Introduction to Scientific Machine Learning

Jian Tao

jtao@tamu.edu

HPRC Short Course

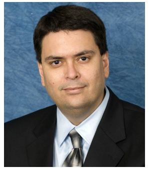
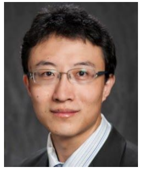
03/26/2021

Texas A&M Engineering Experiment Station

High Performance Research Computing
DIVISION OF RESEARCH

TEXAS A&M
Institute of Data Science

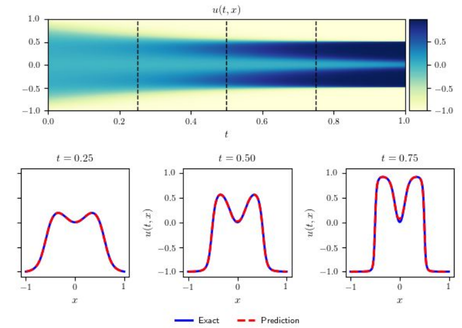
LAB MEMBERS



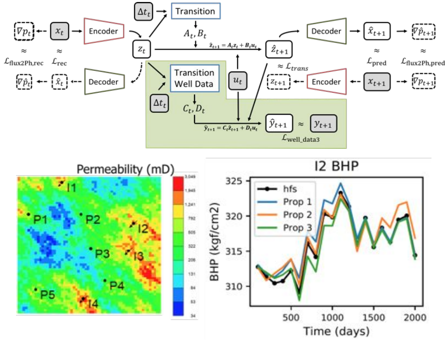



Pilot Project 1 - Microstructure Informatics



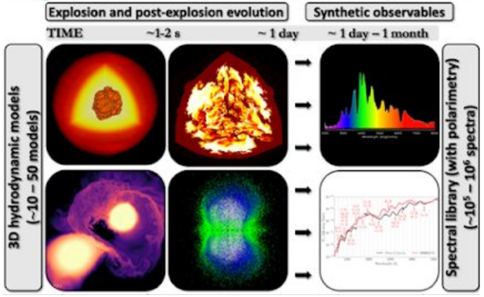
Raymundo Arroyave, Ulisses Braga-Neto, Levi McClenny, Vahid Attari

Pilot Project 2 - Reservoir Simulation



Eduardo Gildin, Ulisses Braga-Neto, Yalchin Efendiev

Pilot Project 3 - Thermonuclear Supernovae



Lifan Wang, Jian Tao, Lisa Perez

TensorDiffeq

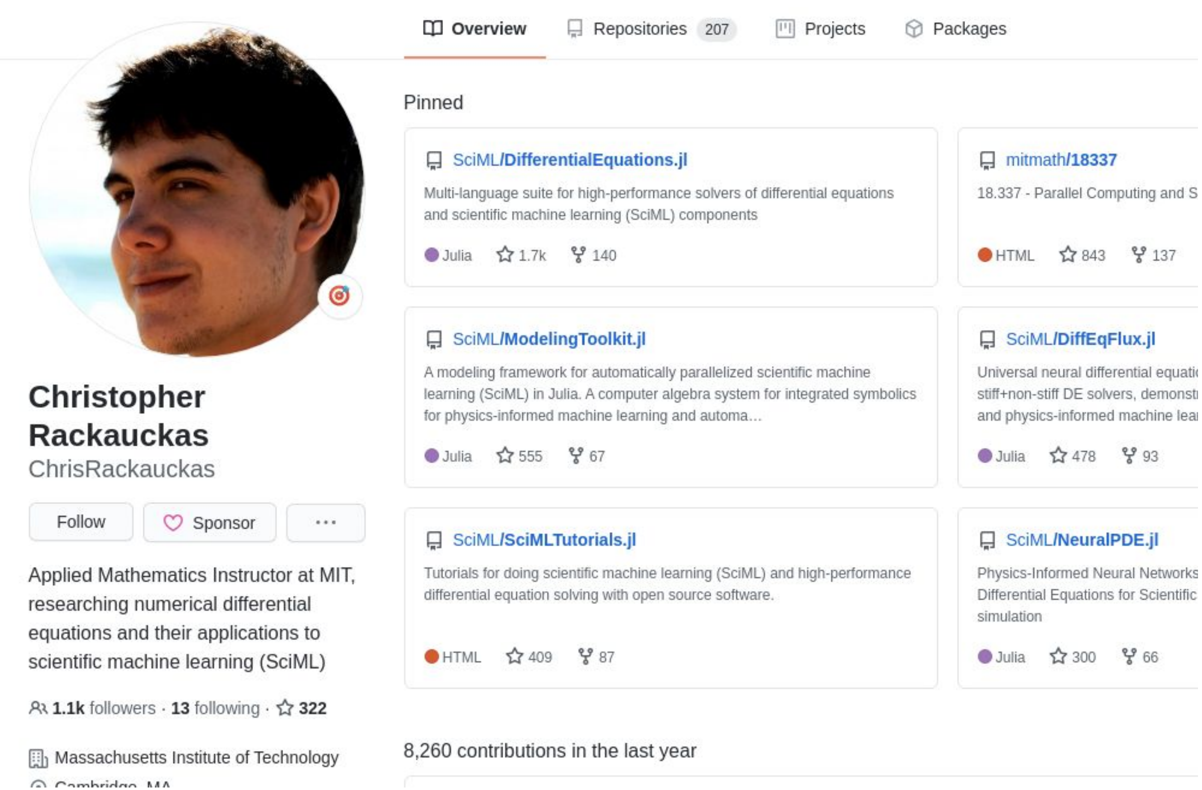
TensorDiffEq is a python package built on top of Tensorflow to provide scalable and efficient PINN solvers. TensorDiffEq's primary purpose is for scalable solving of PINNs (inference) and inverse problems (discovery).

Additionally, TensorDiffEq is the only package that fully supports and implements Self-Adaptive PINN solvers and is the only Multi-GPU PINN solution suite that is fully open-source.

