Autoencoders and Variants & Data-driven Solution and Discovery of Differential Equations

Yibo Yang, Georgios Kissas

Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104, USA

ybyang@seas.upenn.edu
gkissas@seas.upenn.edu

November 7, 2018

Overview

1. Autoencoders, and Variants
   - What is unsupervised learning? Why care?
   - AE: What are standard autoencoders? How to train them?
   - VAE: What are variational autoencoders?
   - AAE: The natural coupling of AE and GAN.
   - ACAI: How to improve interpolation in autoencoders?
   - Image to Image regression.

2. Data-driven Solution and Discovery of Differential Equations
   - What is the problem of data-driven discovery of differential equations?
   - What are the "classical" non-deep approaches.
   - Deep learning approaches: Physics Informed Neural Networks

Unsupervised Learning

- Supervised Learning: Given $\{(x_i, y_i)\}_{i=1}^m$, learn $f : \mathcal{X} \to \mathcal{Y}$. Examples: classification, regression, ...
- Unsupervised Learning: Inferring a function that describes the structure of given unlabeled data $\{x_i\}_{i=1}^m$. Example: clustering, density estimation, dimensionality reduction, ...
- Semi-supervised Learning: Given a (small) amount of labeled data $\{(x_i, y_i)\}_{i=1}^m$ and a (large) amount of unlabeled data $\{x_i\}_{i=m+1}^p$, learn $f : \mathcal{X} \to \mathcal{Y}$. Example: learning parse trees, image search, ...

The Reason for Unsupervised Learning

Benefits:

- It is a natural thing to want to do with large data.
- Can reveal a lot about the structure of data $\rightarrow$ exploratory data analysis. e.g., similar looking images, patients with similar disease profiles, ...
- Allows us to compress data by replacing points by their latent representation

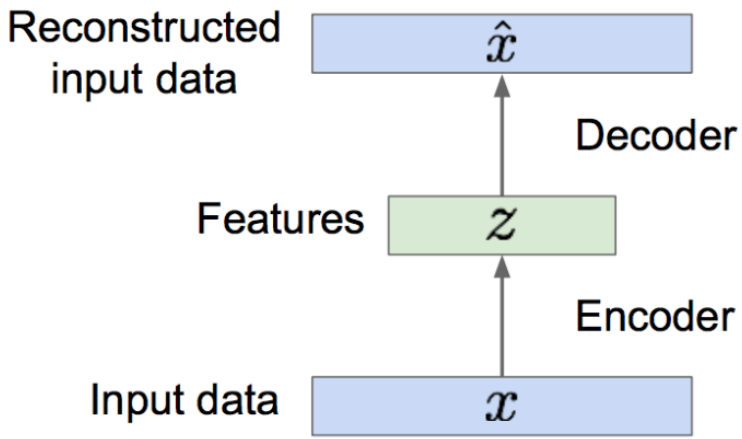
Difficulties:

- Unsupervised problem is always harder to formalize.
- Ill-defined: different objective functions possible, no clear winner. Even if we have managed to do unsupervised learning, it is hard to say whether the result is good or bad $\rightarrow$ subjective.
- Compared to supervised Machine Learning, the theory is in its infancy.

Auto-encoders

Auto-Encoders are an unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data.

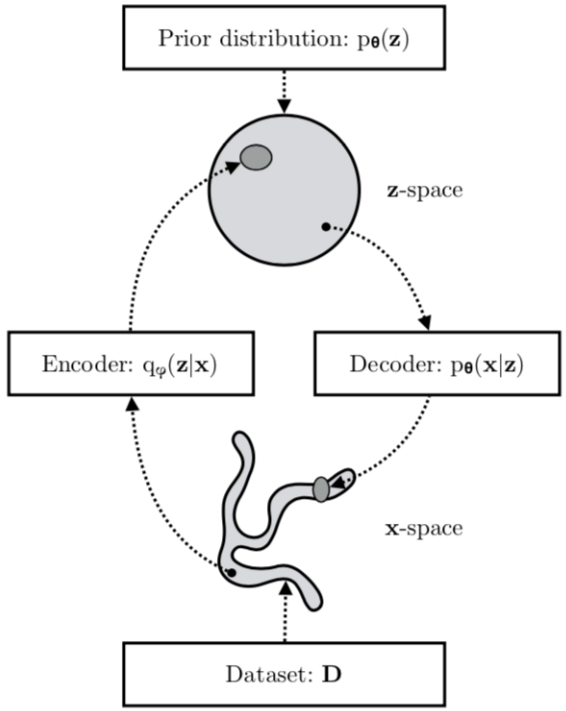
Auto-Encoder



An auto-encoder neural network is an unsupervised learning algorithm that applies back-propagation, setting the target values to be equal to the inputs. e.g. minimize the L2 loss function: $\|\hat{x} - x\|^2$.
Probably not good at generating new meaningful images.

Variational Auto-encoders

Prior distribution $p_\theta(\mathbf{z})$.
Conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$.
Conditional distribution $q_\phi(\mathbf{z}|\mathbf{x})$.



The parameters of the VAE can be estimated efficiently in the stochastic gradient variational Bayes (SGVB) framework, where the variational lower bound of the log-likelihood is used as a surrogate objective function. The variational lower bound is written as:

$$\log p_\theta(x^i) \geq \mathbb{E}_z[\log p_\theta(x^i|z)] - \mathbb{KL}[q_\phi(z|x^i)||p_\theta(z)] \quad (1)$$

Variational Auto-encoders

We want to maximize the data likelihood:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz \tag{2}$$

$$\begin{aligned}
\log p_\theta(x^i) &= \mathbb{E}_{z\sim q_\phi(z|x^i)}[\log p_\theta(x^i)] \\
&= \mathbb{E}_z[\log \frac{p_\theta(x^i|z)p_\theta(z)}{p_\theta(z|x^i)}] \\
&= \mathbb{E}_z[\log \frac{p_\theta(x^i|z)p_\theta(z)}{p_\theta(z|x^i)} \frac{q_\phi(z|x^i)}{q_\phi(z|x^i)}] \\
&= \mathbb{E}_z[\log p_\theta(x^i|z)] - \mathbb{E}_z[\log \frac{q_\phi(z|x^i)}{p_\theta(z)}] + \mathbb{E}_z[\log \frac{q_\phi(z|x^i)}{p_\theta(z|x^i)}] \\
&= \mathbb{E}_z[\log p_\theta(x^i|z)] - \mathbb{KL}[q_\phi(z|x^i)||p_\theta(z)] + \mathbb{KL}[q_\phi(z|x^i)||p_\theta(z|x^i)]
\end{aligned} \tag{3}$$

The last KL-Divergence is greater than or equal to 0. Define:

$$\mathcal{L}(x^i, \theta, \phi) = \mathbb{E}_z[\log p_\theta(x^i|z)] - \mathbb{KL}[q_\phi(z|x^i)||p_\theta(z)] \tag{4}$$

Variational Auto-encoders

The sampling process of VAEs:

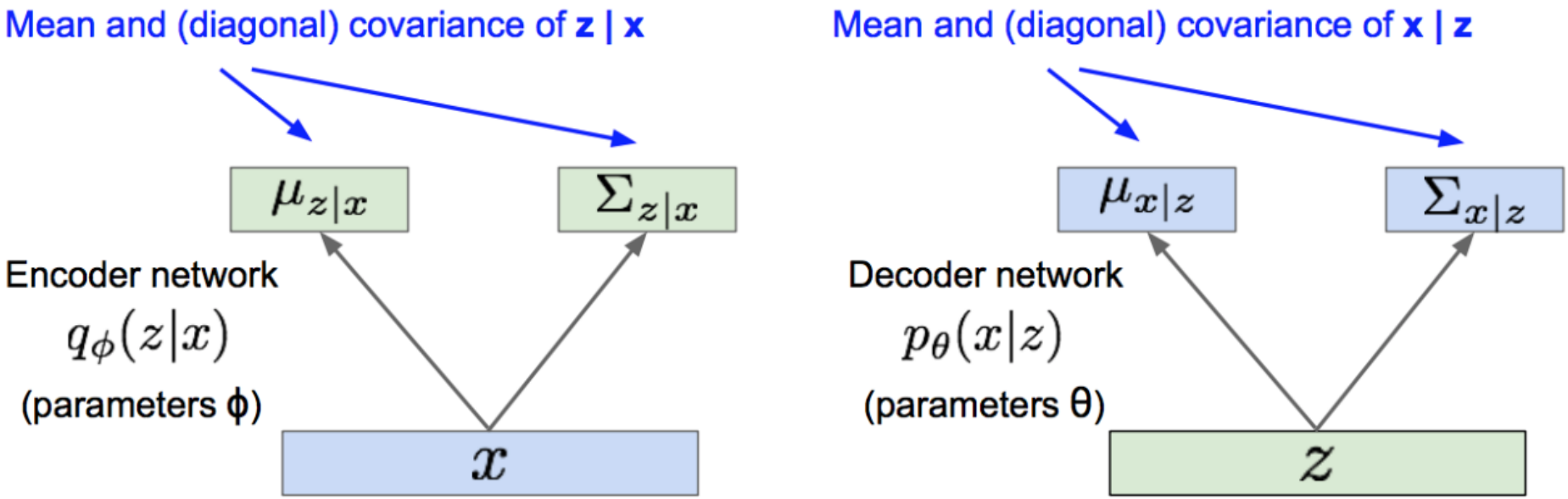


Figure: Sampling process of VAEs.

Variational Auto-encoders

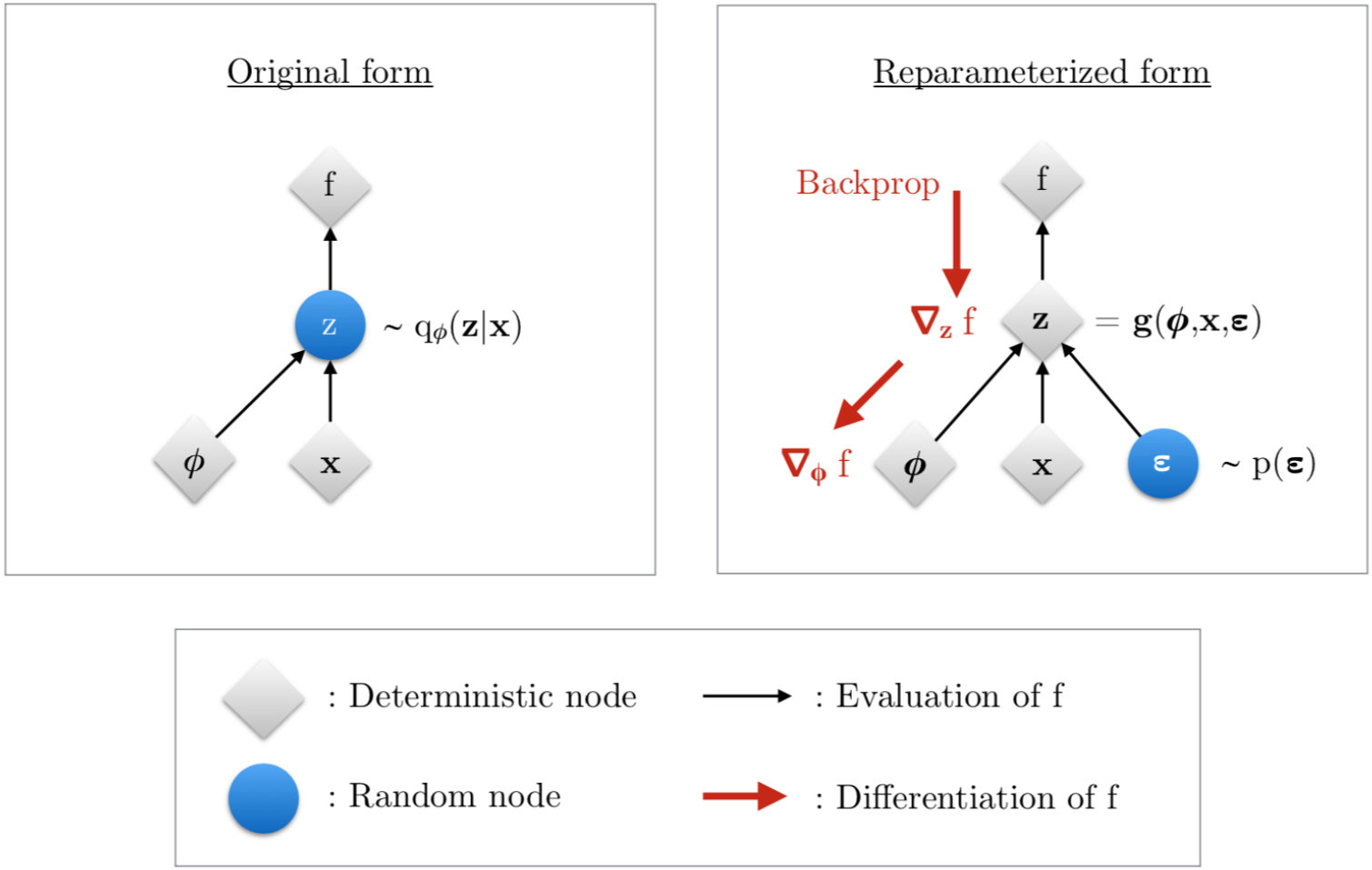


Figure: Reparameterization trick for VAEs.

