ENM 5310: Data-driven Modeling and Probabilistic Scientific Computing

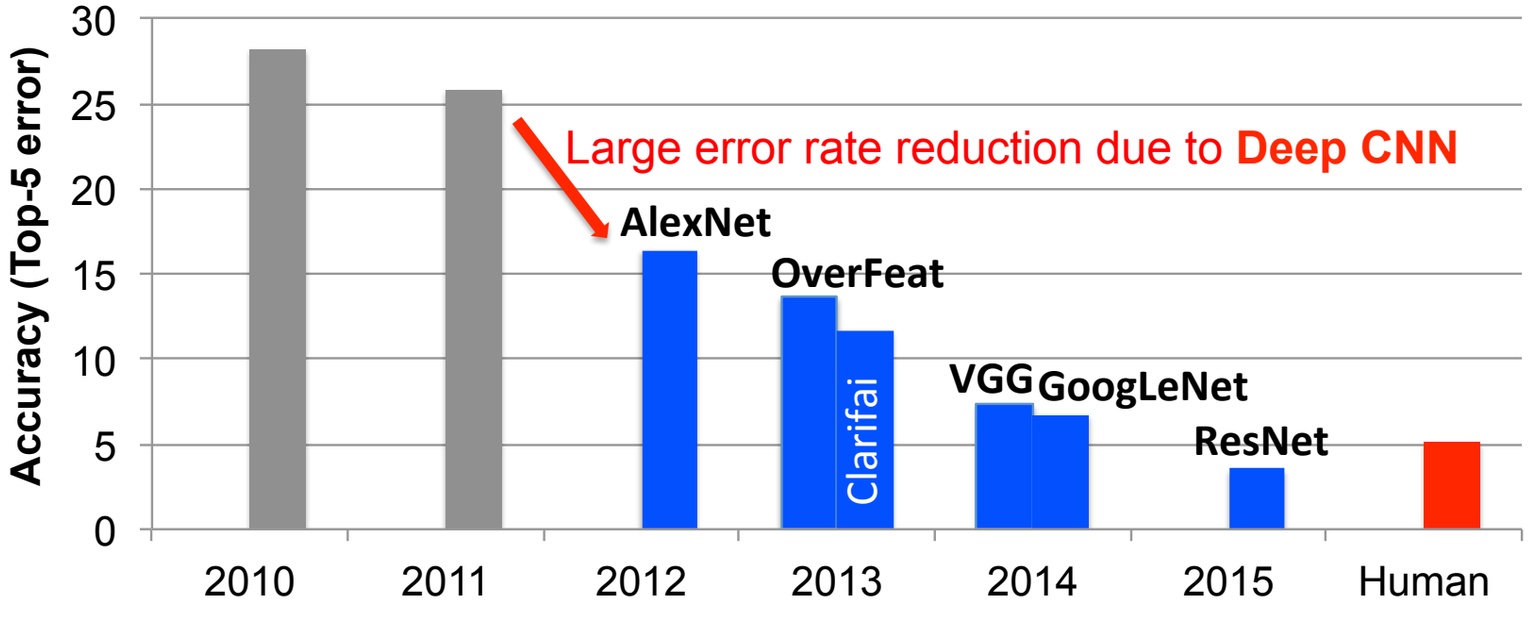
Lecture #16: Deep learning highlights and applications

Paris Perdikaris
March 28, 2023

Deep learning: History and recent success

Deep learning: History and recent success

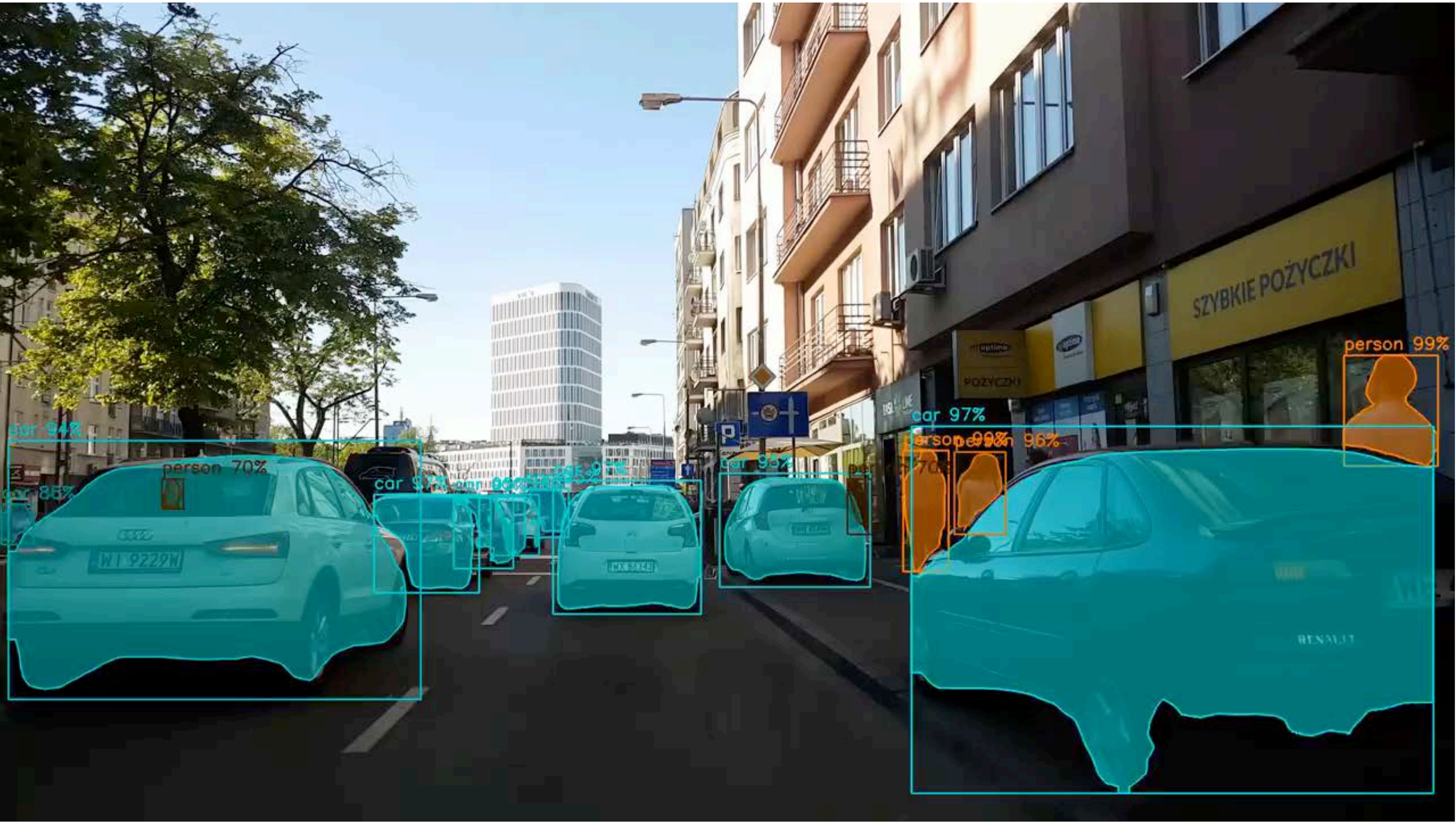
- 1940s - Neural networks were proposed
- 1960s - Deep neural networks were proposed
- 1989 - Neural networks for recognizing digits (LeNet)
- 1990s - Hardware for shallow neural nets (Intel ETANN)
- 2011 - Breakthrough DNN-based speech recognition (Microsoft)
- 2012 - DNNs for vision start supplanting hand-crafted approaches (AlexNet)
- 2014+ - Rise of DNN accelerator research (Neuflow, DianNao...)



In 2015, the ImageNet winning entry, ResNet, exceeded human-level accuracy with a top-5 error rate below 5%.

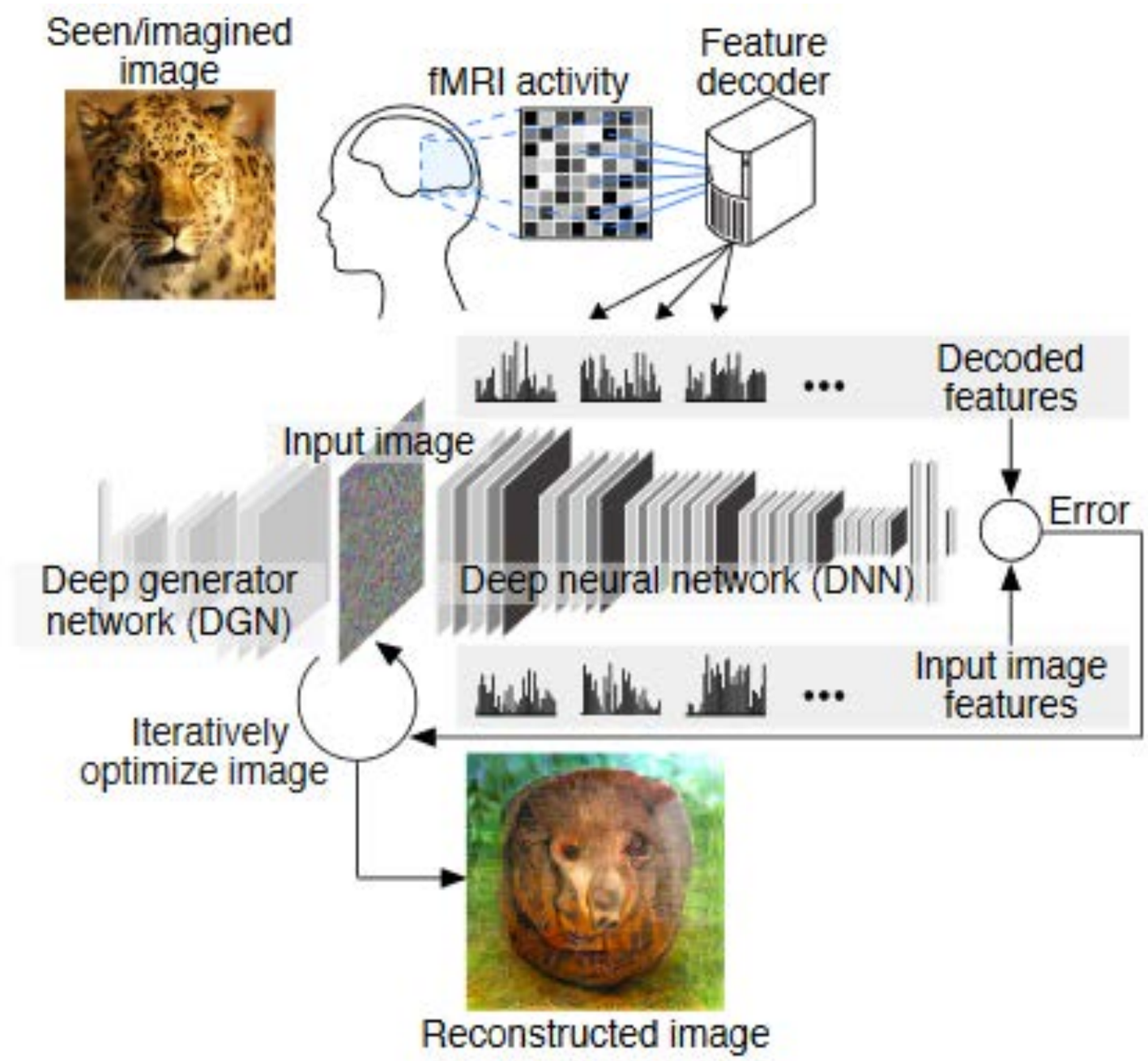
Since then, the error rate has dropped below 3% and more focus is now being placed on more challenging components of the competition, such as object detection and localization.

Object segmentation and tracking



He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017, October). Mask r-cnn. 2017 IEEE International Conference on Computer Vision (ICCV).

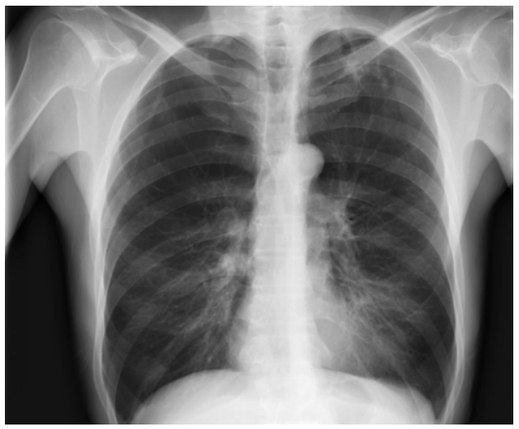
Deep image reconstruction from human brain activity



Overview of deep image reconstruction. The pixels' values of the input image are optimized so that the DNN features of the image are similar to those decoded from fMRI activity. A deep generator network (DGN) is optionally combined with the DNN to produce natural-looking images, in which optimization is performed at the input space of the DGN.

Shen, G., Horikawa, T., Majima, K., & Kamitani, Y. (2017). Deep image reconstruction from human brain activity. bioRxiv, 240317.

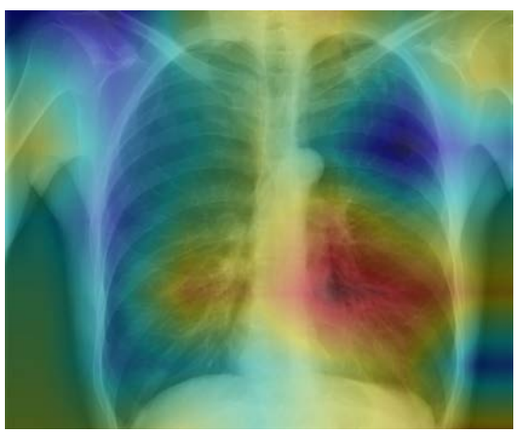
Diagnosing Pneumonia at Radiologist-Level Accuracy



Input
Chest X-Ray Image

CheXNet
121-layer CNN

Output
Pneumonia Positive (85%)



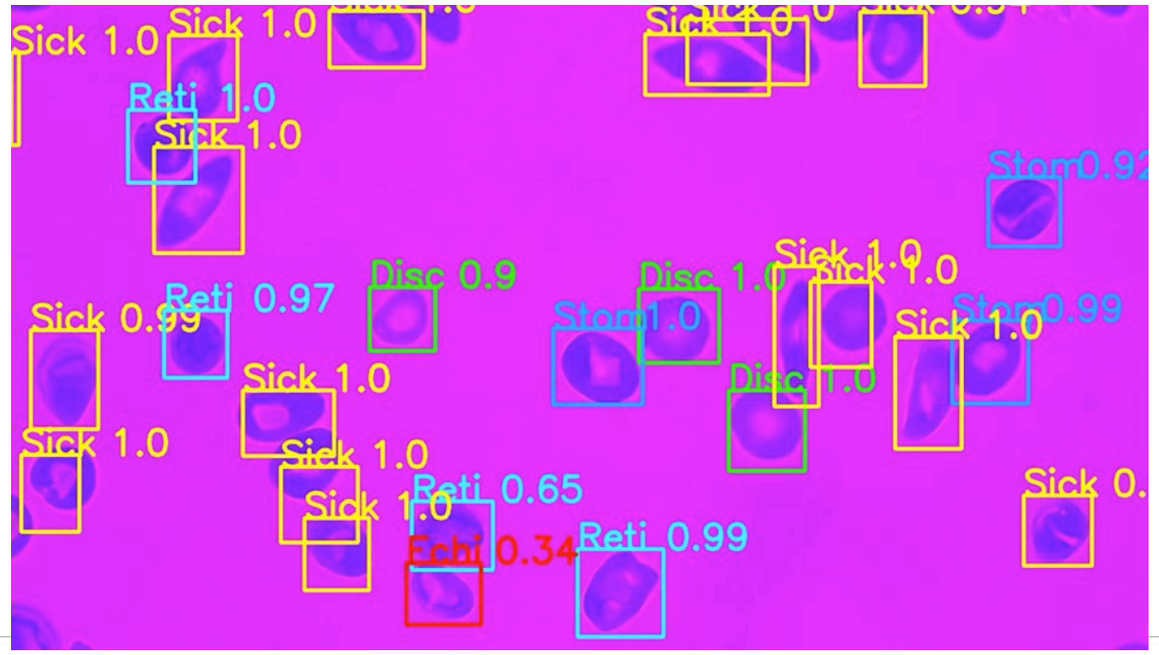
Abstract

We develop an algorithm that can detect pneumonia from chest X-rays at a level exceeding practicing radiologists. Our algorithm, CheXNet, is a 121-layer convolutional neural network trained on ChestX-ray14, currently the largest publicly available chest X-ray dataset, containing over 100,000 frontal-view X-ray images with 14 diseases. Four practicing academic radiologists annotate a test set, on which we compare the performance of CheXNet to that of radiologists. We find that CheXNet exceeds average radiologist performance on the F1 metric. We extend CheXNet to detect all 14 diseases in ChestX-ray14 and achieve state of the art results on all 14 diseases.

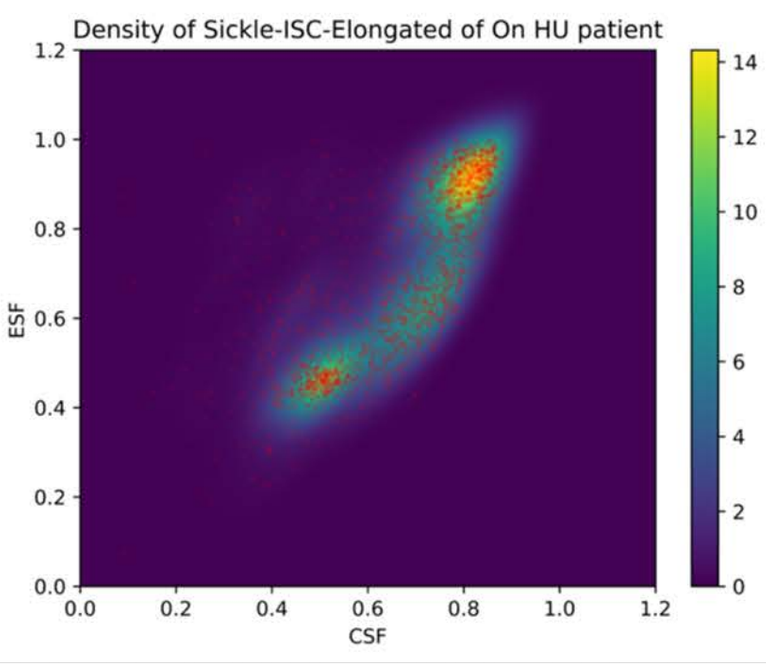
	F1 Score (95% CI)
Radiologist 1	0.383 (0.309, 0.453)
Radiologist 2	0.356 (0.282, 0.428)
Radiologist 3	0.365 (0.291, 0.435)
Radiologist 4	0.442 (0.390, 0.492)
Radiologist Avg.	0.387 (0.330, 0.442)
CheXNet	0.435 (0.387, 0.481)

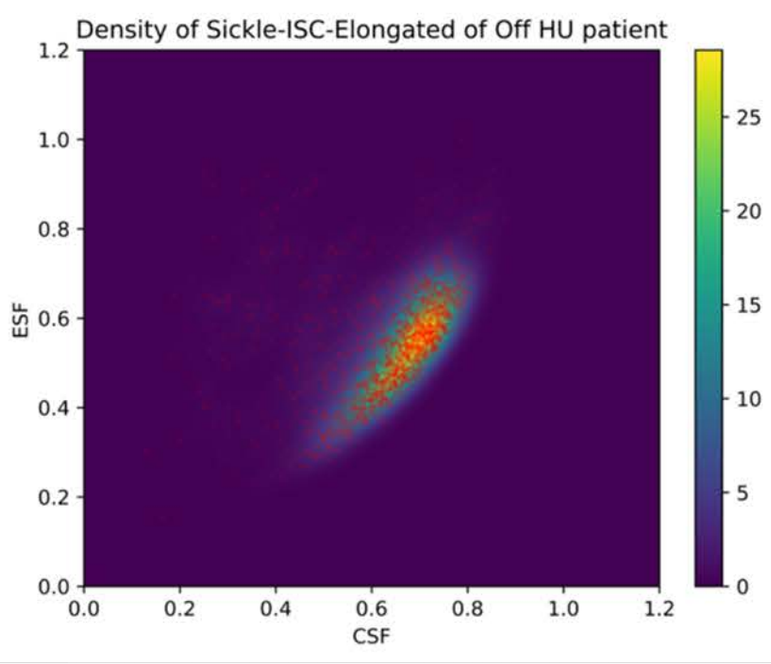
Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Lungren, M. P. (2017). CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. arXiv preprint

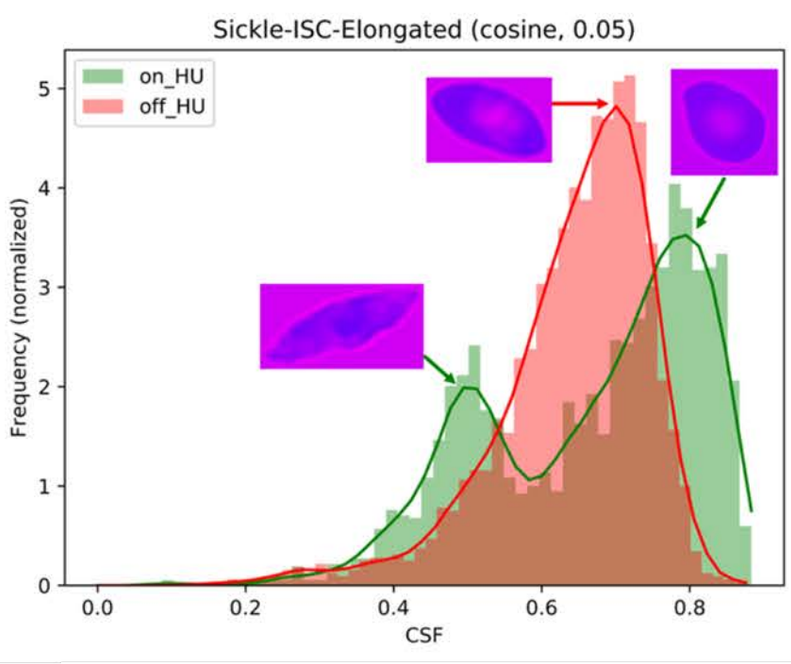
A deep learning system for blood drop analysis



	On Drug		Off Drug	
	Count	% of Total	Count	% of Total
Discocytes_Oval	2412	27.29	637	13.31
Echinocytes-II-III	124	1.40	162	3.38
Granular-Echino-I	116	1.31	49	1.02
Reticulocytes	40	0.45	211	4.41
Sickle-ISC-Elongated	2316	26.20	2144	44.80
Stomatocytes	3831	43.34	1583	33.08
Total	8839	100	4786	100

