	0.54 s
GAN Video Generation (Mathieu et al. (2015))	4.73	0.096 s
Proposed model with regularization	**1.42**	0.040 s
Proposed model without regularization	2.01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

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



Combining NN and differential solvers (Le Guen et al. 2021)

- Context
  - Data driven models are insufficient to predict complex physical dynamics, e.g. extrapolating is still an open problem
  - Physical models extrapolate well if they adequately describe the dynamics
    - But: backround may not be available or only partially, unknown external factors,…
- Objective
  - Hyp: Incomplete background knowledge is available, e.g. PDE
  - Provide a principled framework to make model based and data based framework cooperate
    - Identify correctly the physical parameters
    - The NN component should learn to describe the information that cannot be captured by the physics

Combining NN and differential solvers (Le Guen et al. 2021)

- We consider dynamics of the form
  - $\frac{dX_t}{dt} = F(X_t)$
- We consider 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 (Le Guen et al. 2021)

- Solving the ill posed problem
  - Suppose available sequences of size n: $(\mathrm{X}^{\mathrm{i}}_{\mathrm{t_0}}, \ldots, \mathrm{X}^{\mathrm{i}}_{\mathrm{t_n}})$
  - Instead of solving:

$$Min_{\{\theta_a,\theta_p\}} \sum_{\{k=1\}}^{n} ||\tilde{X}^{\mathrm{i}}_{\mathrm{t_k}} - \mathrm{X}^{\mathrm{i}}_{\mathrm{t_k}}||$$

$$\text{with } \tilde{X}^{\mathrm{i}}_{\mathrm{t_k}} = X_0 + \int_{t_0}^{t_k} (F_a + F_p)(X_t)dt$$

  - We solve:

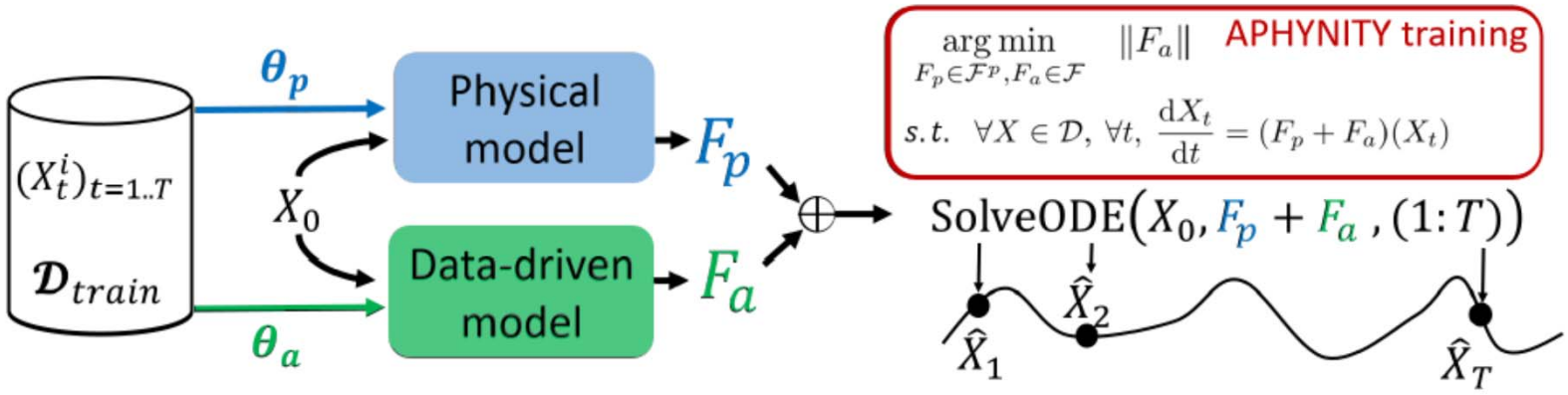
$$Min_{\{\theta_a,\theta_p\}} \left\| F_a^{\theta_a} \right\| \qquad \text{subject to } \forall i, \forall k, \tilde{X}^{\mathrm{i}}_{\mathrm{t_k}} = \mathrm{X}^{\mathrm{i}}_{\mathrm{t_k}}$$

with $\tilde{X}^{\mathrm{i}}_{\mathrm{t_k}}$ obtained by a differentiable ODE solver

    - Under mild assumptions ($\mathcal{F}_p$ closed convex sets or Chebychev), the problem admits one unique solution
    - We derive an algorithm for solving the problem

