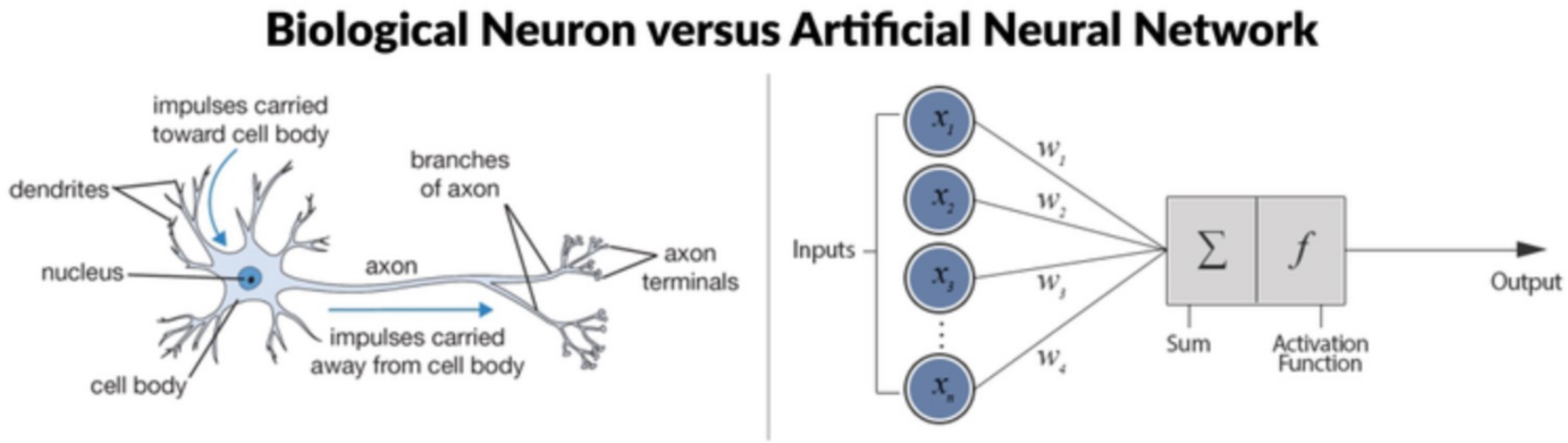
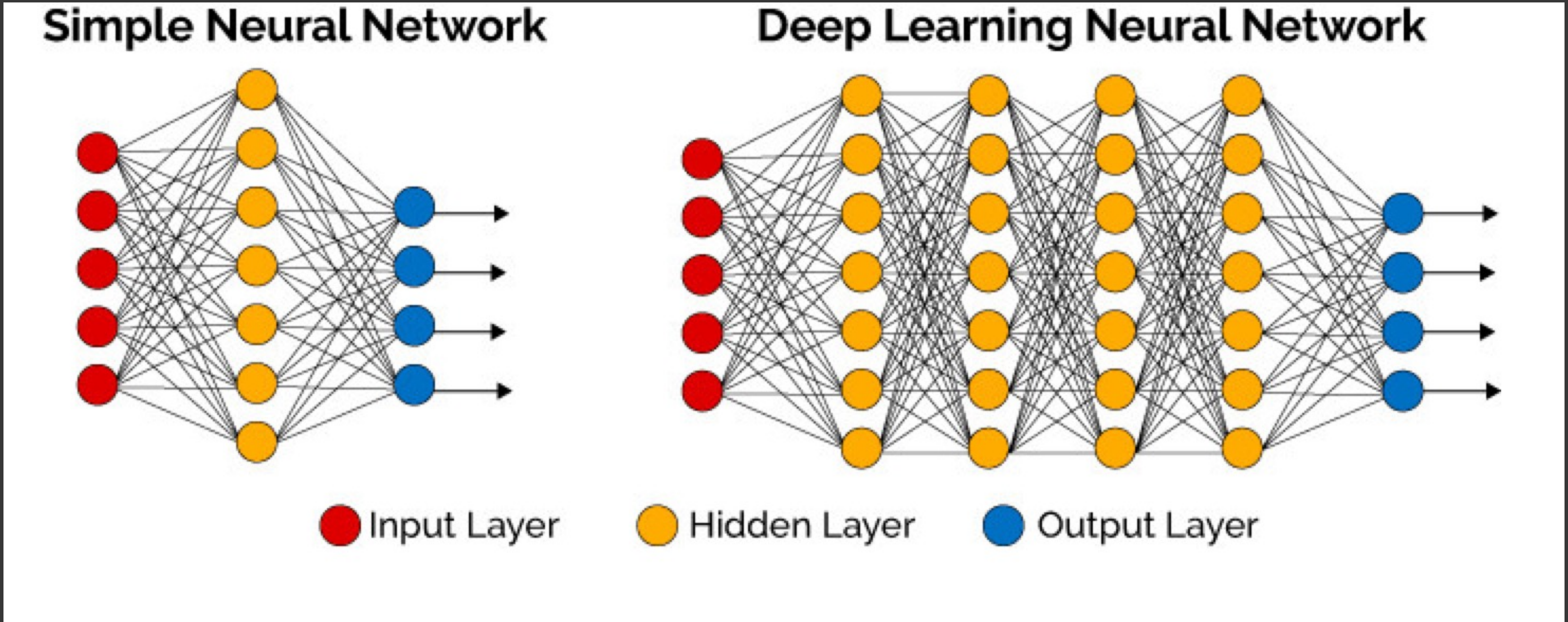
CINNS: A GENERAL PURPOSE ALGORITHM TO SOLVE (DIFFICULT) INVERSE PROBLEMS

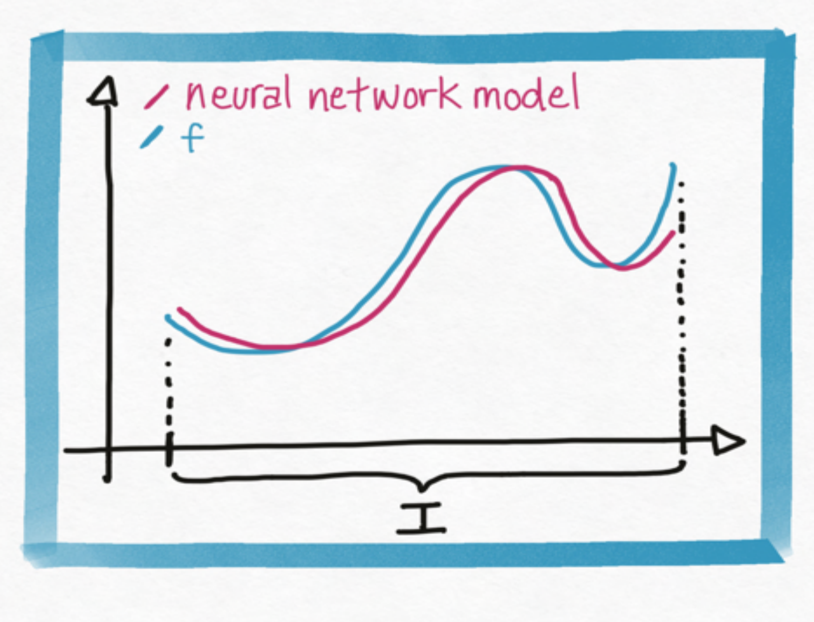
RAUL JIMENEZ (ICREA & UNIVERSITY OF BARCELONA) & PAVLOS PROTOPAPAS (SEAS, HARVARD UNIVERSITY)

The Perceptron: first example of a neural network (in the 60s…totally useless)




Simple Neural Network
Deep Learning Neural Network
Input Layer
Hidden Layer
Output Layer

NEURAL NETWORKS AS UNIVERSAL



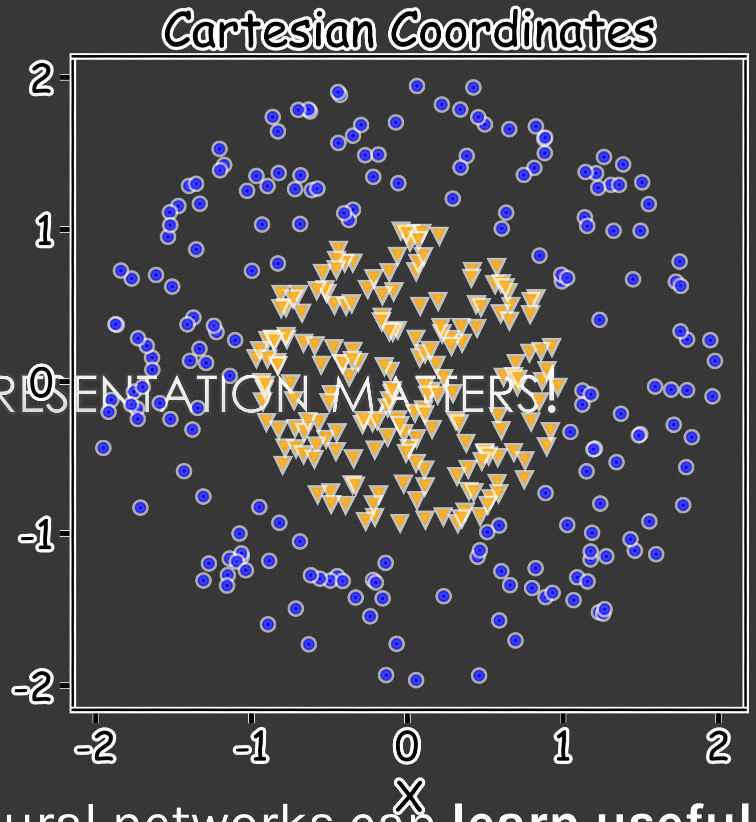
- WE HAVE SEEN THAT NEURAL NETWORKS CAN REPRESENT COMPLEX FUNCTIONS, BUT ARE THERE LIMITATIONS ON WHAT A NEURAL NETWORK CAN EXPRESS?
- **THEOREM:**
- *FOR ANY CONTINUOUS FUNCTION F DEFINED ON A BOUNDED DOMAIN, WE CAN FIND A NEURAL NETWORK THAT APPROXIMATES F WITH AN ARBITRARY DEGREE OF ACCURACY.*

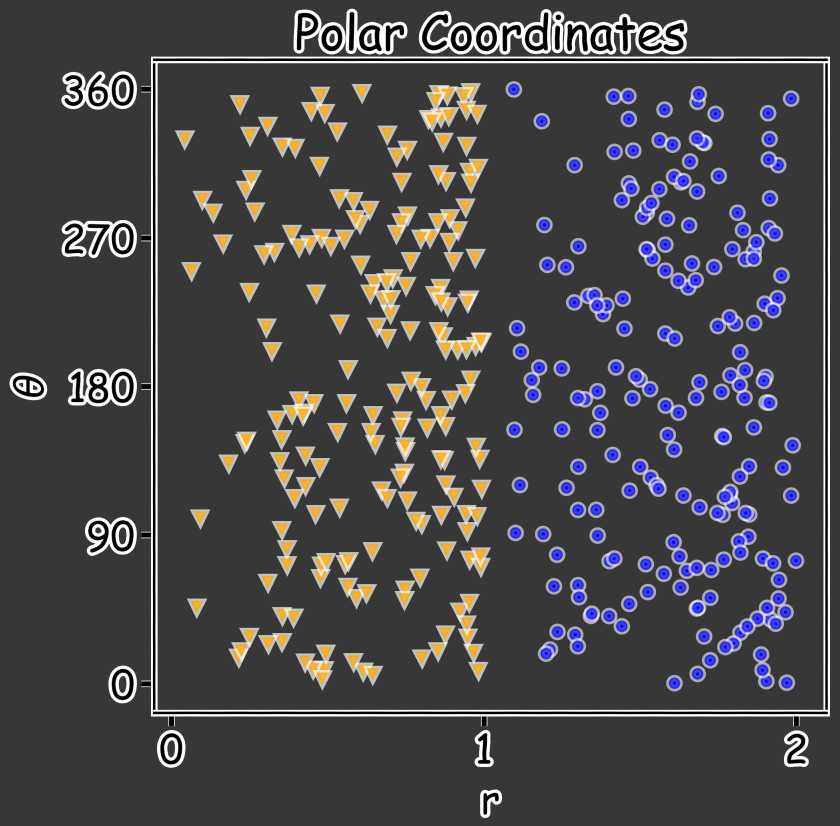
One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy.

So why deeper?

WHY LAYERS?

- REPRESENTATION MATTERS!





Neural networks can **learn useful representations** for the problem. This is another reason why they can be so powerful!

MOTIVATION

In every corner of **science**, dynamics and relationships are described with (partial) differential equations.

Only data science/AI is ignoring this fact.