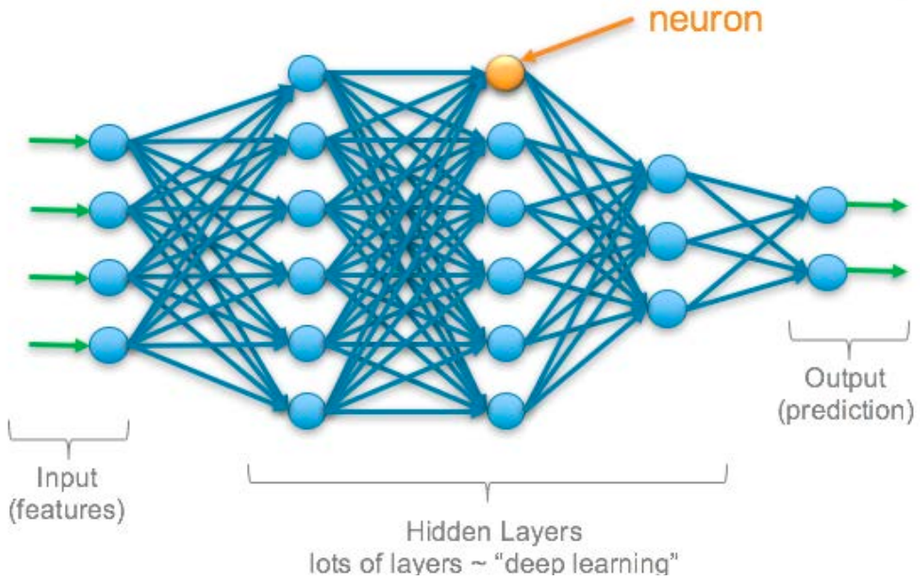


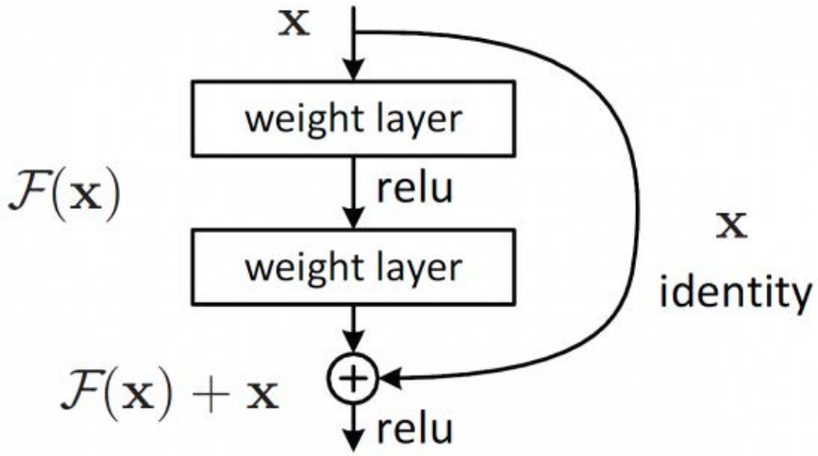




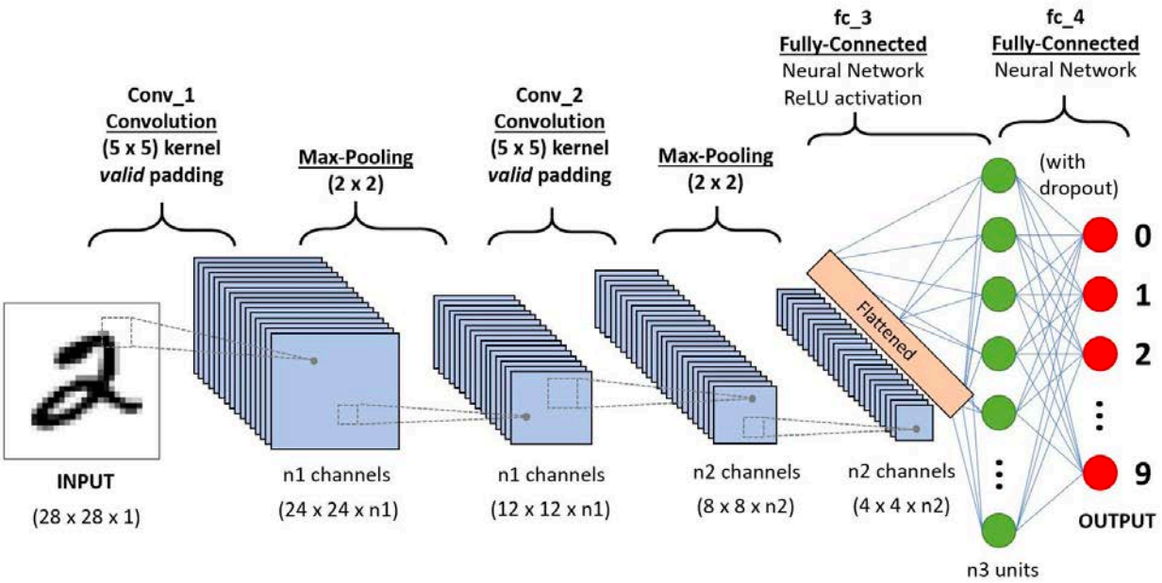
Credit: Lingzhi Zhang (Penn), Kuan-Cheng Chiu (Penn), Dimitrios Papageorgiou (MIT)

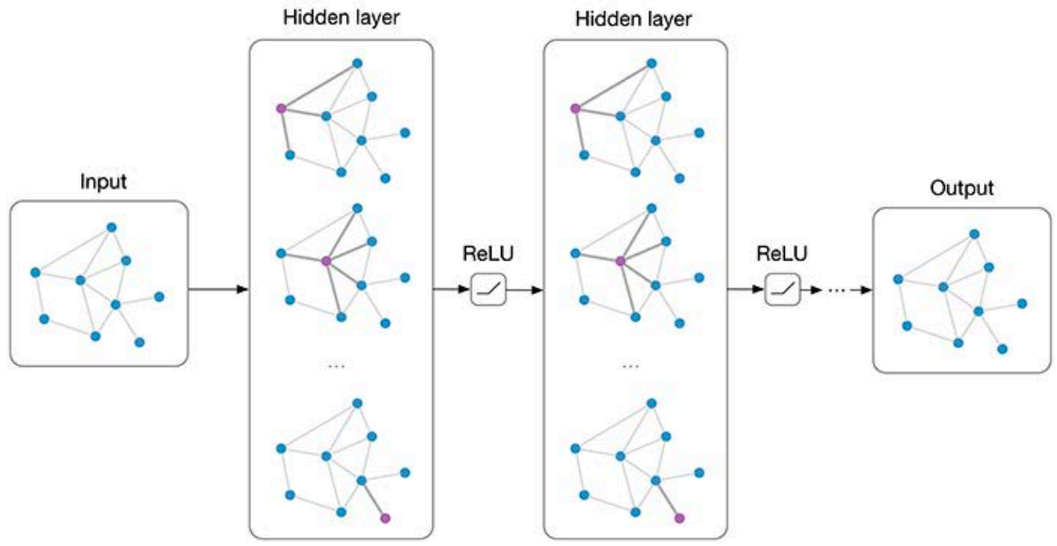
Things we didn't cover



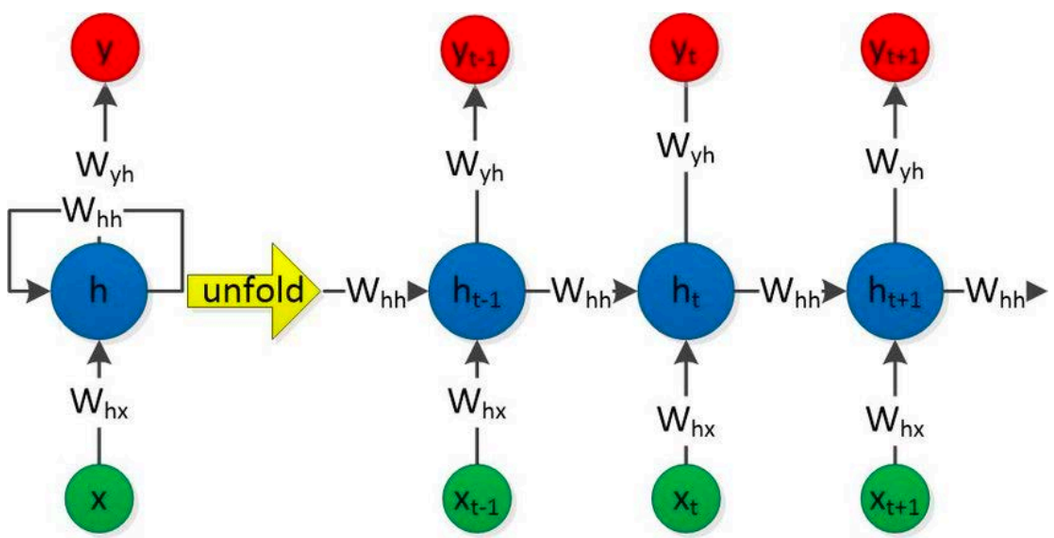


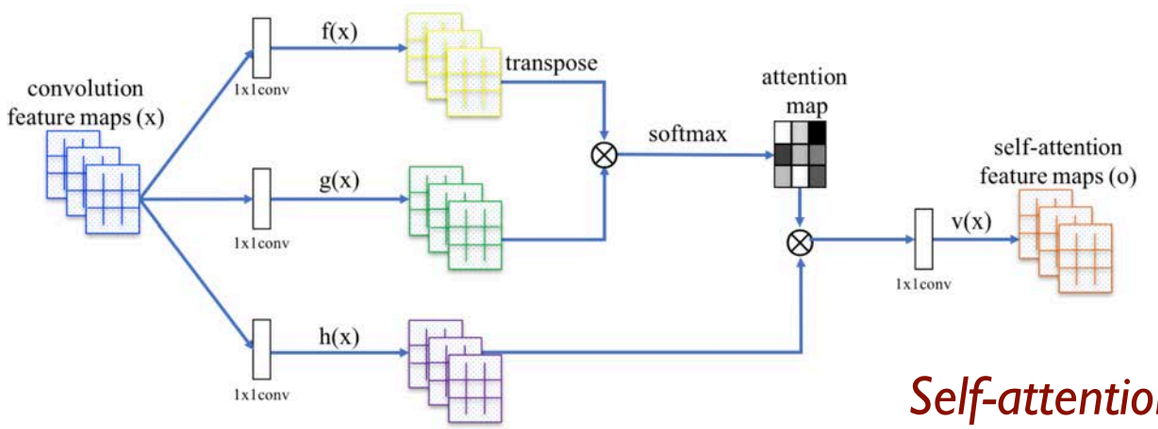
Residual/skip connections





Graph neural networks





Self-attention

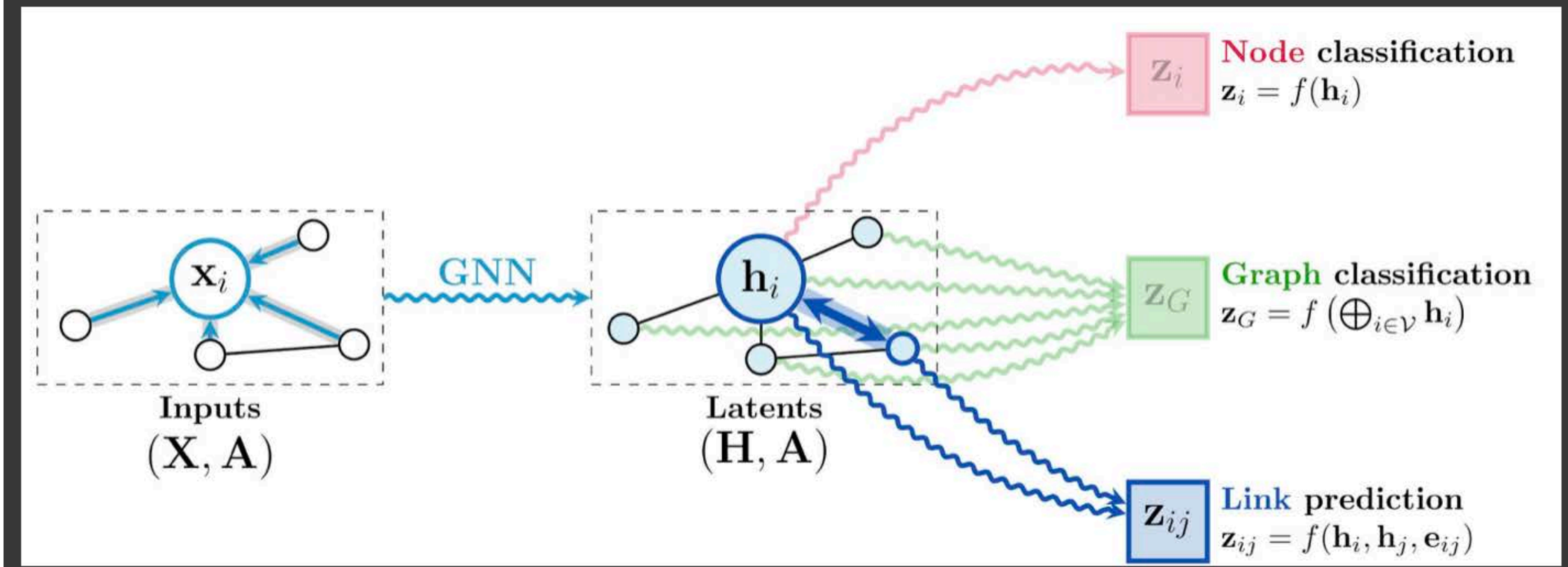
Geometric deep learning

Graph Prediction Tasks

What are the kinds of problems we want to solve on graphs?

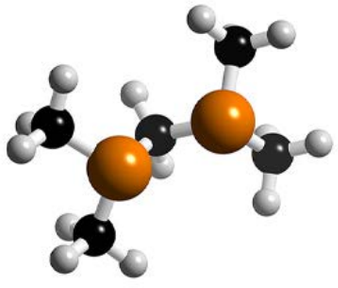
The tasks fall into roughly three categories:

1. **Node Classification**: E.g. what is the topic of a paper given a citation network of papers?
2. **Link Prediction / Edge Classification**: E.g. are two people in a social network friends?
3. **Graph Classification**: E.g. is this protein molecule (represented as a graph) likely going to be effective?

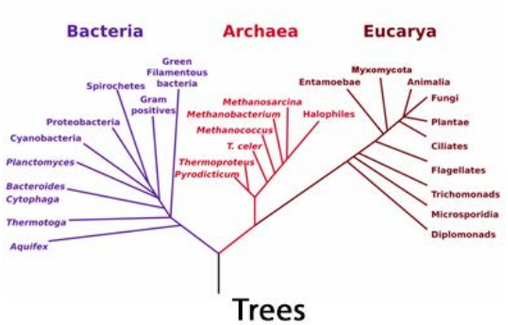


The three main graph learning tasks. Image source: Petar Veličković.