Combining NN and differential solvers (Le Guen et al. 2021)

- Summary



Combining NN and differential solvers (Le Guen 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
  - State $X = (u, v)$ defined over the compact $\Omega = [-1,1]^2$ with periodic boundary conditions.
  - Background physical knowledge
    - Diffusion with coefficients to be estimated
    - Reaction terms are ignored and shall be estimated by F_a

Combining NN and differential solvers (Le Guen et al. 2021)

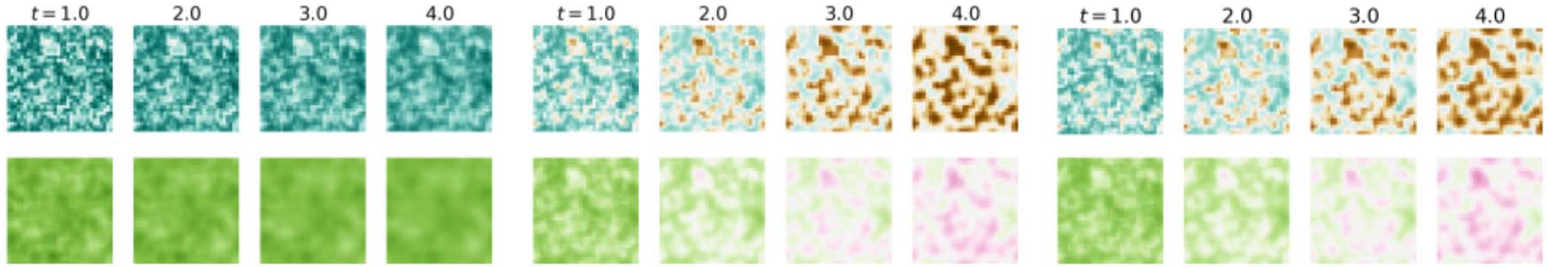
- Results
  - Accurate forecast even when the physical model is incomplete
  - Accurate physical parameter estimation

Table 1: Forecasting and identification results on the (a) reaction-diffusion, (b) wave equation, and (c) damped pendulum datasets. We set for (a) $a = 1 \times 10^{-3}, b = 5 \times 10^{-3}, k = 5 \times 10^{-3}$, for (b) $c = 330$, $k = 50$ and for (c) $T_0 = 6$, $\alpha = 0.2$ as true parameters. log MSEs are computed respectively over 25, 25, and 40 predicted time-steps. %Err param. averages the results when several physical parameters are present. For each level of incorporated physical knowledge, equivalent best results according to a Student t-test are shown in bold. n/a corresponds to non-applicable cases.

Dataset		Method	log MSE	%Err param.	$\|\|F_a\|\|^2$
(a) Reaction-diffusion	Data-driven	Neural ODE	-3.76±0.02	n/a	n/a
		PredRNN++	-4.60±0.01	n/a	n/a
	Incomplete physics	Param PDE (a, b)	-1.26±0.02	67.6	n/a
		APHYNITY Param PDE (a, b)	**-5.10±0.21**	**2.3**	67
	Complete physics	Param PDE (a, b, k)	**-9.34±0.20**	0.17	n/a
		APHYNITY Param PDE (a, b, k)	**-9.35±0.02**	**0.096**	1.5e-6
		True PDE	-8.81±0.05	n/a	n/a
		APHYNITY True PDE	**-9.17±0.02**	n/a	1.4e-7

Combining NN and differential solvers (Le Guen et al. 2021)

- Reaction Diffusion equations



(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 ($t = 2.5$).

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019) Problem

- Forecasting non linear dynamical systems from observations only
- Hypothesis
  - the underlying system follows an unknown differential equation
- Objective
  - Learn the evolution of this system (observations and state) with a NN
  - Discover automatically the relation between states (dynamics)
- Results
  - Cast the problem in a general control framework
  - Derive a generic algorithm
    - for solving the optimization problem
    - Correspondance with the training of a NN through an explicit ODE solver for partially observed states
  - Experiments on different spatio temporal data

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019) Setting

- Dynamical system with initial conditions
  - $$\begin{cases} X_0 \\ \frac{dX_t}{dt} = F(X_t) \\ Y_t = H(X_t) \end{cases}$$
- Variables
  - H: observation operator linking state to observation (known)
  - X : spatio temporal field – system state (unknown)
    - Hyp: X contains the quantities of interest for describing the system
    - Here X is a function of time t and space x (spatio-temporal)
    - e.g. velocity, pressure (Ocean), 3D information, …
  - Y : observation
    - e.g. temperature, salinity, ocean color, waves height, …