Quantum Mechanics: **Schrodinger's** equation

Fluids: **Navier Stokes equation**

Electromagnetism: **Maxwell's** equations

Gravity: **Einstein's** equations; **Holography AdS-CFT correspondence**

Financial markets: **Black Scholes** equation

Epidemiology: **SIR** (*susceptible, infected, recovered*)

FORMULATION: ORDINARY DE (ODE)

Independent variable, x

Differential operator:

$$\sum_{i=1}^{k} \theta_i \frac{d^{(i)}}{dx^{(i)}}$$

The order of DE is determined by k.

θ_i are the parameters of the DE

$$Du + h(x)g(u) = f(x)$$

Any function of u: Special case linear DE $g(u) = u$

Dependent variable, u: what we are looking for

Forcing function, f

FORMULATION: **SYSTEM** OF ODE

$$D\boldsymbol{u} + \boldsymbol{h}(x)g(\boldsymbol{u}) = f(x)$$

Differential operator is a matrix:

$$\sum_{i=1}^{k} \boldsymbol{\Theta}_i \frac{d^{(i)}}{dx^{(i)}}$$

Dependent variables: $\boldsymbol{u} \in \boldsymbol{R}^m$

FORMULATION: **PARTIAL DIFFERENTIAL EQUATION** (PDE)

$$Du + h(\boldsymbol{x})g(u) = f(\boldsymbol{x})$$

Differential operator:

$$\sum_{i=1}^{k} \theta_i \frac{\partial^{(i)}}{\partial x^{(i)}}$$

Independent variables, $x \in \boldsymbol{R}^{\mathrm{d}}$

FORMULATION: **SYSTEM** OF **PARTIAL DIFFERENTIAL EQUATION** (PDE)

$$D\boldsymbol{u} + \boldsymbol{h}(\boldsymbol{x})g(\boldsymbol{u}) = f(\boldsymbol{x})$$

Differential operator:

$$\sum_{i=1}^{k} \theta_i \frac{\partial^{(i)}}{\partial x^{(i)}}$$

Independent variables, $x \in \boldsymbol{R}^{\mathrm{d}}$

Dependent variables: $\boldsymbol{u} \in \boldsymbol{R}^{m}$

FORMULATION: INITIAL AND BOUNDARY CONDITIONS

$$Du + h(x)g(u) = f(x)$$

Initial Conditions are referring to when x is time, t:

$$u(x = t = 0) = u_i$$

Boundary Conditions are referring to when x is anything else:

$$u(x = x_o) = u_b$$ Dirichlet boundary condition

$$\left.\frac{du}{dx}\right|_{x=x_o} = u_b$$ Neumann boundary condition

NEURAL NETWORK EDITION

FORMALISM

$x \longrightarrow$ Neural Network **W** $\longrightarrow u^{NN}$

How about this?

$$Du^{NN} + \boldsymbol{h}(\boldsymbol{x})g(\boldsymbol{u}^{NN}) = f(\boldsymbol{x})$$

FORMALISM

$$x \longrightarrow \boxed{\text{Neural Network } \mathbf{W}} \longrightarrow u^{NN}$$

Residuals

$$\left\{ \begin{array}{l} \mathcal{R}_D = D\boldsymbol{u}^{\boldsymbol{NN}} + \boldsymbol{h}(\boldsymbol{x})g\left(\boldsymbol{u}^{\boldsymbol{NN}}\right) - f(\boldsymbol{x}) \\ \\ \mathcal{R}_B = u^{NN}(\mathrm{t} = 0) - u_{\mathrm{i}} \end{array} \right.$$

Loss function:

$$\mathcal{L} = \mathcal{R}_D^2 + \mathcal{R}_B^2$$

FORMALISM

$x \longrightarrow$ **Neural Network W** $\longrightarrow u^{NN}$

$$\mathcal{R}_D = \; D\boldsymbol{u}^{NN} + \boldsymbol{h}(\boldsymbol{x})g\left(\boldsymbol{u}^{NN}\right) - f(\boldsymbol{x})$$

$$\mathcal{R}_B = u^{NN}(\mathrm{t} = 0) - u_{\mathrm{i}}$$

If we have solution or measurements we include them

$$\mathcal{R}_d = u^{NN}(\mathrm{x}^d) - u^d\left(x^d\right)$$

$$\mathcal{L} = \mathcal{R}_D^2 + \mathcal{R}_B^2 + \mathcal{R}_d^2$$

FORMALISM

x → **Neural Network W** → u^{NN}

$$\mathcal{L} = \mathcal{R}_D^2 + \mathcal{R}_B^2$$

FORMALISM

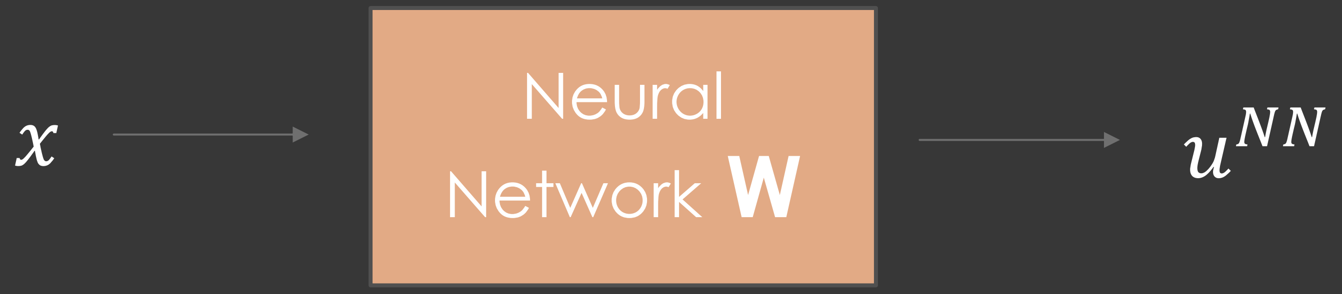
x → Neural Network **W** → u^{NN}

$$\mathcal{L} = \mathcal{R}_D^2 + \mathcal{R}_B^2$$

Find W that minimize the loss by backprop

$$W^* = \underset{W}{\operatorname{argmin}}\, \mathcal{L}$$

FORMALISM



$$\mathcal{L} = \mathcal{R}_D^2 + \mathcal{R}_B^2$$

$$W^* = \underset{W}{\operatorname{argmin}}\, \mathcal{L}$$

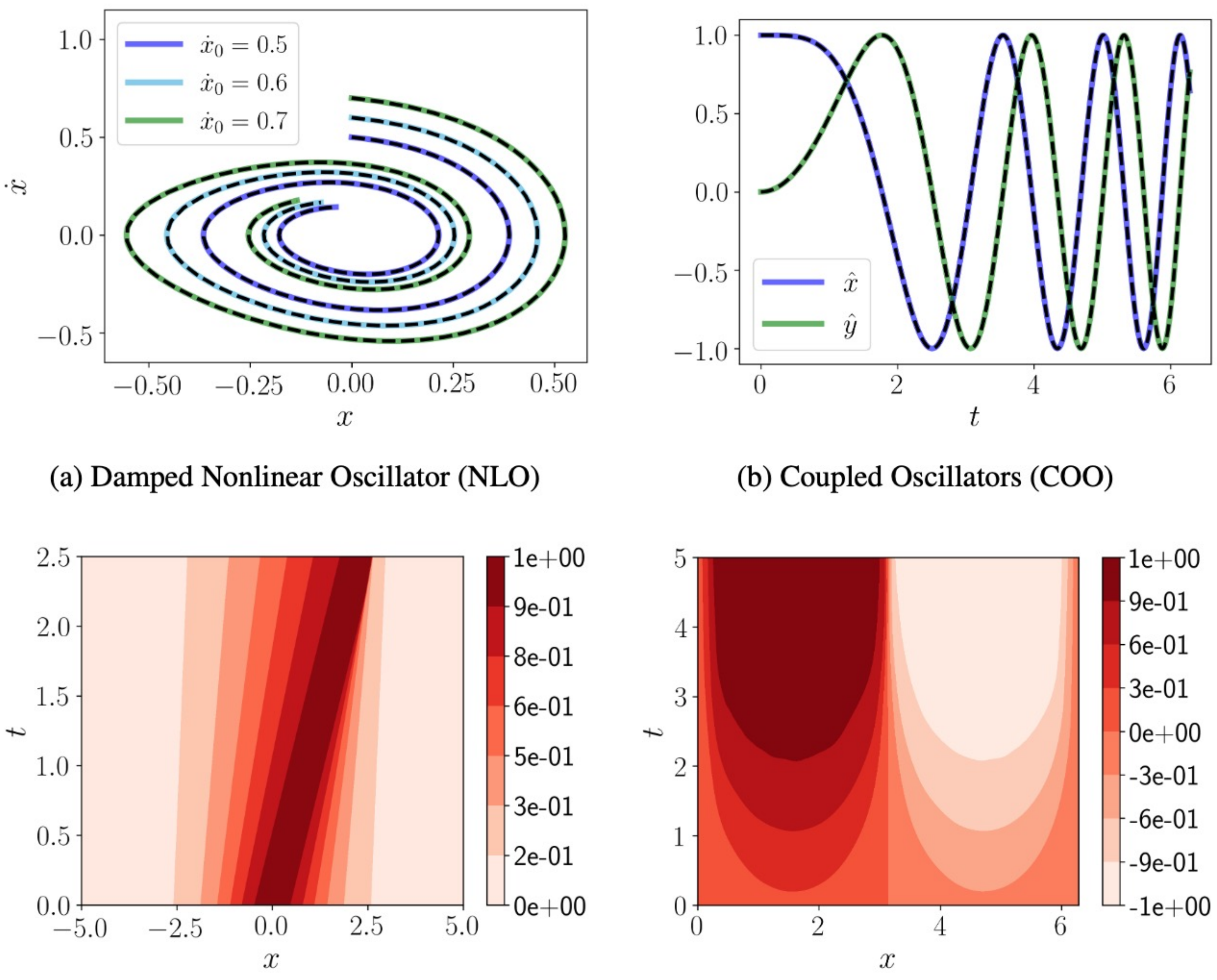
Choose $\mathrm{x} \in X$ and minimize the loss.

Choose x in regular intervals, or randomly. Randomly works better, eliminates trapped in trivial solutions. No meaning of epochs here (infinite data set)

FORMULATION: EXAMPLES

Key	Equation	Class	Order	Linear
EXP	$\dot{x}(t) + x(t) = 0$	ODE	1st	Yes
SHO	$\ddot{x}(t) + x(t) = 0$	ODE	2nd	Yes
NLO	$\ddot{x}(t) + 2\beta\dot{x}(t) + \omega^2 x(t) + \phi x(t)^2 + \epsilon x(t)^3 = 0$	ODE	2nd	No
COO	$\begin{cases} \dot{x}(t) = -ty \\ \dot{y}(t) = tx \end{cases}$	ODE	1st	Yes
SIR	$\begin{cases} \dot{S}(t) & = -\beta I(t)S(t)/N \\ \dot{I}(t) & = \beta I(t)S(t)/N - \gamma I(t) \\ \dot{R}(t) & = \gamma I(t) \end{cases}$	ODE	1st	No
HAM	$\begin{cases} \dot{x}(t) & = p_x \\ \dot{y}(t) & = p_y \\ \dot{p_x}(t) & = -V_x \\ \dot{p_y}(t) & = -V_y \end{cases}$	ODE	1st	No
EIN	$\begin{cases} \dot{x}(z) & = \frac{1}{z+1}(-\Omega - 2v + x + 4y + xv + x^2) \\ \dot{y}(z) & = \frac{-1}{z+1}(vx\Gamma(r) - xy + 4y - 2yv) \\ \dot{v}(z) & = \frac{-v}{z+1}(x\Gamma(r) + 4 - 2v) \\ \dot{\Omega}(z) & = \frac{\Omega}{z+1}(-1 + 2v + x) \\ \dot{r}(z) & = \frac{-r\Gamma(r)x}{z+1} \end{cases}$	ODE	1st	No
POS	$u_{xx} + u_{yy} = 2x(y-1)(y-2x+xy+2)e^{x-y}$	PDE	2nd	Yes
HEA	$u_t = \kappa u_{xx}$	PDE	2nd	Yes
WAV	$u_{tt} = c^2 u_{xx}$	PDE	2nd	Yes
BUR	$u_t + uu_x - \nu u_{xx} = 0$	PDE	2nd	No
ACA	$u_t - \epsilon u_{xx} - u + u^3 = 0$	PDE	2nd	No

FORMULATION: EXAMPLES



(a) Damped Nonlinear Oscillator (NLO)

(b) Coupled Oscillators (COO)

(c) Burgers' Equation (BUR)