Levi McClenny, Ulisses Braga-Neto

https://sciml.tamids.tamu.edu/

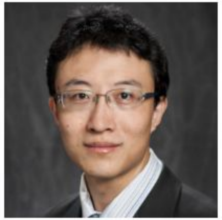
Upcoming TAMIDS SciML Lab Talk (April 14)



https://github.com/ChrisRackauckas

TAMIDS-SciML
DIVISION OF RESEARCH

Upcoming Hackathon on Material Design with Graph Learning (April 19 - 23)



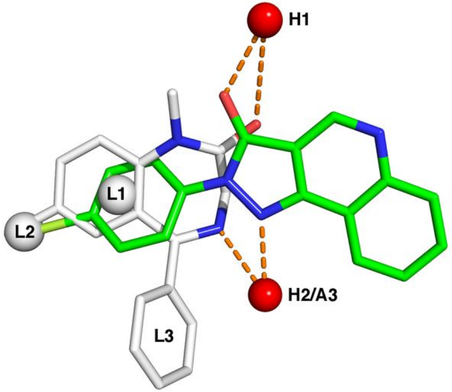
ASSOCIATE DIRECTOR

Jian Tao
TEES Research Scientist / Computational Scientist / Adjunct Professor
Strategic Initiatives, Texas A&M Engineering Experiment Station
Texas A&M Institute of Data Science
High Performance Research Computing, Texas A&M University

Numerical analysis; workflow management; data science; HPC; scientific computation.

Raymundo Arroyave
Professor
Department of Materials Science and Engineering

Materials Science, Numerical Methods



DiveIntoGraphs

One week long Hackathon to explore potential applications of graphical learning in material design.
Please contact jtao@tamu.edu if you are interested.

https://github.com/divelab/DIG/

TAMIDS-SciML
DIVISION OF RESEARCH

Upcoming Tutorial on TensorDiffeq (Early May)

Levi McClenny
Research Assistant
Department of Electrical Engineering

Physics-Informed Deep Learning, Physics-Explainable AI, PINNs, Materials Science

TensorDiffeq

Package Build passing | Package Release passing | pypi v0.1.6.7 | downloads 135/month | python 3.6 | 3.7 | 3.8

Efficient and Scalable Physics-Informed Deep Learning

Collocation-based PINN PDE solvers for prediction and discovery methods on top of Tensorflow 2.X for multi-worker distributed computing.

Use TensorDiffEq if you require:

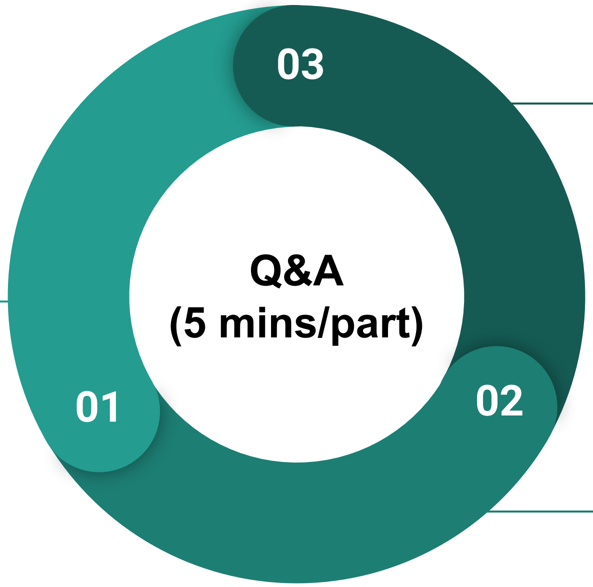
- A meshless PINN solver that can distribute over multiple workers (GPUs) for forward problems (inference) and inverse problems (discovery)
- Scalable domains - Iterated solver construction allows for N-D spatio-temporal support
  - support for N-D spatial domains with no time element is included
- Self-Adaptive Collocation methods for forward and inverse PINNs
- Intuitive user interface allowing for explicit definitions of variable domains, boundary conditions, initial conditions, and strong-form PDEs

https://github.com/tensordiffeq/TensorDiffEq

Introduction to Scientific Machine Learning

Part I

Setting up a working environment on Terra (15 mins)



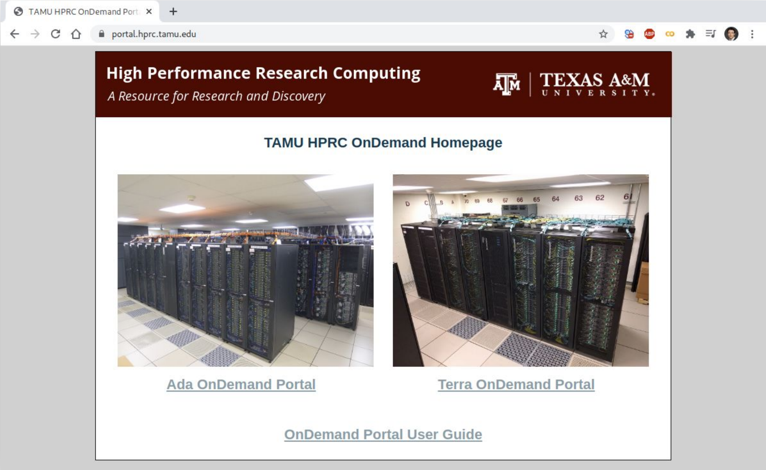
Part III

Hands-on Demos (30 mins)