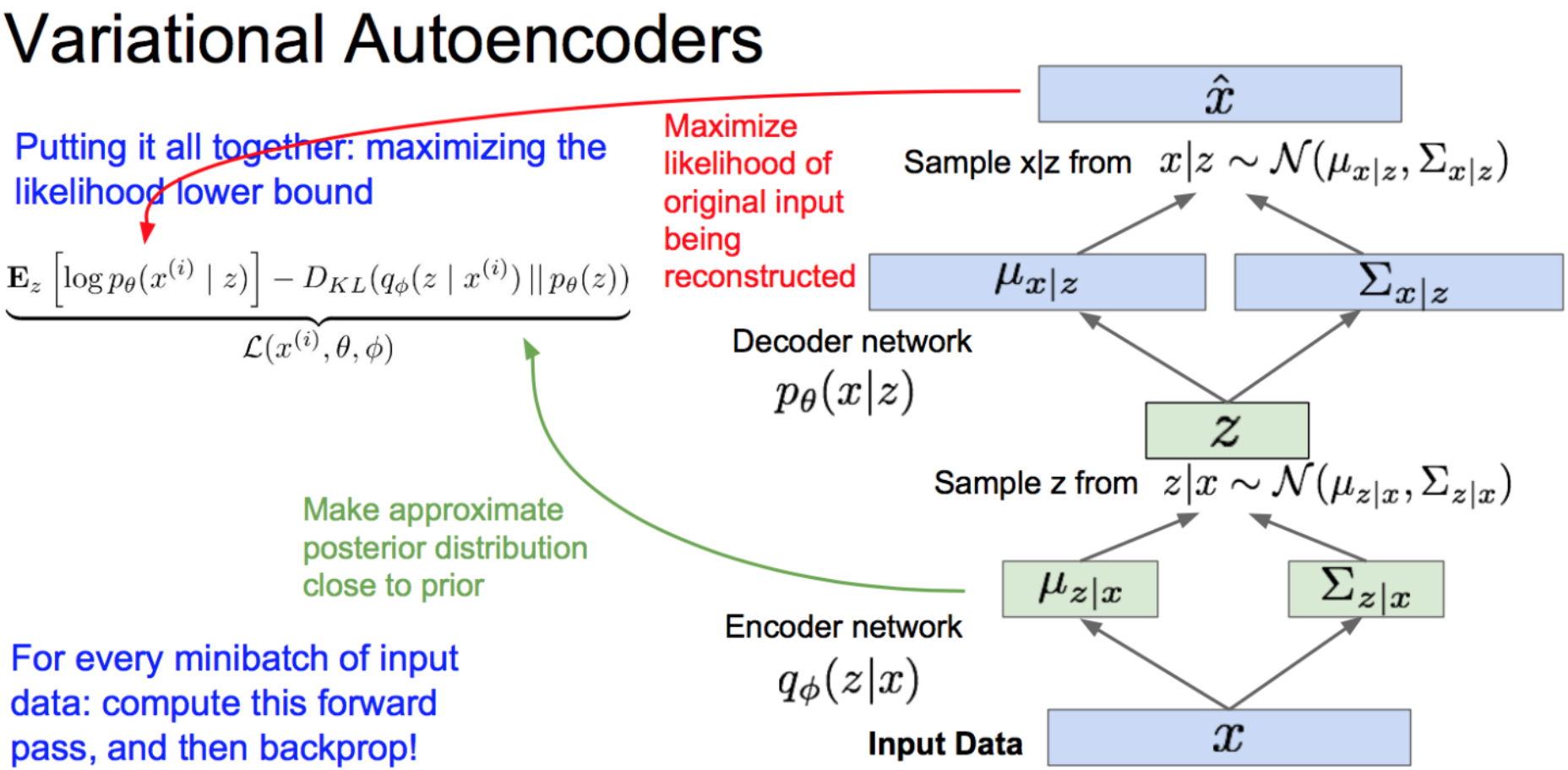
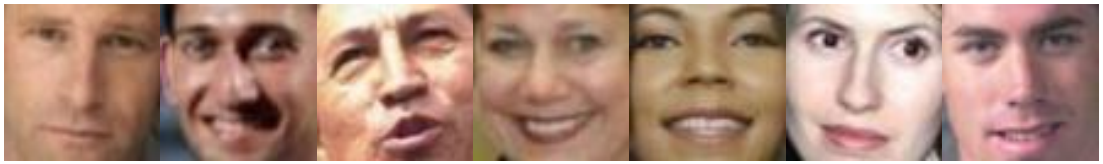


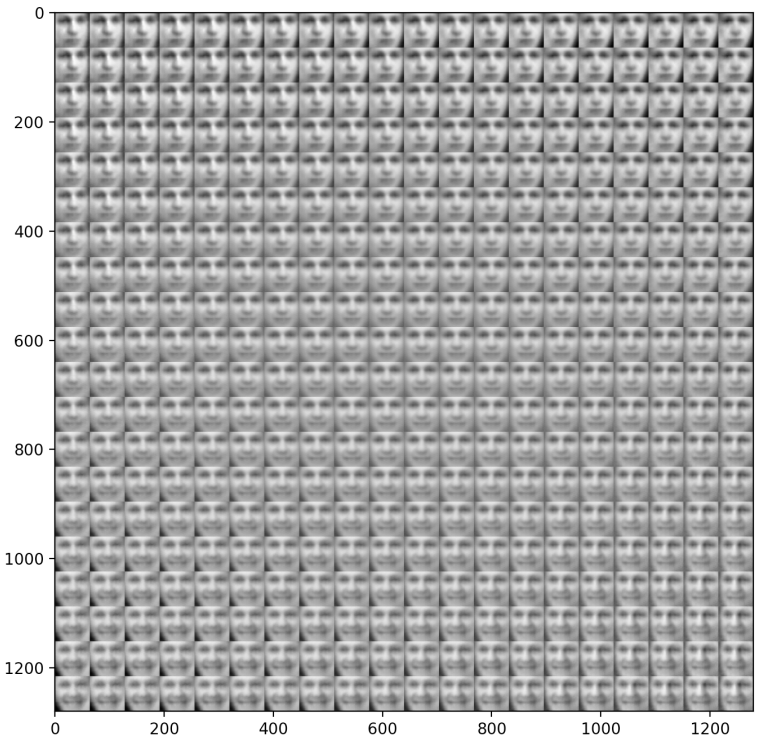
Figure: Framework of VAEs.

Variational Auto-encoders

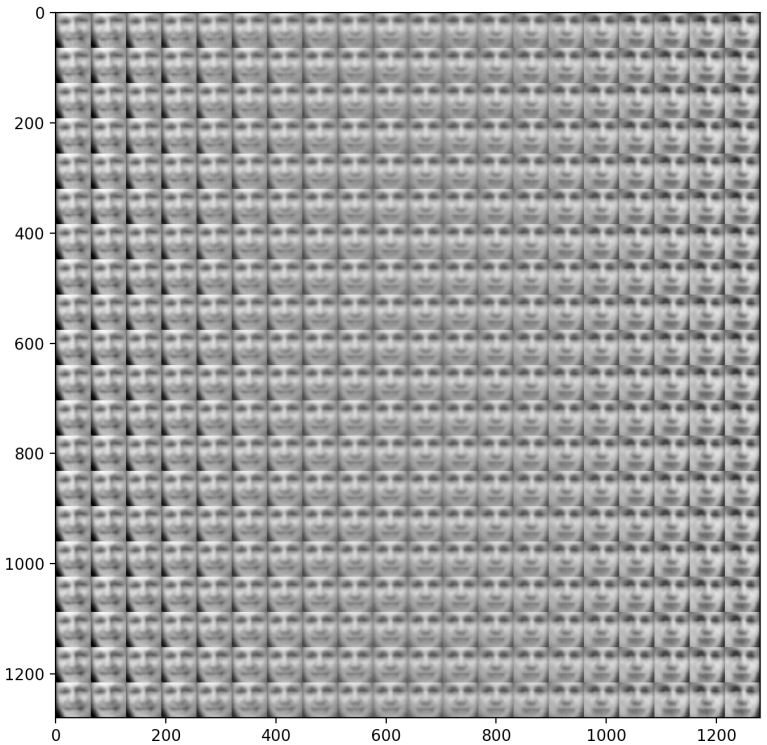
Application on Faces:



Generated faces



Generated faces



https://distill.pub/2017/aia/

Generative Adversarial Nets

Ian Goodfellow and his co-authors proposed the Generative Adversarial Nets (GANs) in 2014. They use samples from a simple distribution $z \sim p(z)$, pass them through a Neural network $x = f_\theta(z)$ to approximate an unknown distribution $q(x)$ via their samples. The Loss function is of the following form:

$$\frac{1}{m}\sum_{i=1}^{m}[\log D_\psi(\mathbf{x}^{(i)}) + \log(1 - D_\psi(G_\theta(\mathbf{z}^{(i)})))] \tag{5}$$

The results of the adversarial training are:

$$\begin{aligned}\hat{\psi} &= \arg\max_\psi \frac{1}{m}\sum_{i=1}^{m}[\log D_\psi(\mathbf{x}^{(i)}) + \log(1 - D_\psi(G_\theta(\mathbf{z}^{(i)})))] \\ \hat{\theta} &= \arg\min_\theta \frac{1}{m}\sum_{i=1}^{m}[\log(1 - D_\psi(G_\theta(\mathbf{z}^{(i)})))]\end{aligned} \tag{6}$$

Here σ is the sigmoid function. ψ are the coefficients of the discriminator, θ are the coefficients for the generator.