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019) Optimization problem

- Loss function
  - $\mathrm{L}(Y, \tilde{Y}) = \int_0^T \left\| Y_t - \tilde{Y}_t \right\|^2 dt$
- Learning problem
  - Minimize $\mathrm{E}_{\mathrm{Y\ observed}}[L(Y, H(X))]$
  - subject to constraints: $\frac{dX_t}{dt} = F_\theta(X_t)$
    $X_0 = g_\theta(Y_{-k}, \tilde{X}_0)$
- Implementation
  - F_θ is implemented as a residual network
  - g_θ is a Unet (for the NEMO experiments)
  - An explicit Euler scheme with a fixed step-size is used to solve the forward equation $\frac{dX_t}{dt} = F_\theta(X_t)$
    - $X^\theta_{t+\delta t} = X^\theta_{t+\delta t} + \delta t F_\theta(X^\theta_t)$

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019) Examples - Datasets

- Navier Stokes equations
  - Discretised on a spatial 64x64 grid
  - State: fluid particle density + velocity
  - Observations: density only
  - Initial state: true full state X_0
- NEMO – Nucleus for European Modelling of the Ocean Engine
  - Observations: Sea Surface Temperature
  - State: 7 variables, we make use only of 2 variables corresponding to the velocity field
  - Initial state: interpolated from previous observations
- For all test, data are partitioned into a training and a test set
  - 6 time steps used for the target sequence for training

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019)
Navier Stokes for incompressible fluids, 10 steps-ahead forecasting

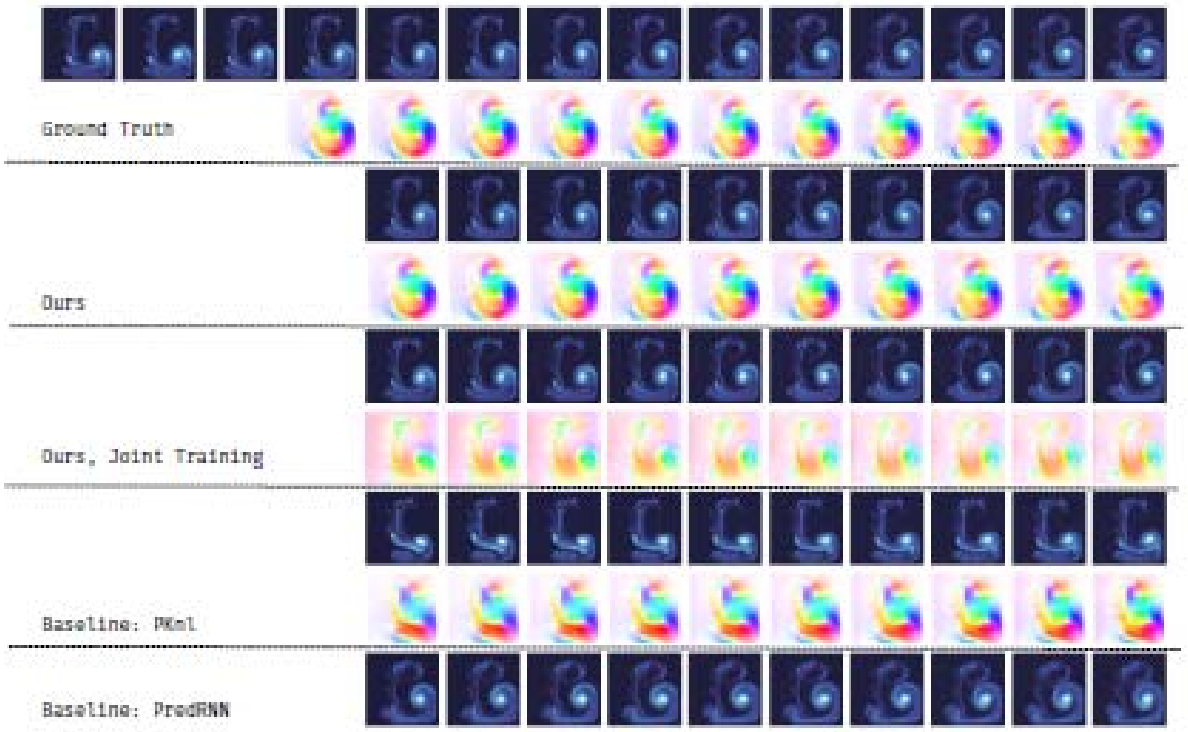


Figure 1: Forecasting the Navier Stokes equations 10 time-steps ahead with different models, starting from a given initial condition.