Part II

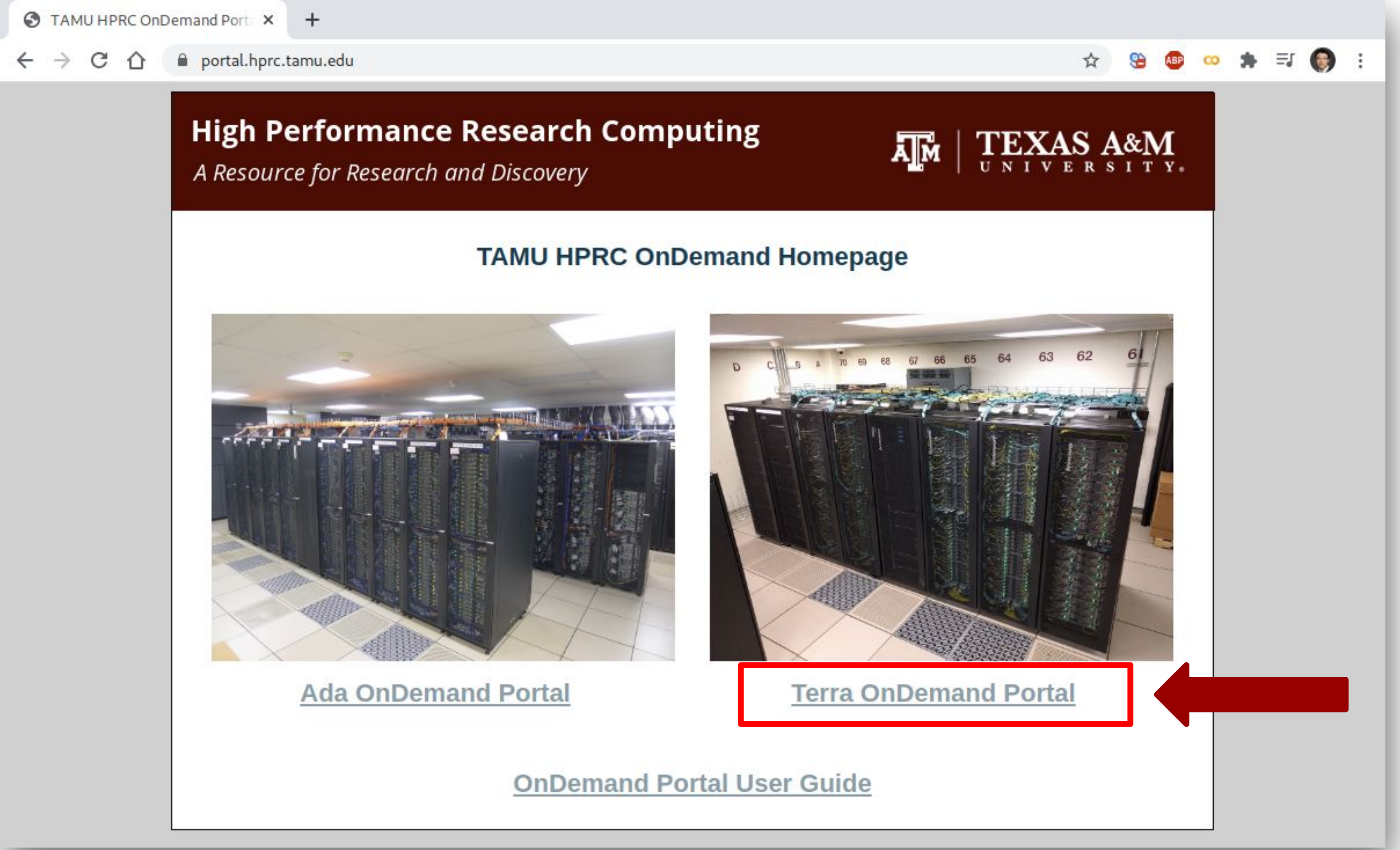
Introduction to Scientific Machine Learning (90 mins)