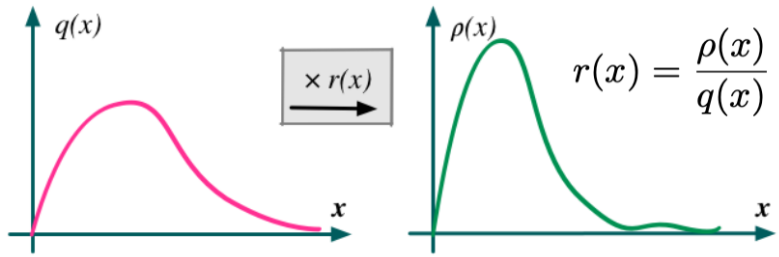
Density ratio estimation by probabilistic classification

$$\mathbb{KL}[p(\boldsymbol{x})||q(\boldsymbol{x})] := \int \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} p(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{p(\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]$$

Estimating density ratios is a challenging task:

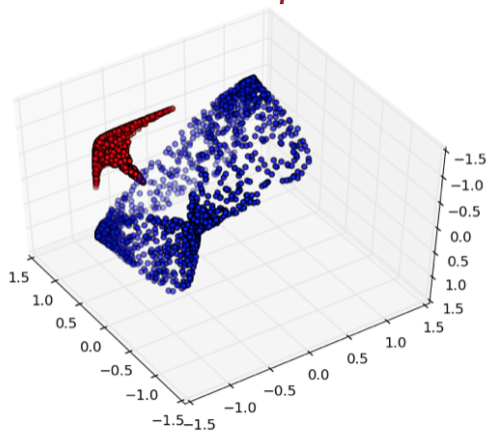
- Each part of the ratio may itself involve intractable integrals
- We often deal with high-dimensional quantities.
- We may only have samples drawn from the two distributions, not their analytical forms.

This is where the ***density ratio trick*** enters: it allows us to construct a binary classifier that distinguishes between samples from the two distributions.



The density ratio gives the correction factor needed to make two distributions equal.

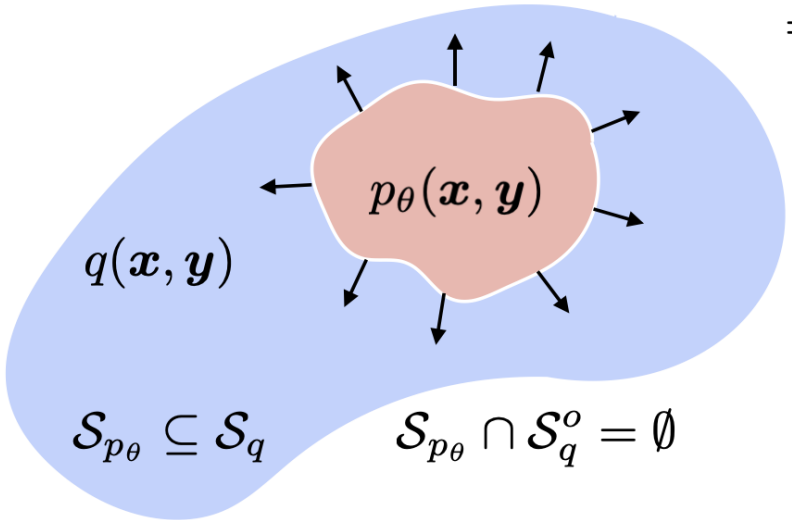
$$\begin{aligned} r(x) &= \frac{\rho(x)}{q(x)} = \frac{p(x|y=+1)}{p(x|y=-1)} \\ &= \frac{p(y=+1|x)p(x)}{p(y=+1)} \Big/ \frac{p(y=-1|x)p(x)}{p(y=-1)} \\ &= \frac{p(y=+1|x)}{p(y=-1|x)} = \frac{p(y=+1|x)}{1-p(y=+1|x)} = \frac{\mathcal{S}(x)}{1-\mathcal{S}(x)} \end{aligned}$$



Joint distribution matching

Reverse KL: $\mathbb{KL}\left[p_\theta(\boldsymbol{x},\boldsymbol{y})||q(\boldsymbol{x},\boldsymbol{y})\right] = -\mathbb{H}\left[p_\theta(\boldsymbol{x},\boldsymbol{y})\right] - \mathbb{E}_{p_\theta(\boldsymbol{x},\boldsymbol{y})}\left[\log q(\boldsymbol{x},\boldsymbol{y})\right]$

$$= -\mathbb{H}\left[p_\theta(\boldsymbol{x},\boldsymbol{y})\right] - \int_{\mathcal{S}_{p_\theta}\cap\mathcal{S}_q} \log q(\boldsymbol{x},\boldsymbol{y}) p_\theta(\boldsymbol{x},\boldsymbol{y}) d\boldsymbol{x}d\boldsymbol{y} - \int_{\mathcal{S}_{p_\theta}\cap\mathcal{S}_q^o} \log q(\boldsymbol{x},\boldsymbol{y}) p_\theta(\boldsymbol{x},\boldsymbol{y}) d\boldsymbol{x}d\boldsymbol{y}$$



$q(\boldsymbol{x},\boldsymbol{y})$: ***Observed data distribution***

$p_\theta(\boldsymbol{x},\boldsymbol{y})$: ***Generated data distribution***

$f_\theta(\boldsymbol{x},\boldsymbol{z}) \longrightarrow$ ***generator***

$T_\psi(\boldsymbol{x},\boldsymbol{y}) \longrightarrow$ ***discriminator***

$q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) \longrightarrow$ ***encoder***

$$\min_\psi -\mathbb{E}_{p(\boldsymbol{z})}[\log(T_\psi(\boldsymbol{x}, f_\theta(\boldsymbol{x},\boldsymbol{z})))] - \mathbb{E}_{q(\boldsymbol{x},\boldsymbol{y})}[\log(1 - T_\psi(\boldsymbol{x},\boldsymbol{y})]$$

$$\min_{\theta,\phi} \mathbb{E}_{p(\boldsymbol{z})}[T_\psi(\boldsymbol{x}, f_\theta(\boldsymbol{x},\boldsymbol{z}))] + (1-\lambda)\mathbb{E}_{p(\boldsymbol{z})}[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}, f_\theta(\boldsymbol{x},\boldsymbol{z}))]$$

Adversarial objective

Li, C., Li, J., Wang, G., & Carin, L. (2018). Learning to Sample with Adversarially Learned Likelihood-Ratio.

Distribution matching

Matching distribution using Generative Adversarial Net (GAN) with mode collapse. e.g.

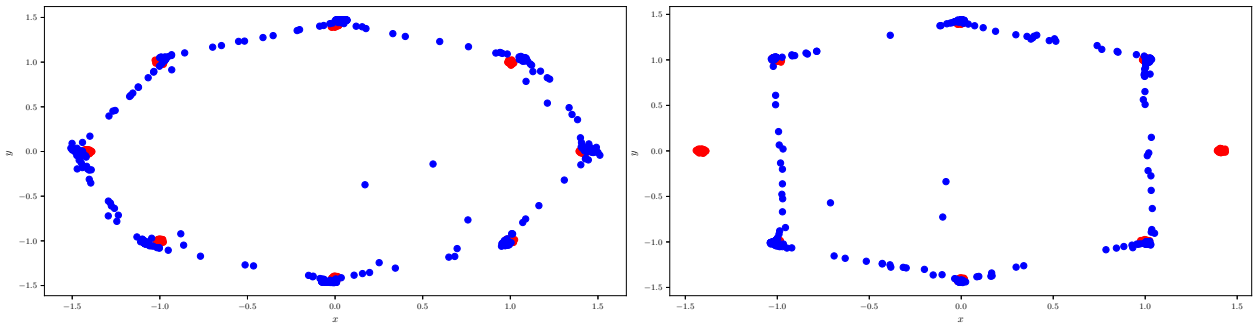


Figure: GAN to approximate 2D distribution. Left: without mode collapse, Right: with mode collapse.

Distribution matching

Matching complex functions using Generative Adversarial Net (GAN) without mode collapse. e.g.

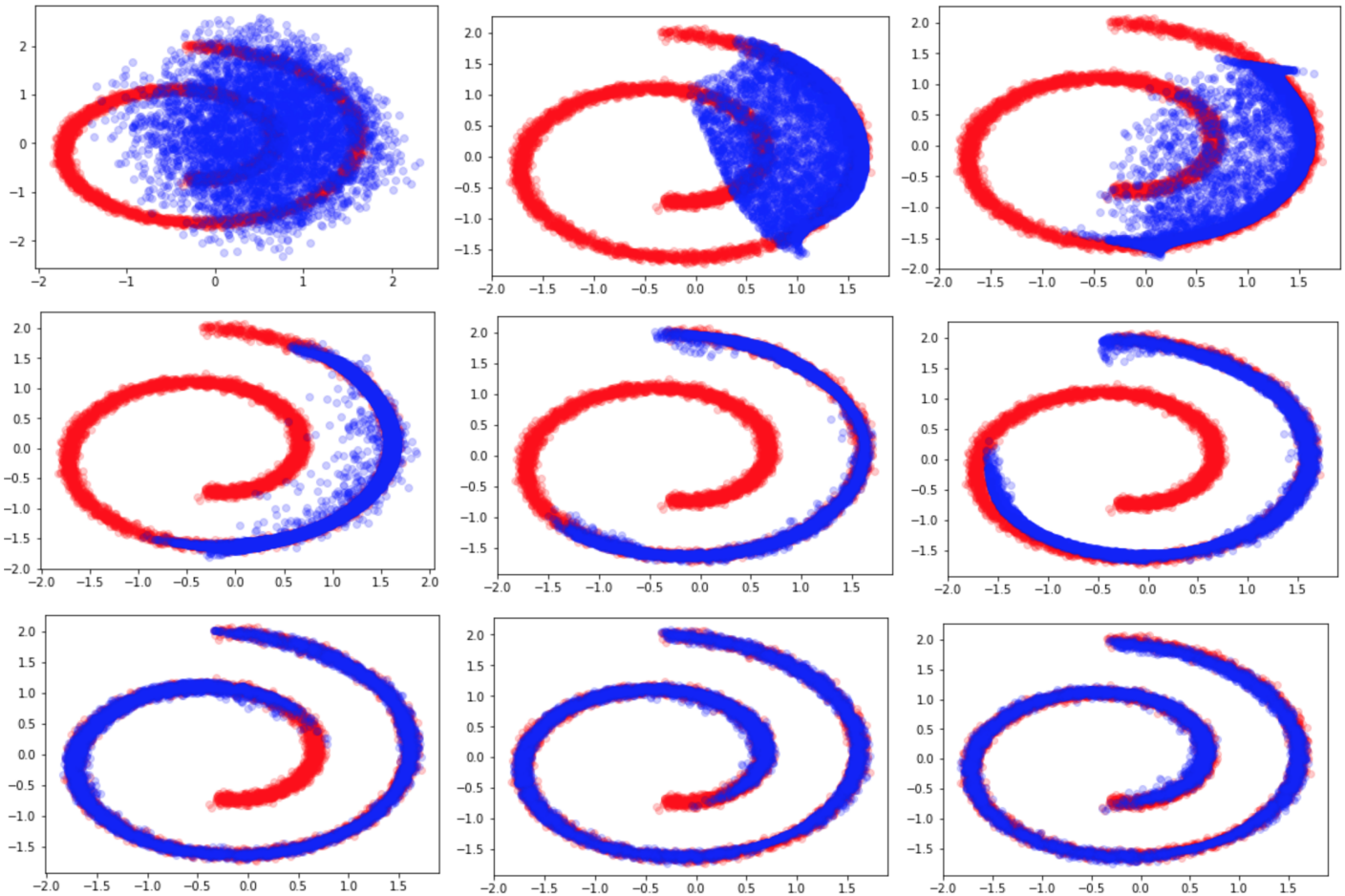


Figure: GAN to approximate 2D Swiss roll data.