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019) Navier Stokes for incompressible fluids, 10 steps-ahead forecasting

- Joint learning what is in the state space?
  - For joint training (i.e. no prior on the initial state), the learned state have no reason to be close to true states
  - Learn a mapping from the learned space values to the true state on NS simulation
    - Can we recover the true state?
    - Preliminary exploration

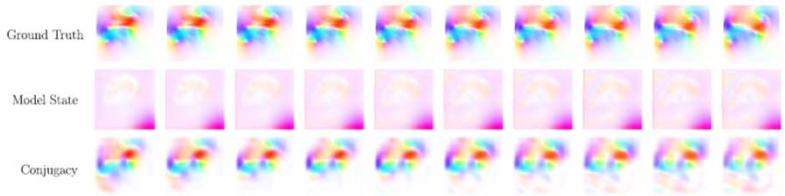


Figure 2: Example of a sequence of hidden states transformed by the calculated conjugacy.

- Cosine betwee rom 0.19 to 0.58 after learning the mapping

Learning Dynamical Systems from Partial Observations (Ayed et al. 2019)
NEMO – Global Ocean Physics Reanalysis

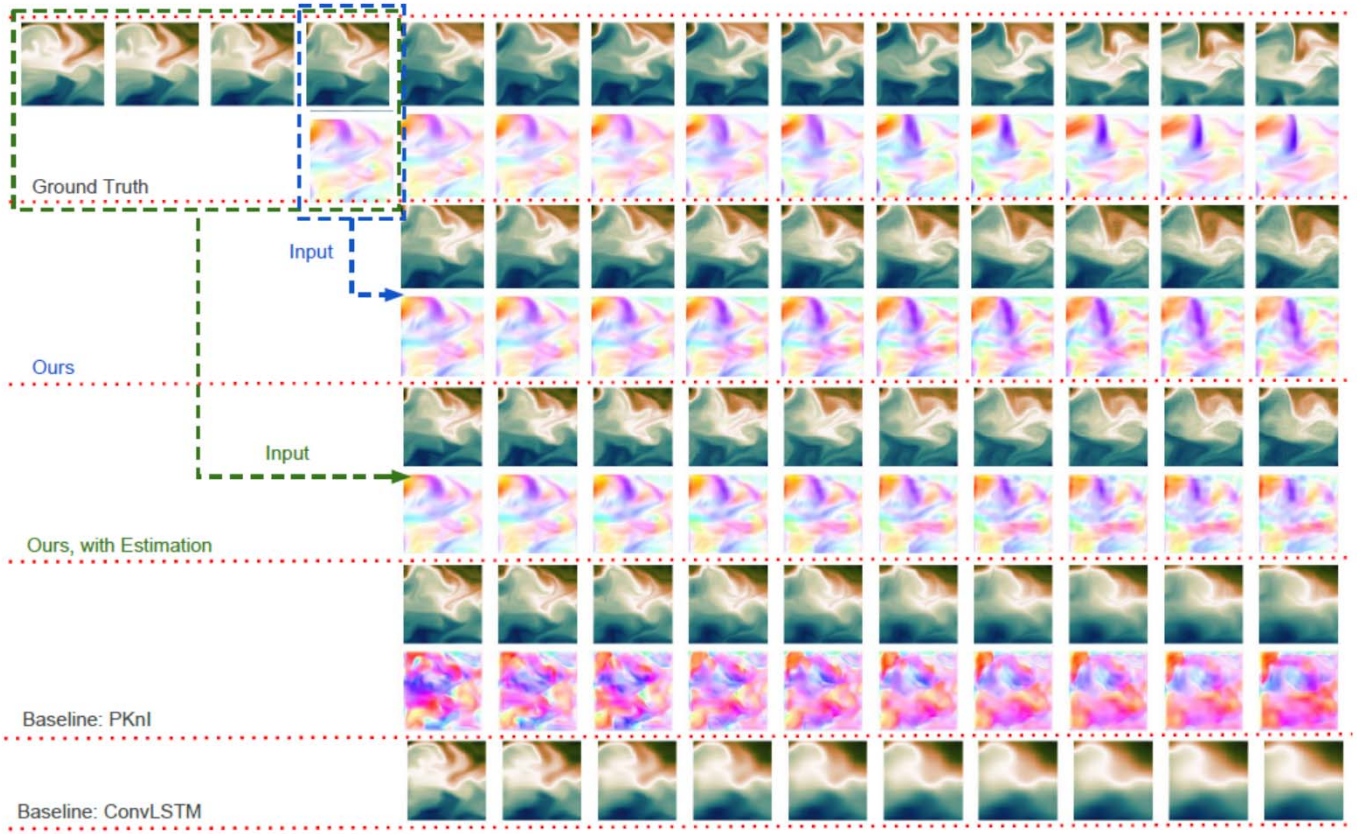


Figure 4. Forecasting Glorys2v4. From top to bottom: input and target observations, along with the associated ground truth partial hidden state, our model's outputs, our model variant when the initial conditions are estimated from the observations, outputs from the PKnI baseline, and from the ConvLSTM.