Part I. Working Environment

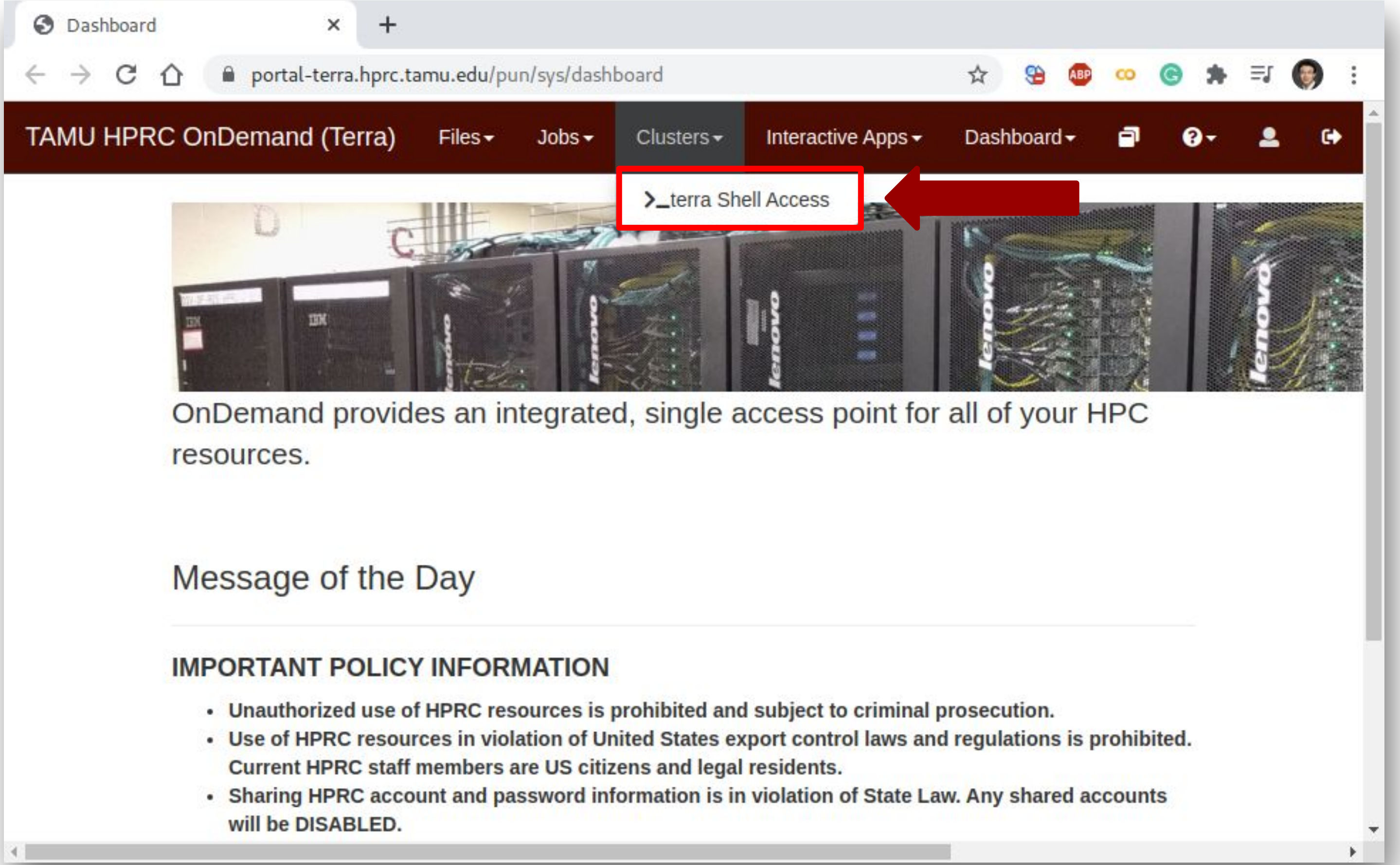


HPRC Portal

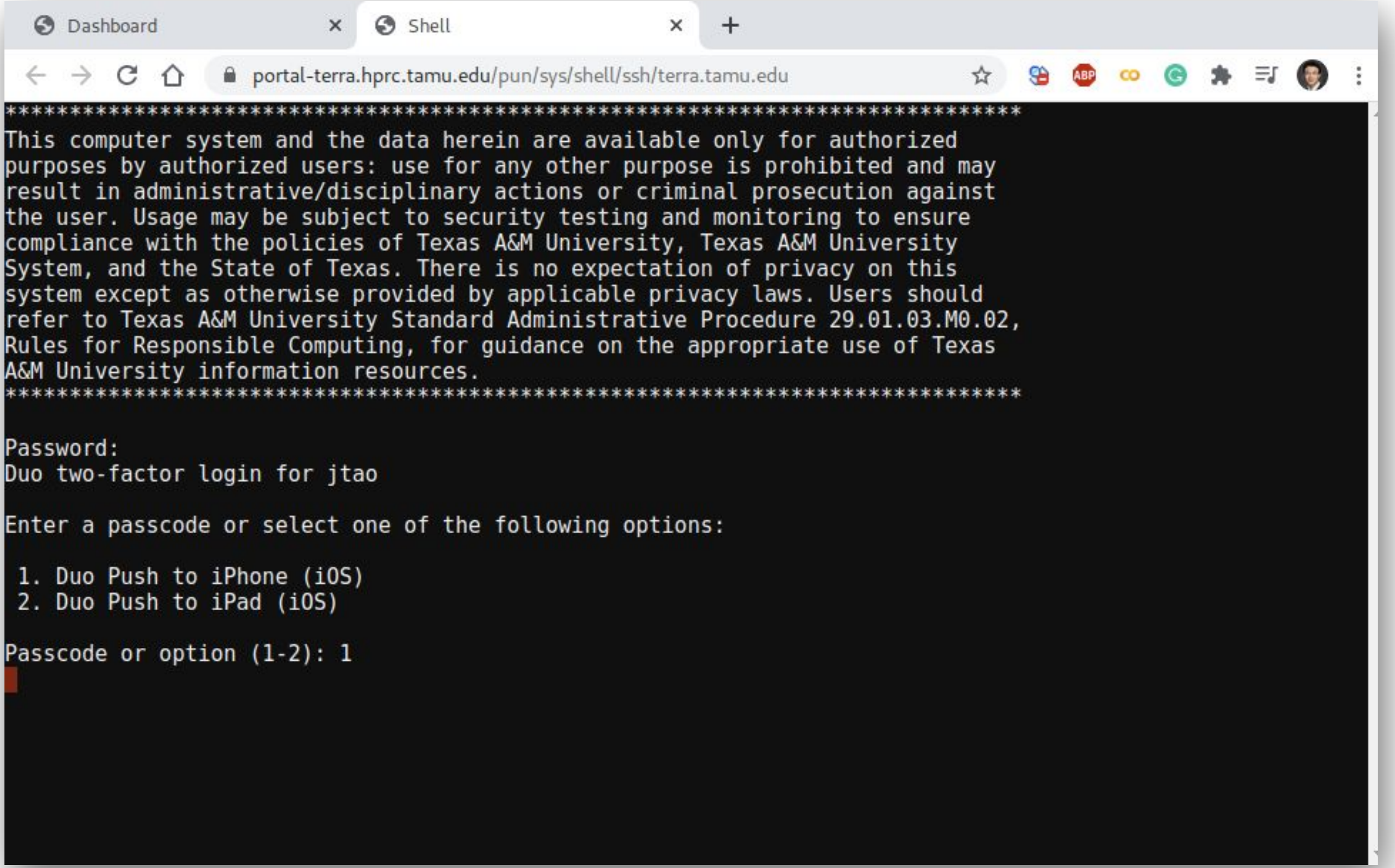
Login HPRC Portal (Terra)



Terra Shell Access - I



Terra Shell Access - II



Using Pre-installed Julia Module

Step 1. Find the module to be loaded

$ module spider julia
...
Julia/0.6.2-intel-2017A-Python-2.7.12-ParMETIS-4.0.3
Julia/0.6.2-intel-2017A-Python-2.7.12-METIS-5.1.0
Julia/0.6.2-intel-2017A-Python-2.7.12-noSuiteSparse
Julia/1.0.5-linux-x86_64
Julia/1.4.1-linux-x86_64
Julia/1.5.1-linux-x86_64
...