Adversarial Auto-Encoder

In 2015 the Adversarial Auto-Encoder was introduced to use GANs to perform variational inference by matching the aggregated posterior of the hidden code vector of the Auto-encoder with an arbitrary prior distribution. Their network structure is:

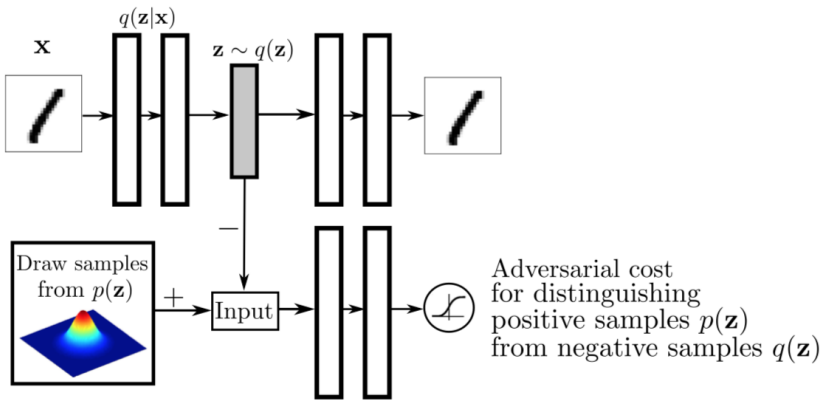


Figure: Architecture of an adversarial Auto-encoder.

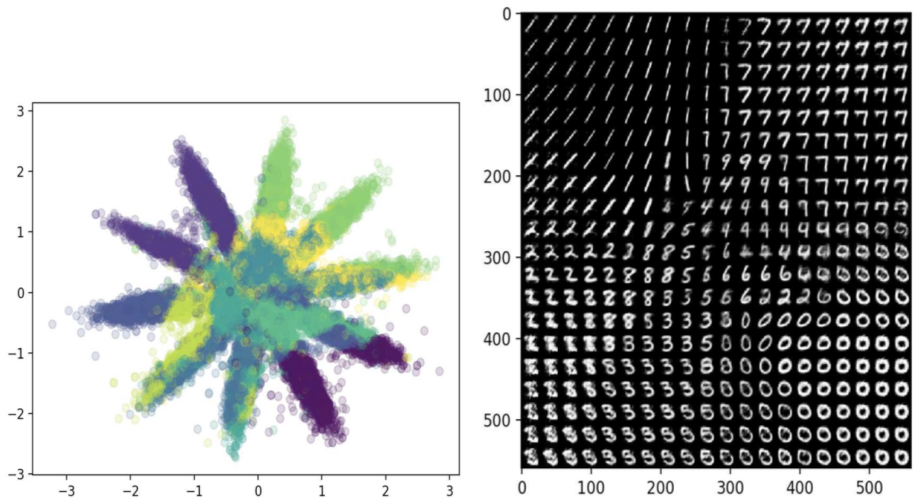


Figure: AAE application on MNIST data set.

Semi-supervised Adversarial Auto-Encoder

Regularizing the hidden code by providing a one-hot vector to the discriminative network.

Semi-supervised AAE

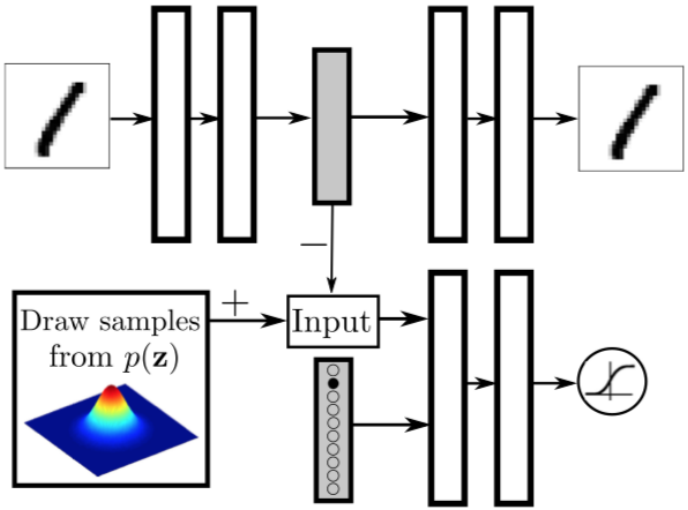


Figure: Architecture of a semi-supervised adversarial Auto-encoder.

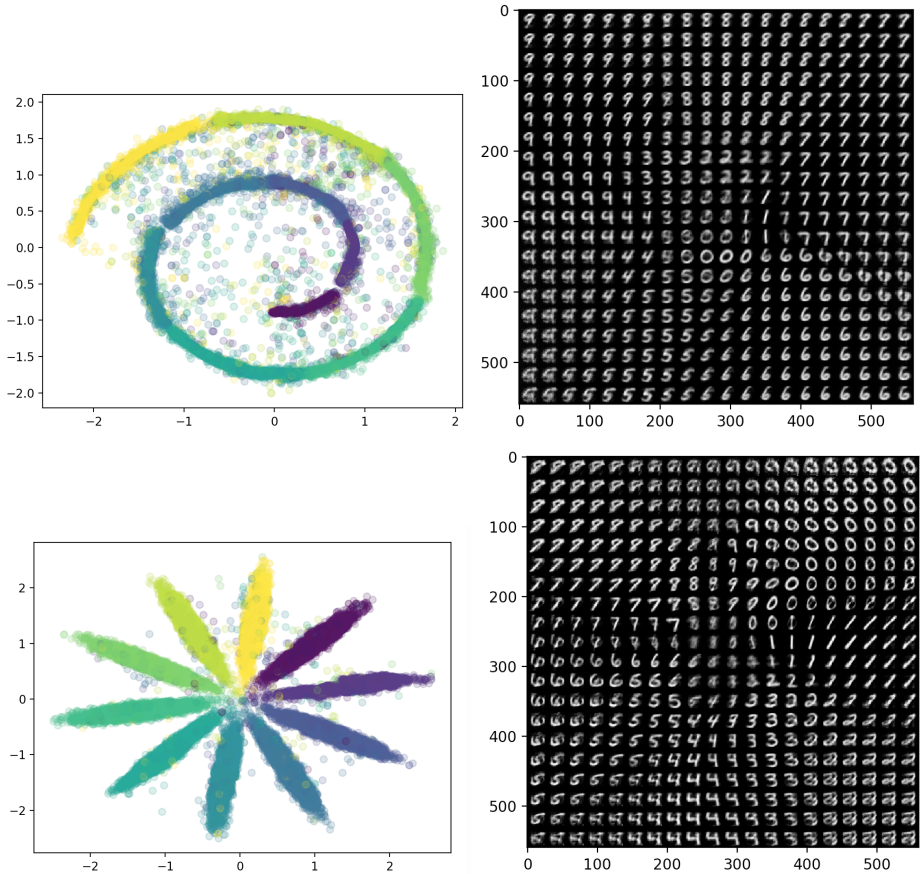


Figure: Semi-supervised AAE with Gaussian Mixture prior and Swiss roll

Understanding and Improving Interpolation in Auto-encoders

In 2018 August, a paper from Goodfellow *et. al.*, proposed a regularization procedure which encourages interpolated outputs to appear more realistic by fooling a critic network which has been trained to recover the mixing coefficient from interpolated data. A critic network tries to predict the interpolation coefficient corresponding to an interpolated datapoint. The auto-encoder is trained to fool the critic into outputting $= 0$.

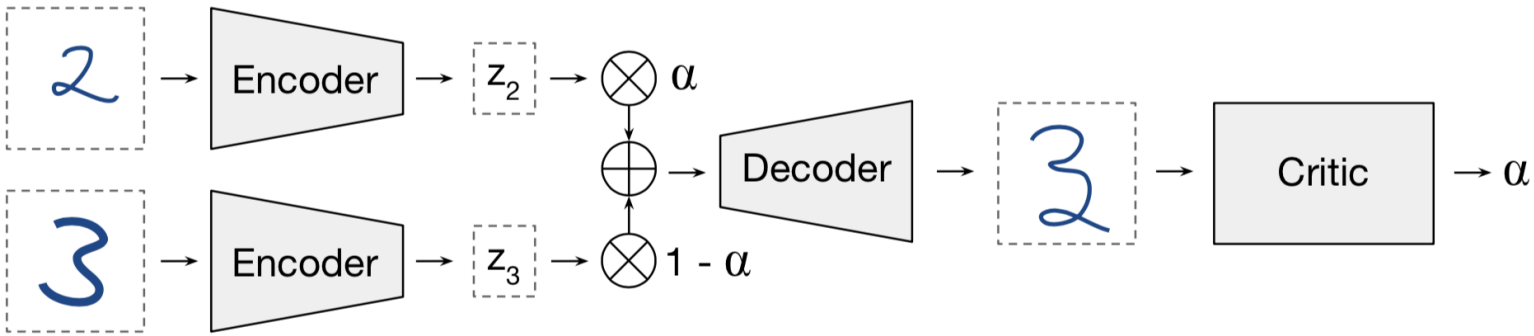


Figure: Adversarially Constrained Auto-encoder Interpolation (ACAI).

Understanding and Improving Interpolation in Auto-encoders

An input $x \in \mathbb{R}^{d_x}$ is passed through an encoder $z = f_\theta(x)$ parametrized by θ to obtain a latent code $z \in \mathbb{R}^{d_z}$. The latent code is then passed through a decoder $\hat{x} = g_\phi(z)$ parametrized by ϕ to reconstruct x. Typically, the latent codes are combined via a convex combination, so that interpolation amounts to computing $\hat{x}_\alpha = g_\phi(\alpha z_1 + (1 - \alpha z_2))$ for some $\alpha \in [0, 1]$ where $z_1 = f_\theta(x_1)$ and $z_2 = f_\theta(x_2)$ are the latent codes corresponding to data points x_1 and x_2.

$$\begin{aligned} \mathcal{L}_d &= \|d_\omega(\hat{x}_\alpha) - \alpha\|^2 + \|d_\omega(\gamma x + (1 - \gamma) g_\phi(f_\theta(x)))\|^2 \\ \mathcal{L}_{f,g} &= \|x - g_\phi(f_\theta(x))\|^2 + \lambda \|d_\omega(\hat{x}_\alpha)\|^2 \end{aligned} \tag{7}$$

The second term of $\mathcal{L}_d$ serves as a regularizer with two functions: First, it enforces that the critic consistently outputs 0 for non-interpolated inputs; and second, by interpolating between x and $g_\phi(f_\theta(x))$ in data space it ensures the critic is exposed to realistic data even when the auto-encoders reconstructions are poor.