NNs meet PDE
Learning Differential Equations from Data - examples

- Data driven discovery of PDEs – (Rudy et al. 2017) - Sparse linear regression
- Deep Hidden Physics models (Raissi 2018) – Neural Networks
- PDE-Nets (Long 2018) - Neural Networks

NNs meet PDE
Objectives

- Uncover the governing equations of dynamical spatio-temporal phenomena from data
- Arguments
  - Data are ubiquitous in many domains
    - Climate, Finance, Epidemiology, ecology, etc
  - Variables or governing dynamics may be partially known or unknown so that using data may help
- Consider non linear equations of the form
  - $u_t = F(u, u_x, u_{xx}, \dots, x, \theta)$
  - Relevant variables and F may be unknown
- Objective (s)
  - Learn the unknown dynamics
  - Learn when possible the explicit form of the underlying PDE
- Current work: tests performed on data simulated from PDE

NNs meet PDE
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

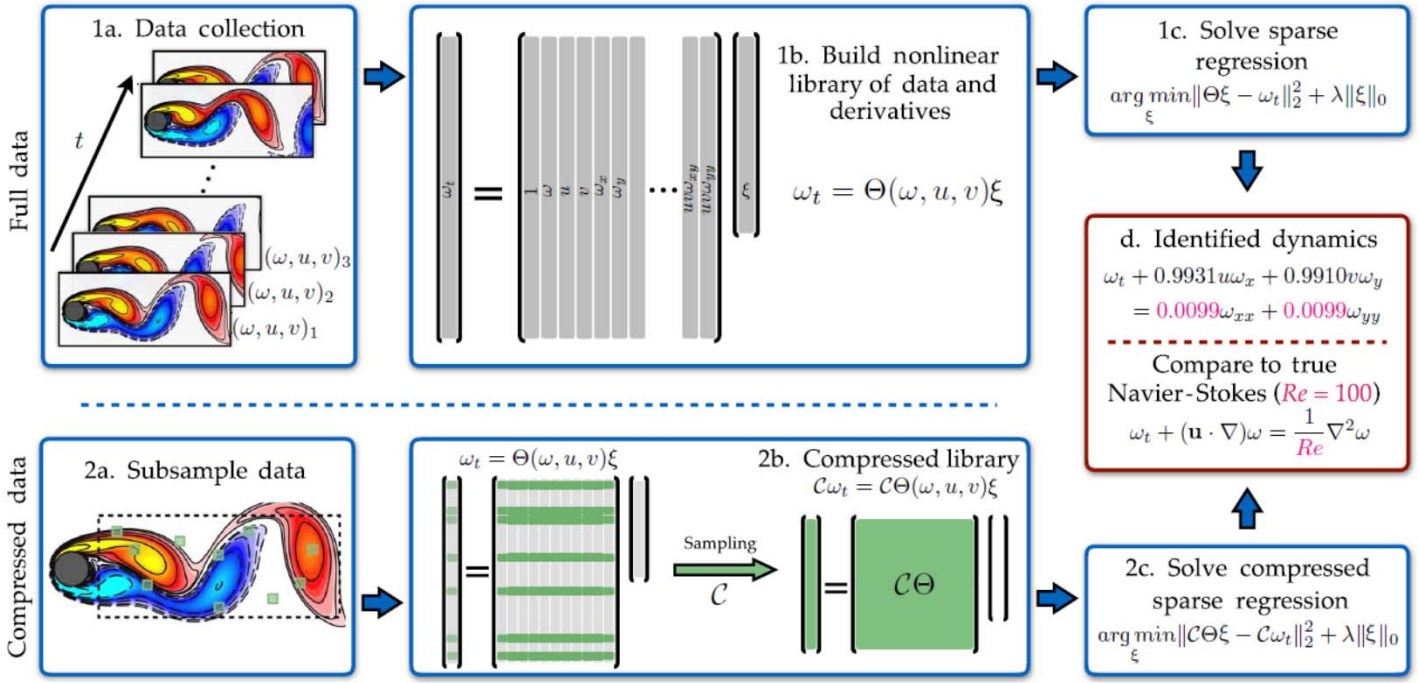
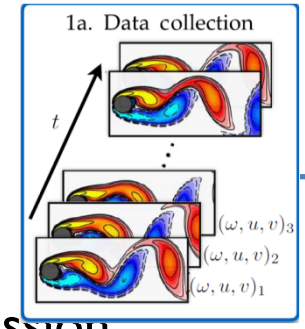