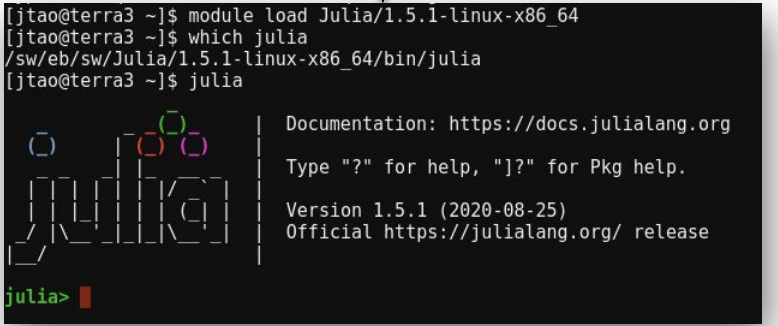
(You can also use the web-based interface to find software modules available on HPRC systems. For experienced users, please jump to Step 2.)

Step 2. Load the module

$ module load Julia/1.5.1-linux-x86_64

Step 3. Start Julia REPL

$ julia



SW:Julia - TAMU HPRC

Using Your Own Julia Installation

Step 1. Find the version to be installed

Current stable release: v1.5.4 (March 11, 2021)

Checksums for this release are available in both MD5 and SHA256 formats.

Windows [help]		ortable)
macOS [help]		
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl)[1] (GPG)	
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)	
Generic FreeBSD on x86 [help]	64-bit (GPG)	
Source	Tarball (GPG)	Tarball with depe

Right click and copy the link.

(You can find different versions of Julia at [Download Julia](), The latest stable version of Julia is highly recommended.)

Step 2. Dow... Unzip

Paste the link here.

```
$ cd $SCRATCH
$ wget https://.../julia-1.5.4-linux-x86_64.tar.gz
$ tar zxvf julia-1.5.4-linux-x86_64.tar.gz
```

Step 3. Start Julia Shell

```
$ cd $SCRATCH/julia-1.5.4/bin; ./julia
```

(Ctrl+D or type exit() to quit Julia shell)

[SW:Julia - TAMU HPRC]()

Install Julia Packages

```
# Set Julia Depot path under $SCRATCH
$export JULIA_DEPOT_PATH=$SCRATCH/.julia

# start Julia
$cd $SCRATCH/julia-1.5.4/bin; ./julia

# type ']' to open Pkg REPL
julia>]
(@v1.5) pkg> add Plots
```

Julia - Quickstart

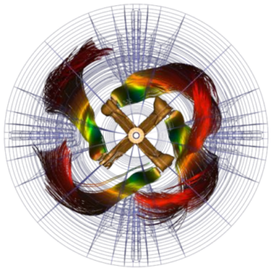
The julia program starts the interactive **REPL**. You will be immediately switched to the **shell mode** if you type a **semicolon**. A **question mark** will switch you to the **help mode**. The <TAB> key can help with autocompletion.

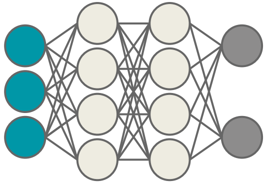
```
julia> versioninfo()
julia> VERSION
```

Special symbols can be typed with the **escape symbol and** <TAB>, but they might not show properly on the web-based terminal.

```
julia> \sqrt <TAB>
julia> for i ∈ 1:10 println(i) end  #\in <TAB>
```

Part II. Introduction to Scientific Machine Learning (SciML)





Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence
by Baker, Nathan, et. al.
https://doi.org/10.2172/1478744

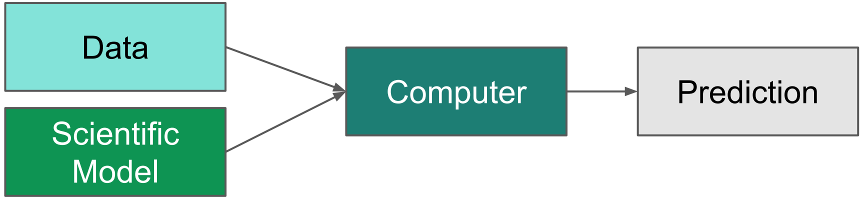
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.
by Raissi, M., Perdikaris, P., & Karniadakis, G. E.
Journal of Computational Physics, 378, 686-707.
https://doi.org/10.1016/j.jcp.2018.10.045

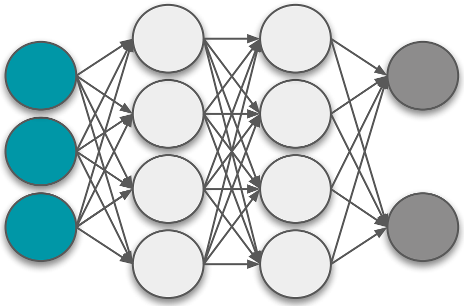
SciML Scientific Machine Learning Software
by Chris Rackauckas et. al.
https://sciml.ai/

Data-Driven vs Theory-Driven

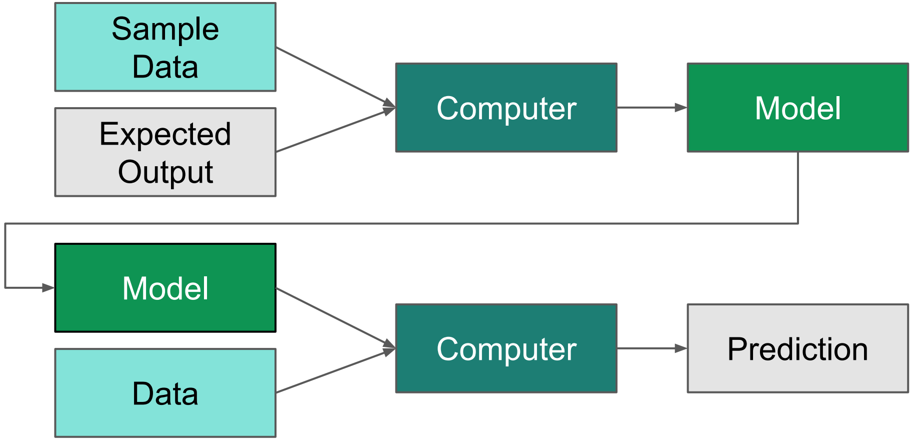
$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \gamma \nabla^2 \mathbf{u} + \frac{1}{\rho}\mathbf{F}$$