Understanding and Improving Interpolation in Auto-encoders

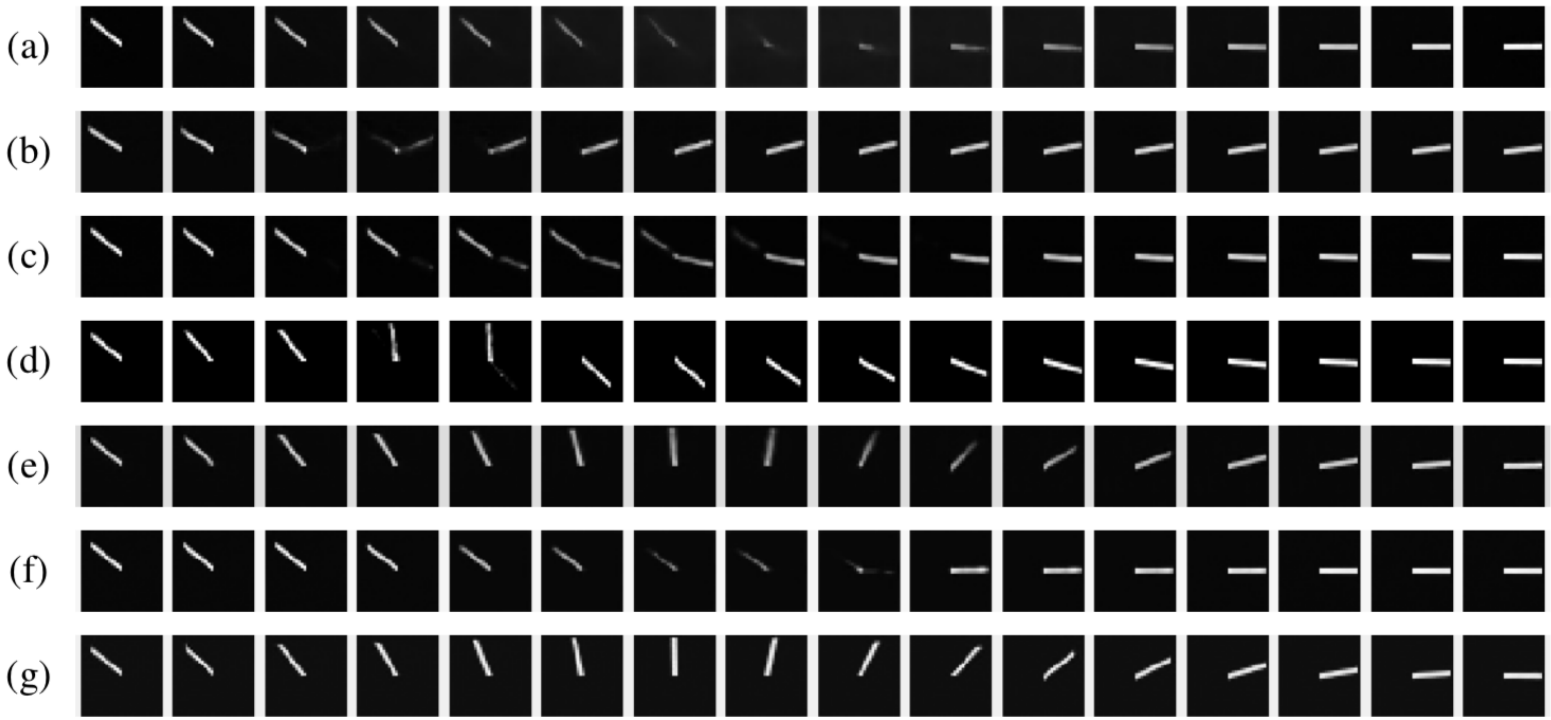


Figure: Interpolations on the synthetic lines benchmark produced by (a) baseline auto-encoder, (b) baseline with latent-space dropout, (c) denoising autoencoder, (d) Variational Autoencoder, (e) Adversarial Autoencoder, (f) Vector Quantized Variational Autoencoder, (g) Adversarially Constrained Autoencoder Interpolation.

Question on ACAI

With the interpolation expression: $\hat{x}_\alpha = g_\phi(\alpha z_1 + (1 - \alpha z_2))$. What if we want to find a way to move this clock with 2π? Which should be the most possible way to do this in the latent space?

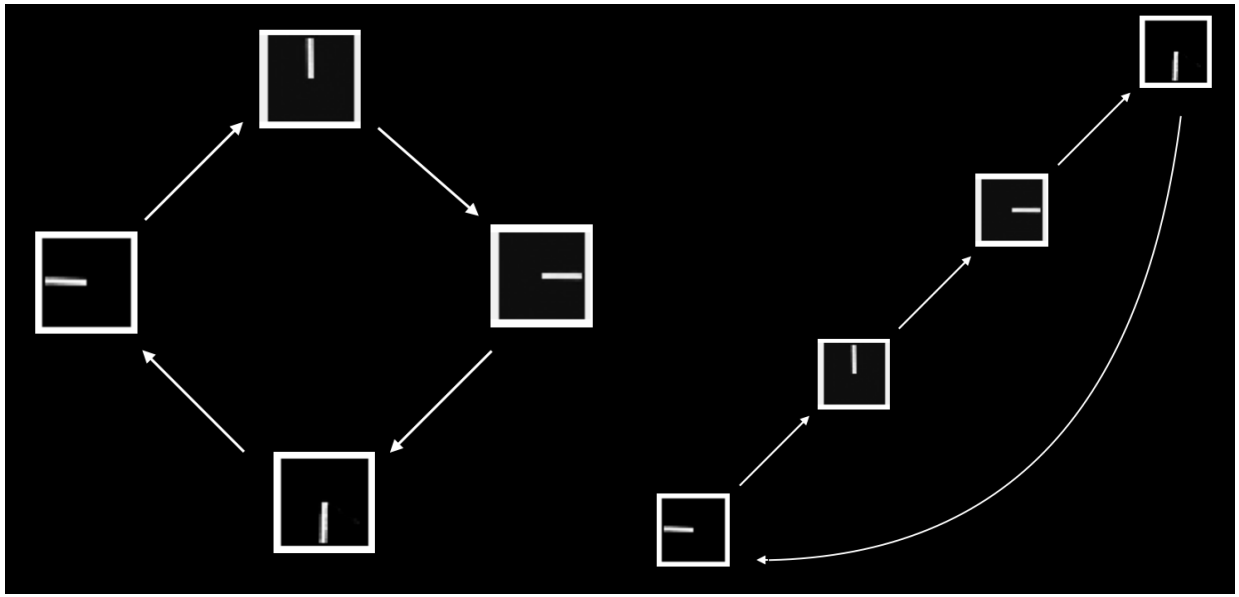


Figure: Two possible ways of Interpolation in the latent space.

Image to Image regression using latent space

Encoding an image into a compact code, and decode the code to reconstruct the image as accurately as possible. These methods have also been shown to learn good feature representations from image pixels.

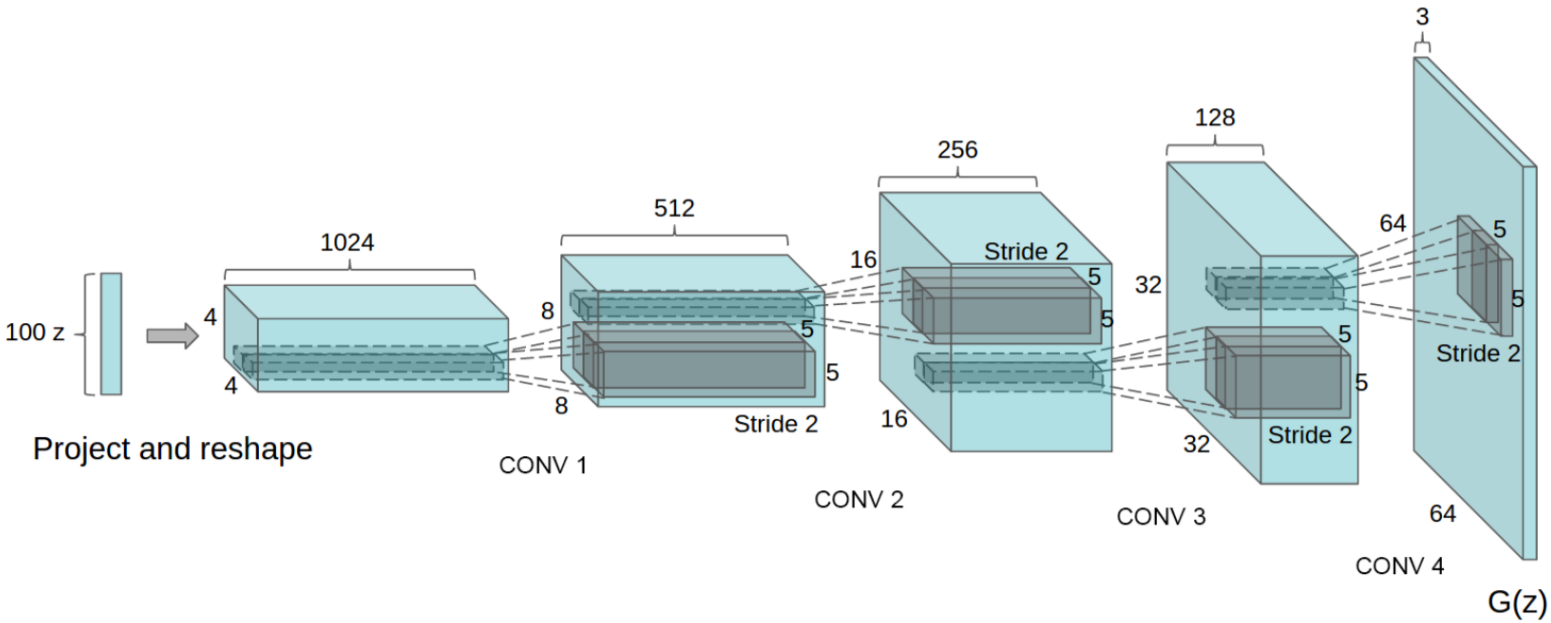


Figure: Decoding the latent variable to derive images (DCGANs).

Image to Image regression using latent space

Application on Darcy's flow problem:

Image to Image regression

Given permeability to predict the velocity field.

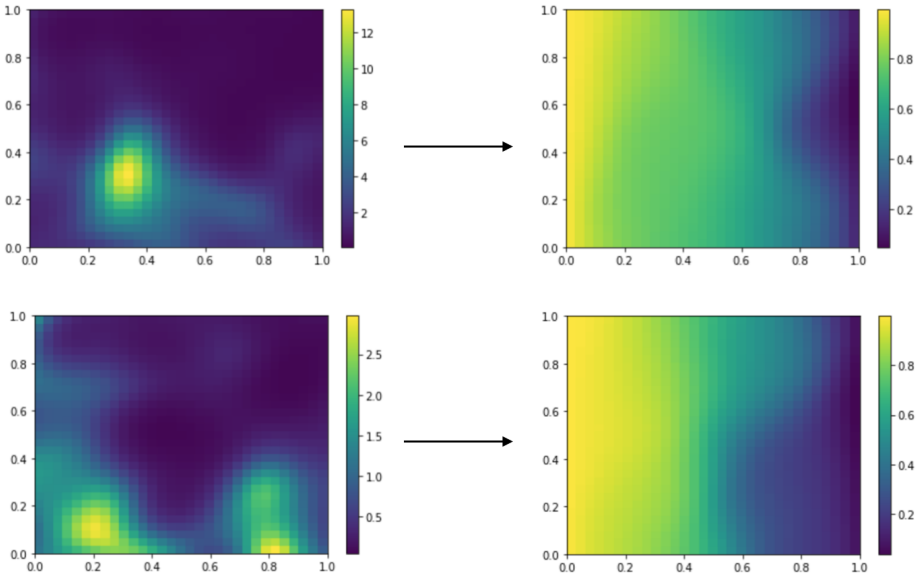


Figure: Image to Image regression on Darcy's flow.