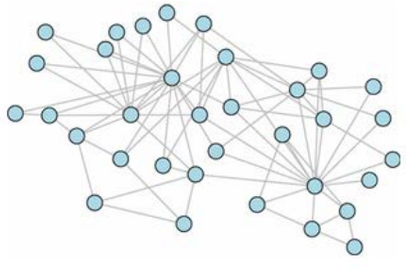
Geometric deep learning



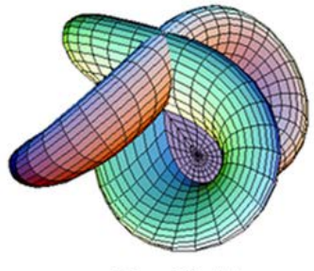
Molecules



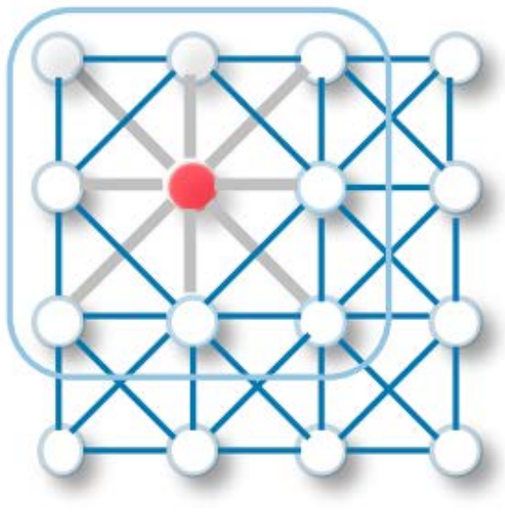
Trees

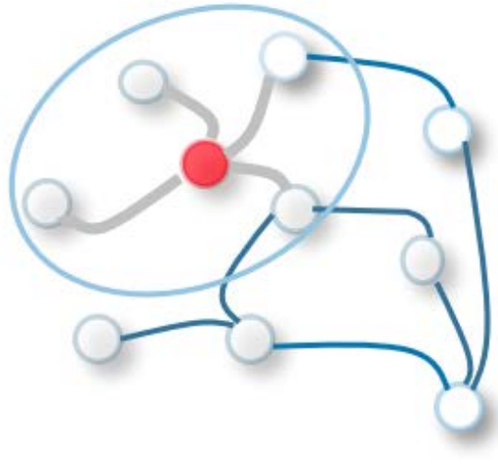


Networks



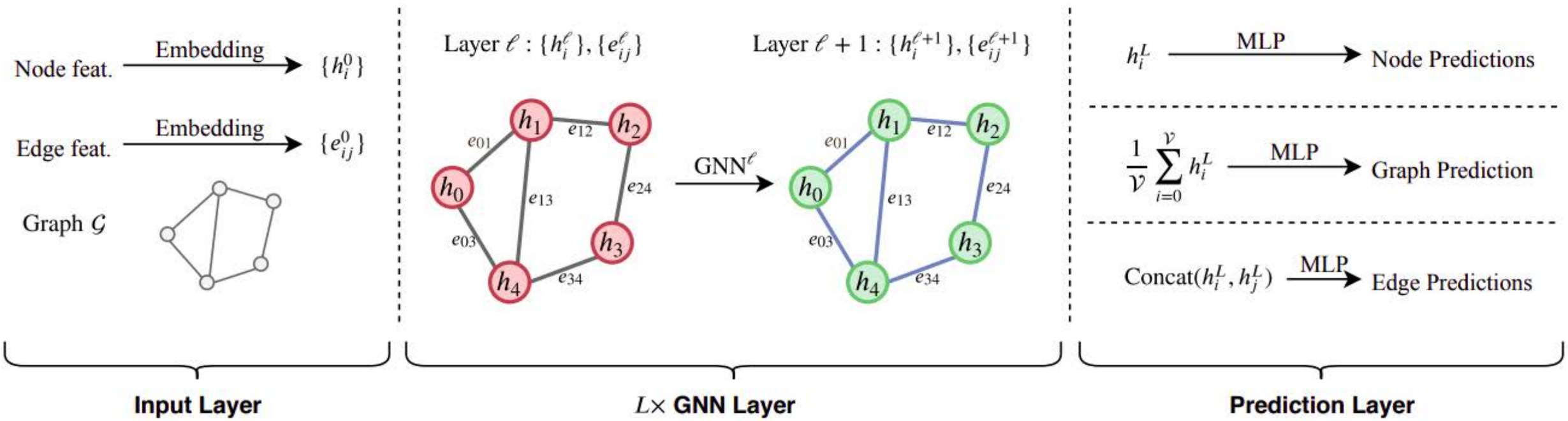
Manifolds





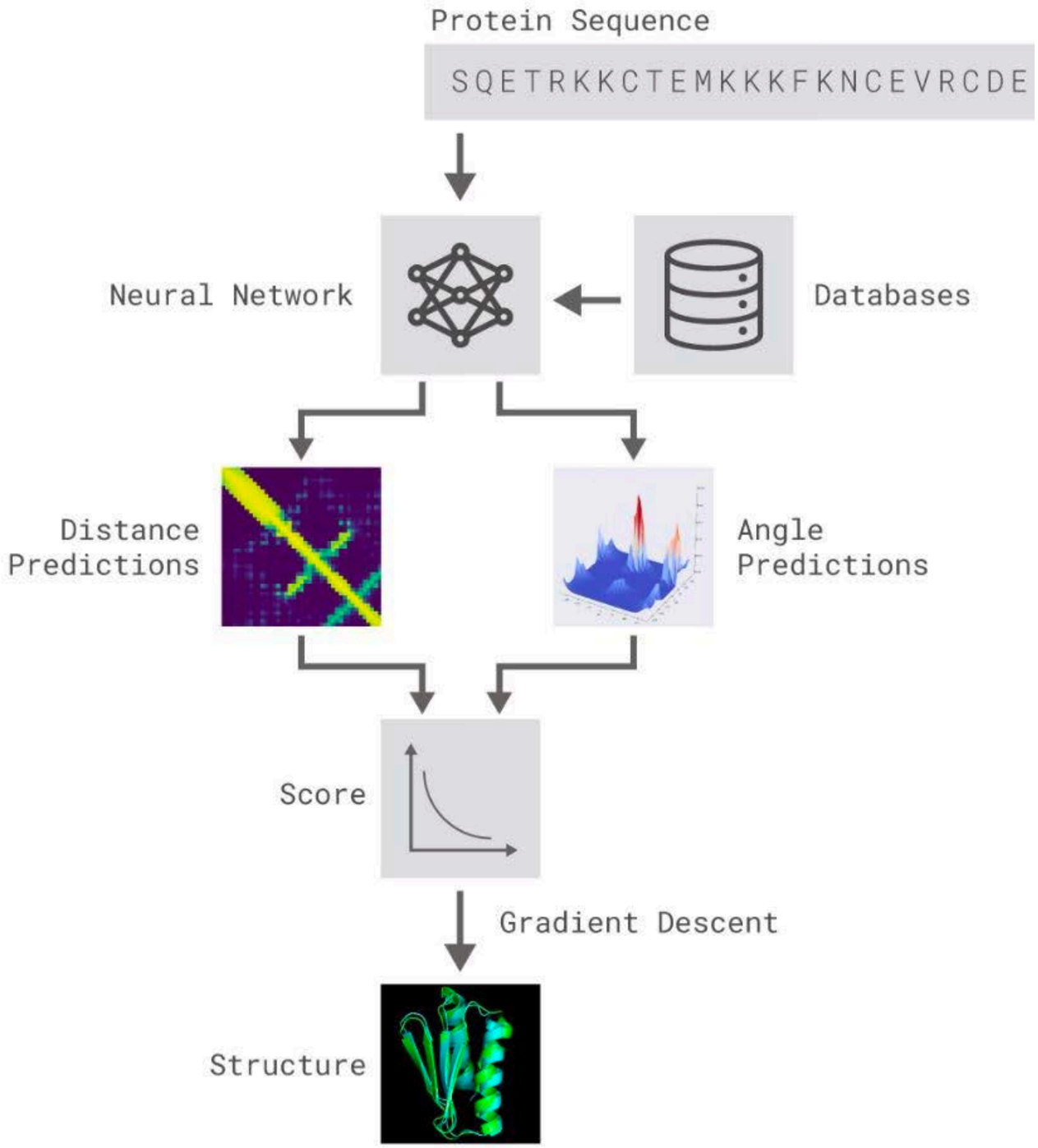
Working with non-Euclidean data

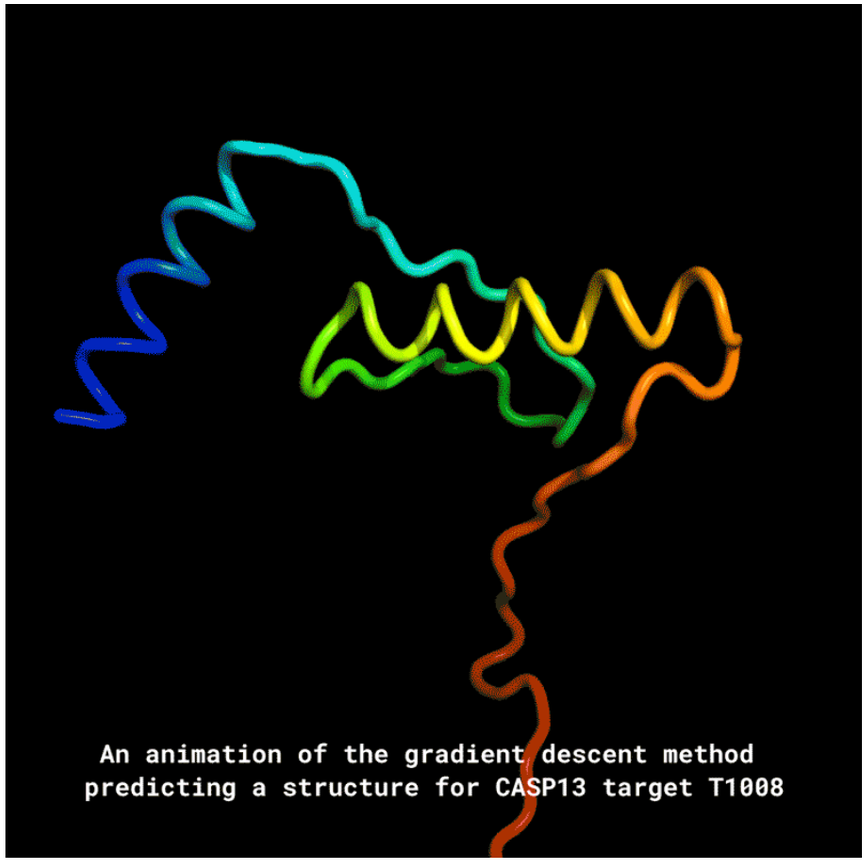
Generalizing convolutions on graphs

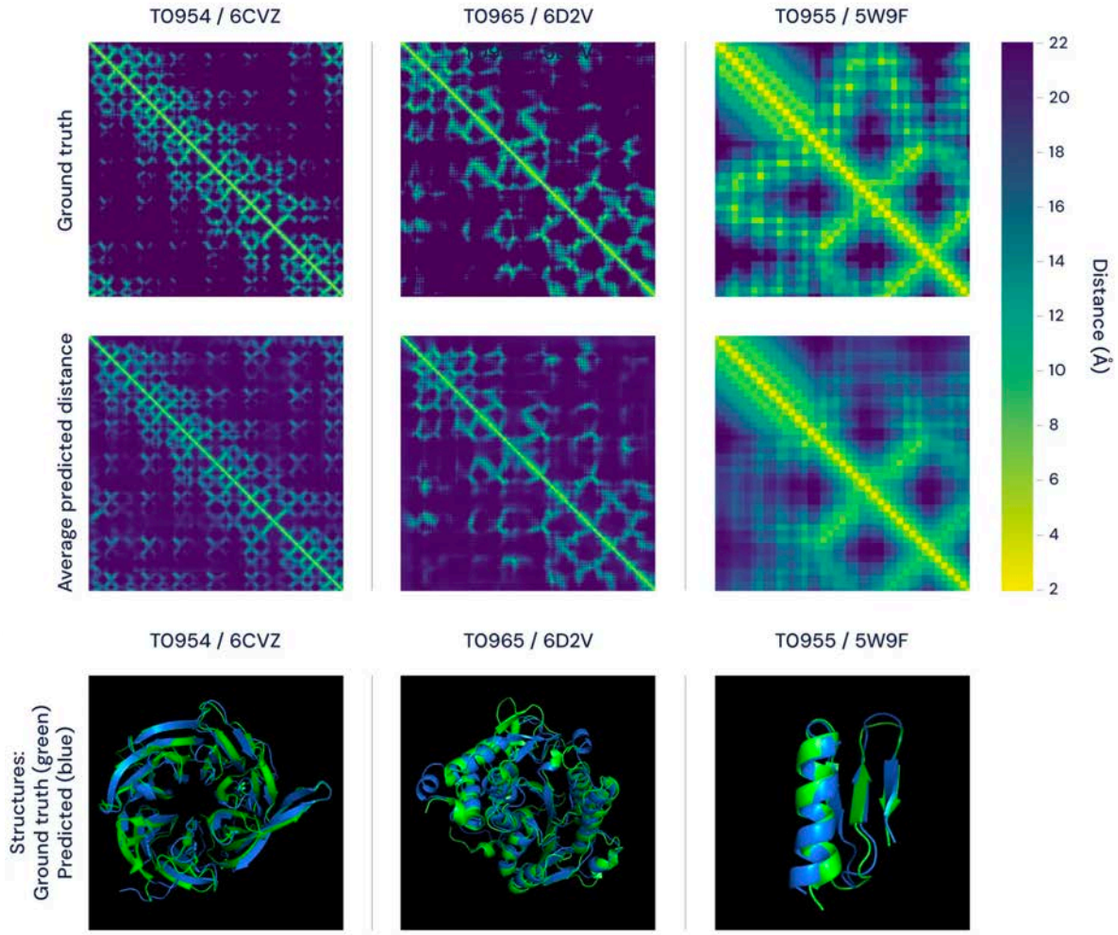


https://colab.research.google.com/github/deepmind/educational/blob/master/colabs/summer_schools/intro_to_graph_nets_tutorial_with_jraph.ipynb

Protein structure prediction



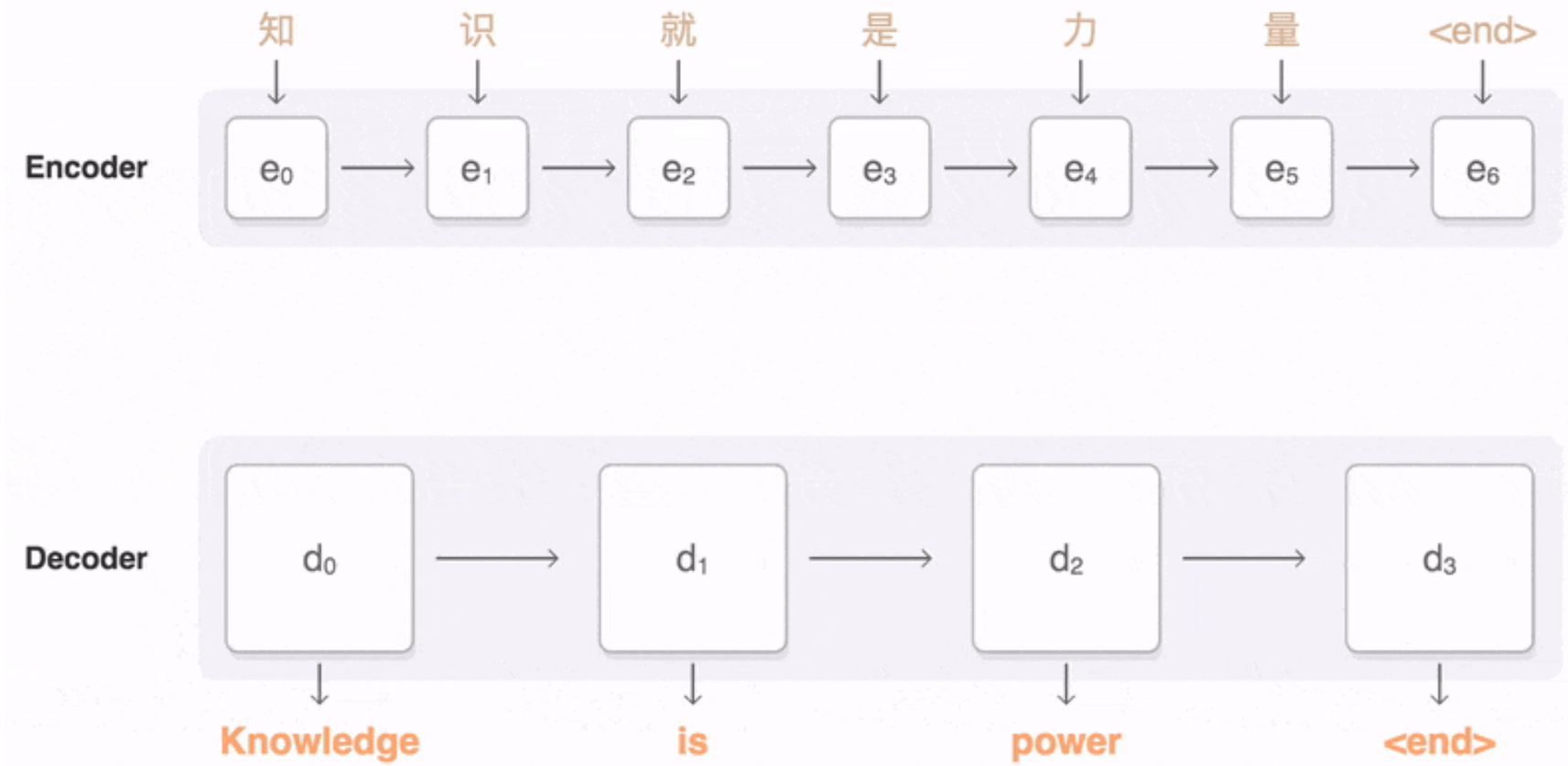




Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., ... & Penedones, H. (2020). Improved protein structure prediction using potentials from deep learning. Nature, 577(7792), 706-710.

Self-attention

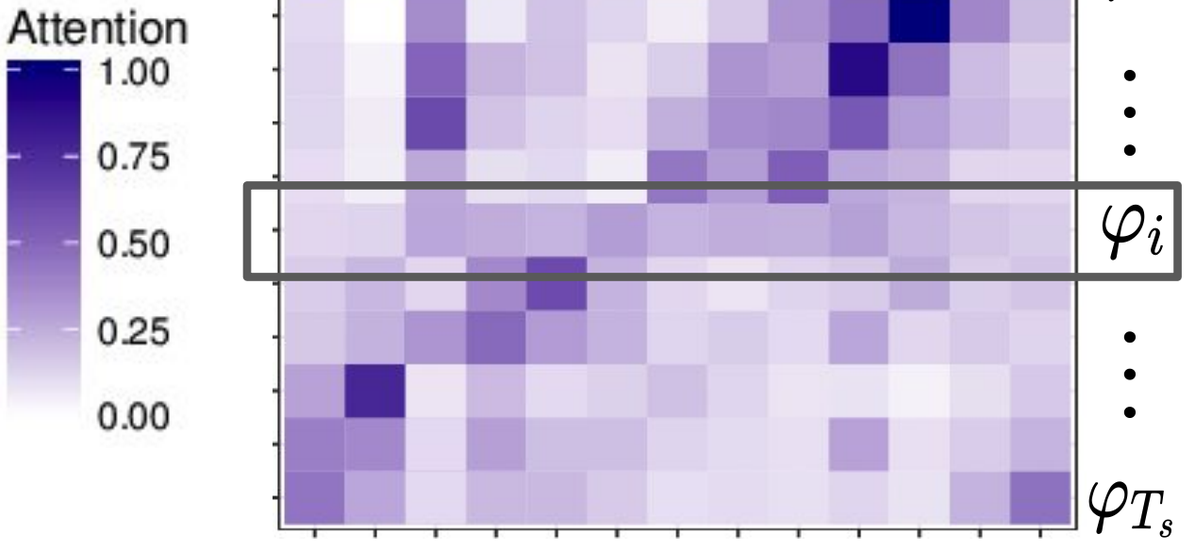
- Goal was to translate $\{u_1, \ldots, u_{T_u}\}$ to $\{s_1, \ldots, s_{T_s}\}$
- First map input to features $\{u_1, \ldots, u_{T_u}\} \longrightarrow \{v_1, \ldots, v_{T_u}\}$
- Each location in the output sentence gets a context vector c_i to determine the probability of a word at that location.
- Context vector is constructed as a position-dependent average of the features.



Self-attention

Input sentence $[u_1, \ldots, u_i, \ldots, u_{T_u}]$

Input features



Attention weights φ_{ij}

$$\begin{bmatrix} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_{T_s} \end{bmatrix} \rightarrow c_i = \sum_{j=1}^{T_u} \varphi_{ij} v_j$$

Context vector

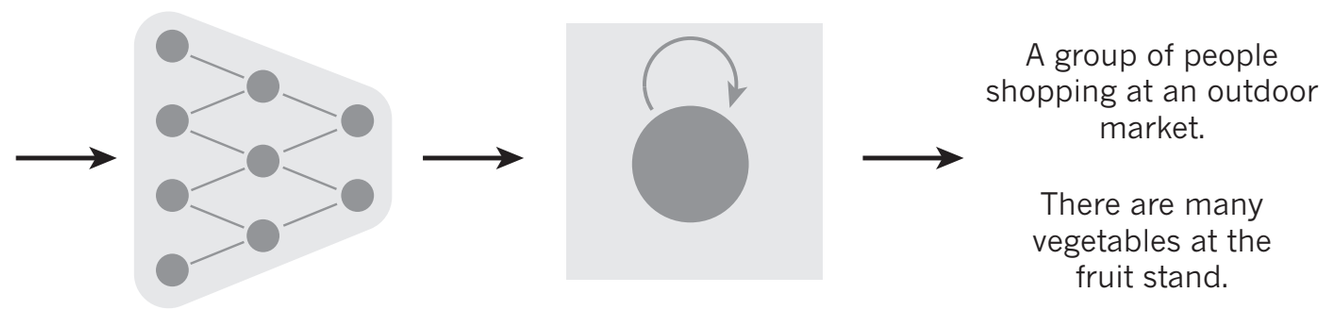
Bahdanau Attention

$p(s_i \mid c_i, u)$ Probability of word s_i at output position i

$[s_1, \ldots, s_i, \ldots, s_{T_s}]$ Output sentence

Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

Attention and captioning



A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436.

Text generation

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \mathrm{gor}_s & & \\ & & \\ & = \alpha' & \longrightarrow \\ & \updownarrow & \\ & = \alpha' & \longrightarrow \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \end{array}$$

$$\mathrm{Spec}(K_\psi) \qquad \mathrm{Mor}_{Sets} \quad \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G})$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

NLP with transformer architectures

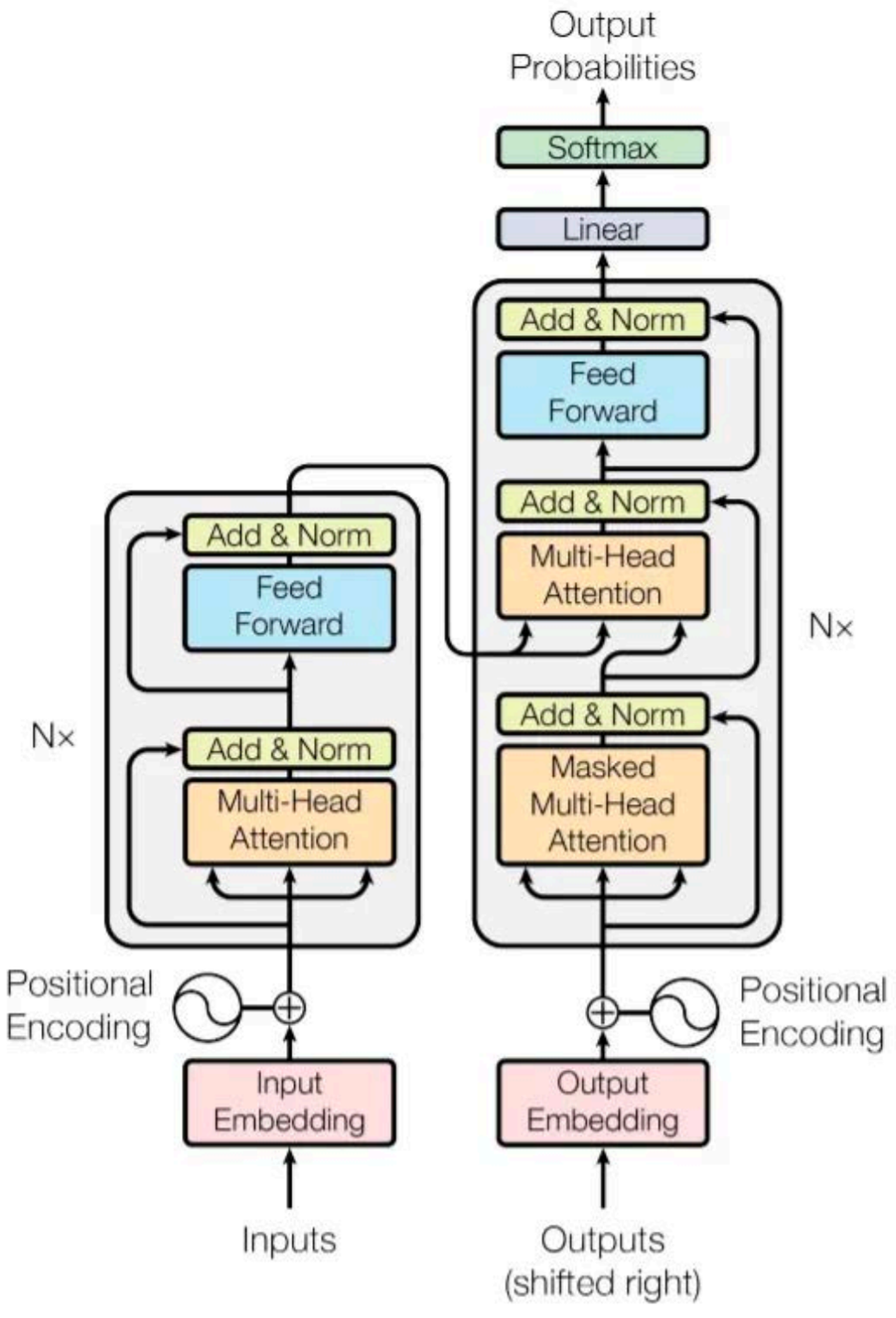
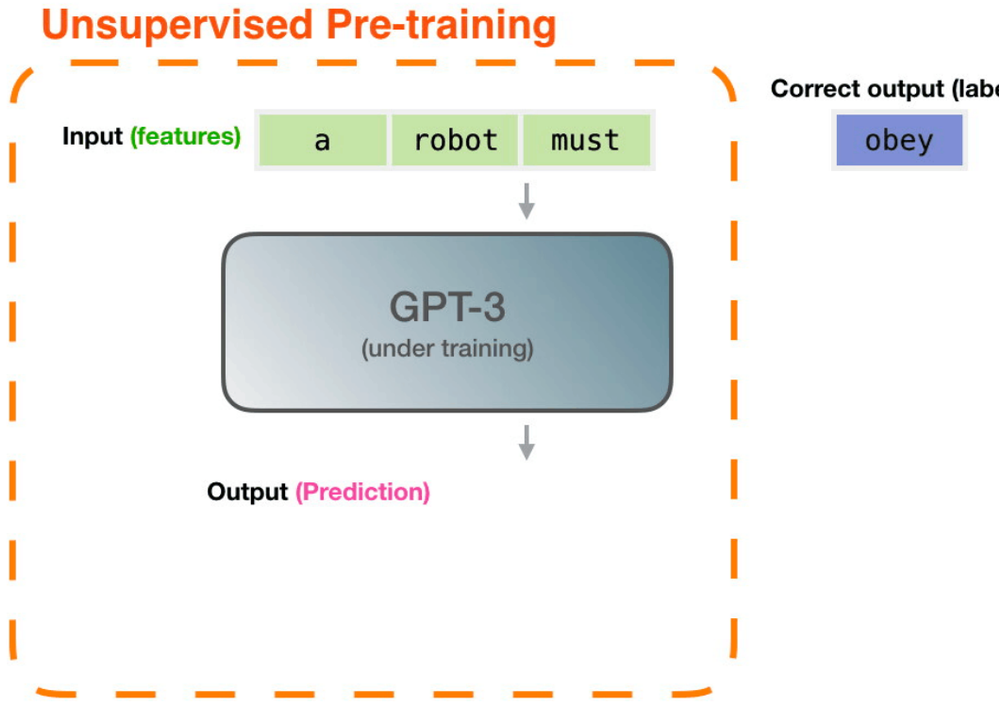
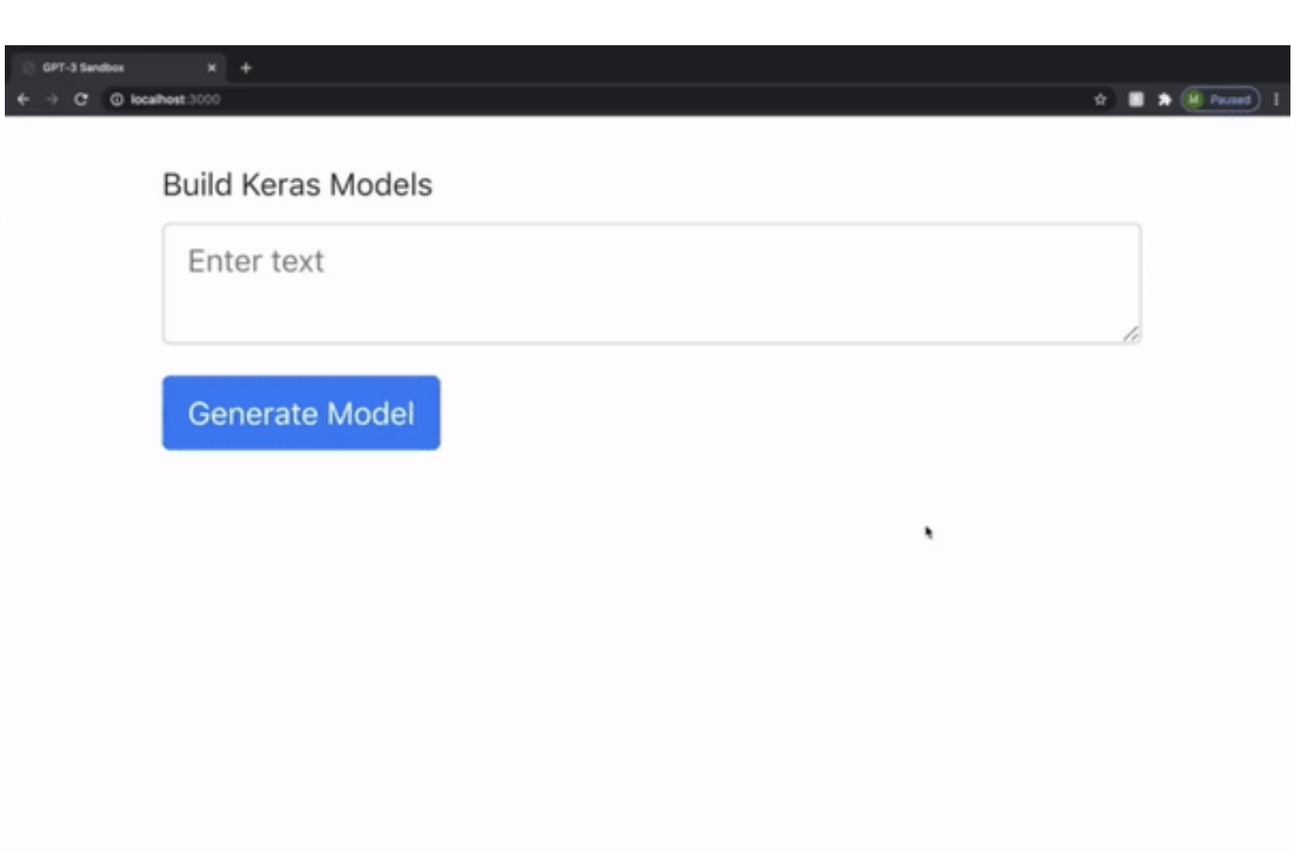


Figure 1: The Transformer - model architecture.