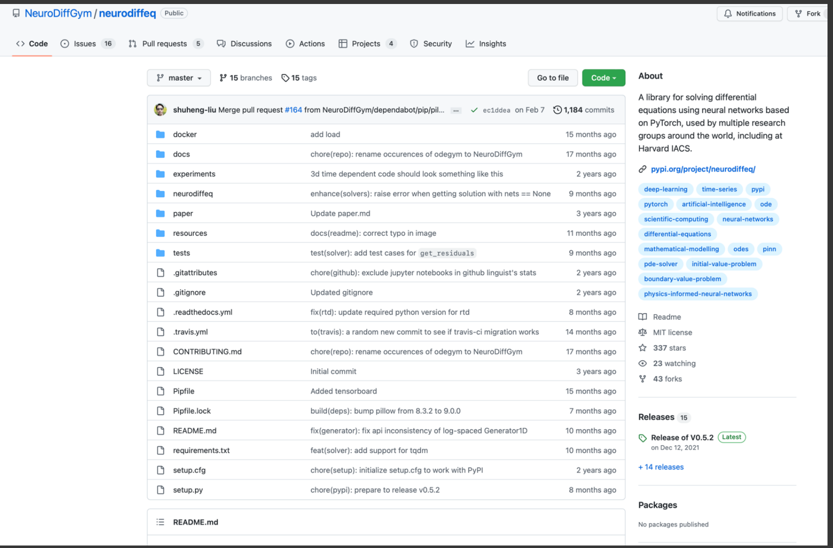
(d) Allen-Cahn Equation (ACA)

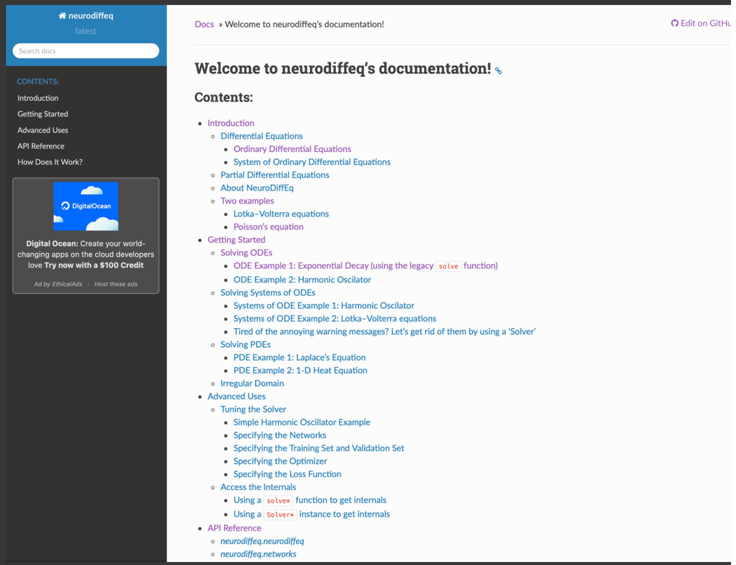
NEURODIFFGYM/NEURODIFFEQ

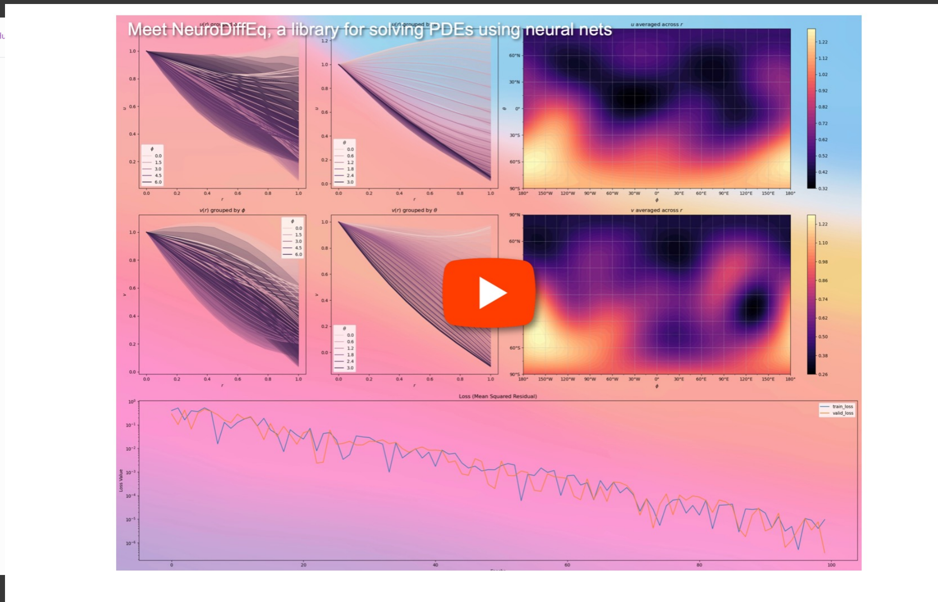
`neurodiffeq` is a package for solving differential equations with neural networks.

`Github` page: https://github.com/NeuroDiffGym/neurodiffeq

Extended documentation and examples: https://neurodiffeq.readthedocs.io/en/latest/intro.html







PHYSICS INFORMED NNS (PINNS) REFERENCES

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: Journal of Computational Physics 378 (2019), pp. 686–707

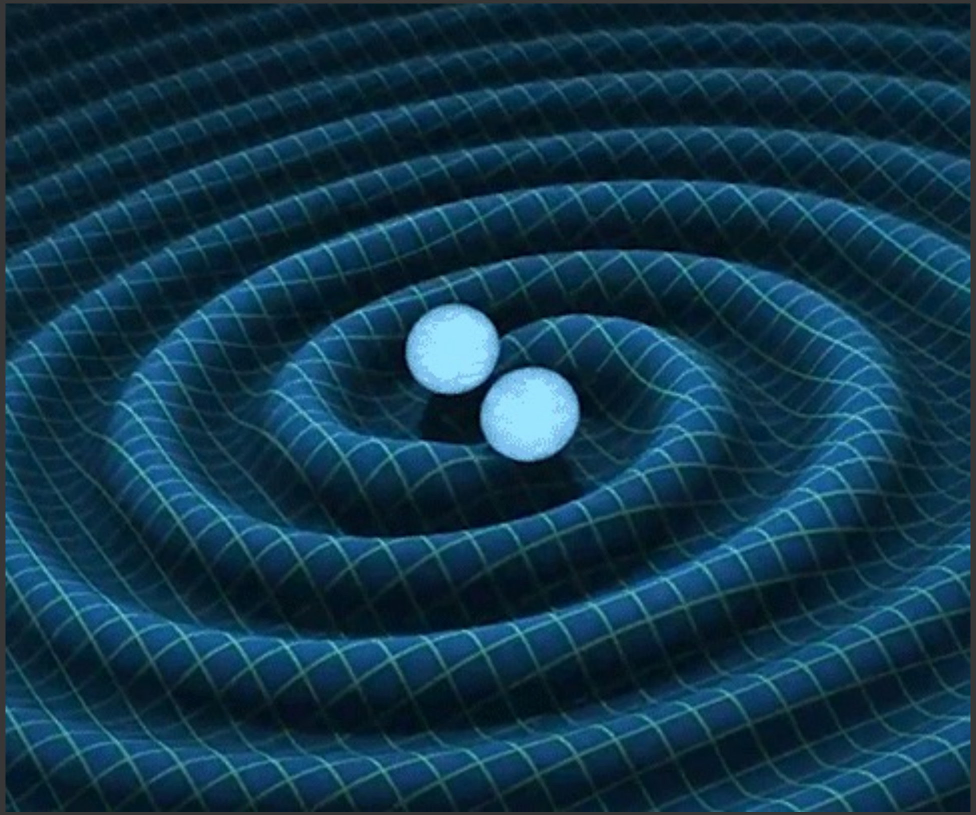
Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: IEEE transactions on neural networks 9.5 (1998), pp. 987–1000

Kevin Stanley McFall and James Robert Mahan. "Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions". In: IEEE Transactions on Neural Networks 20.8 (2009), pp. 1221–1233

N Sukumar and Ankit Srivastava. "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks". In: arXiv preprint arXiv:2104.08426 (2021).

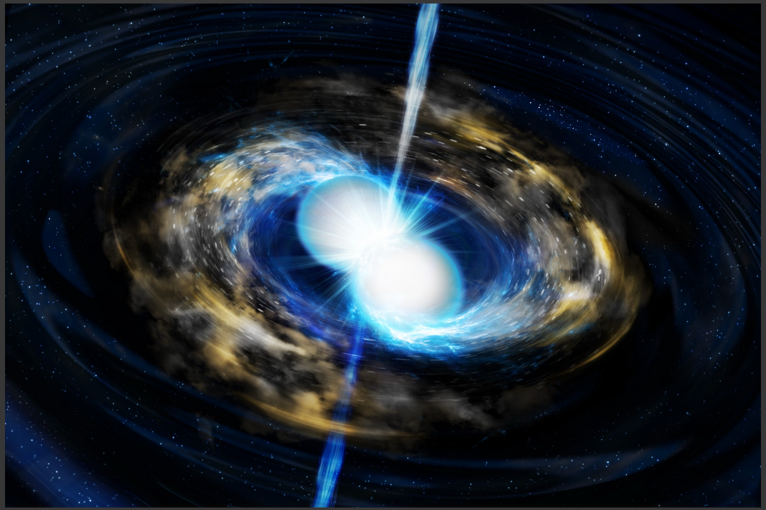
Justin Sirignano and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: Journal of computational physics 375 (2018), pp. 1339–1364.

NEW WINDOWS TO THE UNIVERSE

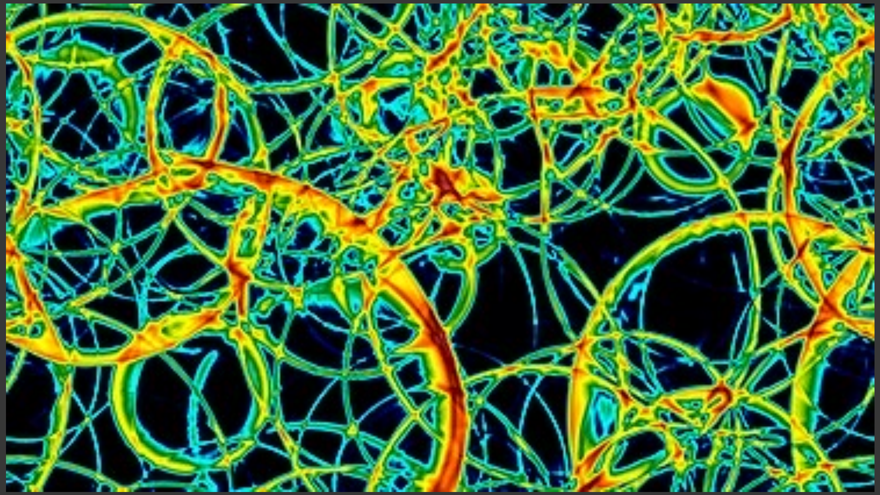


- Strong regime of General Relativity
- Properties of strong-interacting quantum matter

RELEVANT SCENARIOS



Neutron star mergers
SM matter



Phase transition early Universe
BSM matter

Strongly couple Quantum matter + Out of Equilibrium

→ HOLOGRAPHY

OUTLINE

1. Things we have done that worked.
2. Things that we want now to explore.

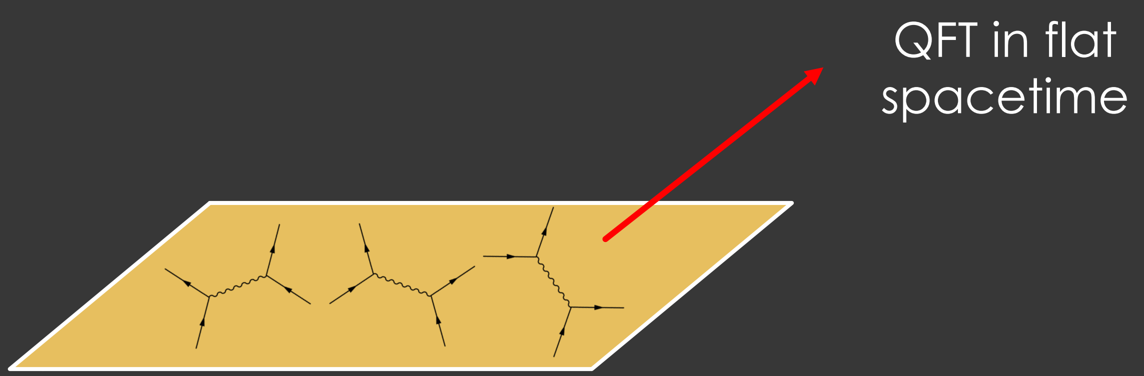
OUTLINE

1. Holography in a Nutshell: presenting the model
2. Physics informed Neural Networks (PINNs)
3. Results
4. Things that we want now to explore