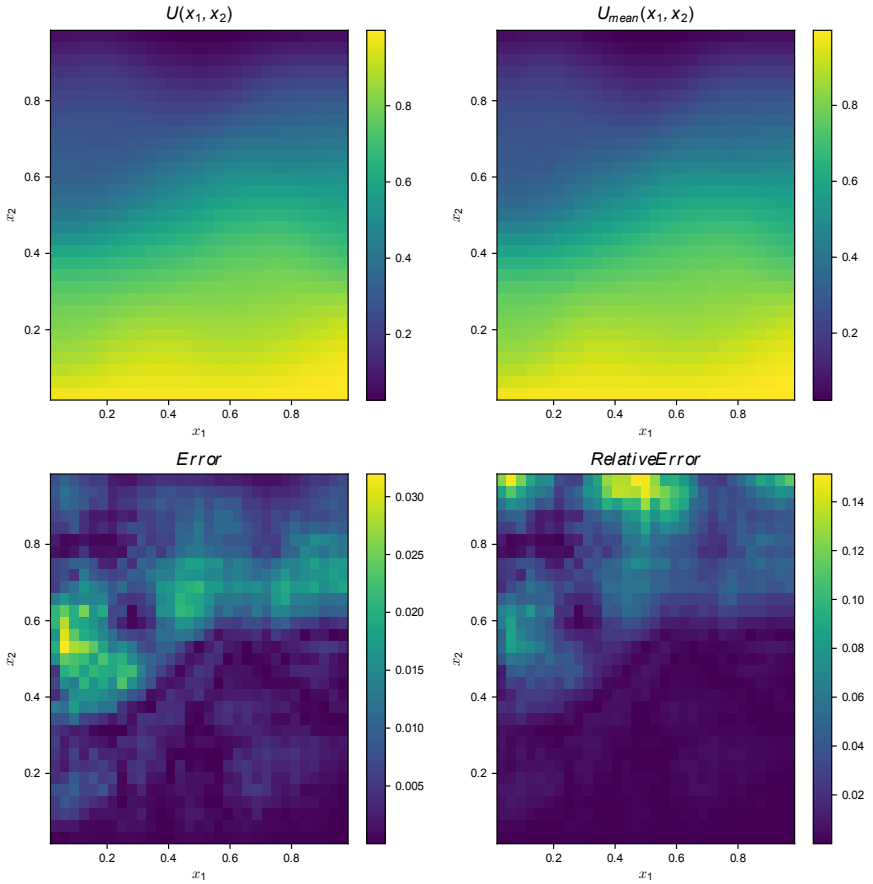


Figure: Exact, prediction, error and relative error.

Physics Informed Machine Learning

Motivation: The need of using data in scientific computing

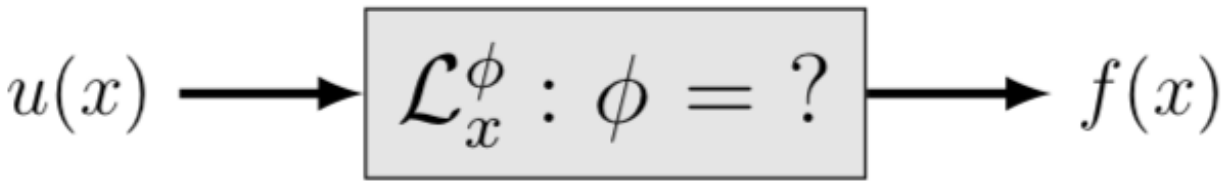
A few reasons for the need of incorporating simulation data into solution algorithms can be:

- Simulations that are close to reality are extremely computationally expensive.
- There is a lack of understanding in many of the physical phenomena and the mathematical models do not capture the complexity of physical world.
- In some scientific disciples there exist no equations for many problems.
- There is an abundance of data from experiments and simulations.

Is there a way to use these data in order to discover new PDEs, accelerate the solution of simulations and find more cost effective ways to calculate parameters?

Problem of data-driven discovery of differential equations

Given noisy observations $\{\mathbf{X}_u, \mathbf{y}_u\}$, $\{\mathbf{X}_f, \mathbf{y}_f\}$ of $u(x)$ and $f(x)$, respectively, the aim is to learn the parameters φ and hence the governing equation which best describes the data. Such problems are ubiquitous in science, and in the mathematical literature are often referred to as inverse problems.



e.g. one always want to model the relationship between two black-box functions u(x) and f(x).

$$f(x) = \mathcal{L}_x^\phi u(x) \tag{8}$$

Data-driven discovery of partial differential equations: Overview

Approaches of data-driven discovery of differential equations

- $\mathcal{F}_\lambda$: Parametrize $\mathcal{F}_\lambda$ using a dictionary of terms
  $\mathcal{F}_\lambda := c_0 \frac{\partial}{\partial t} + c_1 \frac{\partial}{\partial x} + c_3 \frac{\partial^2}{\partial x^2} + c_4 u \frac{\partial}{\partial x} + \cdots$
  Methods used:
  - GPs
  - Neural Networks
  - Sparse Regression (Brunton and Kutz)
- Learn directly the operator from the data:
  $\frac{du}{dt} = \mathcal{F}(x, t, u, u_x, \cdots)$
  In this case we do not parametrize $\mathcal{F}_\lambda$, but learn it directly from the data.
  Methods used:
  - GPs
  - Neural Networks
  - NARMAX

Non-deep approaches: Gaussian Process

Gaussian Process is a powerful tool of using probabilistic model to solve linear differential equations. The algorithm starts by making the assumption that u(x) is Gaussian process:

$$u(x) \sim \mathcal{GP}(0, k_{uu}(x, x'; \theta)) \tag{9}$$

As any linear transformation of a Gaussian process such as differentiation and integration is still a Gaussian process, we have:

$$\mathcal{L}_x^{\phi} u(x) = f(x) \sim \mathcal{GP}(0, k_{ff}(x, x'; \theta)) \tag{10}$$

where the kernel follows the relationship:

$$\begin{aligned} k_{ff}(x, x'; \theta, \phi) &= \mathcal{L}_x^{\phi} \mathcal{L}_{x'}^{\phi} k_{uu}(x, x'; \theta) \\ k_{fu}(x, x'; \theta, \phi) &= \mathcal{L}_x^{\phi} k_{uu}(x, x'; \theta;) \\ k_{uf}(x, x'; \theta, \phi) &= \mathcal{L}_{x'}^{\phi} k_{uu}(x, x'; \theta;) \end{aligned} \tag{11}$$

Non-deep approaches: Gaussian Process

Gaussian process priors used in this work are assumed to have a squared exponential covariance function:

$$k_{uu}(x, x'; \theta) = \sigma_u^2 \exp(-\frac{1}{2} \sum_{d=1}^{D} w_d (x_d - x'_d)^2) \tag{12}$$

By minimizing the negative log marginal likelihood:

$$\mathcal{NLML}(\phi, \theta, \sigma_{n_u}^2, \sigma_{n_f}^2) = -\log p(\mathbf{y}|\phi, \theta, \sigma_{n_u}^2, \sigma_{n_f}^2) \tag{13}$$

where θ denotes the hyper-parameters of the kernel k_{uu} and ϕ represent the parameters of the linear operator $\mathcal{L}_x^\phi$, e.g:
$\mathcal{L}_x^\phi u(x) := \frac{d}{dx} u(x) + \alpha u(x) + \beta \int_0^x u(\xi) d\xi$, $\phi = [\alpha, \beta]$.

Non-deep approaches: Gaussian Process

In the above equation $\mathbf{y} = \begin{bmatrix} \mathbf{y_u} \\ \mathbf{y_f} \end{bmatrix}$, $p(\mathbf{y}|\phi, \theta, \sigma_{n_u}^2, \sigma_{n_f}^2) = \mathcal{N}(\mathbf{0}, \mathbf{K})$ and $\mathbf{K}$ is given by

$$\mathbf{K} = \begin{bmatrix} k_{uu}(\mathbf{X_u}, \mathbf{X_u}; \theta) + \sigma_{n_u}^2 \mathbf{I}_{n_u} & k_{uf}(\mathbf{X_u}, \mathbf{X_f}; \theta, \phi) \\ k_{fu}(\mathbf{X_f}, \mathbf{X_u}; \theta, \phi) & k_{ff}(\mathbf{X_u}, \mathbf{X_f}; \theta, \phi) + \sigma_{n_f}^2 \mathbf{I}_{n_f} \end{bmatrix} \tag{14}$$

For making predictions

$$\begin{aligned} p(u(x)|\mathbf{y}) &= \mathcal{N}(\overline{u}(x), s_u^2(x)), \\ p(f(x)|\mathbf{y}) &= \mathcal{N}(\overline{f}(x), s_f^2(x)), \end{aligned} \tag{15}$$

with

$$\begin{aligned} \overline{u}(x) &= \mathbf{q}_u^T \mathbf{K}^{-1} \mathbf{y}, s_u^2(x) = k_{uu}(x,x) - \mathbf{q}_u^T \mathbf{K}^{-1} \mathbf{q}_u, \mathbf{q}_u^T = [k_{uu}(x, \mathbf{X}_u k_{uf}(x, \mathbf{X}_f)], \\ \overline{f}(x) &= \mathbf{q}_u^T \mathbf{K}^{-1} \mathbf{y}, s_f^2(x) = k_{ff}(x,x) - \mathbf{q}_f^T \mathbf{K}^{-1} \mathbf{q}_f, \mathbf{q}_f^T = [k_{fu}(x, \mathbf{X}_u k_{ff}(x, \mathbf{X}_f)], \end{aligned} \tag{16}$$

Non-deep approaches: Gaussian Process

Heat equation: $\mathcal{L}^{\alpha}_{(t,x)} u(t,x) := \frac{\partial}{\partial t} u(t,x) - \alpha \frac{\partial^2}{\partial x^2} u(t,x) = f(t,x)$
For

$$\begin{aligned} \alpha &= 1 \\ f(t,x) &= e^{-t}(4\pi^2 - 1)sin(2\pi x) \\ u(t,x) &= e^{-t} sin(2\pi x) \end{aligned} \tag{17}$$

The equation is satisfied.By randomly choosing $n_u = n_f = 20$ data points, we get:

Non-deep approaches: Gaussian Process

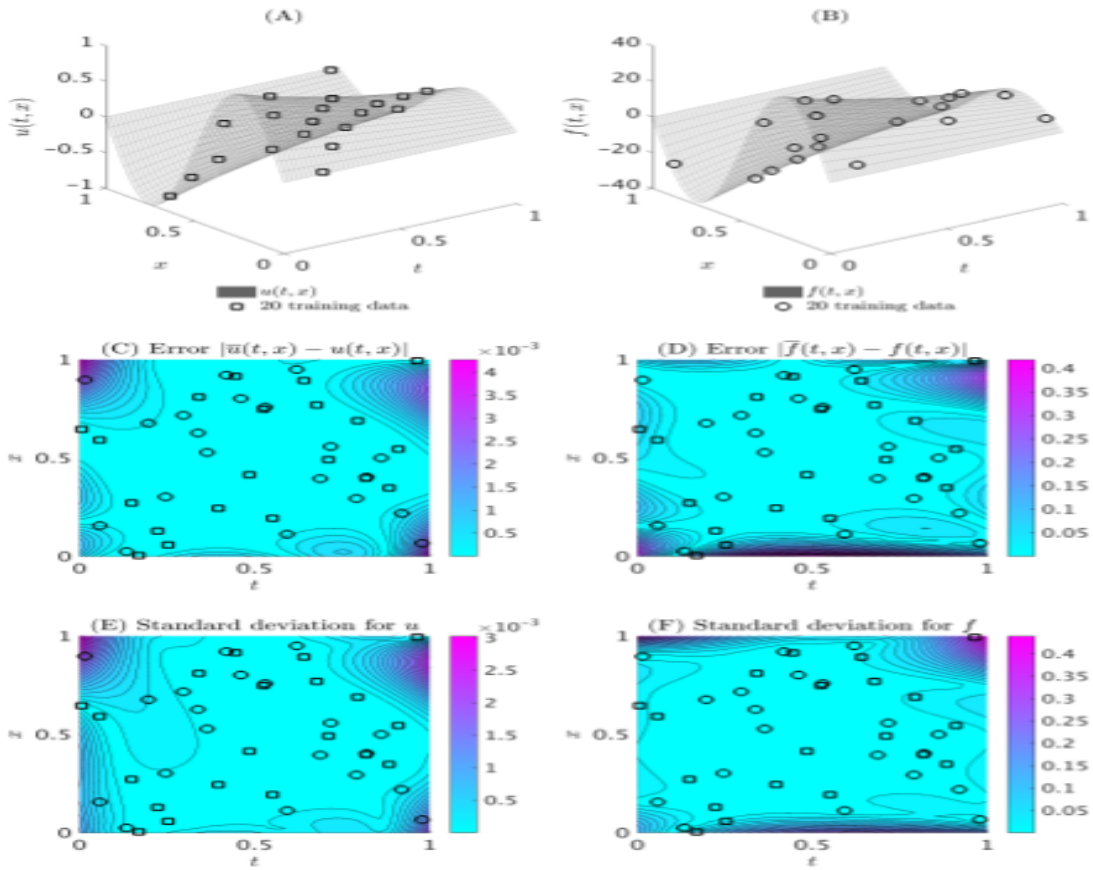


Figure: (A) Exact left hand side function $u(t,x)$ of the equation and training data. (B) Exact right hand side function $f(t,x)$ of the equation and training data.(C),(D) Absolute pointwise error between predictive and exact functions. (E), (F) Standard deviations of u and f respectively.