Fig. 1. Steps in the PDE functional identification of nonlinear dynamics (PDE-FIND) algorithm, applied to infer the Navier-Stokes equations from data. (1a) Data are collected as snapshots of a solution to a PDE. (1b) Numerical derivatives are taken, and data are compiled into a large matrix Θ, incorporating 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.

NNs meet PDE
Data driven discovery of PDEs– (Rudy et al. 2017)
Example

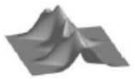
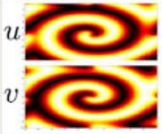
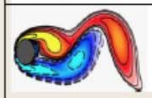


Sample (space and time)
Compute derivatives at the sampled points
Fit the data using sparse regression

- Regression
  - $u_t(x,t) =< u, u_x, u_{xx}, uu_x, \ldots; w >$ sampled at nxm points
  - w parameter vector
- Discrete form for nxm sampled points
  - $U_t = Aw$
    - with U_t : nxm vector, A : $n.m$ x nb *library terms*
- Regression loss : sparse regression
  - $\hat{w} = argmin_w \|Aw - U_t\|_2^2 + \lambda\|w\|_0$
  - Sparse regression allows finding these terms

NNs meet PDE
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%

NNs meet PDE
Data driven discovery of PDEs– (Rudy et al. 2017)

- Hypothesis
  - Full state observable
  - An overcomplete dictionary is available
    - All required terms are available
    - Non linearity requires cross product terms
  - Only a few terms will be involved in the dynamics
  - These are the conditions for sparse regression to work
- Comments
  - Interpretability
  - Requires numerical differentiation
    - Pb for large datasets
  - In the experiments noise level does not exceed 1%
  - Requires « enough » data points

NNs meet PDE
Deep Hidden Physics models (Raissi 2018)

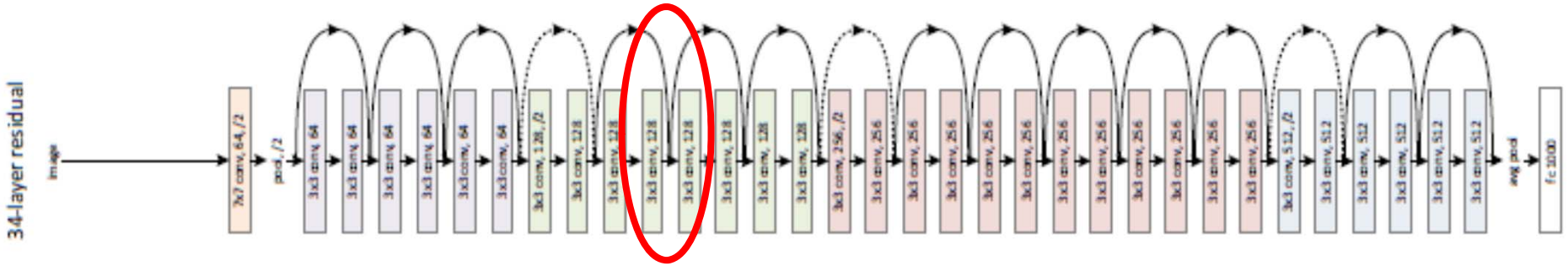
- Same objective and general idea as (Rudy 2017), uncover the dynamics of an underlying PDE
- Novelty
  - Learns a model of the data
    - Allows replacing numerical differentiation by automatic differentiation
  - Replace linear regression by a non linear neural network
- Hypothesis
  - All relevant PDE terms are provided to the model (same as Rudy 2017)
  - Full state observation is assumed (same as Rudy 2017)
  - But no need for cross terms – non linearities are taken into account by the model
- Assesment
  - No more explicit solution – how to assess the quality of the sol. ?