Theory-Driven Modeling (Numerical Simulation)





Data-Driven Modeling (Supervised Learning)



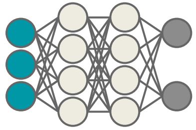
Balance between Data and Theory

Theory-Driven Model

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \gamma \nabla^2 \mathbf{u} + \frac{1}{\rho}\mathbf{F}$$

Data-Driven / ML Model

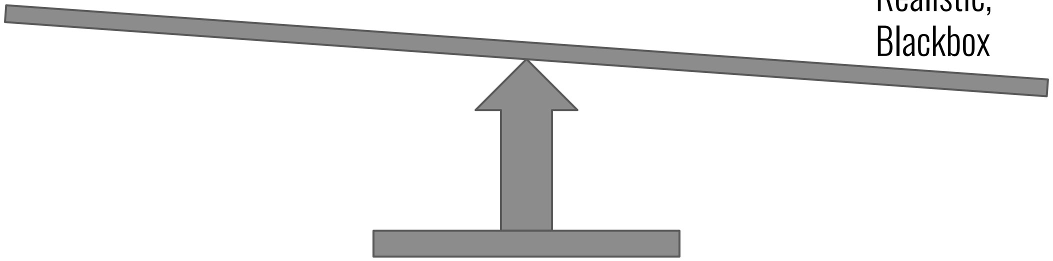
0101010101010
0001011001001



Less Data,
More Theory,
Clean,
Unrealistic,
Explainable

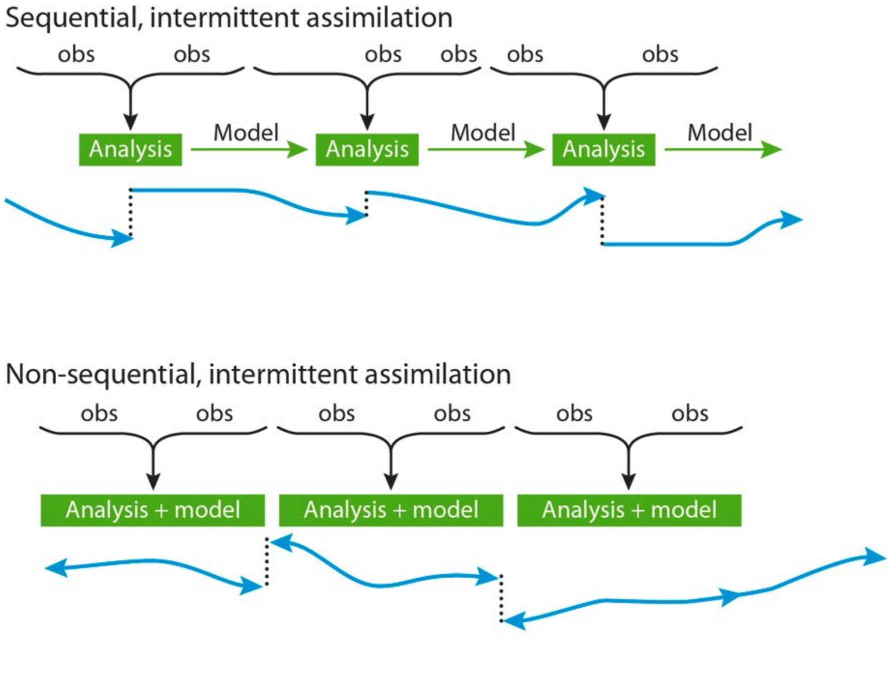
Data Assimilation

More Data,
Less Theory,
Noisy,
Realistic,
Blackbox



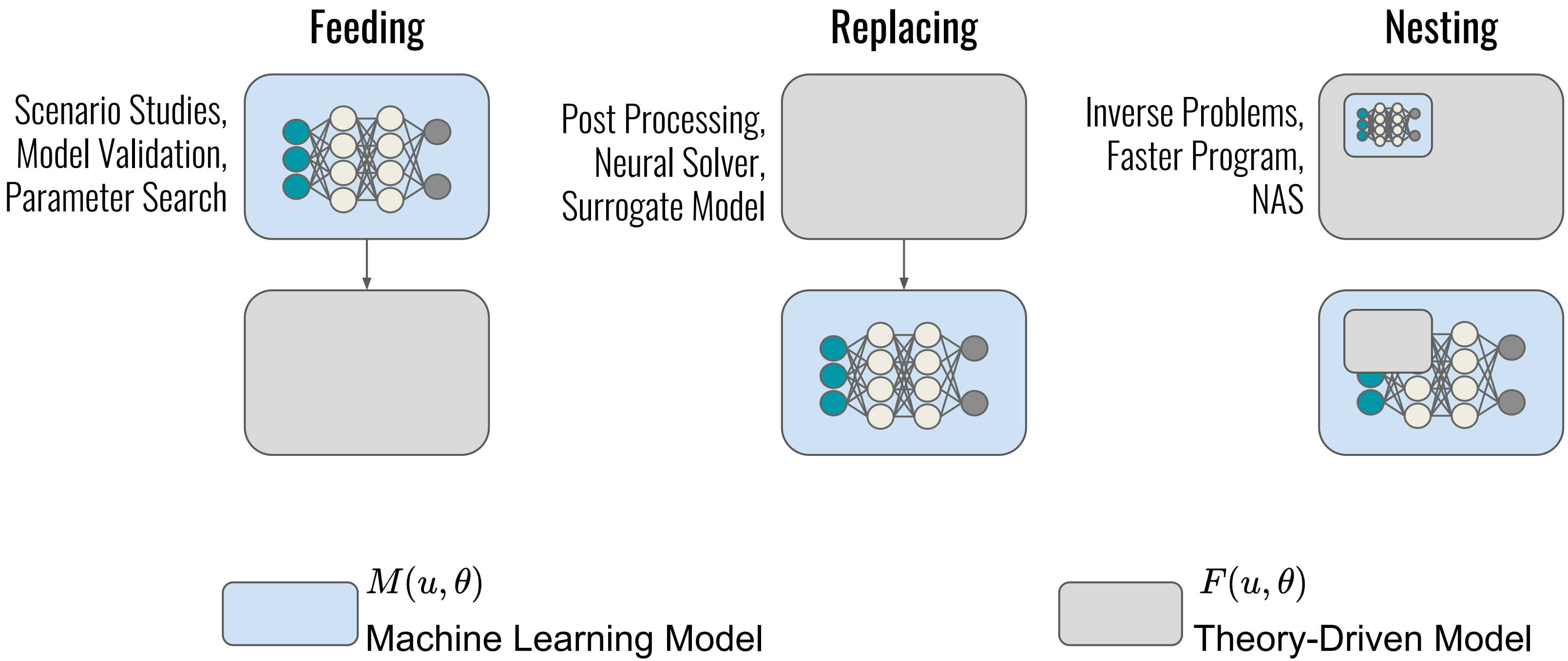
* Data assimilation is somewhere in between but not necessarily balanced.