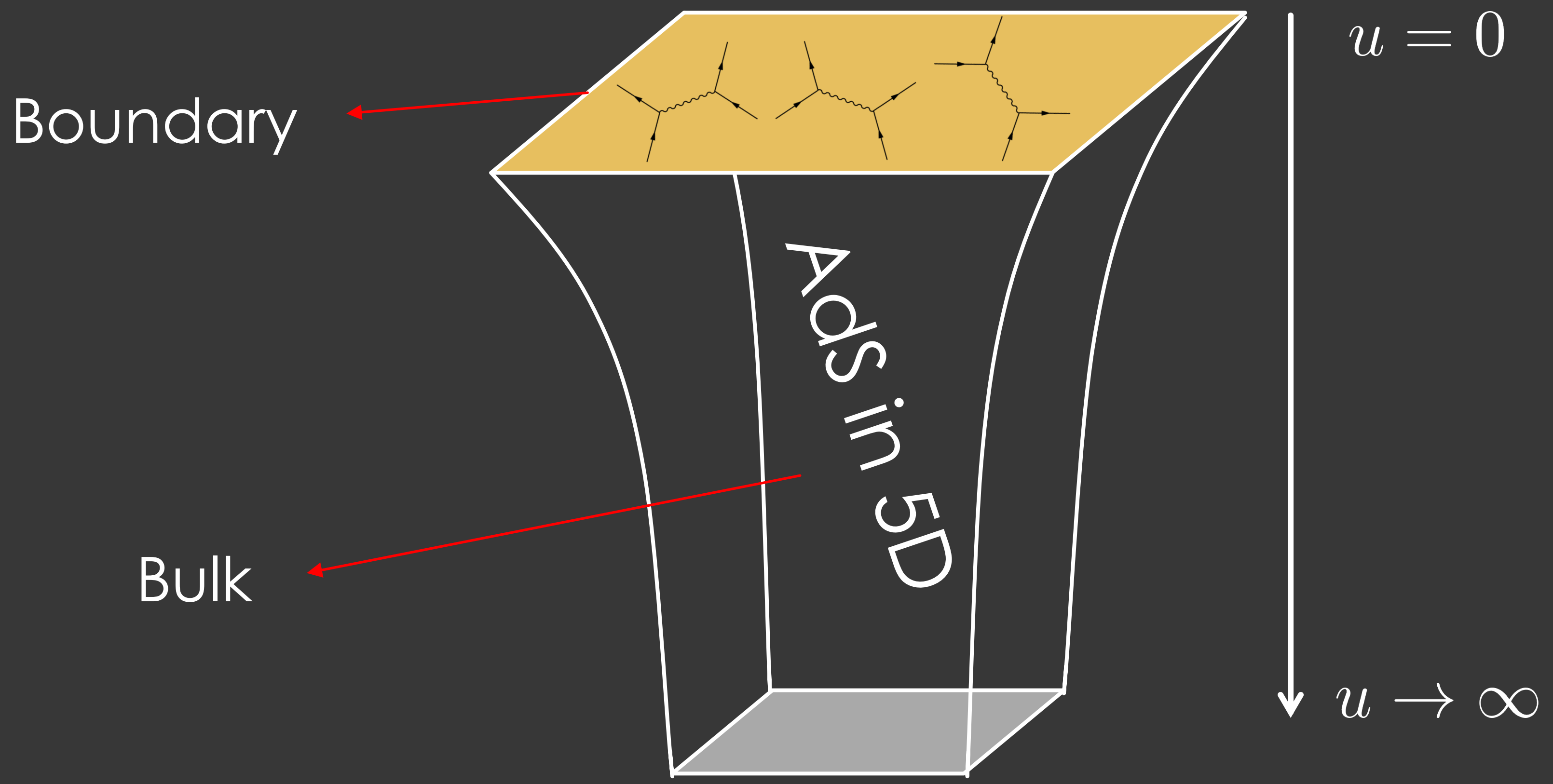
OUTLINE

1. Holography in a Nutshell: presenting the model
2. Physics informed Neural Networks (PINNs)
3. Results
4. Conclusions and Future directions