NNs meet PDE
Deep Hidden Physics models (Raissi 2018)

- Dataset of observations: $D = \{(t^i, x^i, u^i)\}$
- Learn model dynamics F
  - $u_t = F(u, u_x, u_{xx}, \dots, x, \theta)$
- Learn Data model for learning the solution u
  - $u(t, x)$
    - This model will be used to provide derivatives at the sampled points
    - Derivatives computed through automatic differentiation
- Training criterion
  - $L = \sum_{i=1}^{N} (|u(t^i, x^i) - u^i|^2 + f(t^i, x^i)^2)$
    - with $f(\mathrm{x}, \mathrm{t}) = u_t - F(t, x, u, u_x, u_{xx}, \dots, \theta)$
    - u and F are simple non linear Neural Networks

NNs meet PDE
PDE-NET (Long et al. 2018) PDE-NET 2.0 (Long et al. 2019)

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



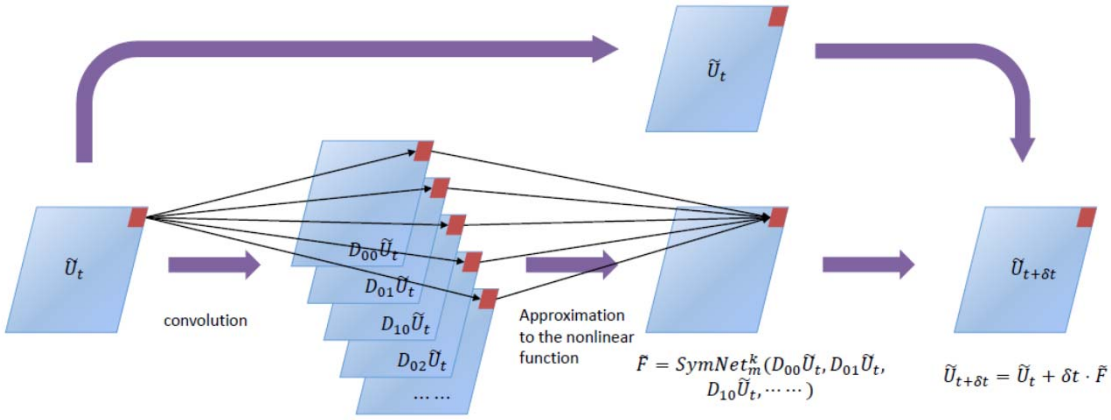
- Each block implements
  - Convolution filters
    - Exploit local properties with learned convolutional filters
    - Here used to approximate spatial differential operators
    - Constraints on the filters forces the approximation of differential operators of given order
  - Skip connections
    - $x_{t+1} = x_t + f(x_t)$
- Implement the F dynamics with a specific NN
  - Represent all the polynomials of its inputs (differential operators) with a limited number of operations

NNs meet PDE
PDE-NET (Long et al. 2018) PDE-NET 2.0 (Long et al. 2019)

- Considers equations of the form
  - $u_t = F(u, u_x, u_{xx}, \dots, \mathrm{x}, \theta)$
- Elementary module (δt −block) implements 1 step of forward Euler
  - $u(t+\delta t, .) \approx u(t, .) + \delta t. Net(D_{00}u, D_{01}u, D_{10}u, \dots)$
  - With
    - D_{ij} convolution operator associated to a filter approximating differential operators
      - $D_{ij}u \approx \frac{\partial^{i+j}u}{\partial^i x \partial^j y}$
    - *Net* is a specific « neural network » approximating F

PDE-NET (Long et al. 2018)