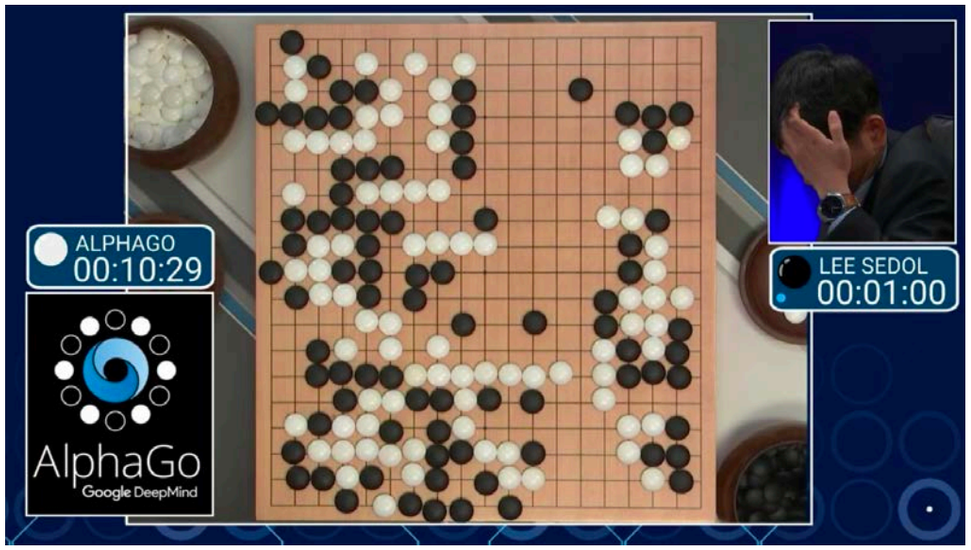
Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Agarwal, S. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

Deep reinforcement learning

Google DeepMind's
Deep Q-learning

The algorithm will play Atari breakout.

The most important thing to know is that all the agent is given is sensory input (what you see on the screen) and it was ordered to maximize the score on the screen.



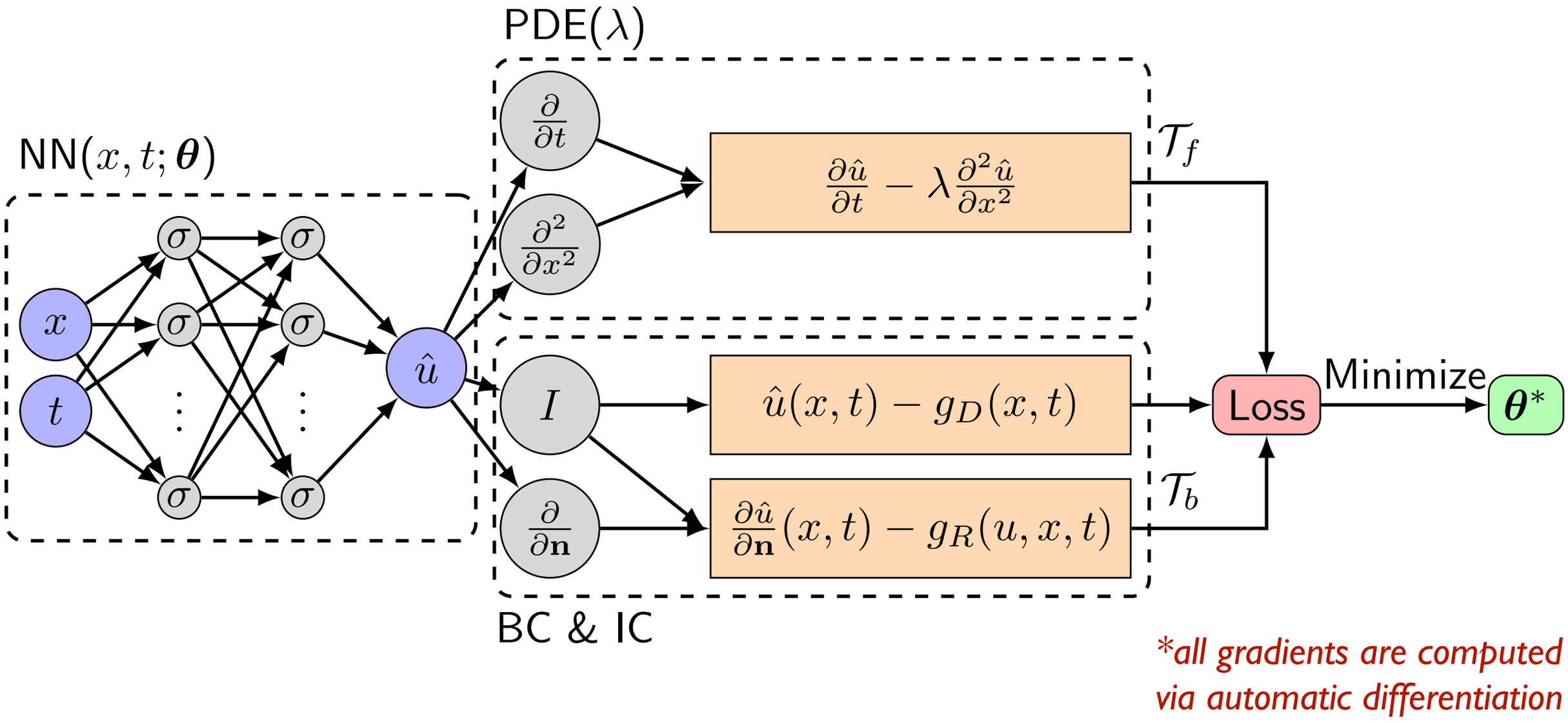
https://github.com/deepmind/rlax

Physics-informed neural networks (PINNs)

We consider partial differential equations (PDEs) of the following general form:

$$\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \quad \boldsymbol{x} \in \Omega, t \in [0, T]$$

subject to appropriate initial and boundary conditions.



**all gradients are computed via automatic differentiation*

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 686-707.

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations.

Psichogios, D. C., & Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling.

General formulation of PINNs

Physics-informed neural networks (PINNs) aim at inferring a continuous latent function $\boldsymbol{u}(\boldsymbol{x}, t)$ that arises as the solution to a system of nonlinear partial differential equations (PDE) of the general form

$$\begin{aligned}
&\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \quad \boldsymbol{x} \in \Omega, t \in [0, T] \\
&\boldsymbol{u}(\boldsymbol{x}, 0) = h(\boldsymbol{x}), \quad x \in \Omega \\
&\boldsymbol{u}(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \quad t \in [0, T], \quad \boldsymbol{x} \in \partial\Omega
\end{aligned}$$

We proceed by approximating $\boldsymbol{u}(\boldsymbol{x}, t)$ by a deep neural network $f_\theta(\boldsymbol{x}, t)$, and define the residual of the PDE as

$$\boldsymbol{r}_\theta(\boldsymbol{x}, t) := \frac{\partial}{\partial t} f_\theta(\boldsymbol{x}, t) + \mathcal{N}_{\boldsymbol{x}}[f_\theta(\boldsymbol{x}, t)]$$

The corresponding loss function is given by

$$\mathcal{L}(\theta) := \underbrace{\mathcal{L}_u(\theta)}_{\text{Data fit}} + \underbrace{\mathcal{L}_r(\theta)}_{\text{PDE residual}} + \underbrace{\mathcal{L}_{u_0}(\theta)}_{\text{ICs fit}} + \underbrace{\mathcal{L}_{u_b}(\theta)}_{\text{BCs fit}}$$

Training via stochastic gradient descent:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n)$$

**all gradients are computed via automatic differentiation*

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 686-707.

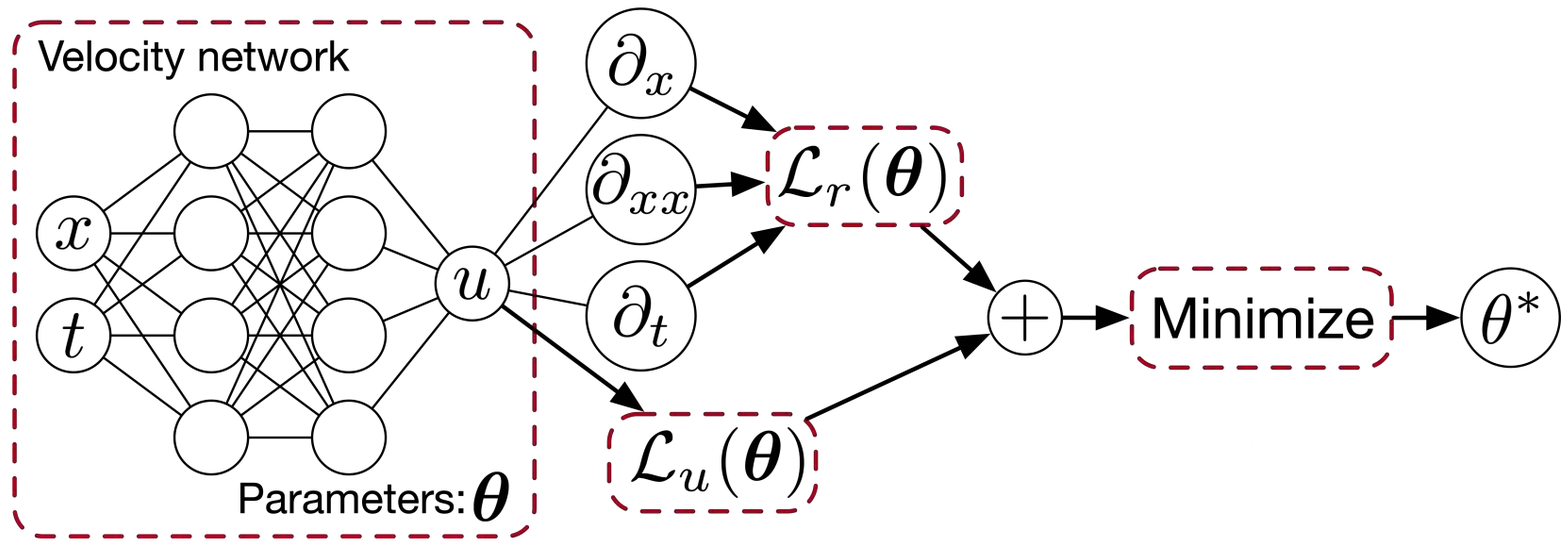
Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations.

Psichogios, D. C., & Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling.

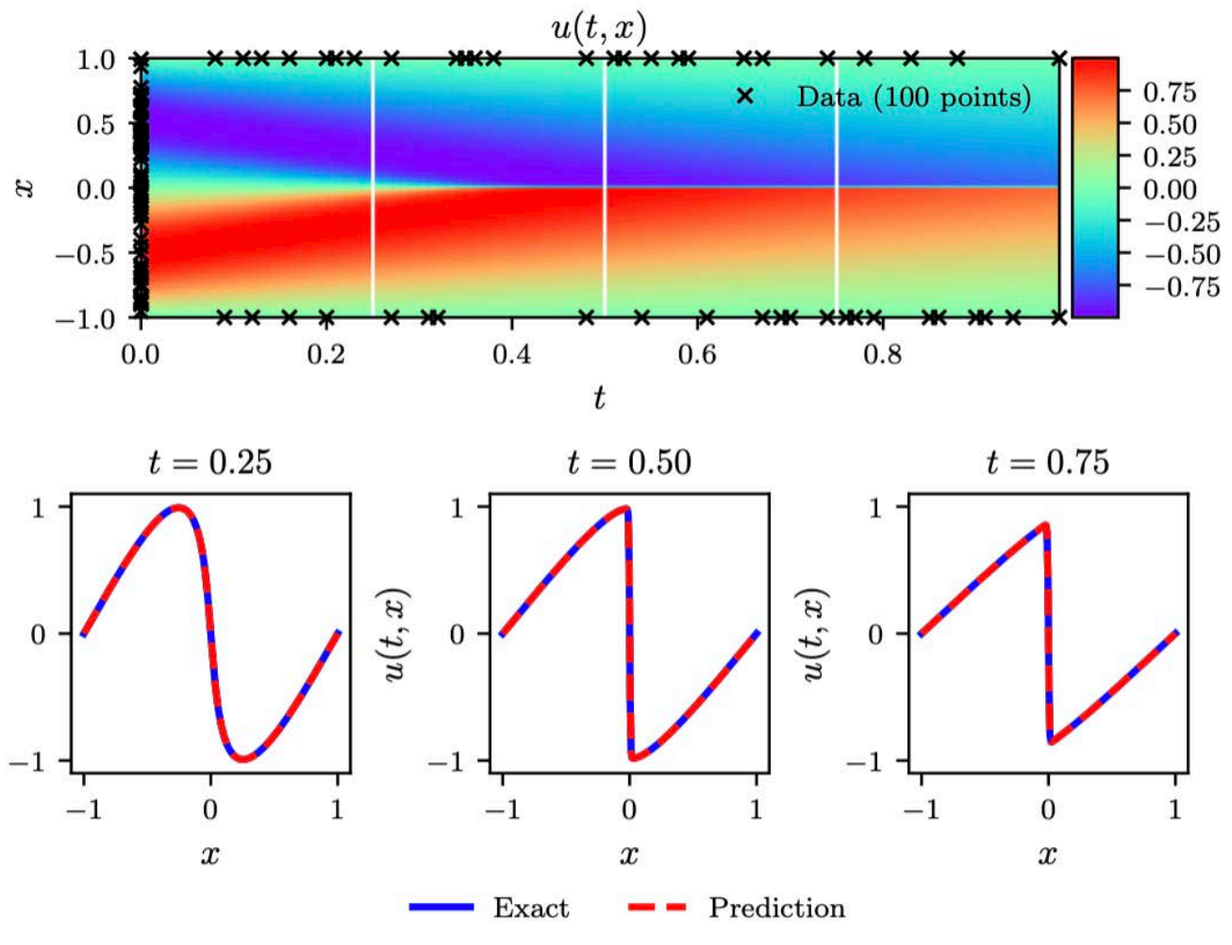
Physics-informed neural networks

Example:
1D Burgers equation

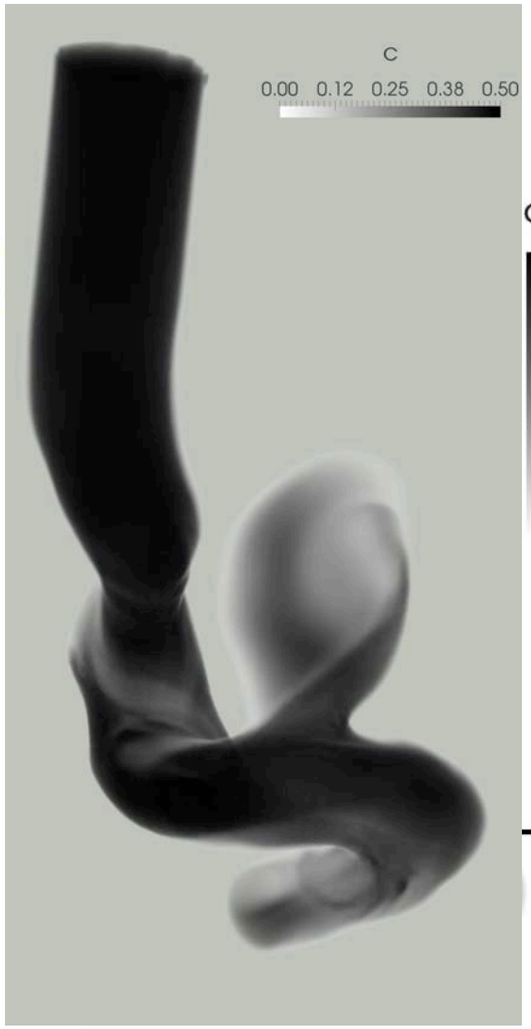
$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1],$$
$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$

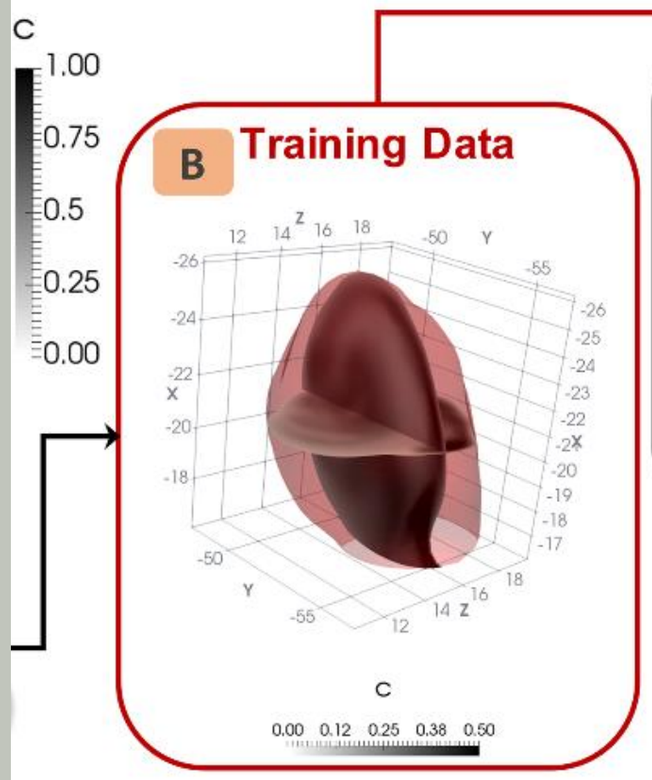


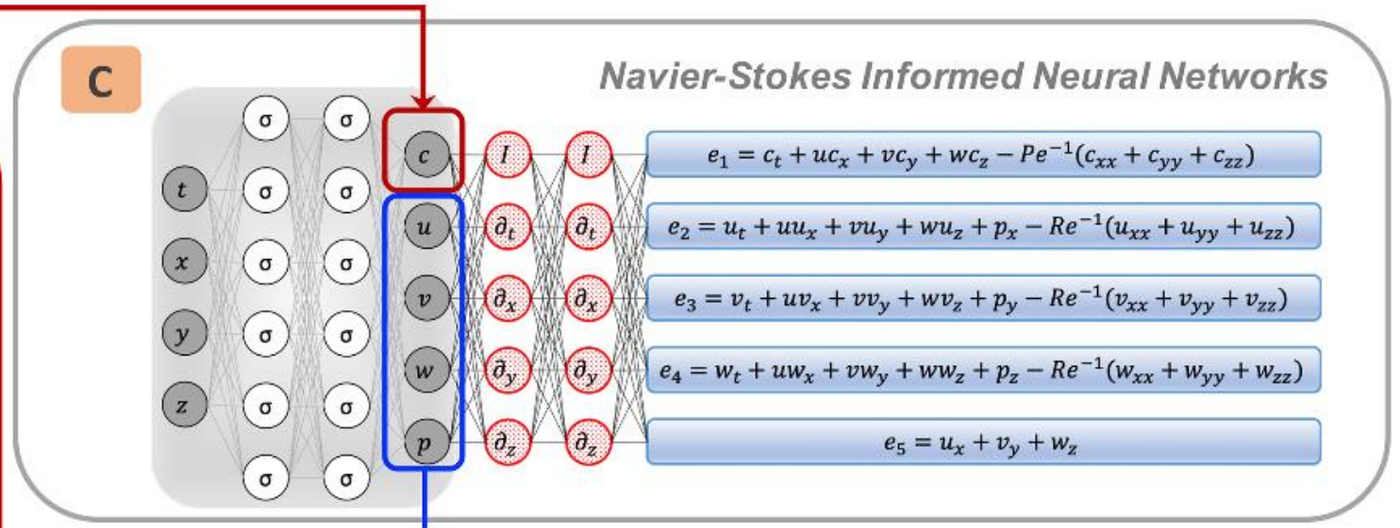

```python
import jax.numpy as np
from jax import grad

def net_u(params, x, t):
    inputs = np.stack([x, t])
    u = self.net_apply(params, inputs)
    return u[0]

def net_r(params, x, t):
    # Compute derivatives
    u = net_u(params, x, t)
    u_t = grad(net_u, 2)(params, x, t)
    u_x = grad(net_u, 1)(params, x, t)
    u_xx = grad(grad(net_u, 1), 1)(params, x, t)
    # Compute residual
    res = u_t + u*u_x - (0.01/np.pi)*u_xx
    return res
```