HOLOGRAPHY IN A NUTSHELL

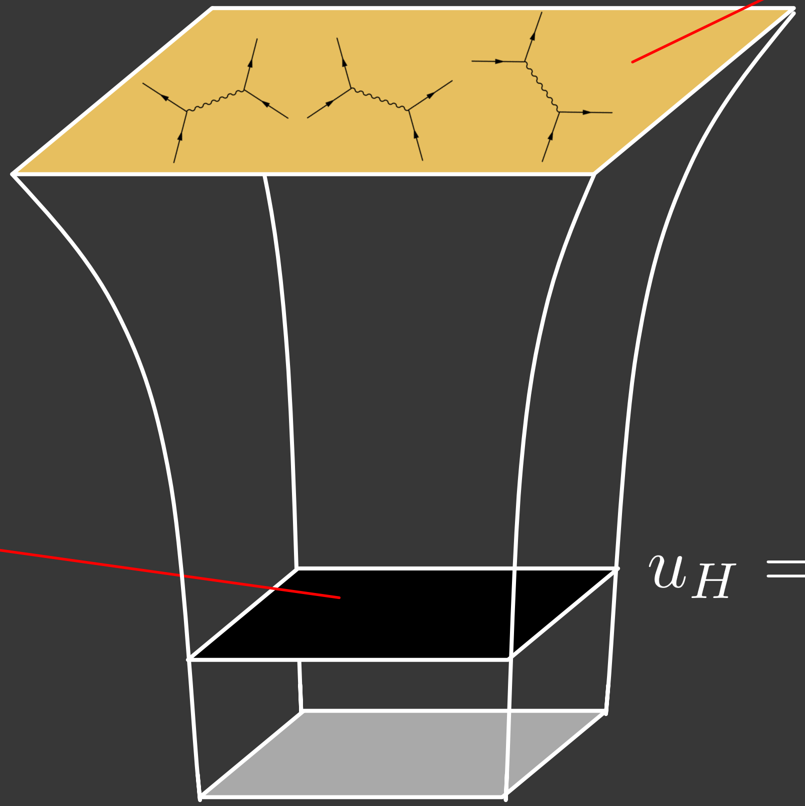


HOLOGRAPHY IN A NUTSHELL



HOLOGRAPHY IN A NUTSHELL

Add temperature to the QFT

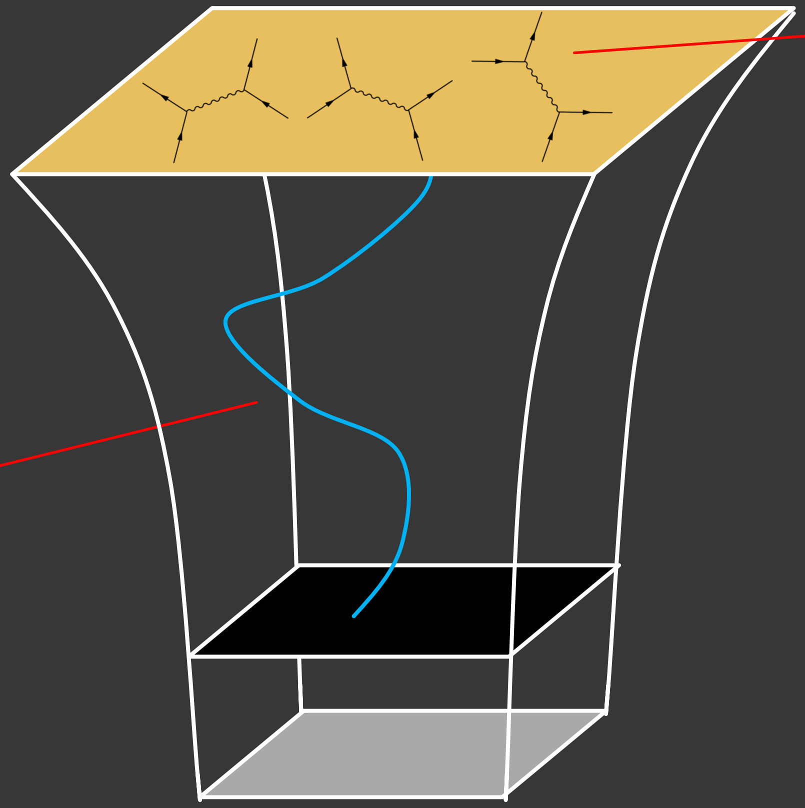


Add a Black hole with $T_H = T$

$u_H = 1$

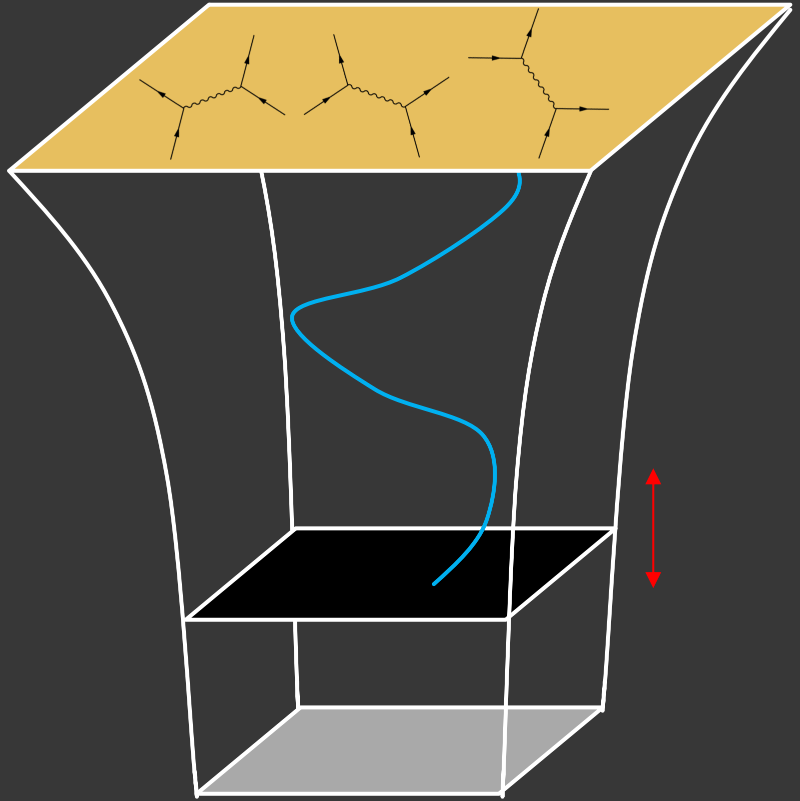
HOLOGRAPHY IN A NUTSHELL

Phase transitions



Scalar field with potential $V(\phi)$

HOLOGRAPHY IN A NUTSHELL

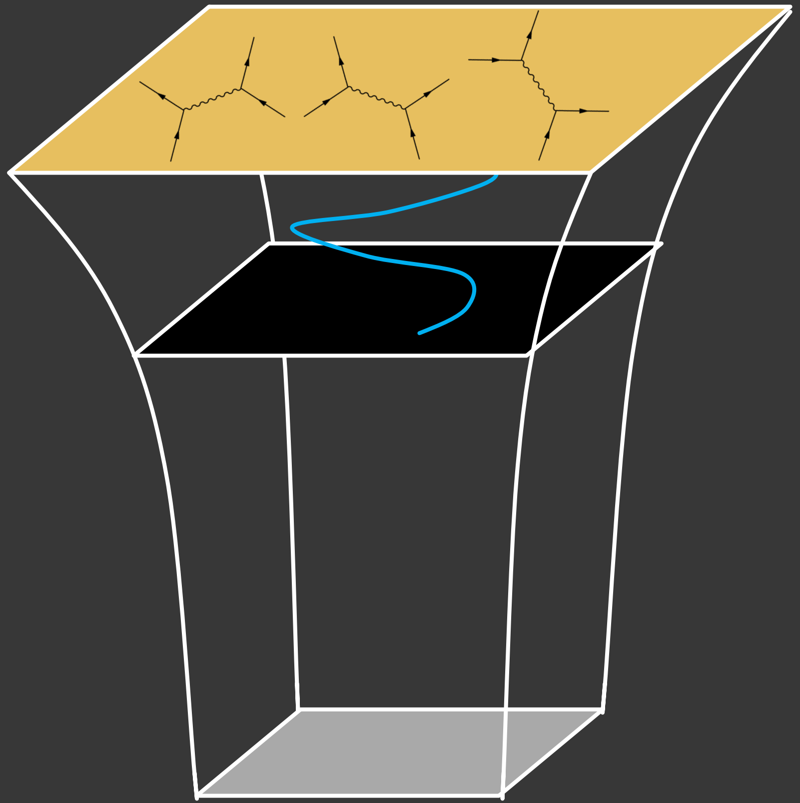


1. Construct different black brane solutions

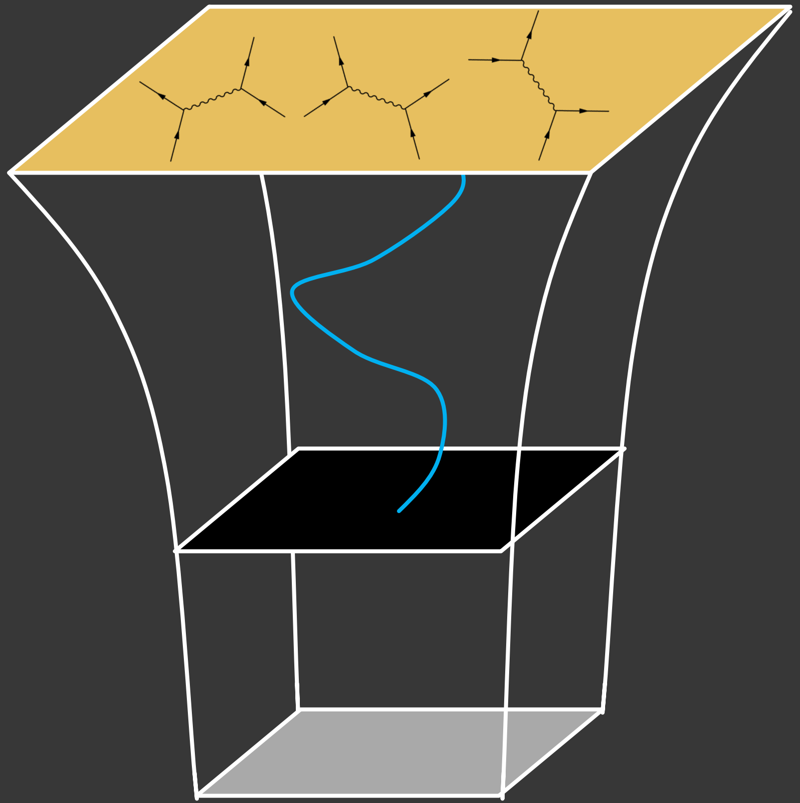
2. Solve the EE+KG

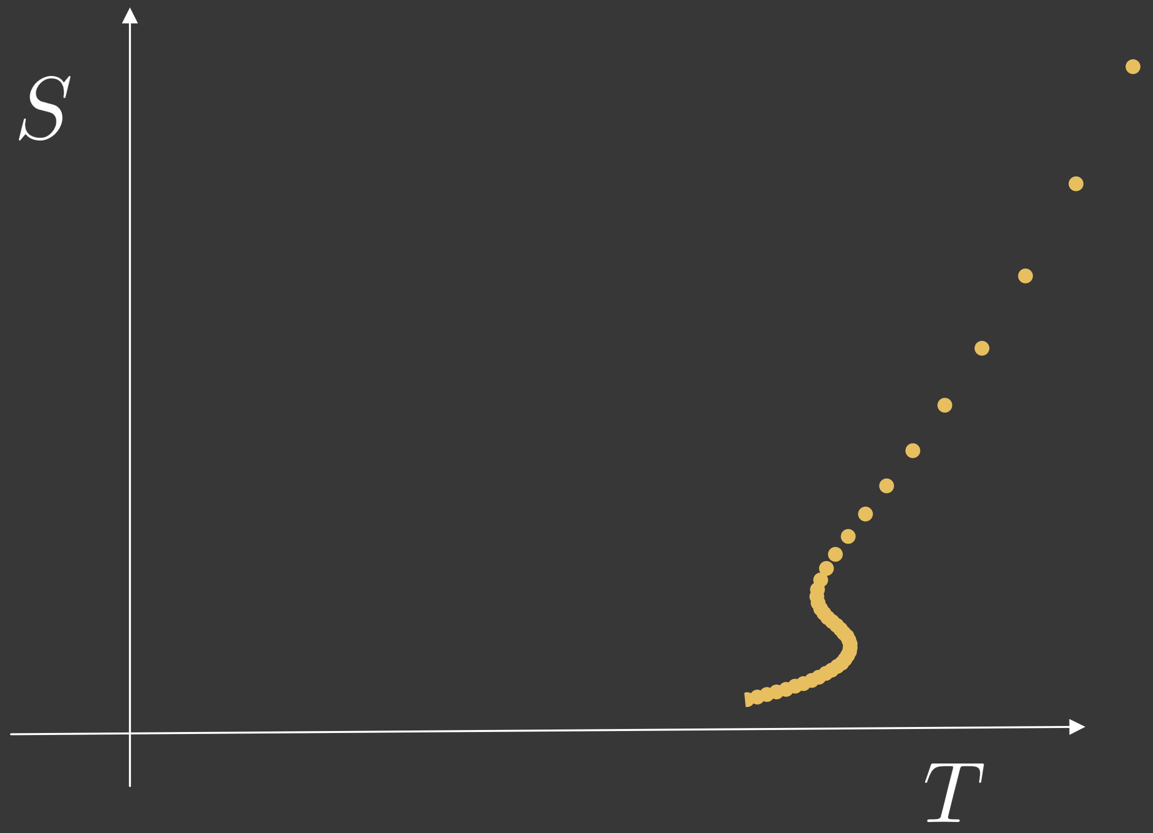
3. Read off the thermodynamics as b.c.