Non-deep approaches: Gaussian Process

Advantages of Gaussian Processes:

- They need a few data points to produce an accurate result.
- The uncertainty quatification is embeded to the method itself.
- They have a robust mathematical background

Disadvantages of Gaussian Processes:

- The computational cost to invert $\mathbf{K}$ is very high and it increases cubically to the amount of data points.
- They are a natural choice for linear systems but cannot be used for non-linear systems without a linearization procedure.

Deep learning for PDEs: Physics Informed Neural Networks

We can use Neural Networks instead of Gaussian Processes for discovering equations. At first we will explain how to use Neural Networks to solve PDEs.
Consider differential equations of the following form:

$$u_t + \mathcal{N}[u] = 0, \quad x \in \Omega, \quad t \in [0, T], \tag{18}$$

where $u(x, t)$ is the solution of the PDE. $\mathcal{N}[\cdot]$ denotes a nonlinear differential operator. $\Omega \in \mathbf{R}^D$ is the spacial domain where the equation holds.

Deep learning for PDEs: Physics Informed Neural Networks

In the physics informed neural network, the authors introduce a regularization mechanism (residual) of the PDE as:

$$f := u_t + \mathcal{N}[u] \tag{19}$$

Then, they propose the regularization function of the PDEs:

$$\mathrm{MSE} = \mathrm{MSE}_u + \mathrm{MSE}_f \tag{20}$$

where

$$\mathrm{MSE}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2, \tag{21}$$

and

$$\mathrm{MSE}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |u(t_u^i, x_f^i)|^2, \tag{22}$$

Where, $\{t_u^i, \mathbf{x}_u^i, u^i\}_{i=1}^{N_u}$ are training data corresponding to the initial conditions and boundary conditions. $\{t_f^i, \mathbf{x}_f^i\}_{i=1}^{N_f}$ are the collocations points within the computational domain.

Deep learning for PDEs: Physics Informed Neural Networks

Consider the Burgers Equations as follows:

$$
\begin{aligned}
&u_t + uu_x - (0.01/\pi)u_{xx} = 0, \\
&u(0, x) = -\sin(\pi x), \\
&u(t, -1) = u(t, 1) = 0.
\end{aligned} \tag{23}
$$

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

We train a neural network to approximate the solution u and residual f:

$$
\begin{aligned}
\hat{u}(x, t; \theta) &= f_\theta(x, t); \\
\hat{f}(x, t; \theta) &= \frac{\partial}{\partial t} f_\theta(x, t) \\
&\quad + f_\theta(x, t) \frac{\partial}{\partial x} f_\theta(x, t) \\
&\quad - (\frac{0.01}{\pi}) \frac{\partial^2}{\partial x^2} f_\theta(x, t) \\
&= f_\theta(x, t)
\end{aligned} \tag{24}
$$

Deep learning for PDEs: Physics Informed Neural Networks

We choose 50 points at the initial value, 100 points on the boundary condition and 10000 points in the computational domain as collocation points. So, we have $N_u = 100$ and $N_f = 10000$. The results are below:

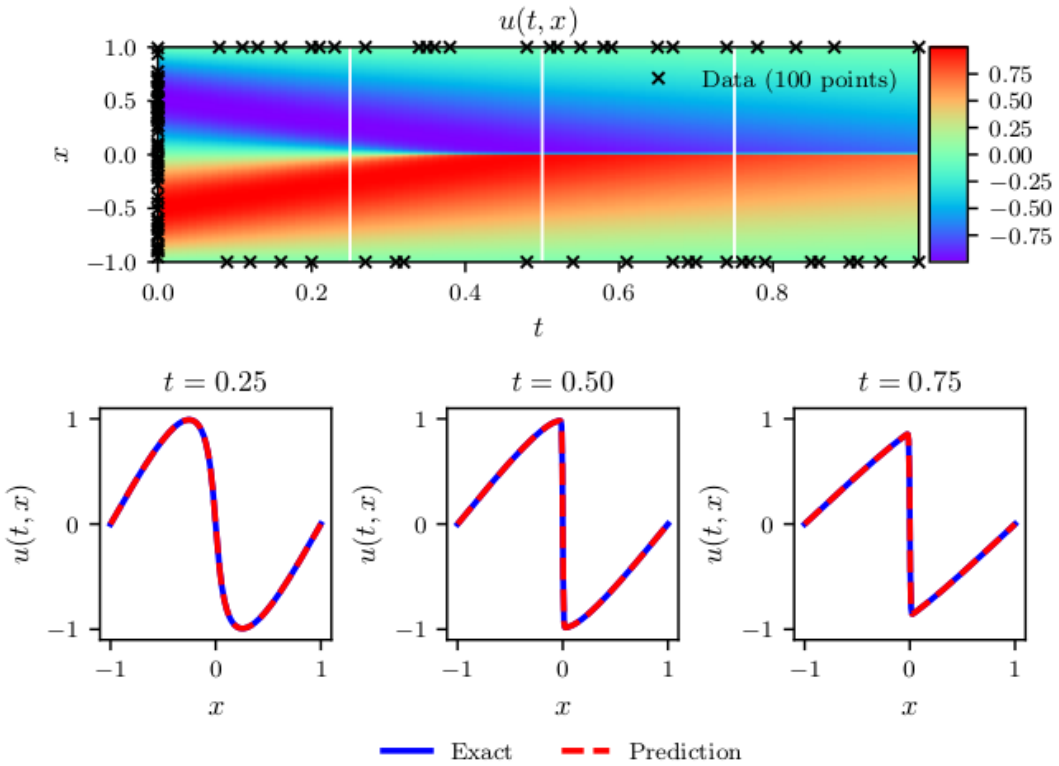


Figure: PINN on Burgers equations.

Deep learning for PDEs: Data-driven discovery

In the physics informed neural network original work, the authors introduce the residual of the PDE which depends on some parameter λ

$$f := u_t + \mathcal{N}[u; \lambda] \tag{25}$$

The funciton used for training remains:

$$\text{MSE} = \text{MSE}_u + \text{MSE}_f \tag{26}$$

where

$$\text{MSE}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2, \tag{27}$$

and

$$\text{MSE}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |u(t_u^i, x_f^i)|^2, \tag{28}$$

Where, $\{t_u^i, \mathbf{x}_u^i, u^i\}_{i=1}^{N_u}$ are training data corresponding to the initial conditions and boundary conditions. $\{t_f^i, \mathbf{x}_f^i\}_{i=1}^{N_f}$ are the collocations points within the computational domain.

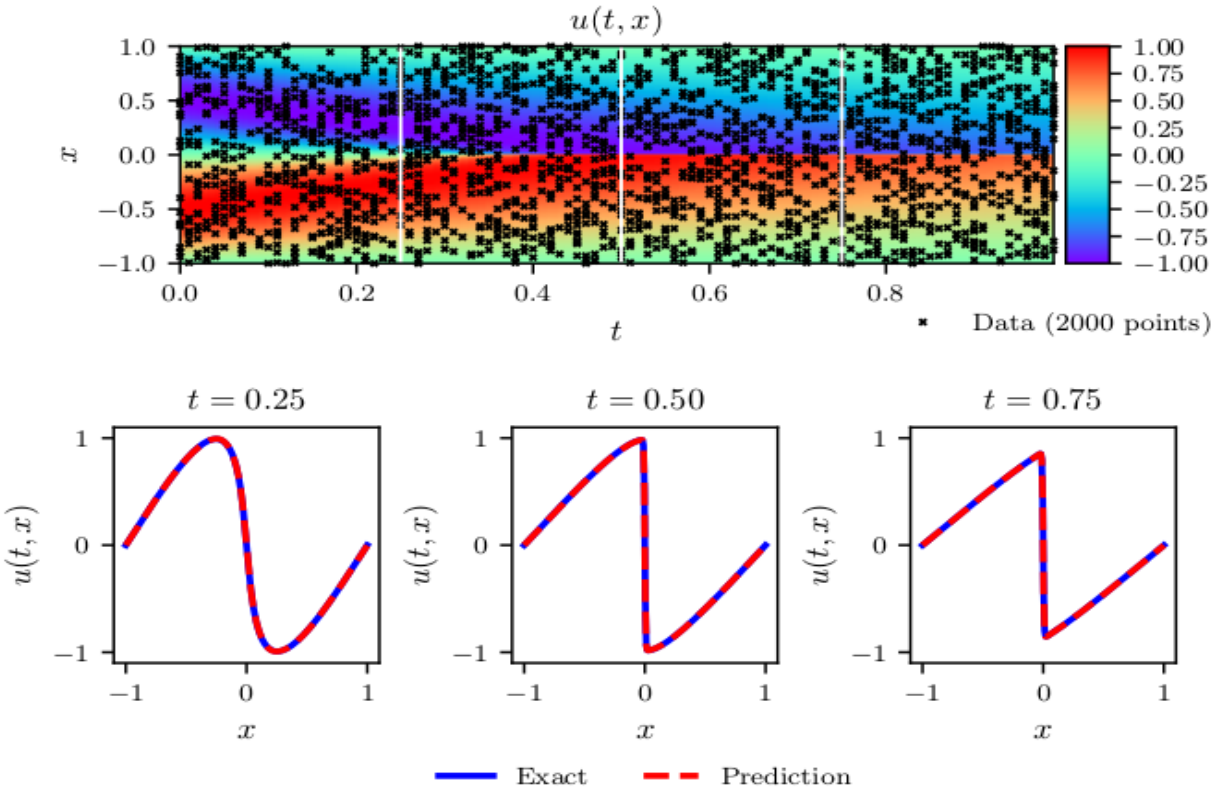
Deep learning for PDEs: Data-driven discovery

The Burgers' equation for the discovery case has the form:

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx} \tag{29}$$

and the unknown quatities are λ_1 and λ_2.
We are using information derived by simulations (2000 points, $\lambda_1 = 1.0$ and $\lambda_2 = 0.01/\pi$) and train the model in order to satisfy this equation.
When the model is trained, we have found the values of these parameters.