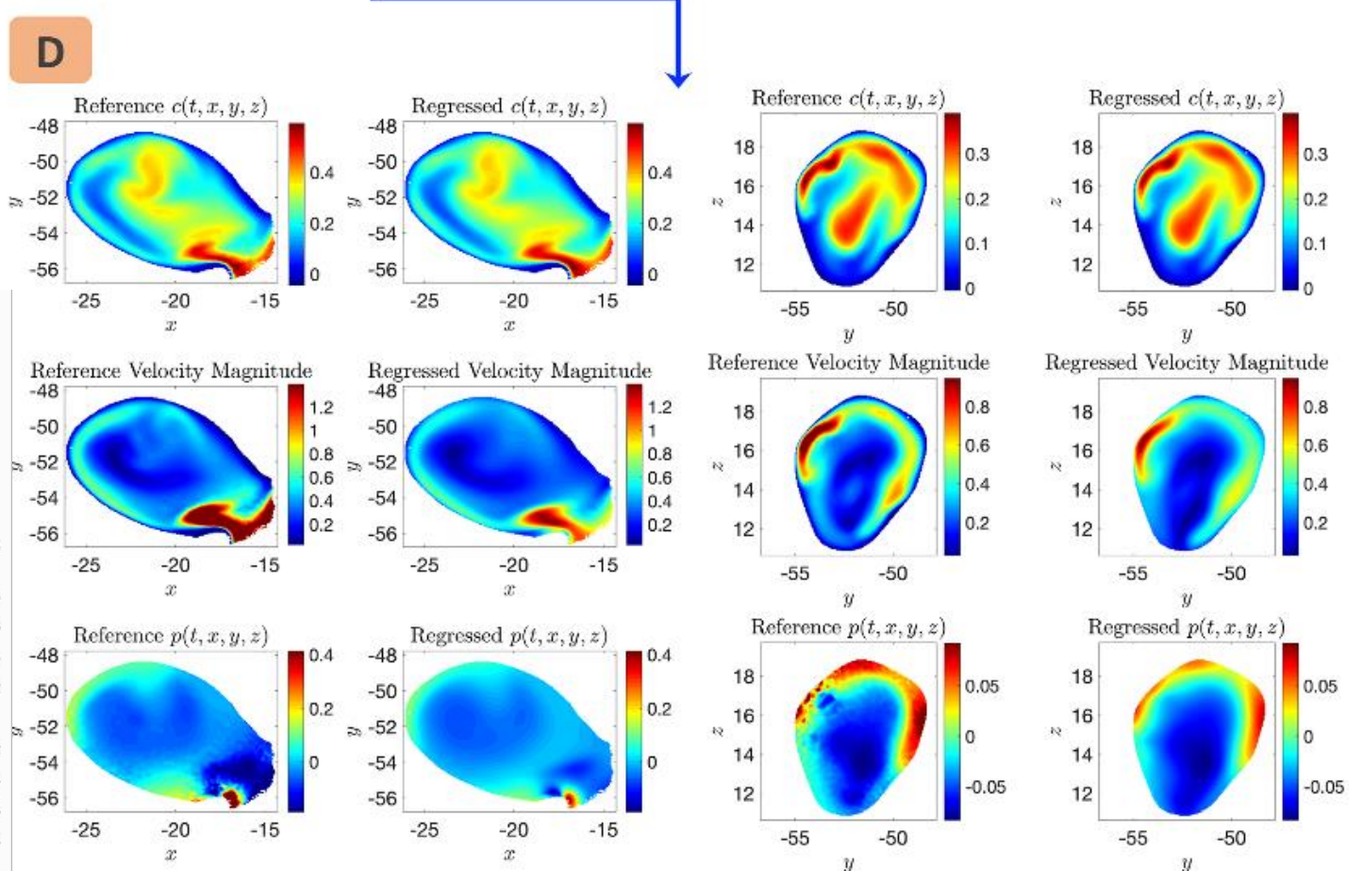


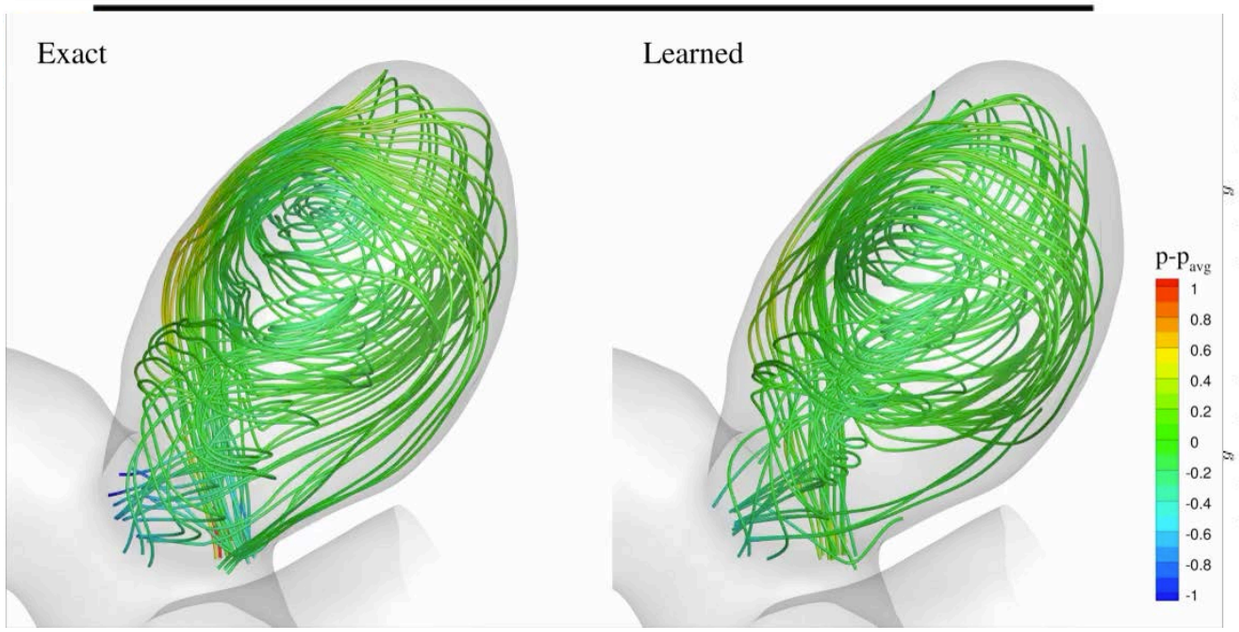
Physics-informed neural networks





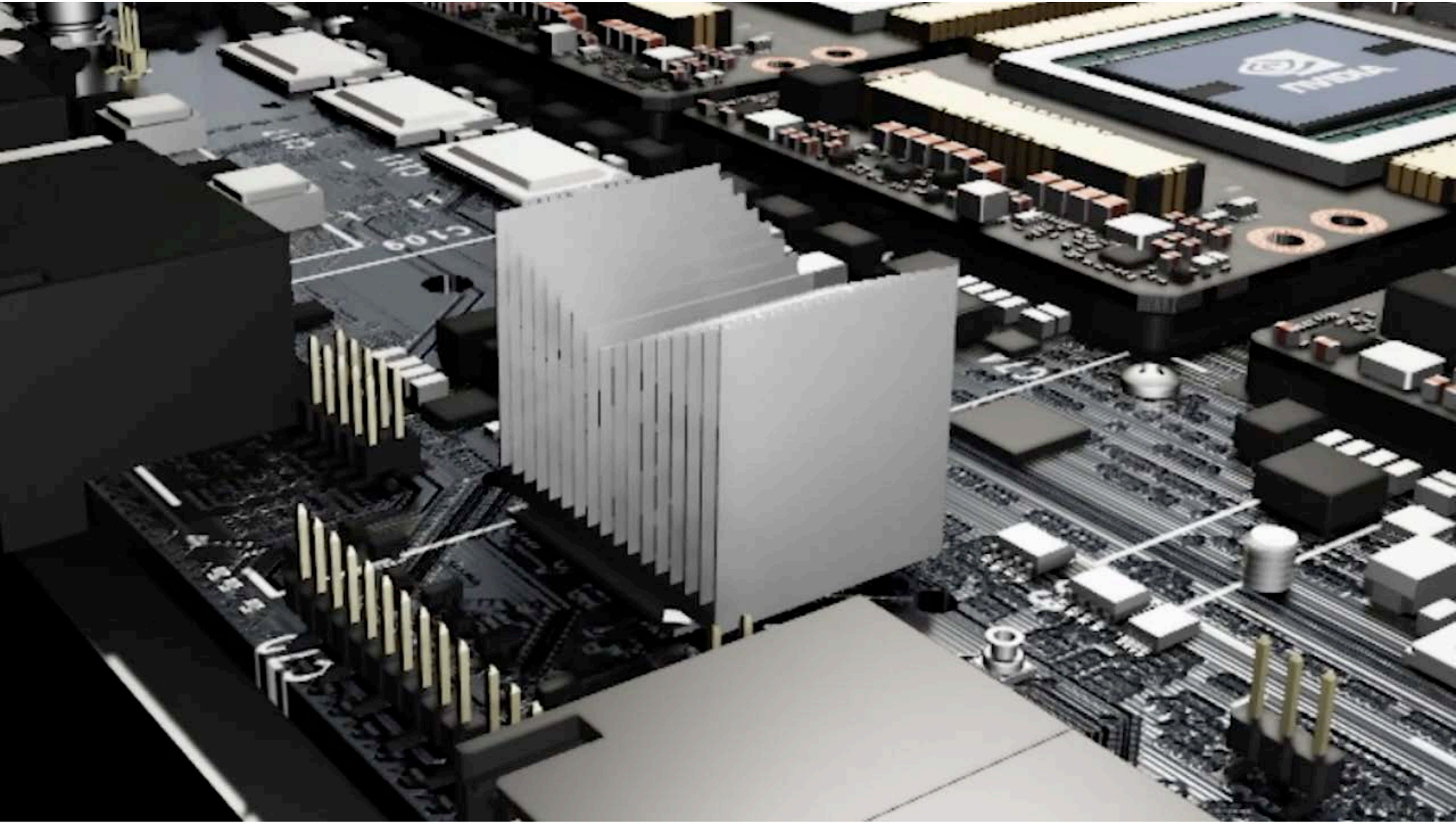






Raissi, M., Yazdani, A., & Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science.

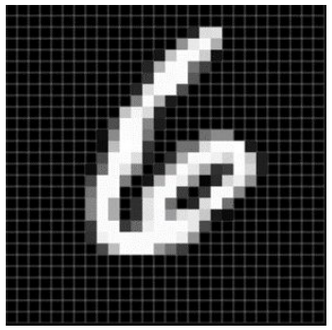
Design exploration



https://developer.nvidia.com/simnet

Supervised (finite-dimensional) learning

- Data science and machine learning methods have traditionally been applied to learn functions of finite dimensional data

$$\mathbb{R}^{784} \ni$$  $$\xrightarrow{f : \mathbb{R}^{784} \to \mathbb{R}^{10}} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^{10}$$

- Given labelled examples $x_i \in \mathbb{R}^{d_x}, y_i \in \mathbb{R}^{d_y}$ find a function

$$f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$$
$$f(x_i) = y_i, \quad \forall i$$

- If input and output data are sequences, the **attention mechanism** can provide state of the art performance
    - e.g. NLP, receptive blocks in vision, etc.

Functional data

- For many physical applications, we are presented with data from the world coming from a function over some domain.

- **Function of time**

  Trajectories from a continuous time dynamical system

$$s : [0, T] \to \mathbb{R}^d$$

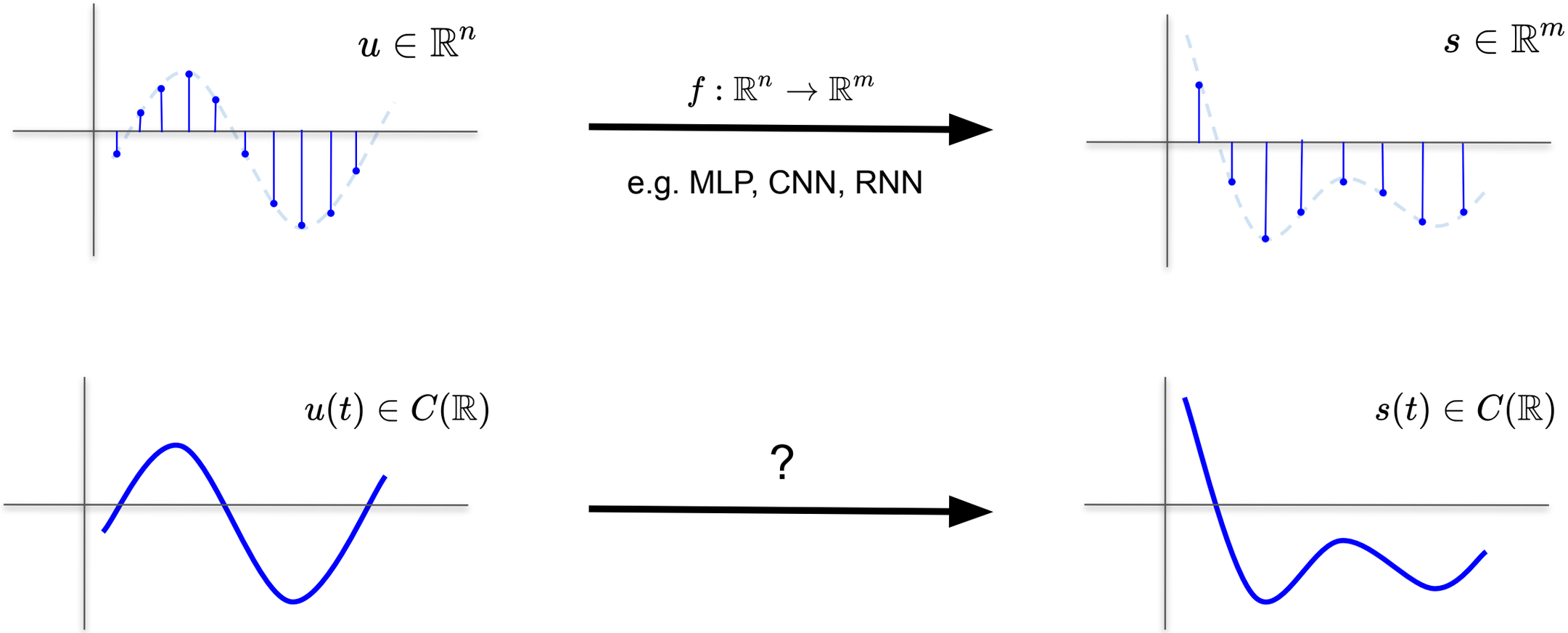
- **Function of space**

  Measurements over a continuous spatial domain $D \subset \mathbb{R}^n$

$$u : D \to \mathbb{R}^d$$

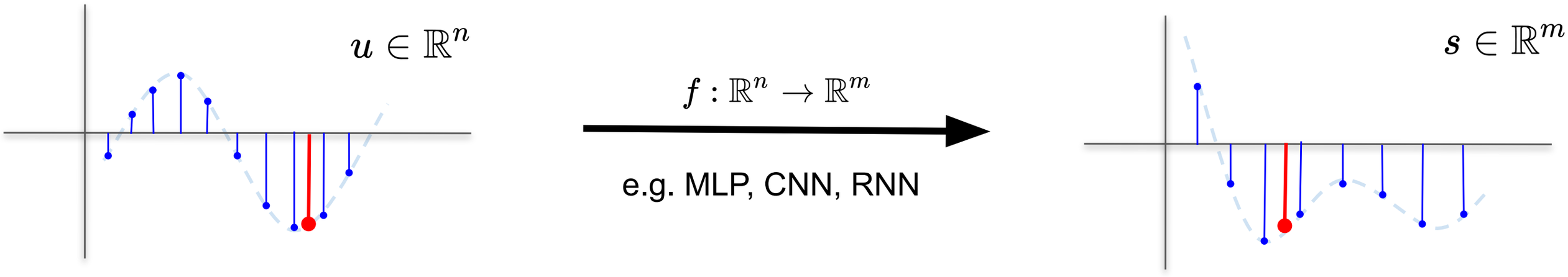
How to learn on function spaces?

- Take discrete measurements of functional data and use standard ML models on the finite dimensional discretization



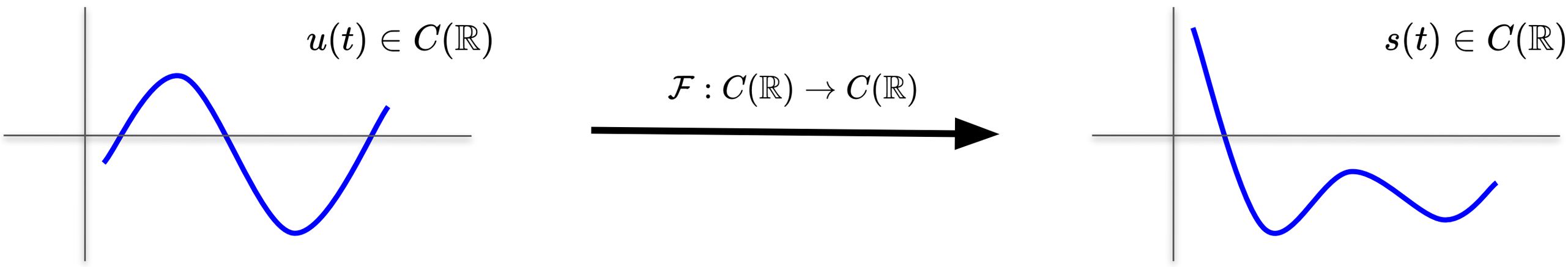
- Model is not built to accept varying numbers of measurements or new measurement locations
- We are completely constrained to our initial choice of discretization

How to learn on function spaces?



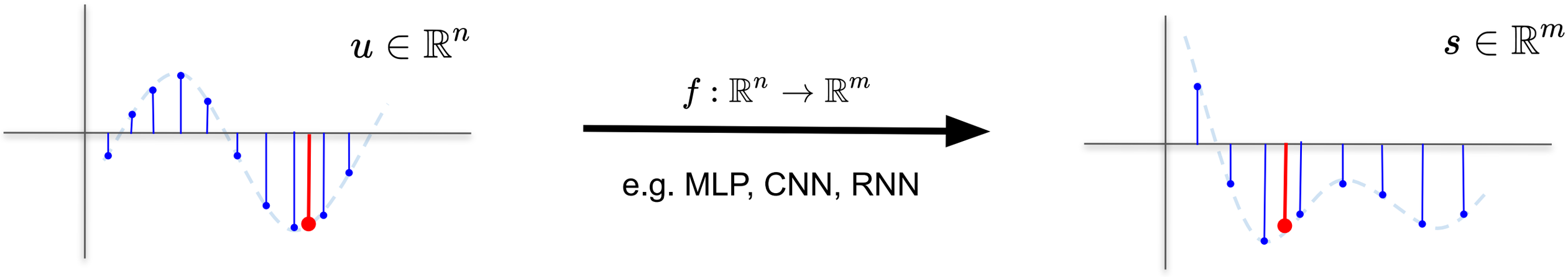
- Model is not built to accept varying numbers of measurements or new measurement locations

- We are completely constrained to our initial choice of discretization



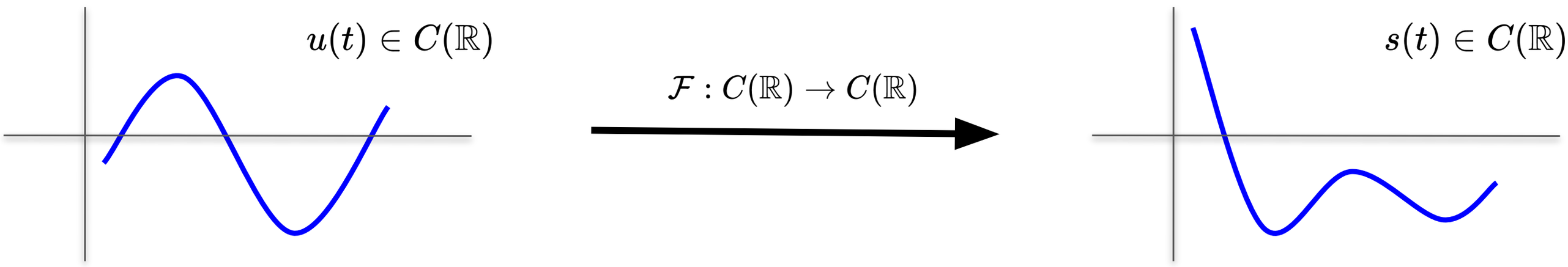
Can we instead learn operators between function spaces directly?

How to learn on function spaces?



- Model is not built to accept varying numbers of measurements or new measurement locations

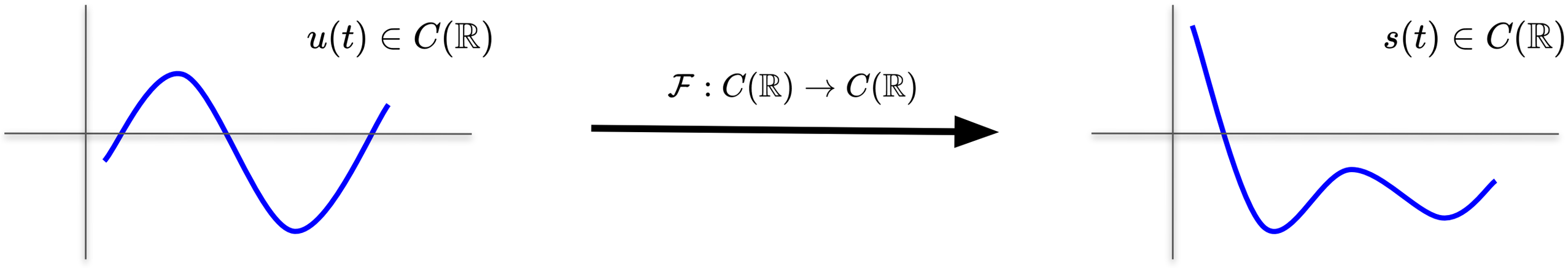
- We are completely constrained to our initial choice of discretization

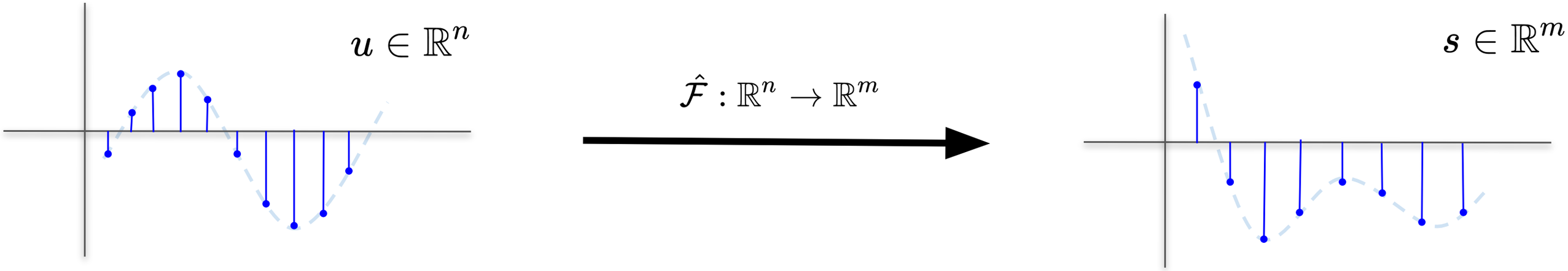


Can we instead learn operators between function spaces directly?