HOLOGRAPHY IN A NUTSHELL

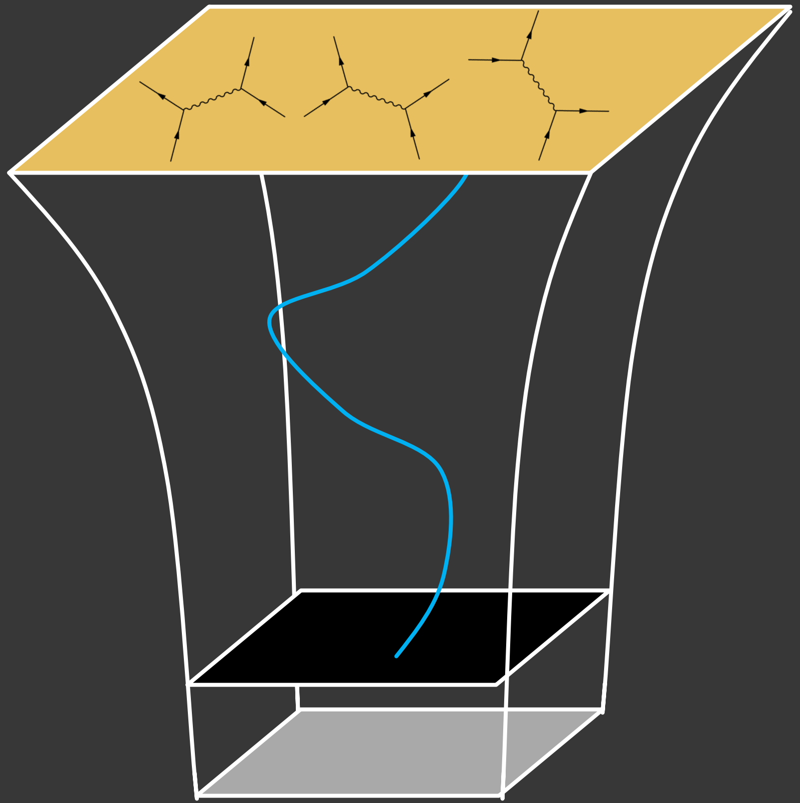


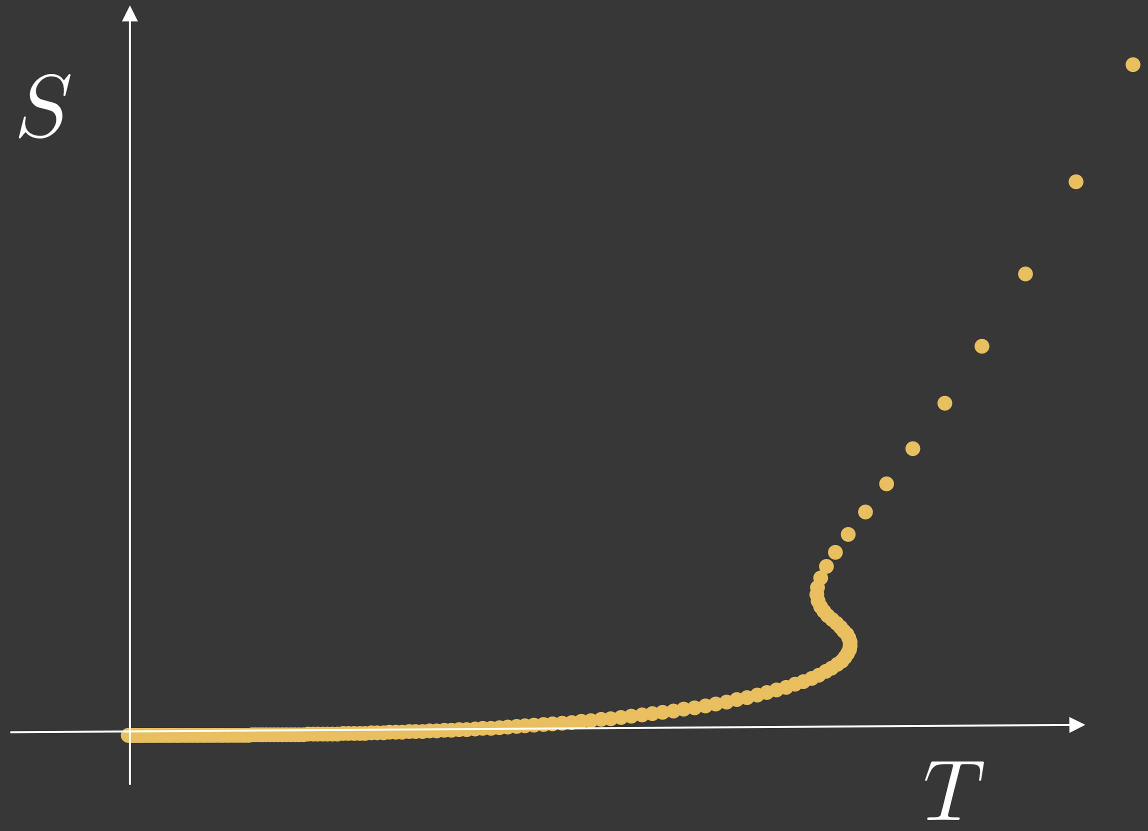
HOLOGRAPHY IN A NUTSHELL



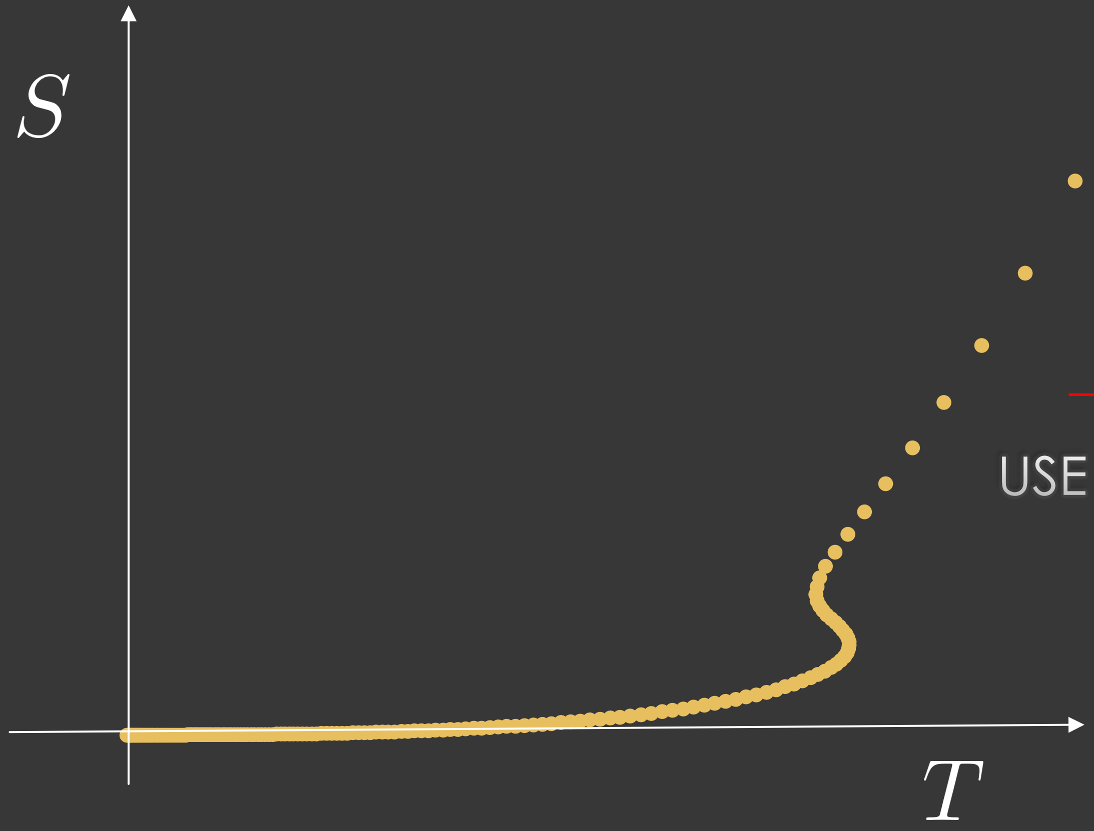


HOLOGRAPHY IN A NUTSHELL

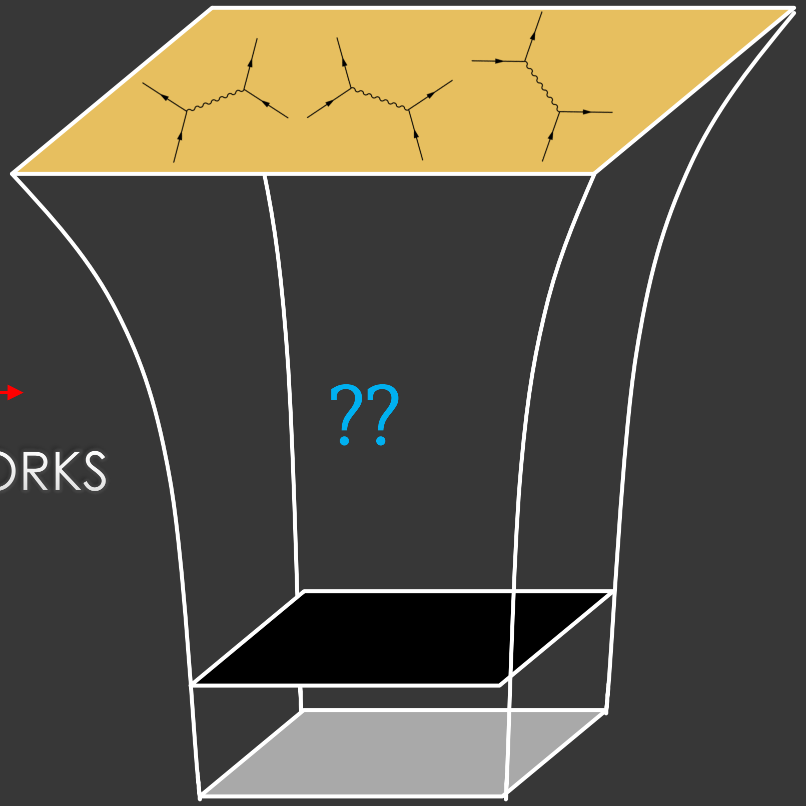




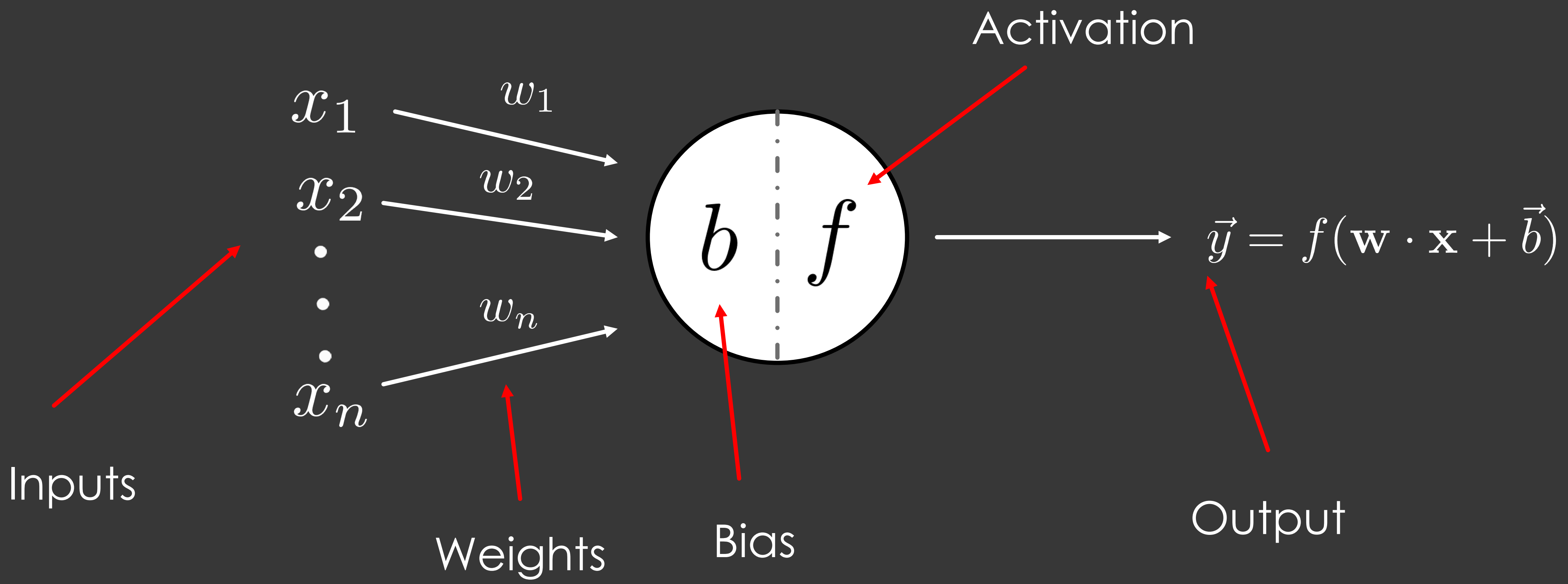
HOLOGRAPHY IN A NUTSHELL



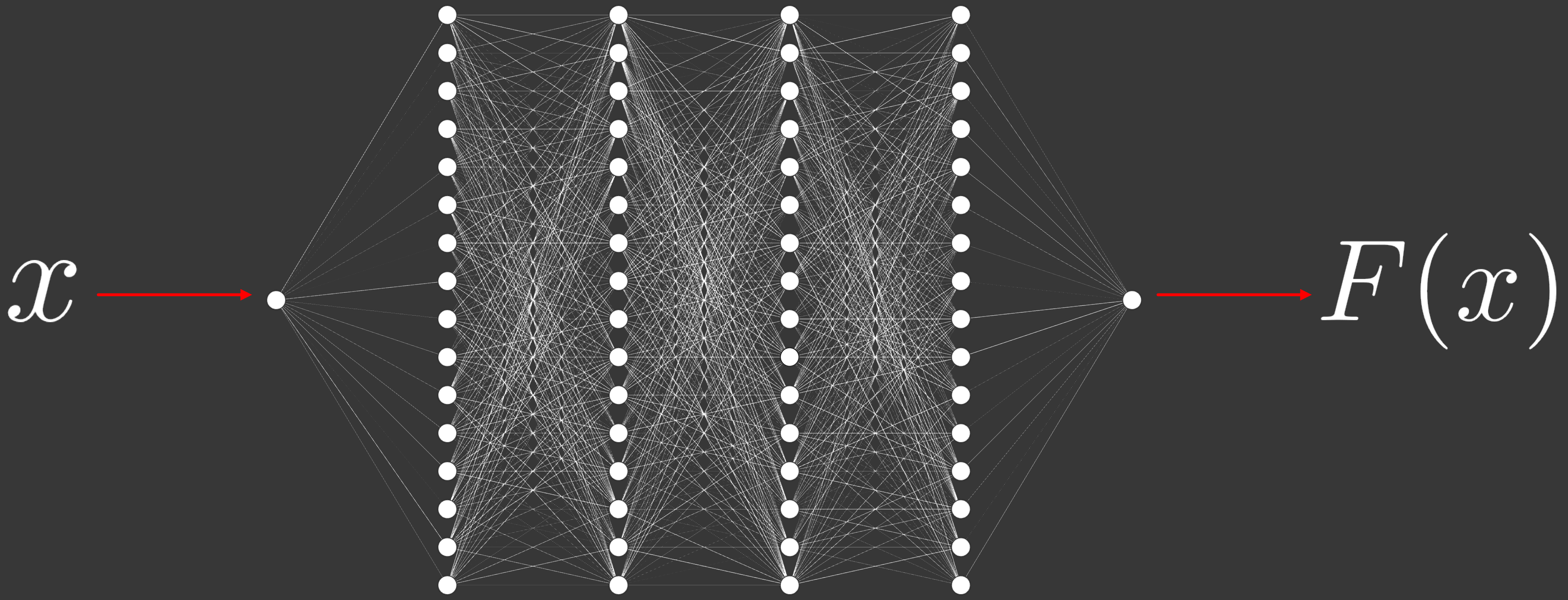
USE NEURAL NETWORKS



PHYSICS INFORMED NEURAL NETWORKS



PHYSICS INFORMED NEURAL NETWORKS



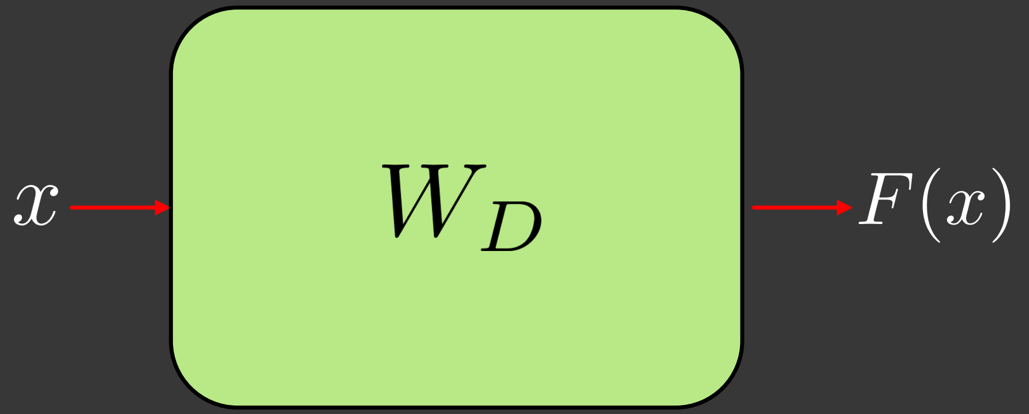
PHYSICS INFORMED NEURAL NETWORKS

$$x \longrightarrow \boxed{W_D} \longrightarrow F(x)$$

PHYSICS INFORMED NEURAL NETWORKS

[Dissanayake, M and Phan-Thien, N. 1994]

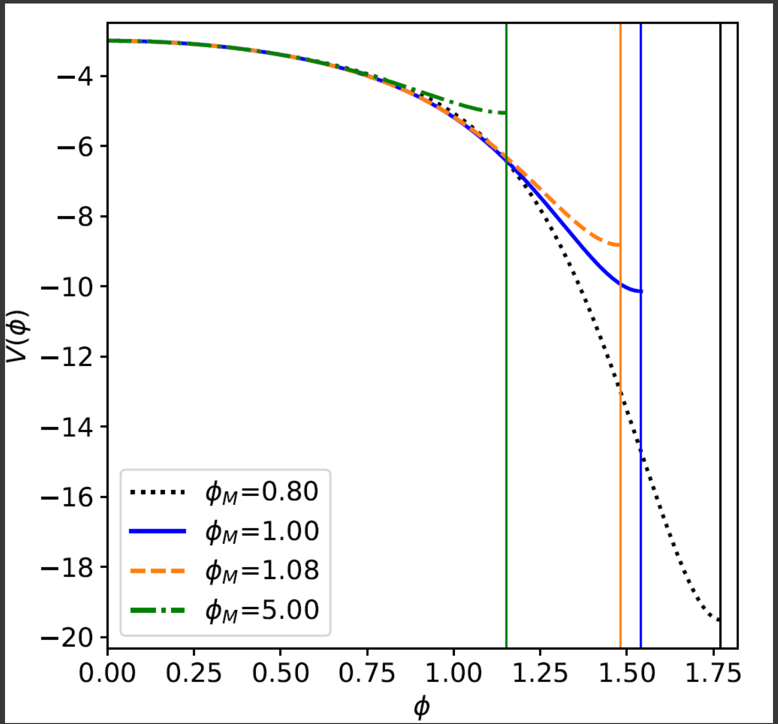
[Lagaris et al. 1997]



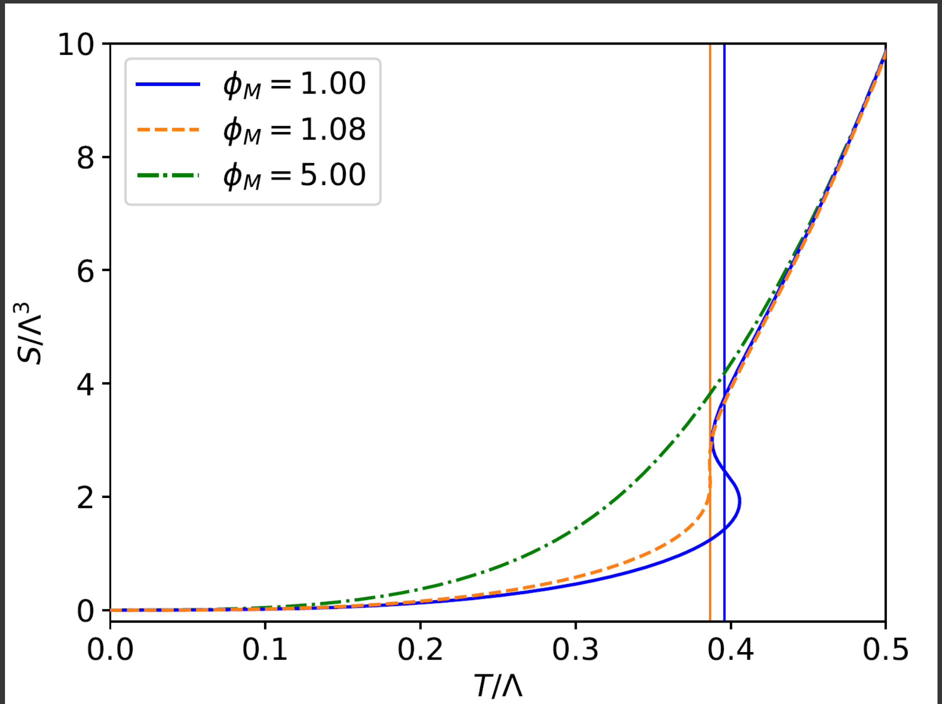
- Use a NN for each unknown function
- BC fulfilled by construction
- Train for different BC at once
- PINNs loss function is defined as the residuals of the differential equations.