Middle Ground - Data Assimilation

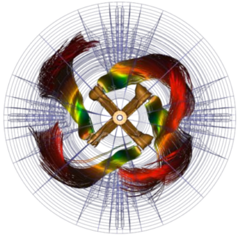


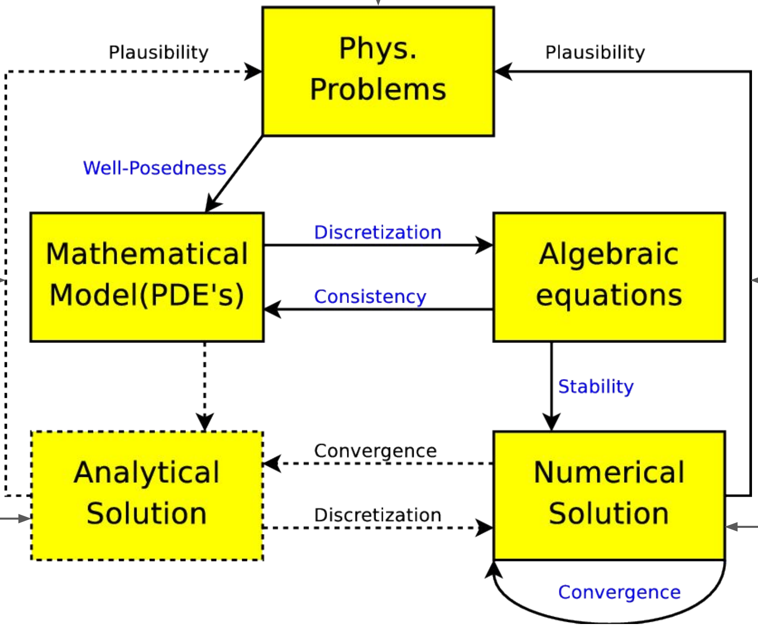
[Simplistic Overview of Reanalysis Data Assimilation Methods | NCAR]

SciML - Best of the Two Worlds



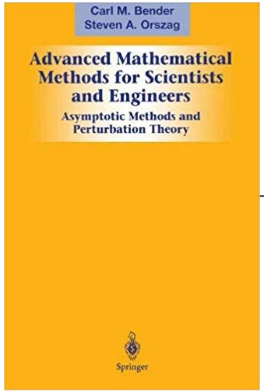
Theory / Numerical Modeling

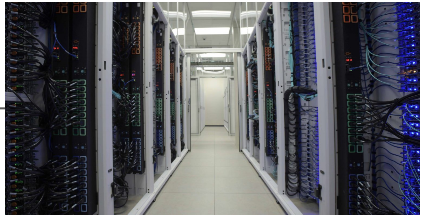




$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

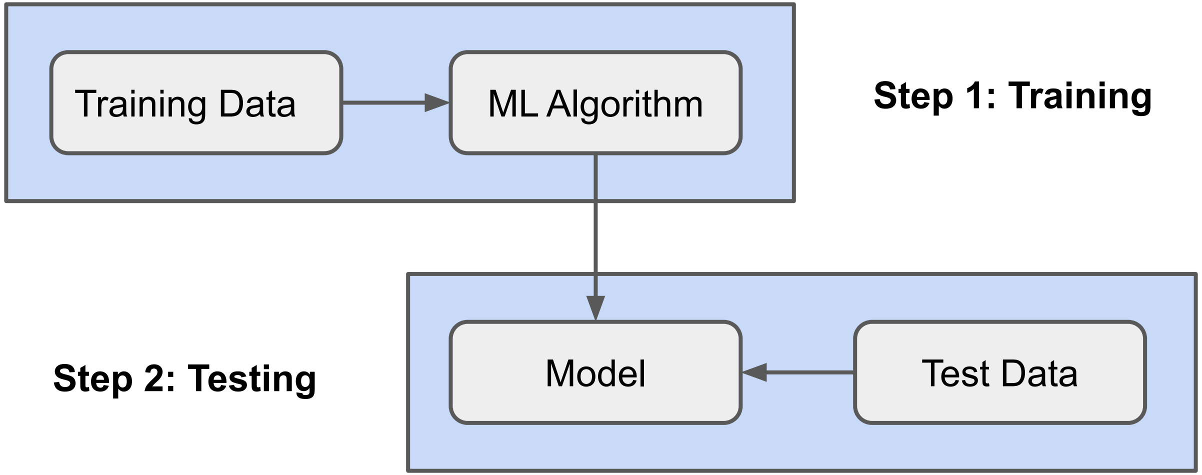
$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)}{\Delta x^2}$$