How to learn on function spaces?

But don't we always have to work with discrete data?





Formulating the architecture in function spaces allows different discretizations to approximate the same operator without rebuilding/retraining the model

What can we use this for?

- ODEs with control input

$$u(x) \mapsto s(t) : \begin{cases} \dot{s} = f(s, u(x)) \\ s(0) = s_0 \end{cases}$$

- PDE forward operator

$$u(x) \mapsto s(t, y) : \begin{cases} L(s(t, y)) = f(t, y) \\ s(0, x) = u(x) \end{cases}$$

- More black box relations between functions (e.g. unknown governing PDE)

What can we use this for?

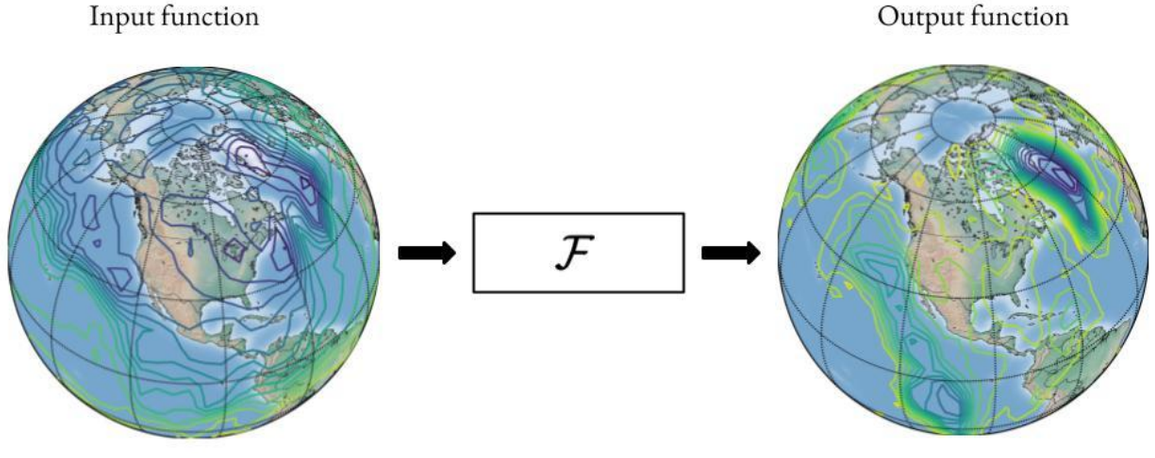
- ODEs with control input

$$u(x) \mapsto s(t) : \begin{cases} \dot{s} = f(s, u(x)) \\ s(0) = s_0 \end{cases}$$

- PDE forward operator

$$u(x) \mapsto s(t, y) : \begin{cases} L(s(t, y)) = f(t, y) \\ s(0, x) = u(x) \end{cases}$$

- More black box relations between functions (e.g. unknown governing PDE)



Supervised operator learning

- Input functions from a domain $\mathcal{X} \subset \mathbb{R}^{d_x}$ to $\mathbb{R}^{d_u}$

$$u : \mathcal{X} \to \mathbb{R}^{d_u} \qquad u(x)$$

$$u \in C(\mathcal{X}, \mathbb{R}^{d_u})$$

"input function location"

- Output functions from a domain $\mathcal{Y} \subset \mathbb{R}^{d_y}$ to $\mathbb{R}^{d_s}$

$$s : \mathcal{Y} \to \mathbb{R}^{d_s} \qquad s(y)$$

$$s \in C(\mathcal{Y}, \mathbb{R}^{d_s})$$

"query" or "query location"

- Given a dataset of N pairs of input and output functions

$$\{(u^1, s^1), \ldots, (u^N, s^N)\}$$

learn an operator

$$\mathcal{F} : C(\mathcal{X}, \mathbb{R}^{d_u}) \to C(\mathcal{Y}, \mathbb{R}^{d_s})$$

such that

$$\mathcal{F}(u^i) = s^i, \quad \forall i$$

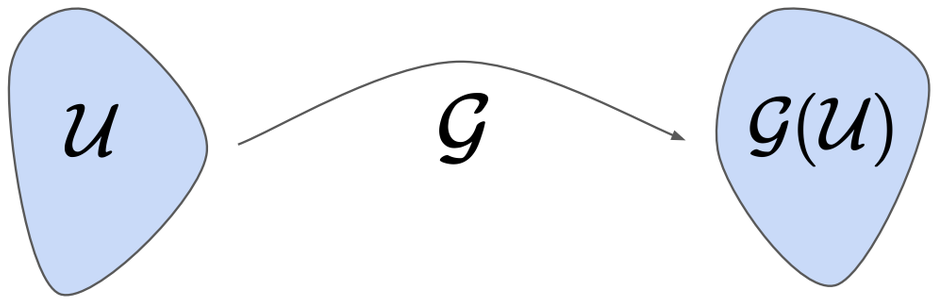
How do we design an architecture for learning operators?

$$F(u)(y) = \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k^\top y + \zeta_k)$$

Theorem (Chen & Chen '95): If $\mathcal{X} \subset \mathbb{R}^{d_x}, \mathcal{Y} \subset \mathbb{R}^{d_y}, \mathcal{U} \subset C(U, \mathbb{R})$ are all compact, given a continuous $\mathcal{G} : \mathcal{U} \to C(D, \mathbb{R})$, for any $\epsilon > 0$, there exists F of the above form such that

$$\sup_{u \in \mathcal{U}} \sup_{y \in \mathcal{Y}} \| F(u)(y) - \mathcal{G}(u)(y) \| < \epsilon$$

- Consider a continuous operator $\mathcal{G} : \mathcal{U} \to C(\mathcal{Y}, \mathbb{R})$ with $\mathcal{U} \subset C(\mathcal{X}, \mathbb{R})$ compact

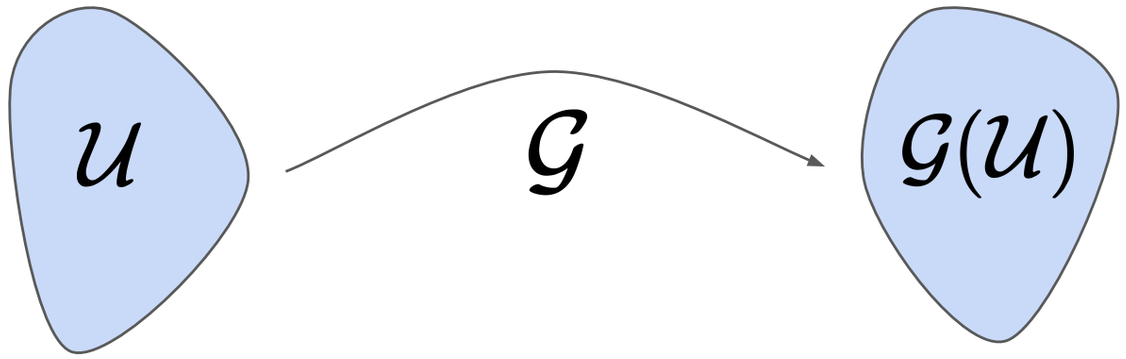


- Given $u \in \mathcal{U}$, we want to be able to evaluate $\mathcal{G}(u)(y)$ for any $y \in \mathcal{Y}$

Thus, our aim is to approximate functions in the set $\mathcal{G}(\mathcal{U})$

Chen, Tianping, and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems." *IEEE Transactions on Neural Networks* 6.4 (1995): 911-917.

The Chen & Chen theorem



Since $\mathcal{U}$ is compact and $\mathcal{G}$ is continuous, $\mathcal{G}(\mathcal{U})$ is also compact

We can find a finite dimensional subspace of $\mathcal{G}(\mathcal{U})$ that is ϵ-close to any $\mathcal{G}(u)$

This subspace of functions has a basis of functions $\{t_1(y), \ldots, t_d(y)\}$

So for every $\mathcal{G}(u)$ there are coordinates in this basis $\{b_1(u), \ldots, b_d(u)\}$ such that

$$\left\| \sum_{k=1}^{d} b_k(u) t_k(y) - \mathcal{G}(u)(y) \right\|_\infty < \epsilon$$

Chen, Tianping, and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems." *IEEE Transactions on Neural Networks* 6.4 (1995): 911-917.

The Chen & Chen architecture

$$\sum_{k=1}^{d} b_k(u) t_k(y)$$

Approximate with neural network sending

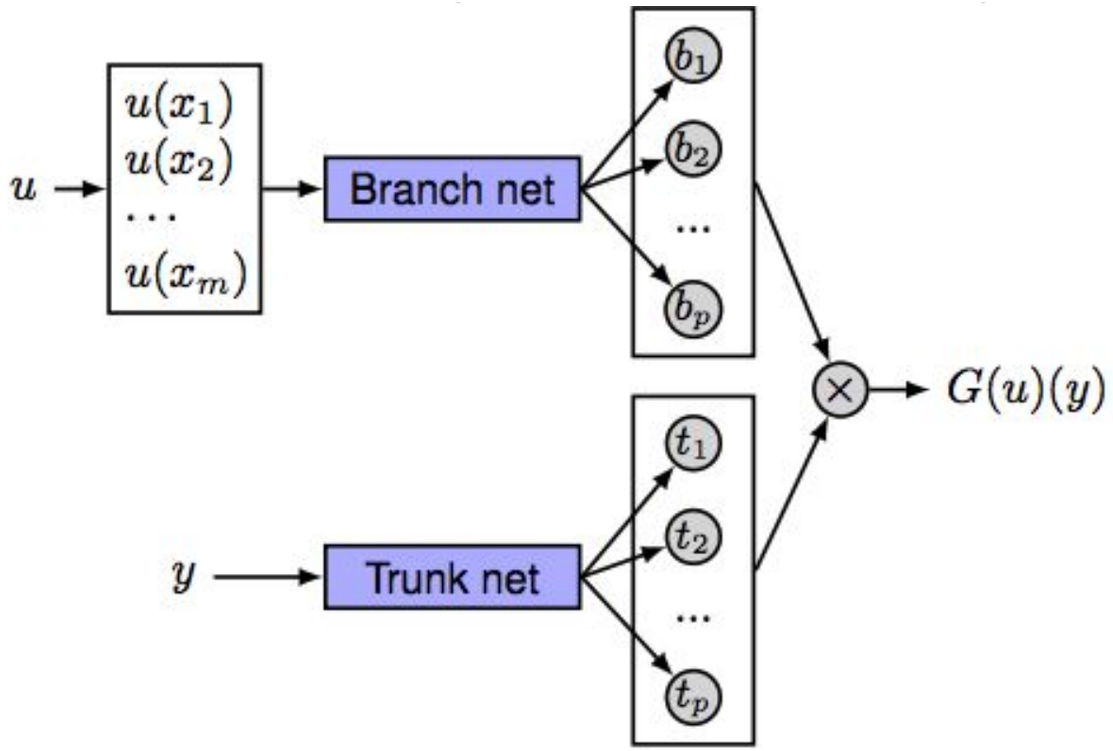
$$\begin{bmatrix} u(x_1) \\ \vdots \\ u(x_m) \end{bmatrix} \longmapsto \begin{bmatrix} b_1(u) \\ \vdots \\ b_d(u) \end{bmatrix}$$

$$b_k(u) = \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)$$

Approximate with neural network sending

$$y \longmapsto \begin{bmatrix} t_1(y) \\ \vdots \\ t_d(y) \end{bmatrix}$$

$$t_k(y) = \sigma(w_k^\top y + \zeta_k)$$

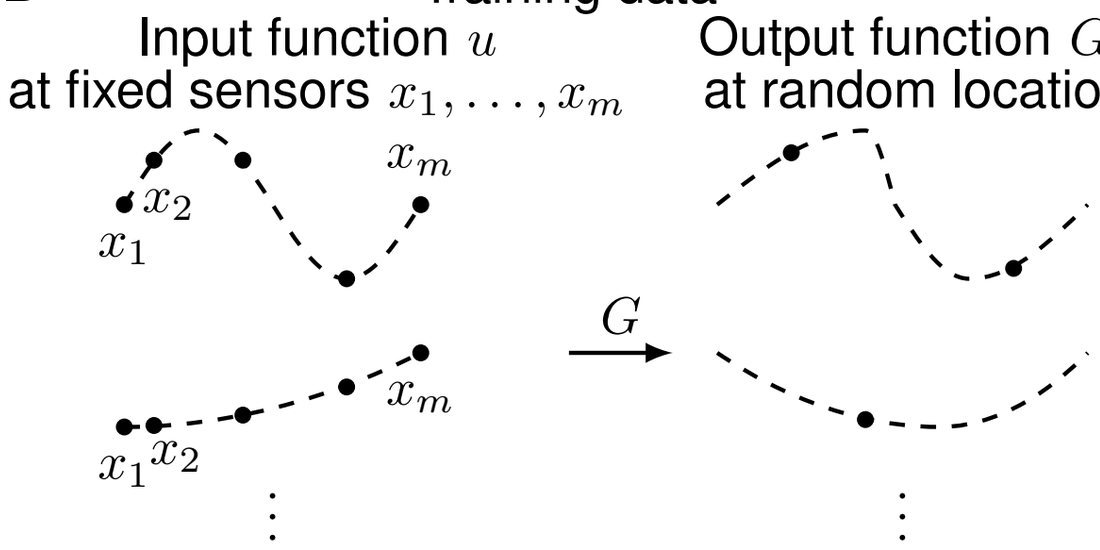


- But nowadays everything is a deep network…
- By making the branch and trunk deep we obtain a "deep learning" extension: the *DeepONet* (Lu et al., 2021)

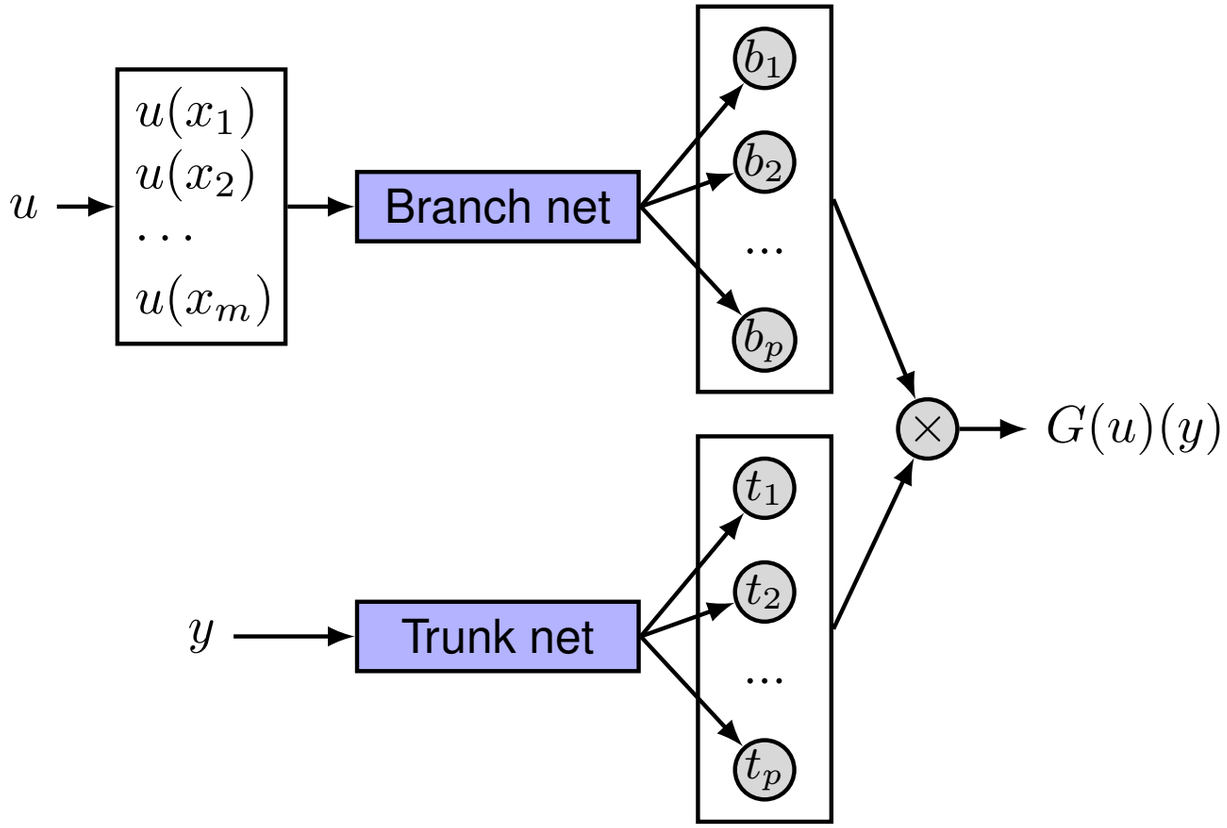
Chen, Tianping, and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems." *IEEE Transactions on Neural Networks* 6.4 (1995): 911-917.

Learning operators with DeepOnets

B Training data



D Unstacked DeepONet



Training data:

$$[\boldsymbol{u}, \boldsymbol{y}, G(\boldsymbol{u})(\boldsymbol{y})] = \left[\begin{bmatrix} \vdots \\ \boldsymbol{u}^{(i)}(\boldsymbol{x}_1), \boldsymbol{u}^{(i)}(\boldsymbol{x}_2), \cdots, \boldsymbol{u}^{(i)}(\boldsymbol{x}_m) \\ \boldsymbol{u}^{(i)}(\boldsymbol{x}_1), \boldsymbol{u}^{(i)}(\boldsymbol{x}_2), \cdots, \boldsymbol{u}^{(i)}(\boldsymbol{x}_m) \\ \vdots \\ \boldsymbol{u}^{(i)}(\boldsymbol{x}_1), \boldsymbol{u}^{(i)}(\boldsymbol{x}_2), \cdots, \boldsymbol{u}^{(i)}(\boldsymbol{x}_m) \\ \vdots \end{bmatrix}, \begin{bmatrix} \vdots \\ \boldsymbol{y}_1^{(i)} \\ \boldsymbol{y}_2^{(i)} \\ \vdots \\ \boldsymbol{y}_P^{(i)} \\ \vdots \end{bmatrix}, \begin{bmatrix} \vdots \\ G(\boldsymbol{u}^{(i)})(\boldsymbol{y}_1^{(i)}) \\ G(\boldsymbol{u}^{(i)})(\boldsymbol{y}_2^{(i)}) \\ \vdots \\ G(\boldsymbol{u}^{(i)})(\boldsymbol{y}_P^{(i)}) \\ \vdots \end{bmatrix} \right]$$

Training loss:

$$\mathcal{L}_{\text{operator}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left| G_\theta\left(u^{(i)}\right)\left(y^{(i)}\right) - s^{(i)}\left(y^{(i)}\right) \right|^2$$

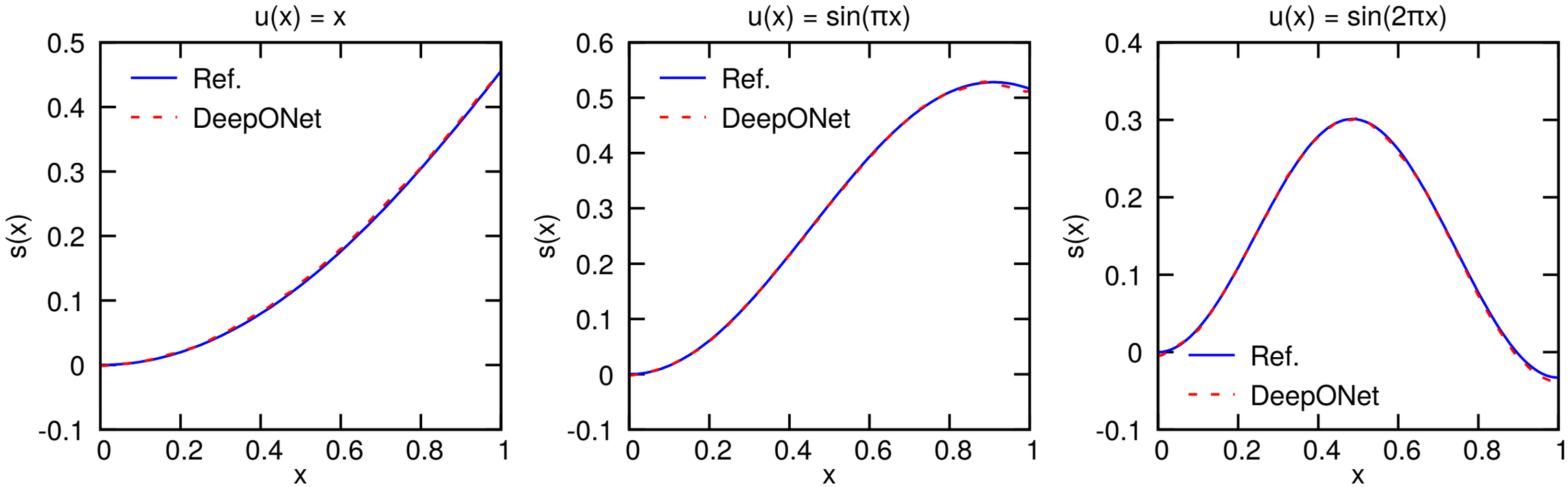
Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence, 3(3), 218-229.