[Chen et al. 2020]

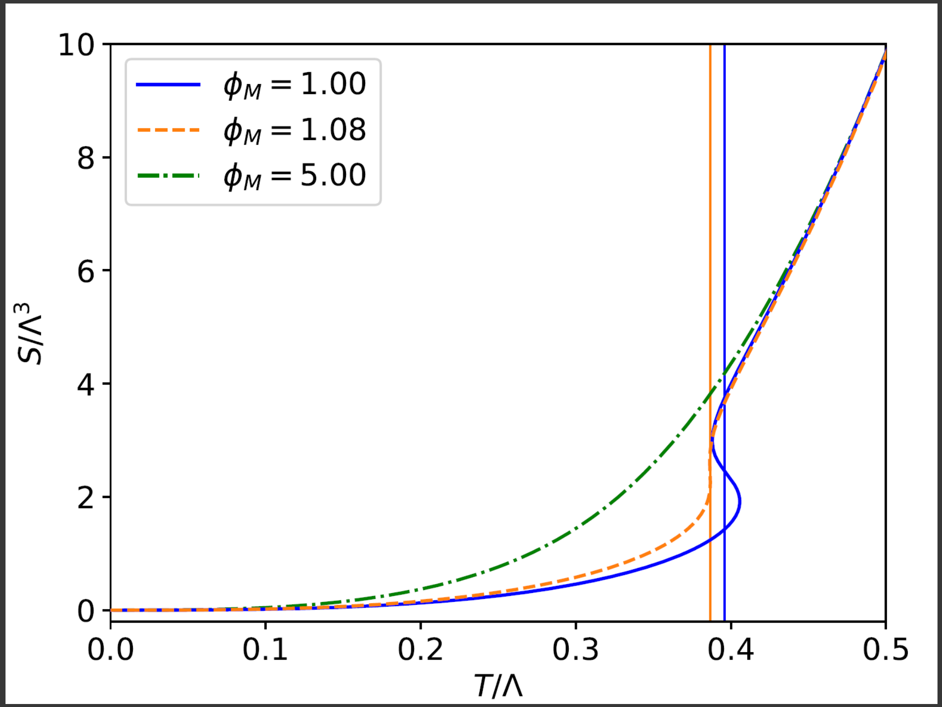
RESULTS



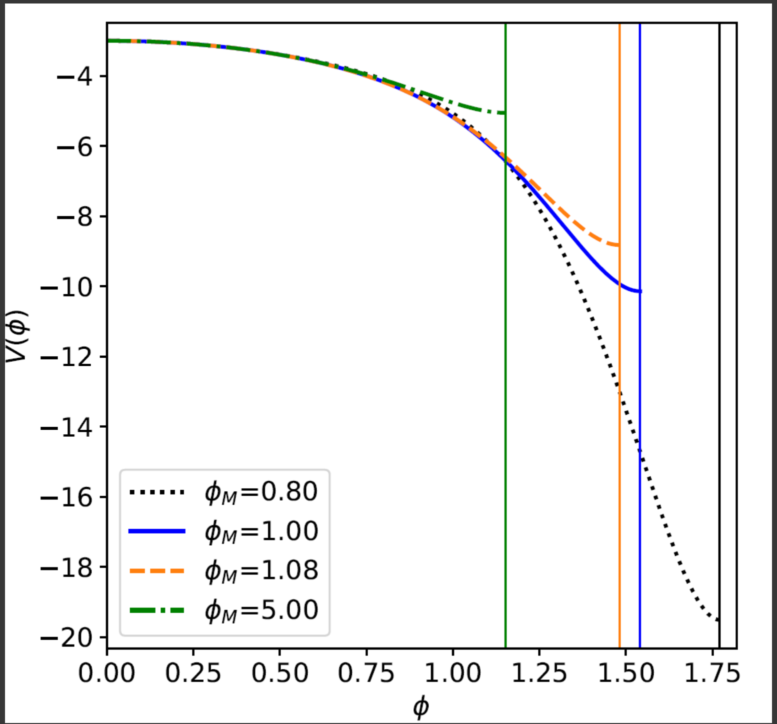
Direct problem



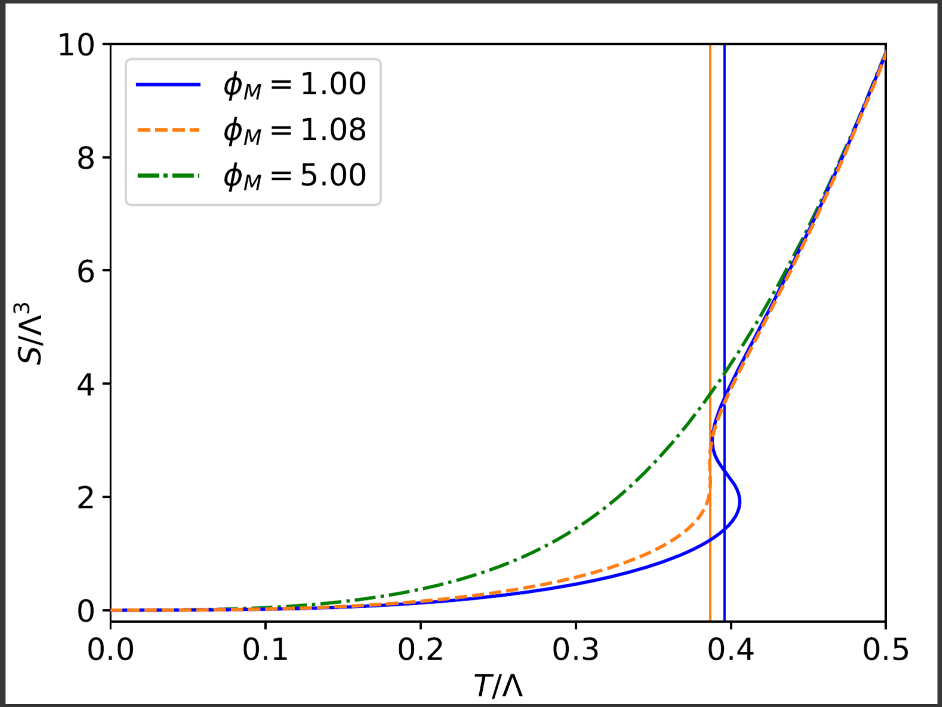
RESULTS



Inverse problem
PINNs

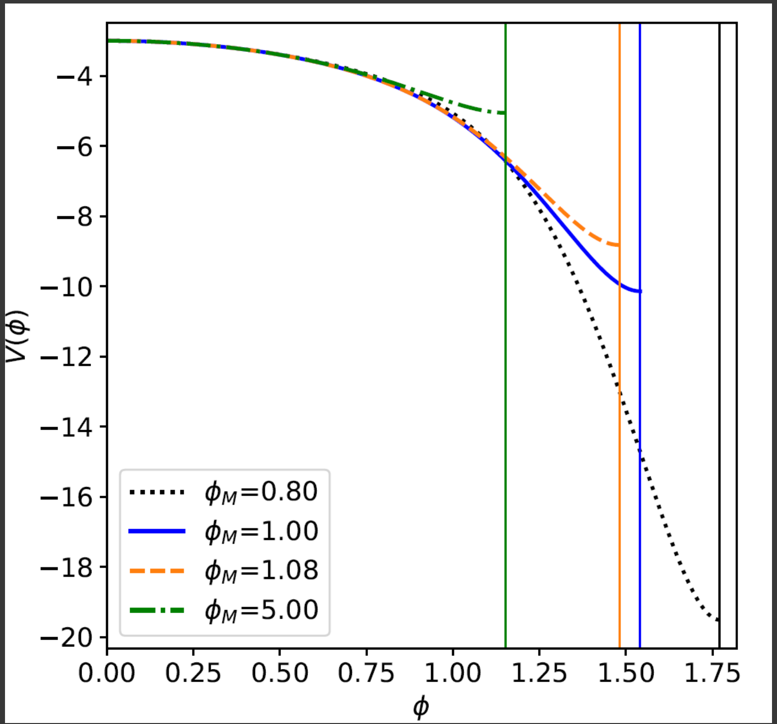


RESULTS



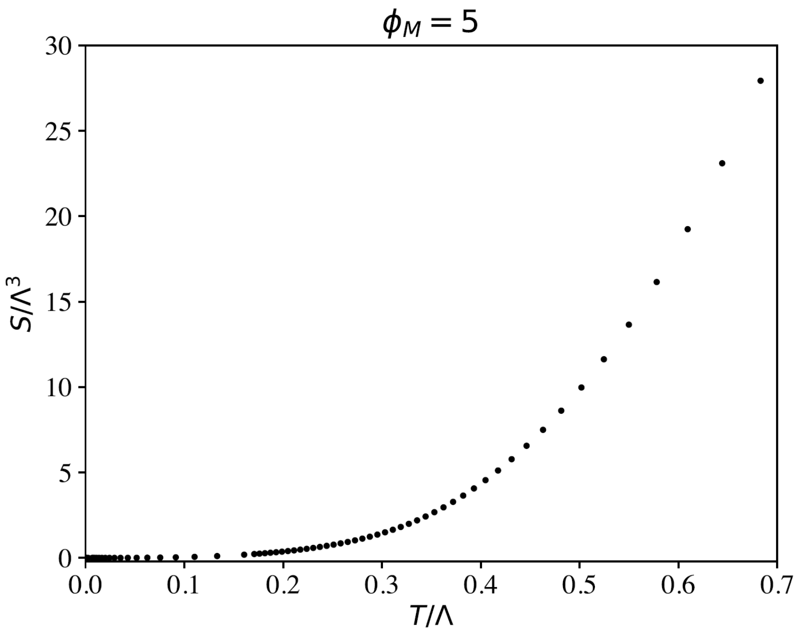
Inverse problem
PINNs

Direct problem
Compare S(T)