- Implementation: δt −block



$$u_{t+\delta t} = u_t + \delta t.F$$

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

- Multiple steps

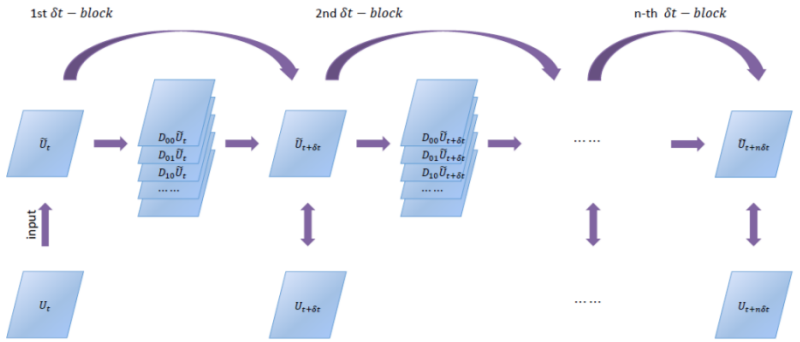
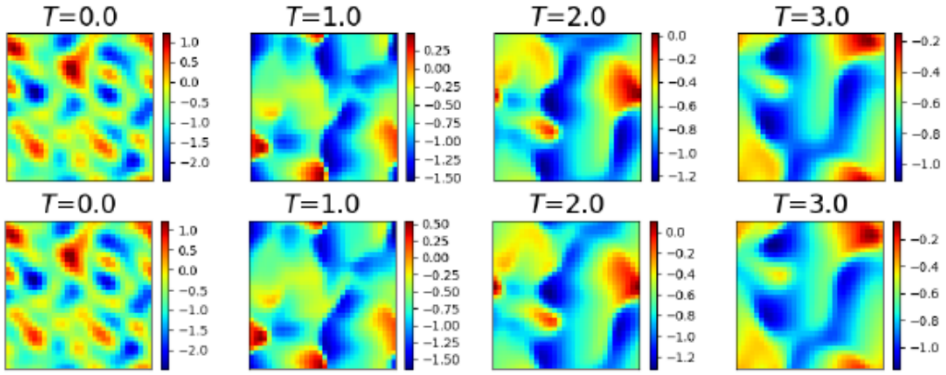


Figure 2: The schematic diagram of the PDE-Net 2.0.

PDE-NET (Long et al. 2018)

- More flexible than e.g. (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, $\delta t = 0.01$



Conclusion

- Several emerging topics at the crossroad of NNs and numerical modeling systems
- Open several perspectives both for statistical machine learning and for physical modeling

  - Cross fertilization of model based approaches and data driven approaches
  - New models for describing complex dynamics exploiting the large amounts of observation data
  - New perspectives for modeling/ training neural networks
    - e.g. as dynamical systems

References used in the presentation

- Ayed I., de Bezenac E., Pajot A., Brajard J. , Gallinari P. , (2019), Learning Dynamical Systems from Partial Observations, arXiv:1902.11136
- de Bezenac, E., Pajot, A., & Gallinari, P. (2018). Deep Learning For Physical Processes: Incorporating Prior Scientific Knowledge. In *ICLR*.
- CHANG, B., MENG, L., HABER, E., RUTHOTTO, L., BEGERT, D., AND HOLTHAM, E. 2018. Reversible Architectures for Arbitrarily Deep Residual Neural Networks. *AAAI*, 2811–2818.
- CHEN, R.T.Q., RUBANOVA, Y., BETTENCOURT, J., AND DUVENAUD, D. 2018. Neural Ordinary Differential Equations. *NIPS*.
- Delasalles E., Ziat A., Denoyer L., Gallinari P, Spatio-temporal neural networks for space-time data modeling and relation discovery, KAIS, 2019, Springer
- Haber, E. and Ruthotto, L. 2018. Stable architectures for deep neural networks. *Inverse Problems*. 34, 1 (2018).
- KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information*, 1106–1114.
- Le Guen, V., Yin, Y., Dona, J., Ayed, I., de Bézenac, E., Thome, N. and Gallinari, P. 2021. Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting. *ICLR* (2021), 1–22.
- Long, Z., Lu, Y., & Dong, B. (2018). PDE-Net 2.0: Learning PDEs from Data with A Numeric-Symbolic Hybrid Deep Network. https://doi.org/arXiv:1812.04426
- LU, Y., ZHONG, A., LI, Q., AND DONG, B. 2018. Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations. *ICML*, 3282–3291.
- 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).
- Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*.
- 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.
- Ziat, A., Delasalles, E., Denoyer, L., & Gallinari, P. (2017). Spatio-temporal neural networks for space-time series forecasting and relations discovery. *Proceedings - IEEE International Conference on Data Mining, ICDM*, *2017–Nov*, 705–714.
- A blog on Neural ODE: https://www.depthfirstlearning.com/2019/NeuralODEs