Learning the anti-derivative operator

Learning the anti-derivative operator:

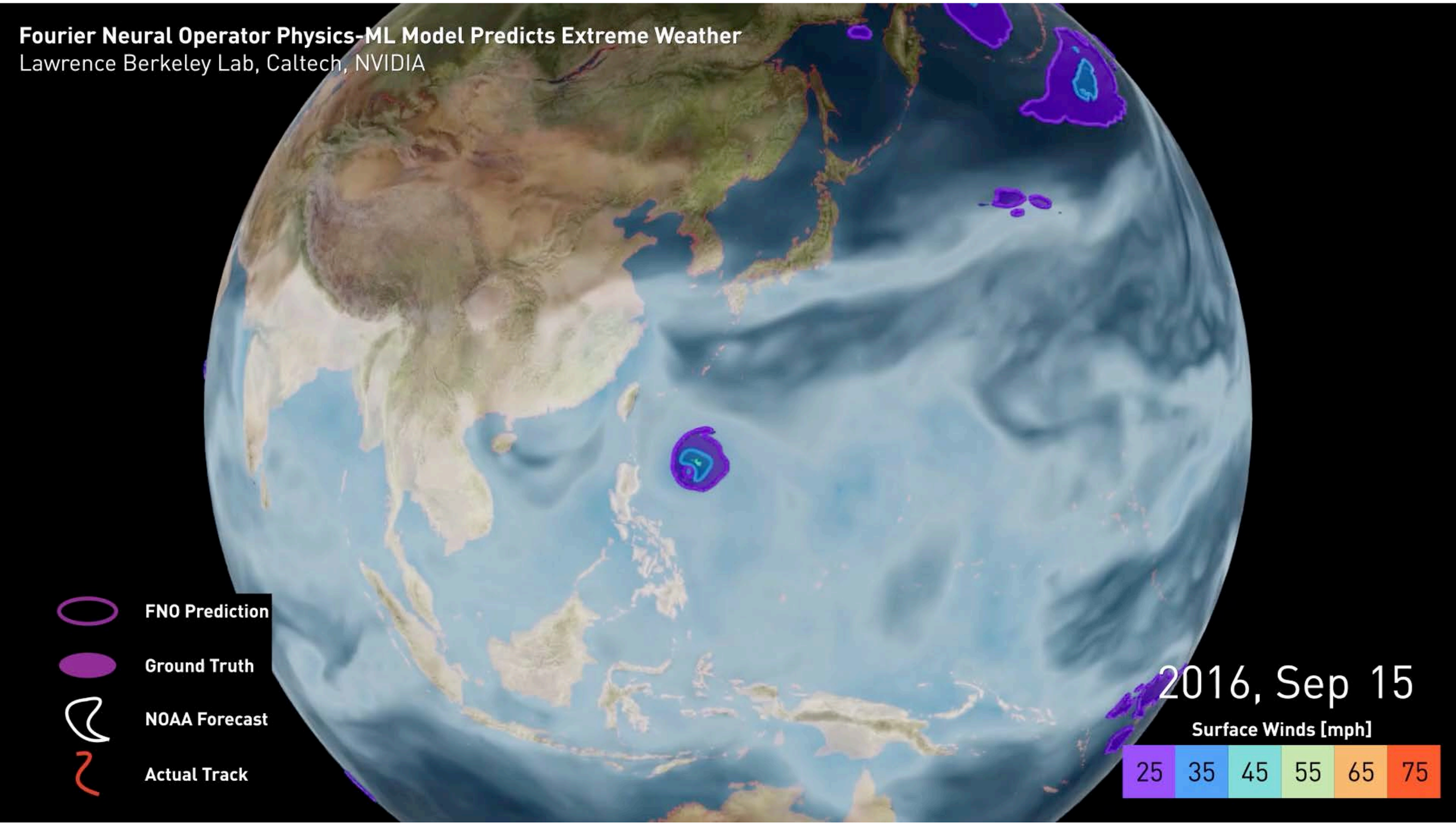
$$\frac{ds(x)}{dx} = u(x), \quad x \in [0,1]$$
$$s(0) = 0$$

$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0,1]$$



Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence, 3(3), 218-229.

Amortizing the cost of numerical solvers



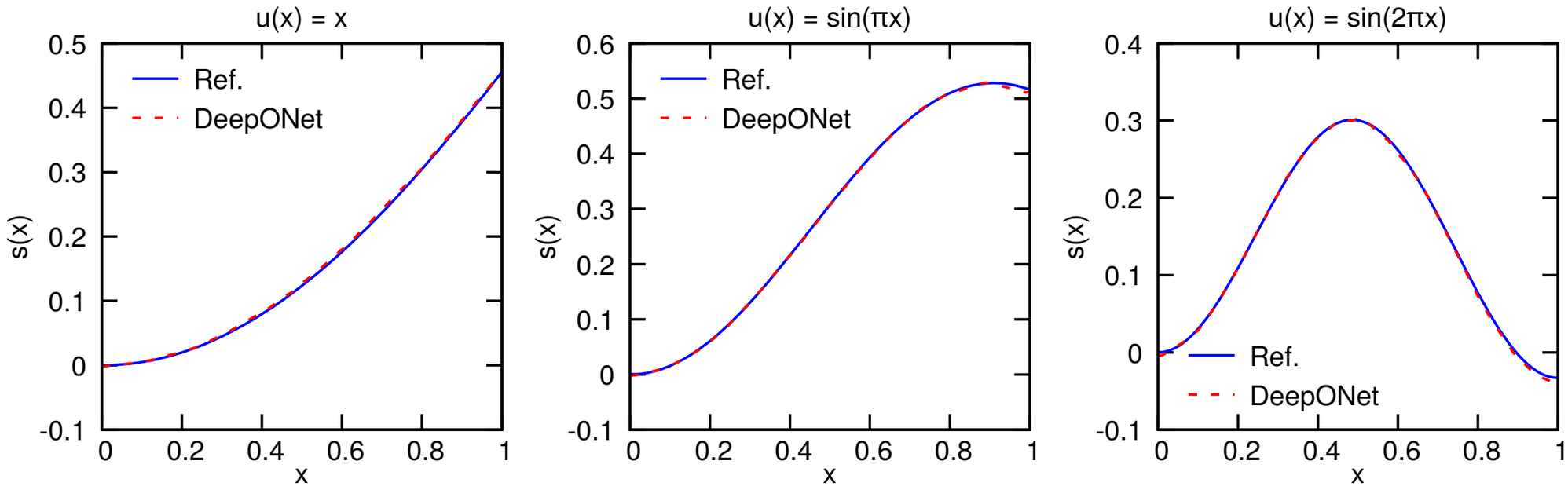
Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020). Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895.

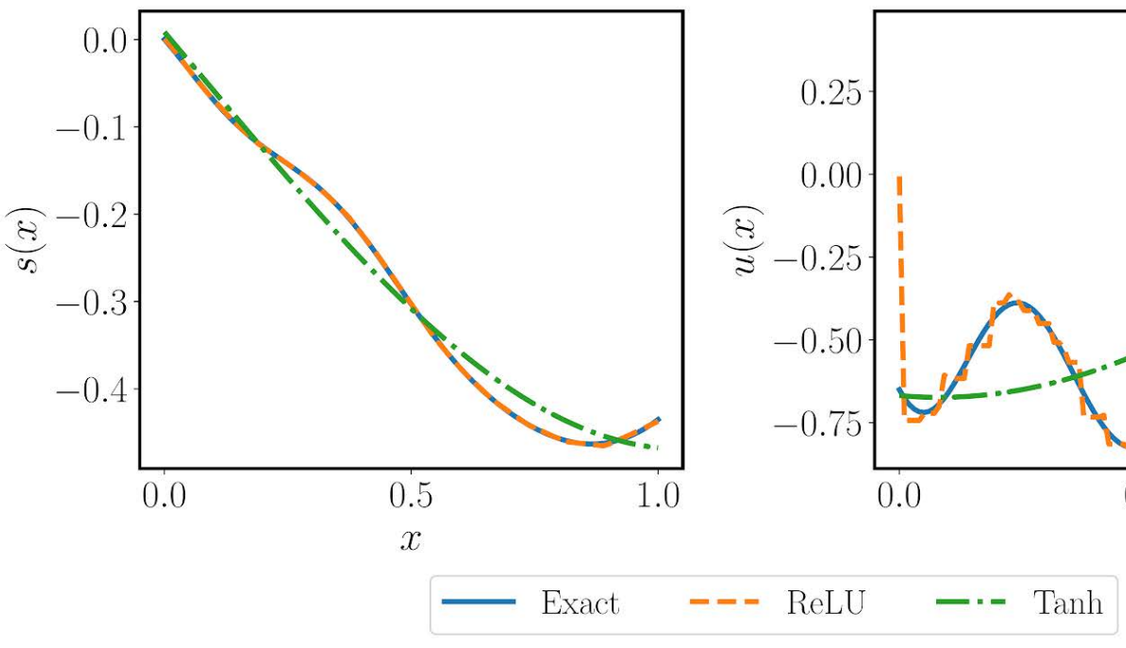
Learning the anti-derivative operator

Learning the anti-derivative operator:

$$\frac{ds(x)}{dx} = u(x), \quad x \in [0,1]$$
$$s(0) = 0$$

$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0,1]$$



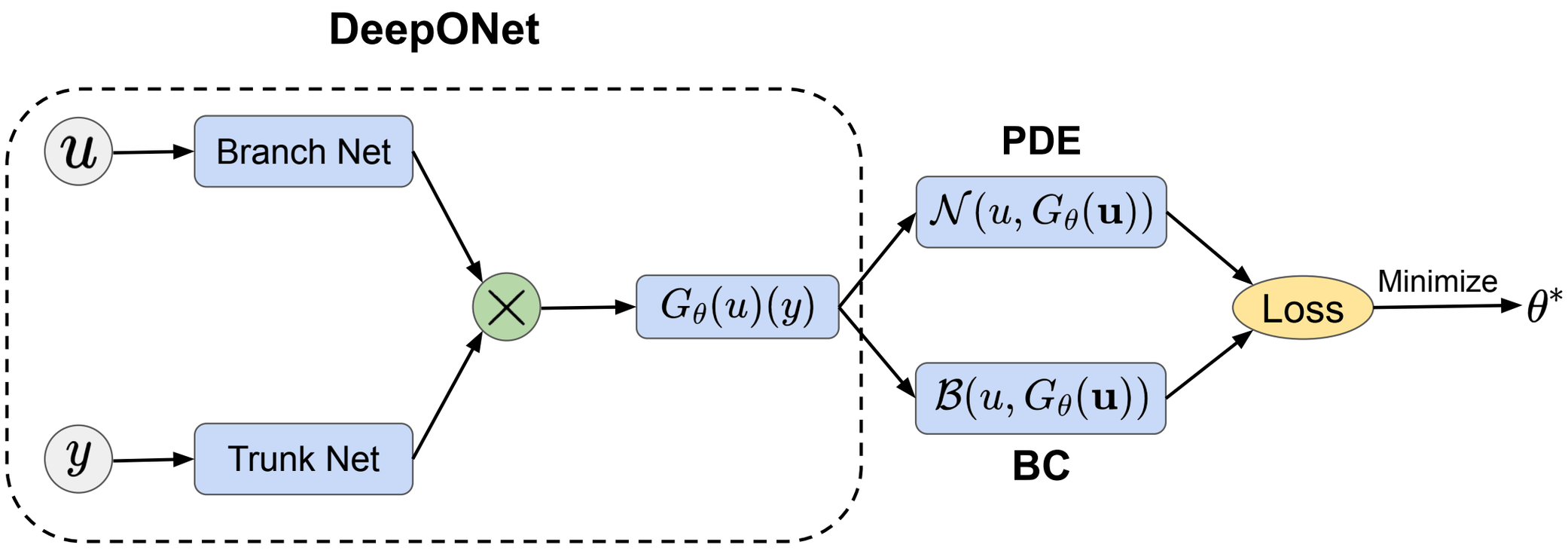


Operator learning is great, but:

- It requires large training data-sets consisting of paired input-output observations.
- Predictions may be incompatible with respect to the true operator that generated the data.

Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence, 3(3), 218-229.

Physics-informed DeepOnets



$$\mathcal{L}(\theta) = \mathcal{L}_{\text{operator}}(\theta) + \mathcal{L}_{\text{physics}}(\theta)$$

$$\mathcal{L}_{\text{operator}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left|G_\theta\left(u^{(i)}\right)\left(y^{(i)}\right) - s^{(i)}\left(y^{(i)}\right)\right|^2$$

$$\mathcal{L}_{\text{physics}}(\theta) = \frac{1}{NQm}\sum_{i=1}^{N}\sum_{j=1}^{Q}\sum_{k=1}^{m}\left|\mathcal{N}\left(u^{(i)}(x_k), G_\theta\left(u^{(i)}\right)\left(y_j^{(i)}\right)\right)\right|^2$$

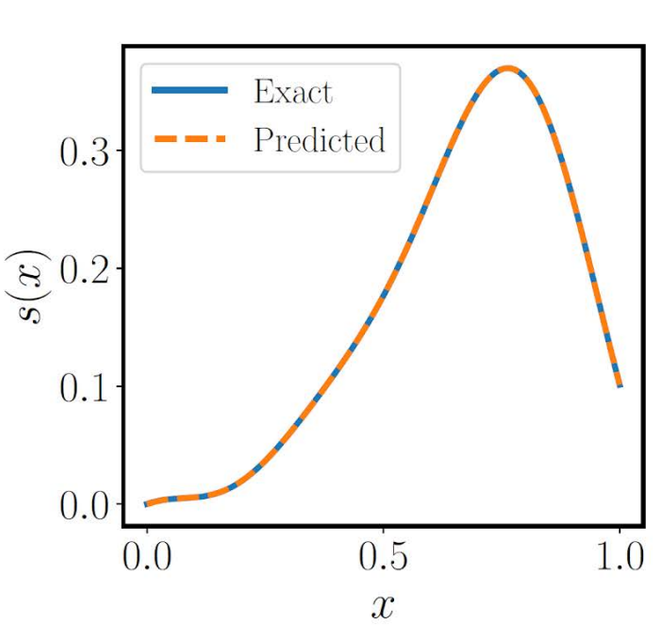
Wang, S., Wang, H., & Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets

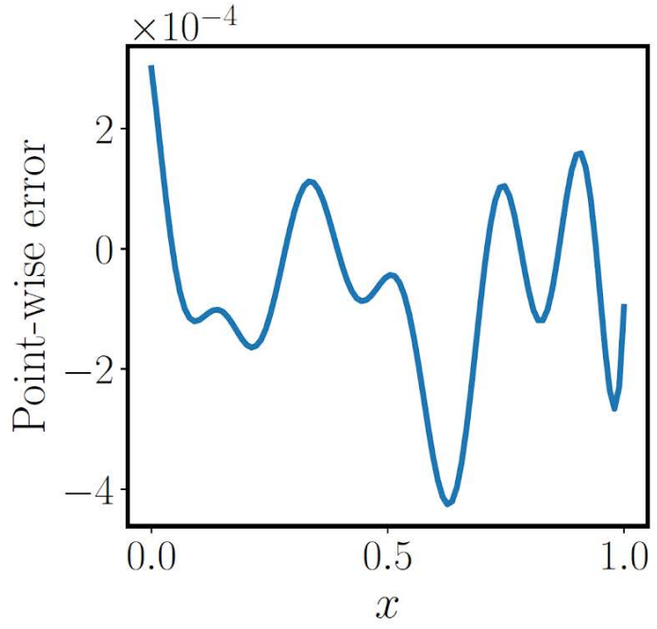
Data-efficient operator learning

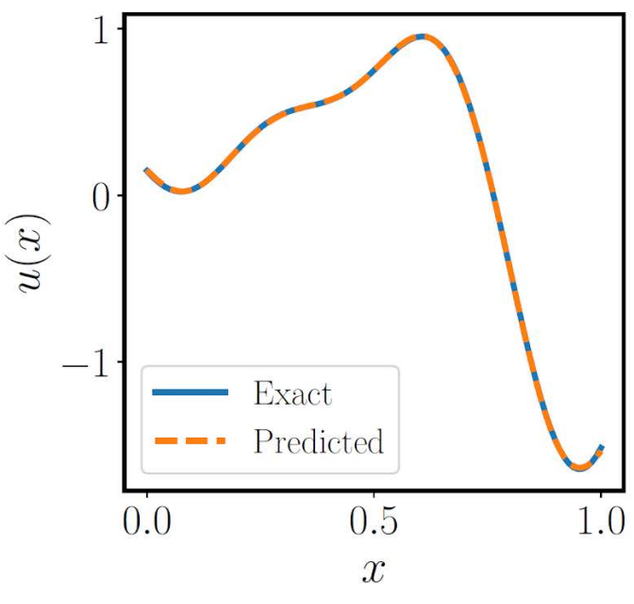
Learning the anti-derivative operator:

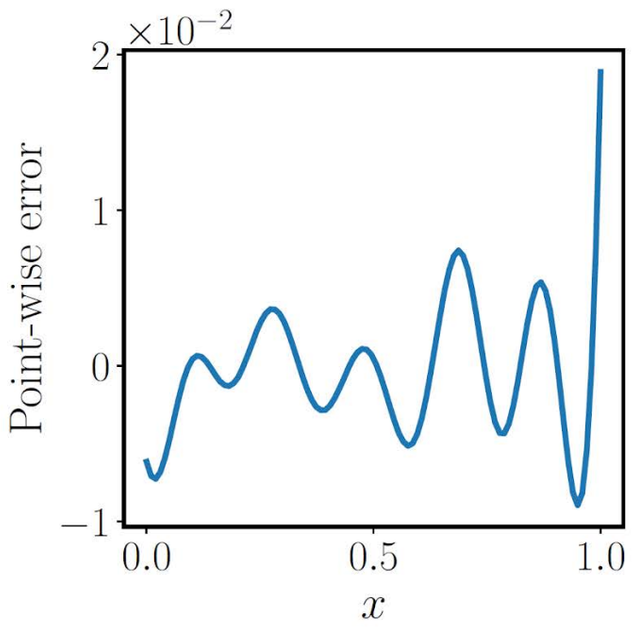
$$\frac{ds(x)}{dx} = u(x), \quad x \in [0,1]$$
$$s(0) = 0$$

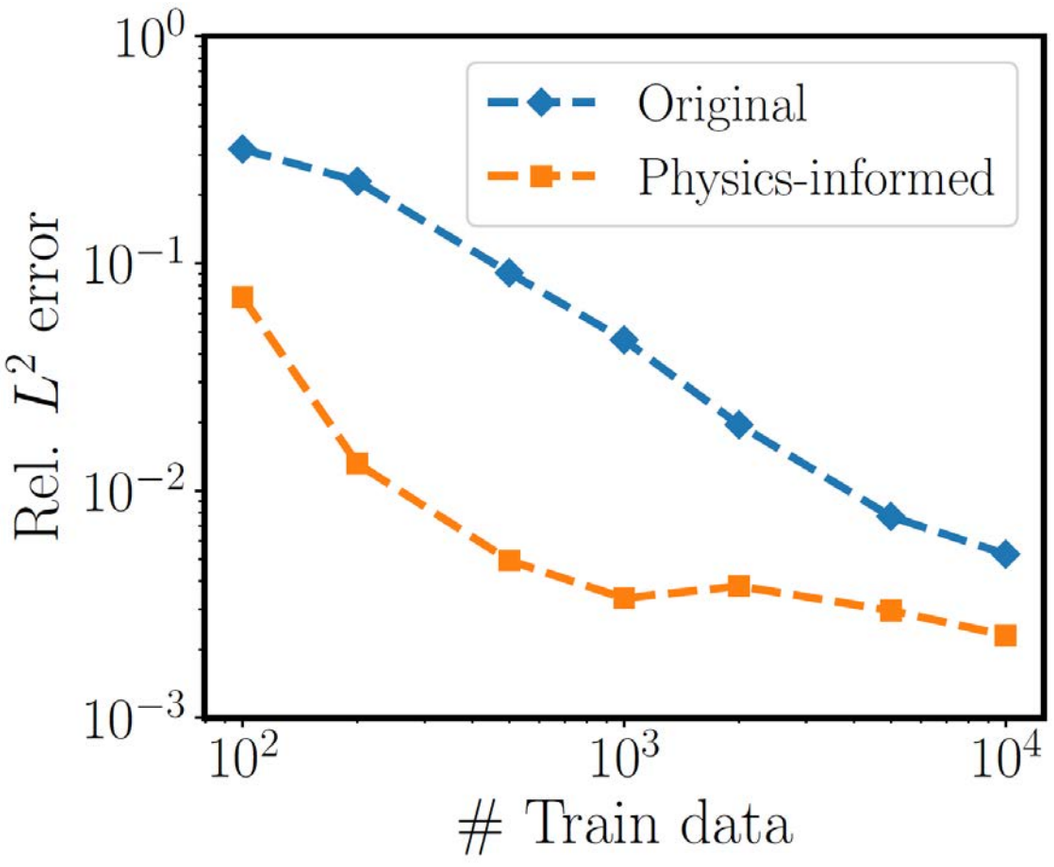
$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0,1]$$











Wang, S., Wang, H., & Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets

SCIENCE ADVANCES | RESEARCH ARTICLE

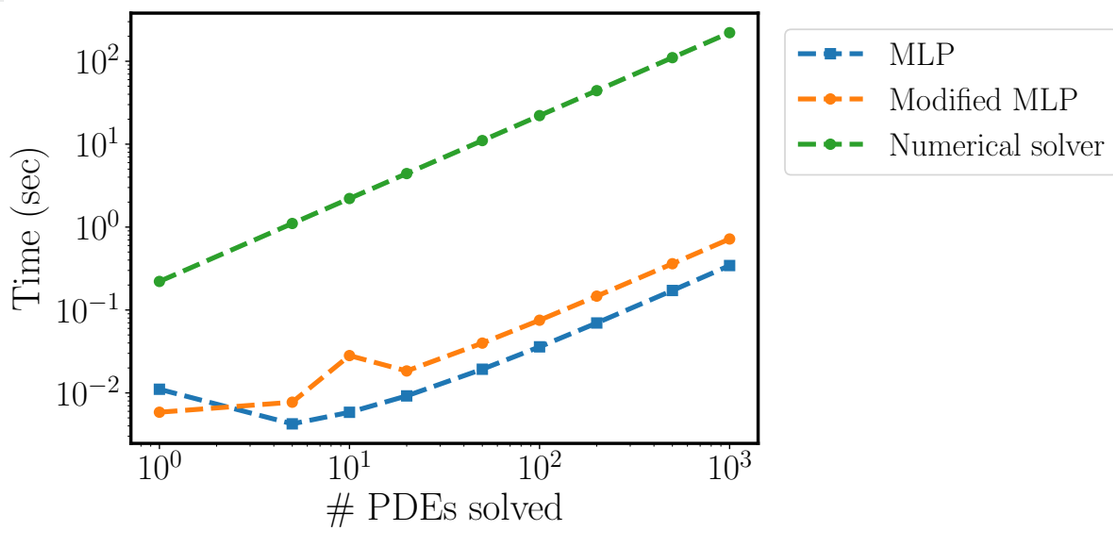
APPLIED PHYSICS

Learning the solution operator of parametric partial differential equations with physics-informed DeepONets

Sifan Wang[1], Hanwen Wang[1], Paris Perdikaris[2]*

Table 1. Summary of benchmarks for assessing the performance of physics-informed DeepONets across various types of parametric differential equations. The reported test error corresponds to the relative L^2 prediction error of the trained model, averaged over all examples in the test dataset (see eq. S20).