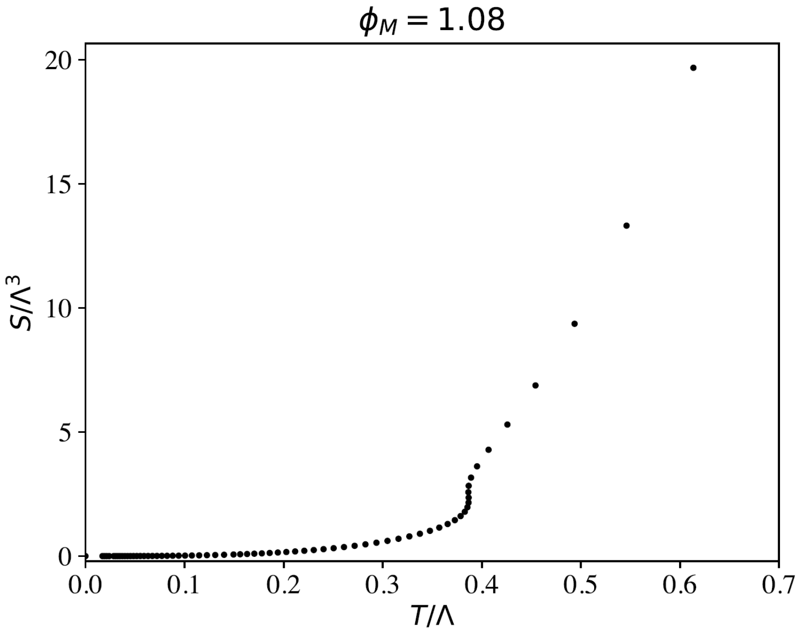


RESULTS

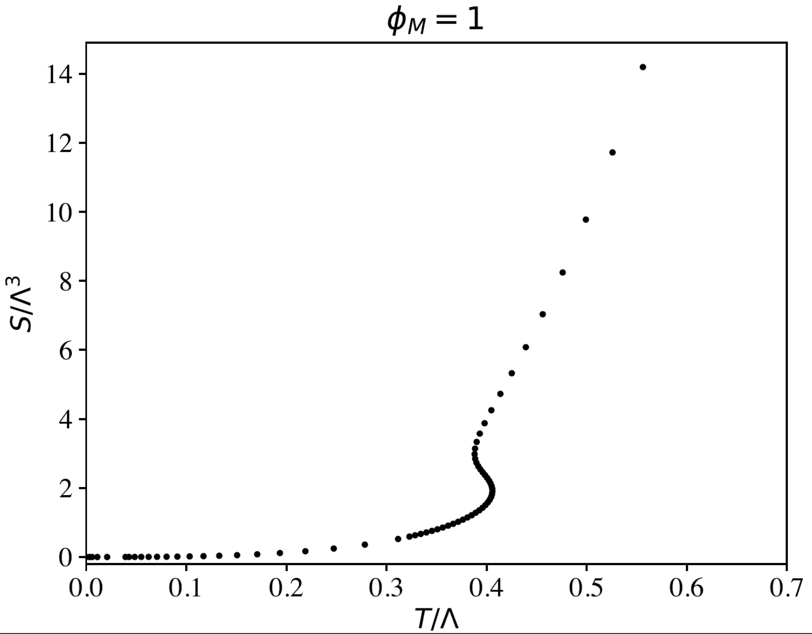
Crossover



Second Order

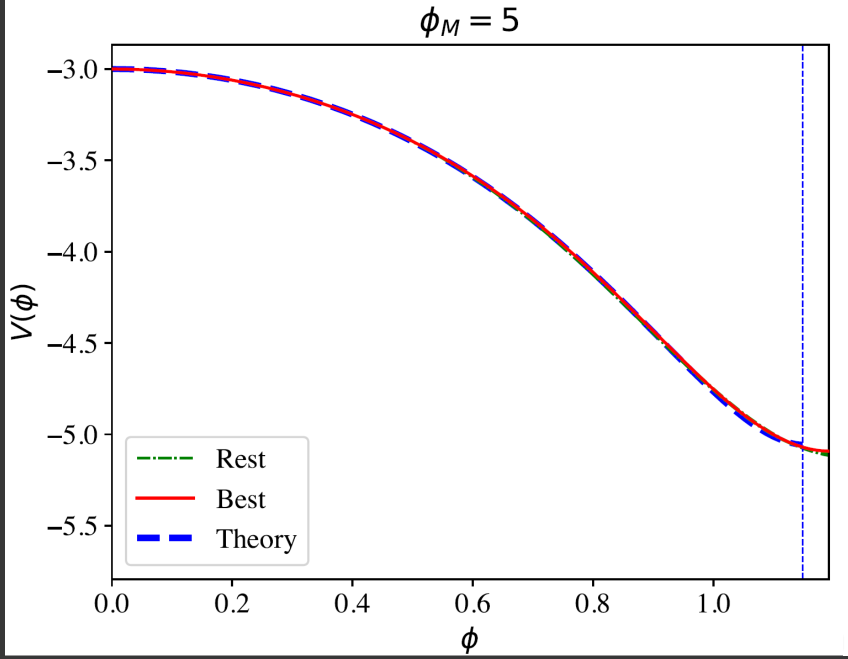


First Order

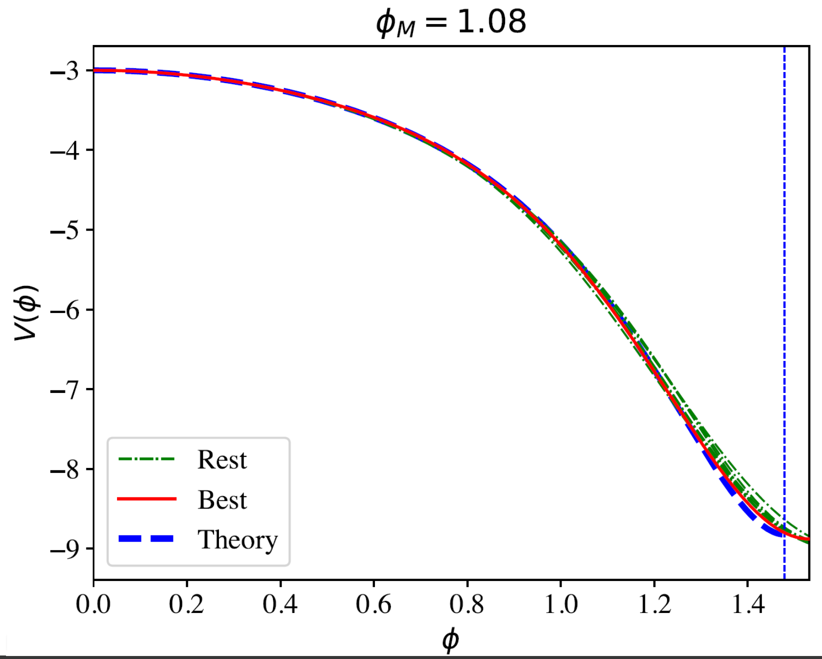


RESULTS

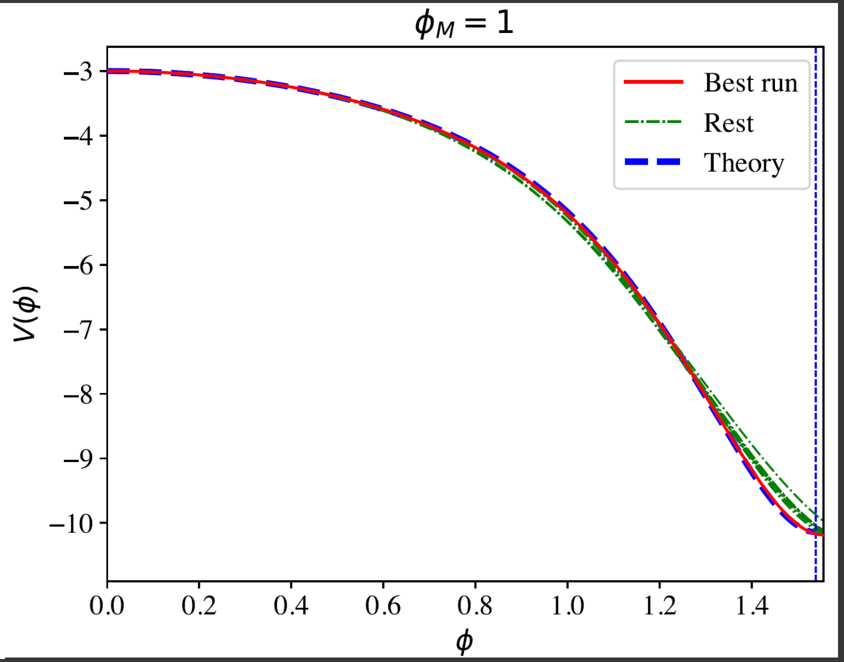
Crossover



Second Order

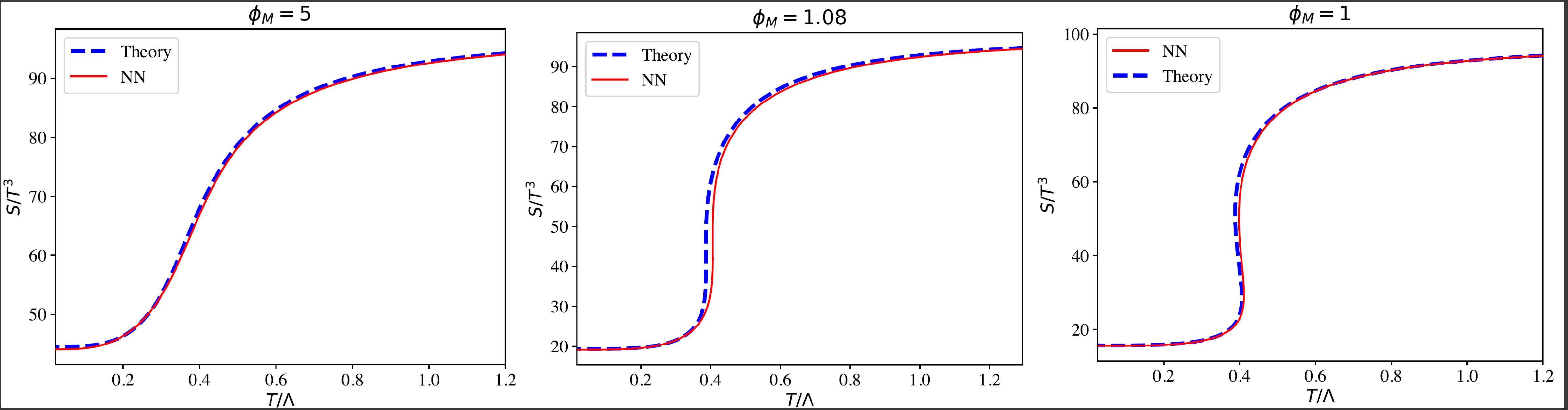


First Order

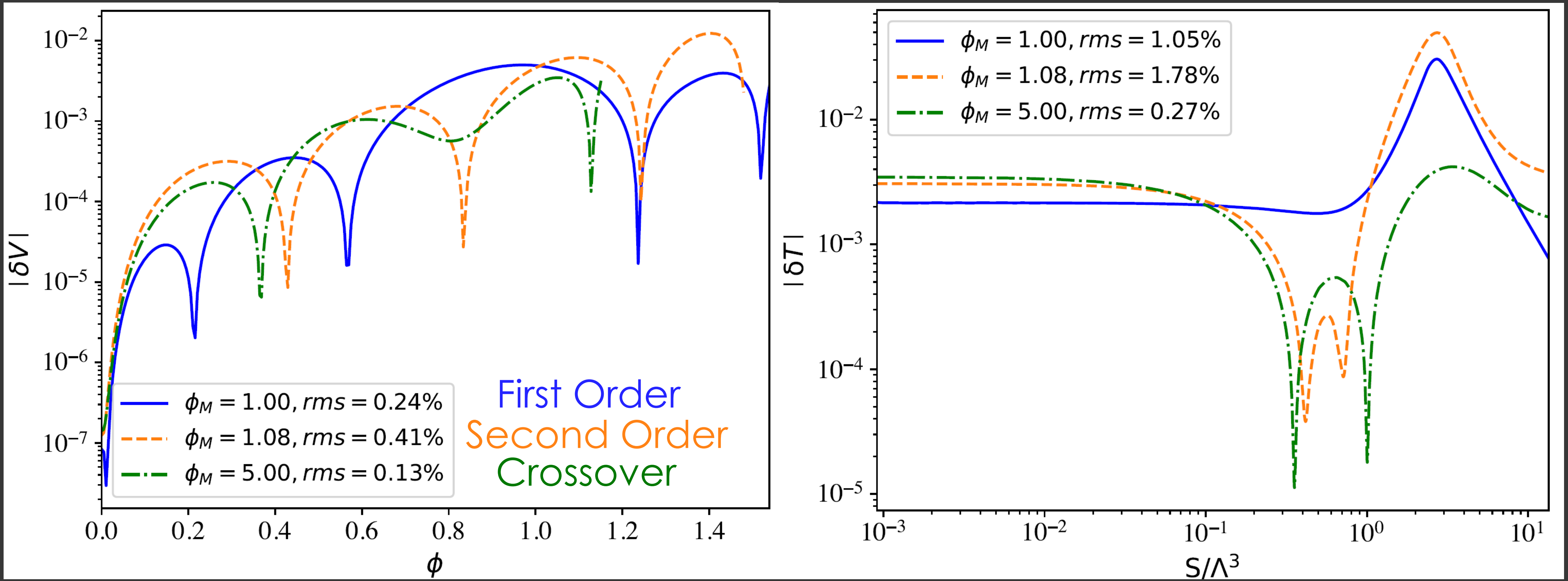


RESULTS

Crossover | Second Order | First Order



RESULTS



$$|\delta V(\phi)| = \left| \frac{V_{\text{theory}}(\phi) - V_{\text{PINN}}(\phi)}{V_{\text{theory}}(\phi)} \right|$$

$$|\delta T(S)| = \left| \frac{T_{\text{input}}(S) - T_{\text{PINN}}(S)}{T_{\text{input}}(S)} \right|$$