Deep learning for PDEs: Data-driven discovery



Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.99915uu_x - 0.0031794u_{xx} = 0$
Identified PDE (1% noise)	$u_t + 1.00042uu_x - 0.0032098u_{xx} = 0$

Deep learning for PDEs: Navier Stokes equation

In this case we will present the case of data-driven discovery of incompressible Navier-Stokes equation:

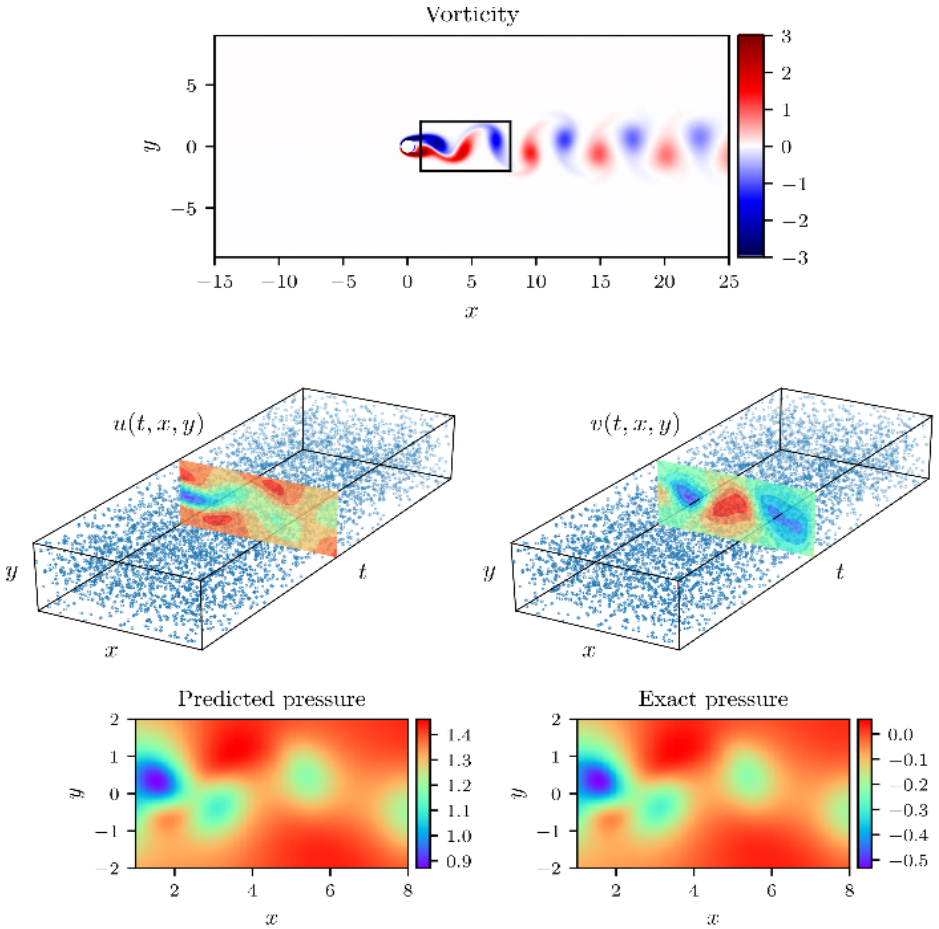
$$\begin{aligned} &\nu_x + \nu_y = 0, \\ &f := u_t + \lambda_1(uu_x + \nu u_y) + p_x - \lambda_2(u_{xx} + u_{yy}), \\ &g := \nu_t + \lambda_1(u\nu_x + \nu\nu_y) + p_y - \lambda_2(\nu_{xx} + \nu_{yy}) \end{aligned} \tag{30}$$

with

$$\begin{aligned} \mathrm{MSE} = &\frac{1}{N}\sum_{i=1}^{N_f}(|u(t^i, x^i, y^i) - u^i|^2 + |\nu(t^i x^i, y^i) - \nu^i|^2) \\ &+ \frac{1}{N}\sum_{i=1}^{N_f}(|f(t^i, x^i, y^i)|^2 + |g(t^i x^i, y^i)|^2) \end{aligned} \tag{31}$$

The input data for the training procedure is produced by spectral/hp-element solver NekTar and the training is conducted using 5000 simulation points and 9 layers, 20 neurons feed-forward Neural Network.

Deep learning for PDEs: Navier Stokes equation



Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y - 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999(uu_x + vu_y) = -p_x + 0.01047(u_{xx} + u_{yy})$ $v_t + 0.999(uv_x + vv_y) = -p_y + 0.01047(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Deep learning for PDEs: Stochastic particle diffusion

Particle density is governed by a diffusion equation of the type:

$$f = \frac{\partial \rho}{\partial t} - \frac{\partial}{\partial x}(\mathbf{K}(\rho)\frac{\partial \rho}{\partial x}) \tag{32}$$

In this case we want to discover the form of the function $\mathbf{K}(\rho)$ for different realizations with different averaging.

Deep learning for PDEs: Stochastic particle diffusion

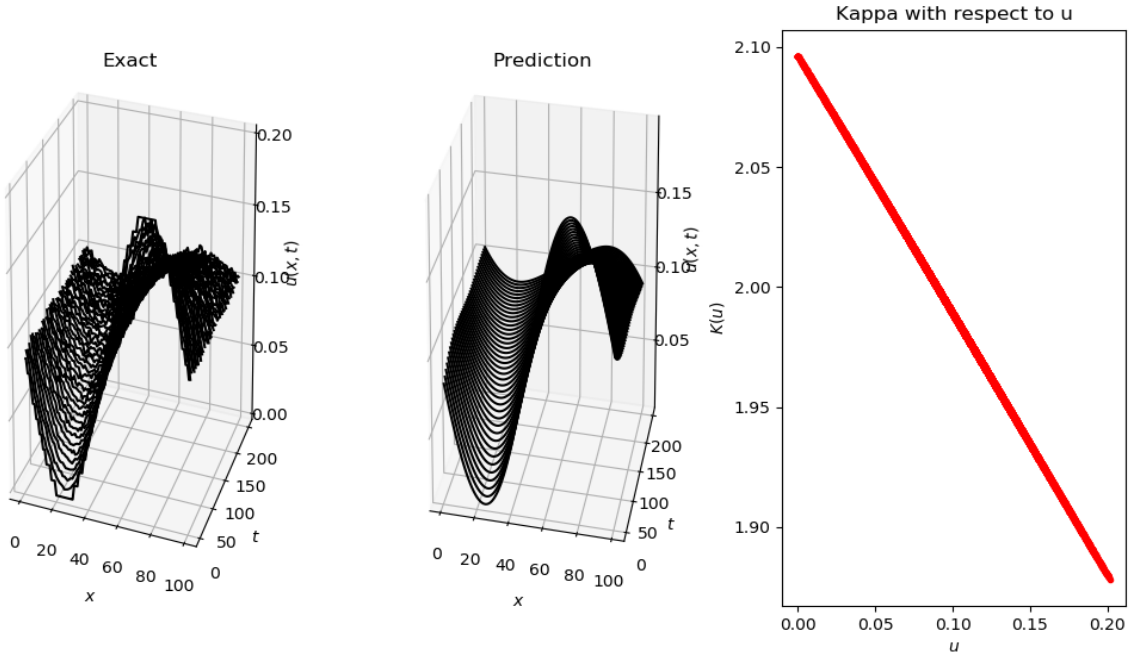


Figure: The plot on the left is the exact solution of the density of the system, in the middle is the predicted solution and on the right are the predicted values of parameter **K** with respect to density.

Deep learning for PDEs: Discovery in the latent space

This method can be categorized as a discretization-free method which provides certain advantages in manipulating the solution procedure of the PDE.
Meaning, we are solving the Burgers' equation using a PINN/Conditional-Autoencoder scheme, while constraining the latent space to satisfy a PDE of the form:

$$f := z_t + \lambda_1 z + \lambda_2 z_x + \lambda_3 z_{xx} \tag{33}$$

We are constraining the latent space using the whole domain information and training the model using a part of the domain. When the model is trained we pass the remaining collocation points through the decoder to solve the equation.

Deep learning for PDEs: Discovery in the latent space

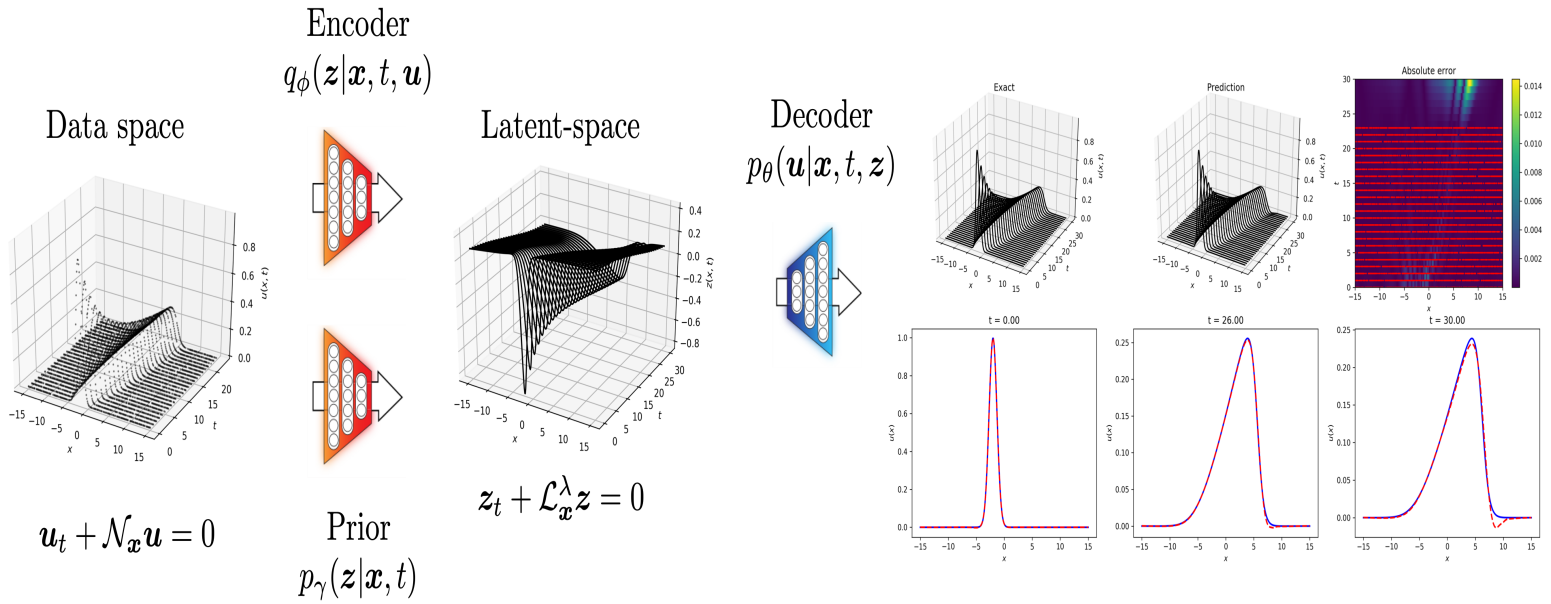


Figure: On the far left we see the exact solution of the PDE and the latent space constraint. On the right we see absolute error combined with the discretization points and the predicted solution.

References


Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders." arXiv preprint arXiv:1511.05644 (2015).


Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven solutions of nonlinear partial differential equations." arXiv preprint arXiv:1711.10561 (2017).


Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.


Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).


LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436.


Doersch, Carl. "Tutorial on variational autoencoders." arXiv preprint arXiv:1606.05908 (2016).


Li, Chunyuan, et al. "Learning to Sample with Adversarially Learned Likelihood-Ratio." (2018).


Sohn, Kihyuk, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models." Advances in Neural Information Processing Systems. 2015.

References


Berthelot, David, et al. "Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer." arXiv preprint arXiv:1807.07543 (2018).


Zhu, Yinhao, and Nicholas Zabaras. "Bayesian deep convolutional encoderdecoder networks for surrogate modeling and uncertainty quantification." Journal of Computational Physics 366 (2018): 415-447.


Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).


Ma, Chao, Yingzhen Li, and Jos Miguel Hernndez-Lobato. "Variational Implicit Processes." arXiv preprint arXiv:1806.02390 (2018).


Perdikaris, Paris, and George Em Karniadakis. "Fractional-order viscoelasticity in one-dimensional blood flow models." Annals of biomedical engineering 42.5 (2014): 1012-1023.


Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. "Machine learning of linear differential equations using Gaussian processes." Journal of Computational Physics 348 (2017): 683-693.


Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (Part II): data-driven discovery of nonlinear partial differential equations." arXiv preprint arXiv:1711.10566 (2017).