Governing law	Equation form	Random input	Test error
Linear ODE	$\frac{ds(x)}{dx} = u(x)$	Forcing terms	0.33 ± 0.32%
Diffusion reaction	$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + k s^2 + u(x)$	Source terms	0.45 ± 0.16%
Burgers'	$\frac{\partial s}{\partial t} + s\frac{\partial s}{\partial x} - \nu\frac{\partial^2 s}{x^2} = 0$	Initial conditions	1.38 ± 1.64%
Advection	$\frac{\partial s}{\partial t} + u\frac{\partial s}{\partial x} = 0$	Variable coefficients	2.24 ± 0.68%
Eikonal	$\| \nabla s \|_2 = 0$	Domain geometries	0.42 ± 0.11%



Self-supervised operator learning

$$\frac{ds}{dt} + s\frac{ds}{dx} - \nu\frac{d^2s}{dx^2} = 0, \quad (x,t) \in (0,1) \times (0,1],$$

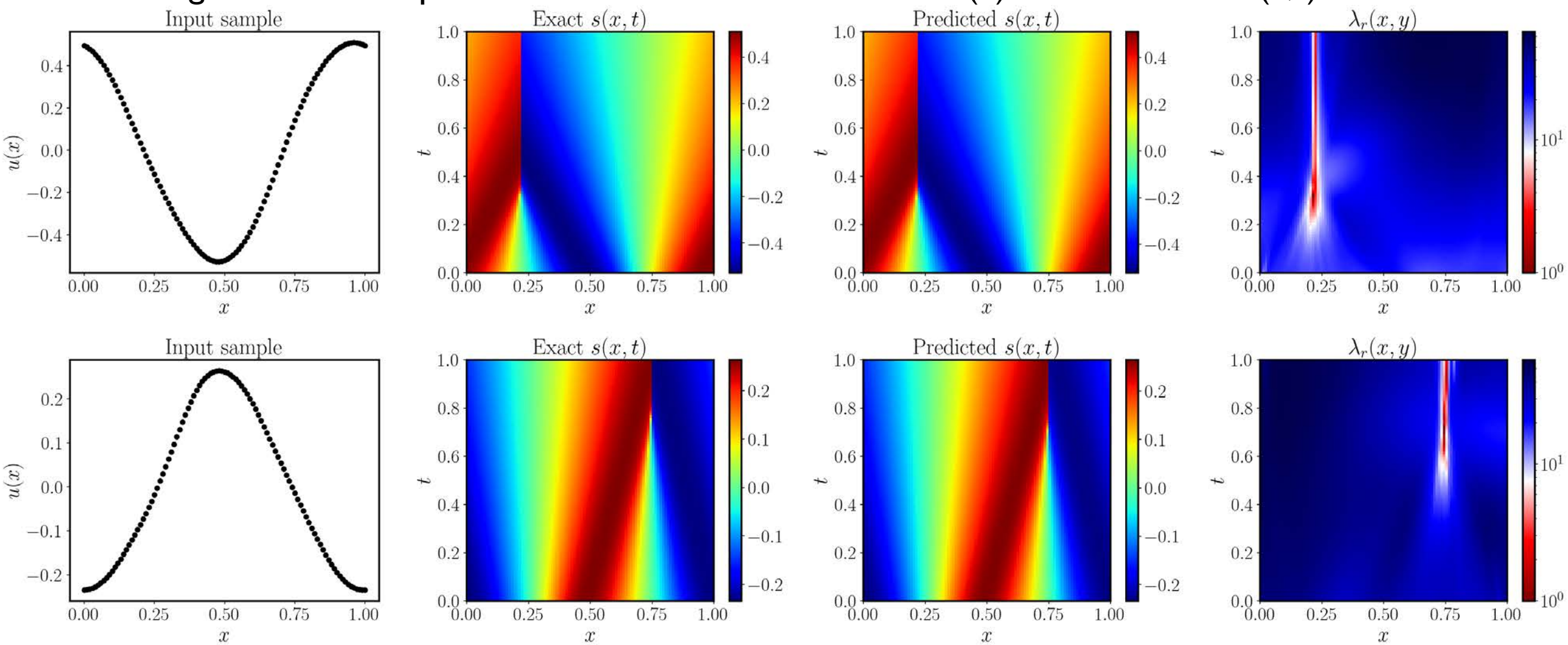
$$s(x,0) = u(x), \quad x \in (0,1),$$

$$\nu = 10^{-4}$$

$$s(0,t) = s(1,t)$$

$$\frac{ds}{dx}(0,t) = \frac{ds}{dx}(1,t)$$

Goal: Learning the solution operator G from the initial condition u(x) to the solution s(x,t).



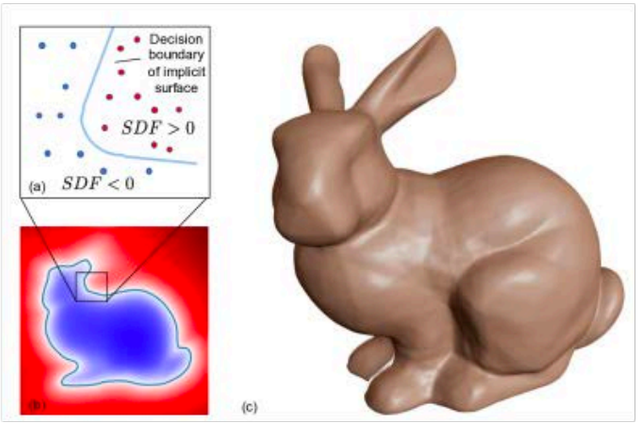
Model \ Viscosity	$\nu = 10^{-2}$	$\nu = 10^{-3}$	$\nu = 10^{-4}$
Plain PI-DeepONet	$3.48\% \pm 3.79\%$	$26.91\% \pm 12.26\%$	$29.21\% \pm 12.38\%$
Improved PI-DeepONet	$1.03\% \pm 1.35\%$	$3.69\% \pm 3.63\%$	$7.95\% \pm 5.29\%$

Wang, S., Wang, H., & Perdikaris, P. (2021). Improved architectures and training algorithms for deep operator networks

Parametrized geometries

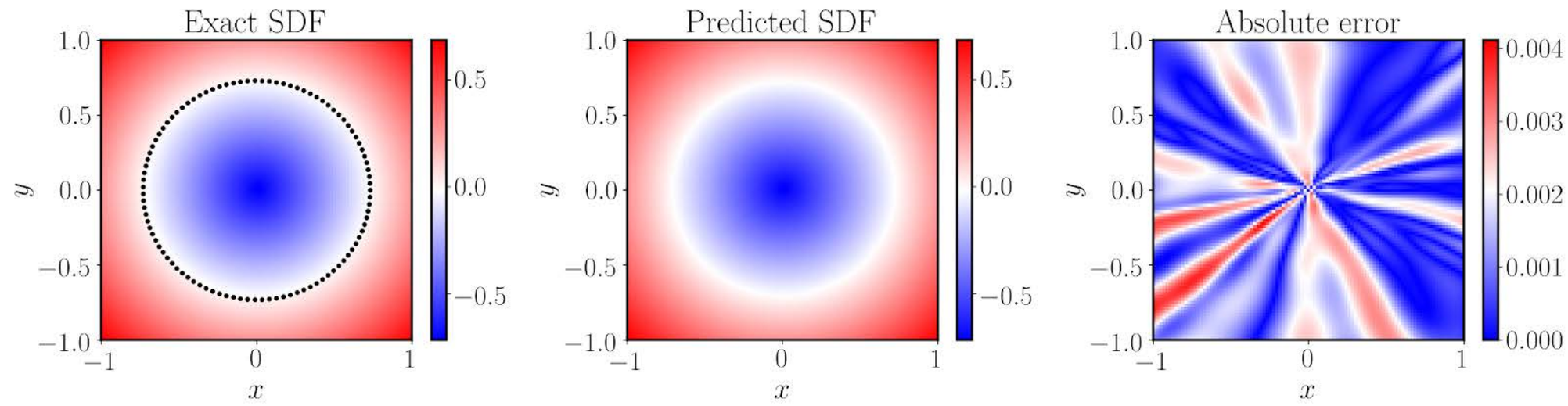
$$\|\nabla s(x)\|_2 = 1$$
$$s(x) = 0, \quad x \in \partial\Omega$$

$$s(\boldsymbol{x}) = \begin{cases} d(\boldsymbol{x}, \partial\Omega) & \text{if } \boldsymbol{x} \in \Omega \\ -d(\boldsymbol{x}, \partial\Omega) & \text{if } \boldsymbol{x} \in \Omega^c \end{cases}$$

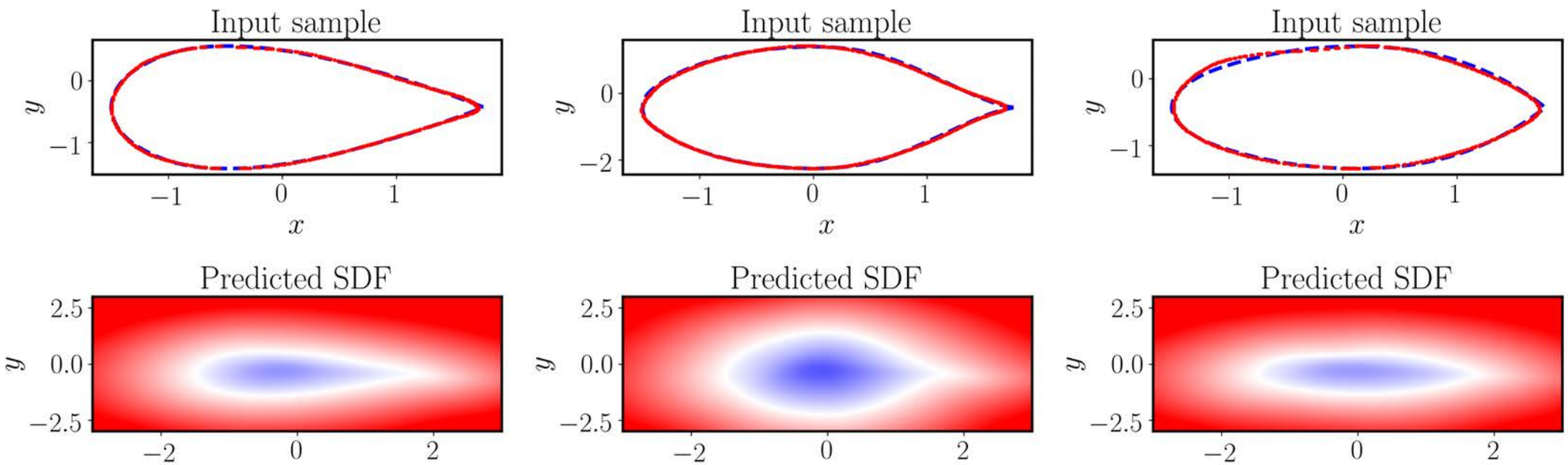


Goal: Learn the solution operator G from a well-behaved closed curve Γ to the associated SDF.

Case I: Circles



Case II: 2D airfoils



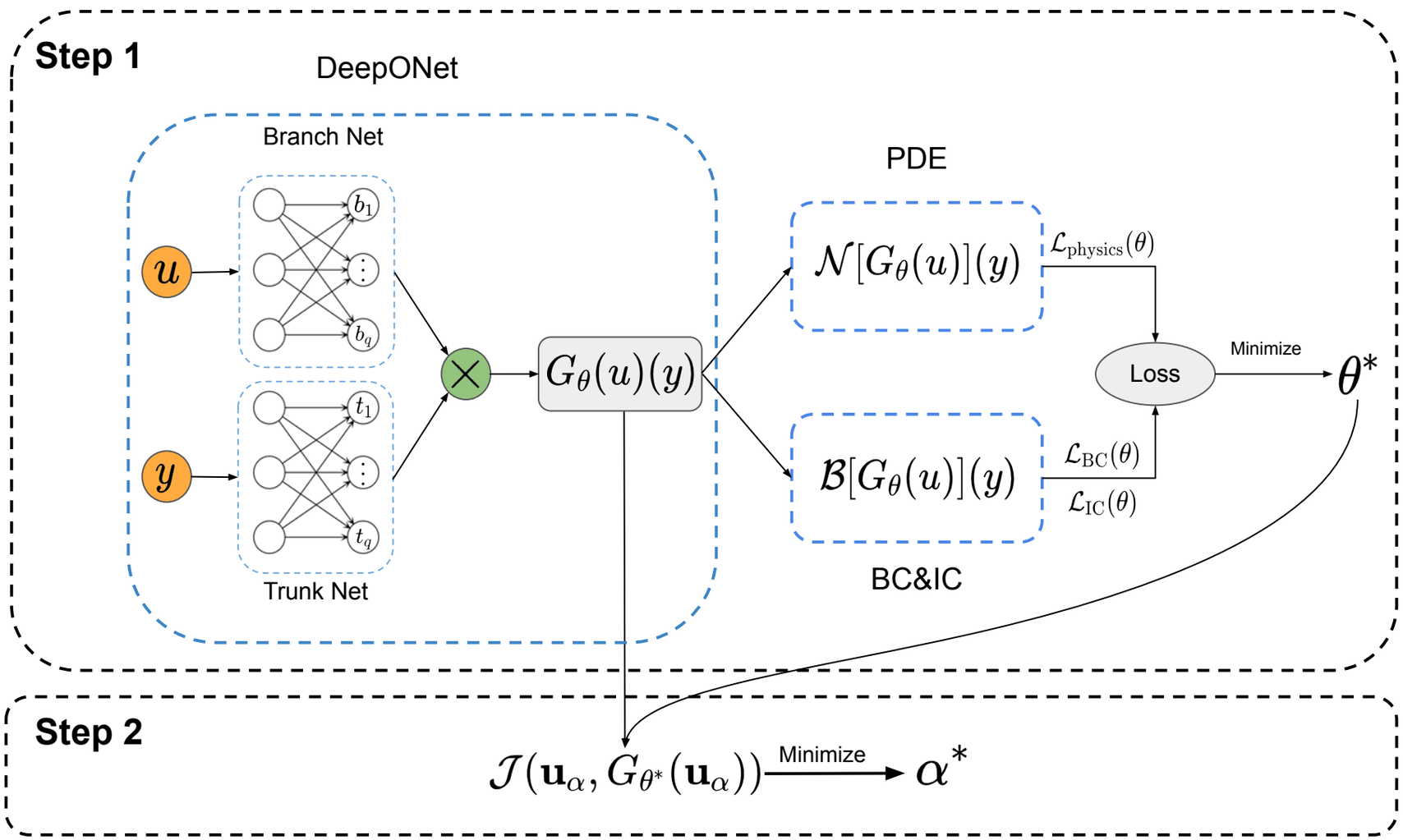
Wang, S., Wang, H., & Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets

PDE-constrained optimization

Consider the following general PDE-constrained optimization problem

$$\begin{aligned} \min \quad & \mathcal{J}(\boldsymbol{u}, \boldsymbol{s}) \\ \text{s.t.} \quad & \mathcal{E}(\boldsymbol{u}, \boldsymbol{s}) = 0 \\ & \boldsymbol{u} \in U_{ad}, \boldsymbol{s} \in S_{ad}, \end{aligned}$$

where $\mathcal{J} : U \times S \to \mathbb{R}$ denotes a cost function to be minimized, and $\mathcal{E} : U \times S \to V$ represents a system of PDEs subject to appropriate initial and boundary conditions.



Train a self-supervised physics-informed DeepONet to learn the PDE solution operator:

$$G_\theta : \mathcal{U} \to \mathcal{S}$$

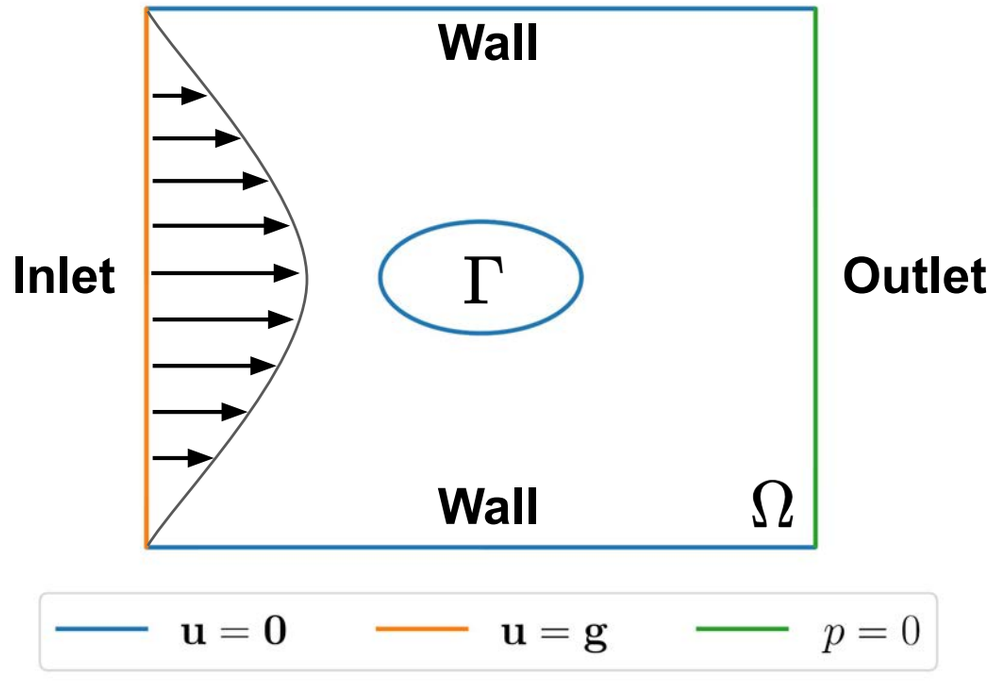
Use G_{θ^*} as a differentiable surrogate to minimize J.

* All gradients (i.e. $\partial \mathcal{L}/\partial \boldsymbol{\theta}$, $\partial \mathcal{J}/\partial \boldsymbol{\alpha}$ and $\partial \boldsymbol{s}/\partial \boldsymbol{x}$) are computed using automatic differentiation).

Drag minimization of obstacles in Stokes flow

$$-\Delta \boldsymbol{u} + \nabla p = 0, \quad (x, y) \in \Omega/\Gamma,$$
$$\nabla \cdot \boldsymbol{u} = 0, \quad (x, y) \in \Omega/\Gamma,$$

$$\boldsymbol{u} = \mathbf{0}, \quad (x, y) \in \Lambda_1 \cup \partial\Gamma,$$
$$\boldsymbol{u} = \boldsymbol{g}, \quad (x, y) \in \Lambda_2$$
$$p = 0, \quad (x, y) \in \Lambda_3,$$



Goal: Learn the operator G that maps different obstacle shapes to the associated PDE solution triplet (u, v, p)

Shape parametrization:

In this example, we parameterize $\partial\Gamma$ by a family of ellipses centered at $(\frac{1}{2}, \frac{1}{2})$, i.e

$$\partial\Gamma = \partial\boldsymbol{\Gamma}(\phi) = (a\cos(\phi) + \frac{1}{2}, b\sin(\phi) + \frac{1}{2}), \quad \phi \in [0, 2\pi),$$

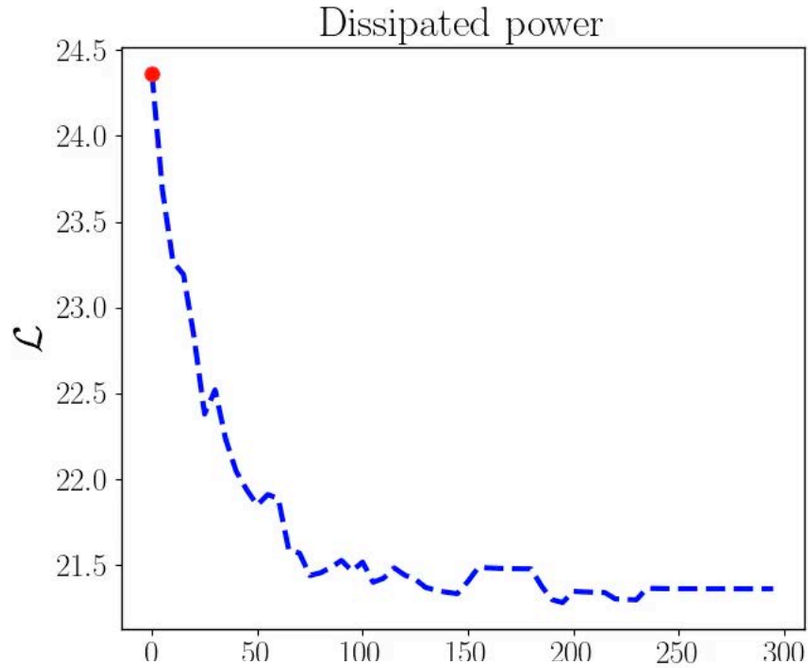
where $a, b > 0$ denote the length of the long or the short axis. We then approximate the solution operator by a DeepONet $G_{\boldsymbol{\theta}}$ with 3-dimensional vector-valued outputs for representing u, v, p, respectively,

$$\partial\boldsymbol{\Gamma} \xrightarrow{G_{\boldsymbol{\theta}}} [G_{\boldsymbol{\theta}}^{(u)}, G_{\boldsymbol{\theta}}^{(v)}, G_{\boldsymbol{\theta}}^{(p)}].$$

where $\partial\boldsymbol{\Gamma} = [\partial\Gamma(\phi_1), \partial\Gamma(\phi_2), \ldots, \partial\Gamma(\phi_m)]$ denotes an input closed curve evaluated at evenly-spaced grid points $\{\phi\}_{i=1}^m$ in $[0, 2\pi]$

Drag minimization of obstacles in Stokes flow

Step 2: Parametrize the obstacle geometry (here using 100 equispaced control points), and minimize J via gradient descent, using the trained DeepONet G_{θ^*} model as a differentiable surrogate.



$$\mathcal{J}(\partial\mathbf{\Gamma}) = \int_{\Omega/\Gamma} \left(\frac{\partial G^{(u)}_{\boldsymbol{\theta}^*}(\partial\mathbf{\Gamma})}{\partial x}\right)^2 + \left(\frac{\partial G^{(u)}_{\boldsymbol{\theta}^*}(\partial\mathbf{\Gamma})}{\partial y}\right)^2 + \left(\frac{\partial G^{(v)}_{\boldsymbol{\theta}^*}(\partial\mathbf{\Gamma})}{\partial x}\right)^2 + \left(\frac{\partial G^{(v)}_{\boldsymbol{\theta}^*}(\partial\mathbf{\Gamma})}{\partial y}\right)^2 \mathrm{d}x + \alpha \left|\mathrm{Vol}(\Gamma) - \mathrm{Vol}\left(\Gamma_0\right)\right|^2 + \beta \sum_{j=1}^{2} \left|\mathrm{Bc}(\Gamma) - \mathrm{Bc}\left(\Gamma_0\right)\right|^2,$$

where $\partial\mathbf{\Gamma} = [\partial\Gamma(\phi_1), \partial\Gamma(\phi_2), \ldots, \partial\Gamma(\phi_m)]$ denotes an input closed curve evaluated at evenly-spaced grid points $\{\phi\}_{i=1}^{m}$ in $[0, 2\pi]$.