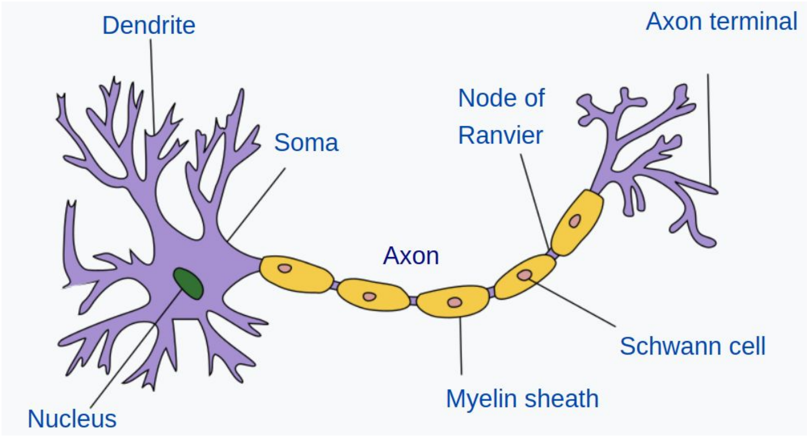


Data-Driven Model - Supervised Learning

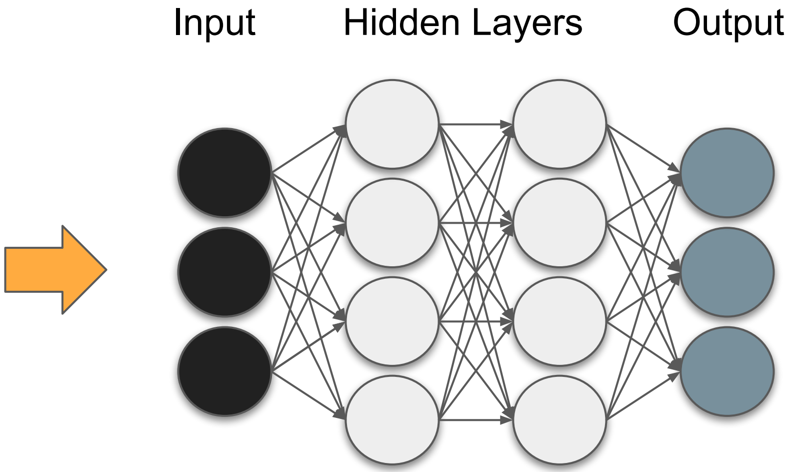
When both input variables - X and output variables - Y are known, one can approximate the mapping function from X to Y.



Artificial Neural Network



(Image Credit: Wikipedia)

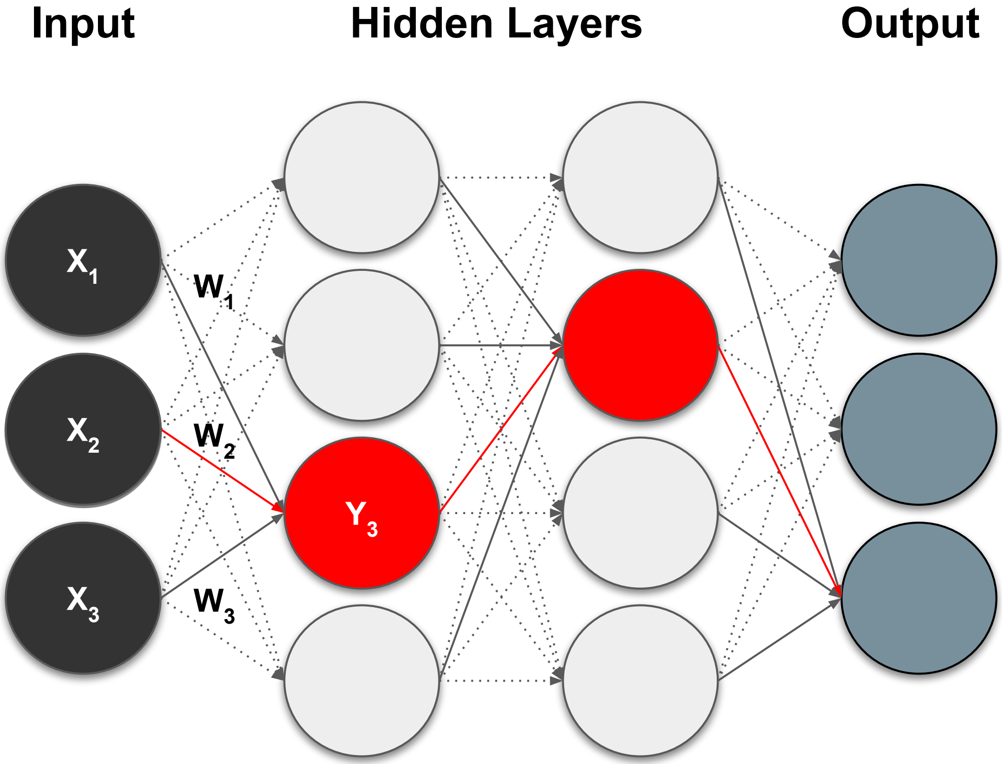


Supervised Deep Learning with Neural Networks

From one layer to the next

$$Y_j = f\left(\sum_i W_i X_i + b_i\right)$$

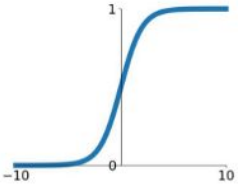
f is the activation function, W_i is the weight, and b_i is the bias.



Activation Functions

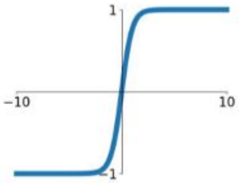
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



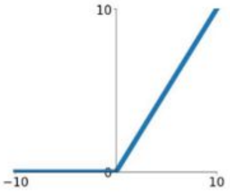
tanh

$\tanh(x)$



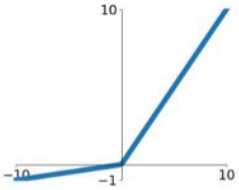
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$



Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

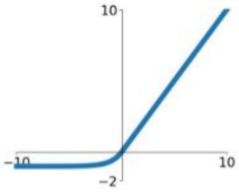


Image Credit: towardsdatascience.com

Training - Minimizing the Loss

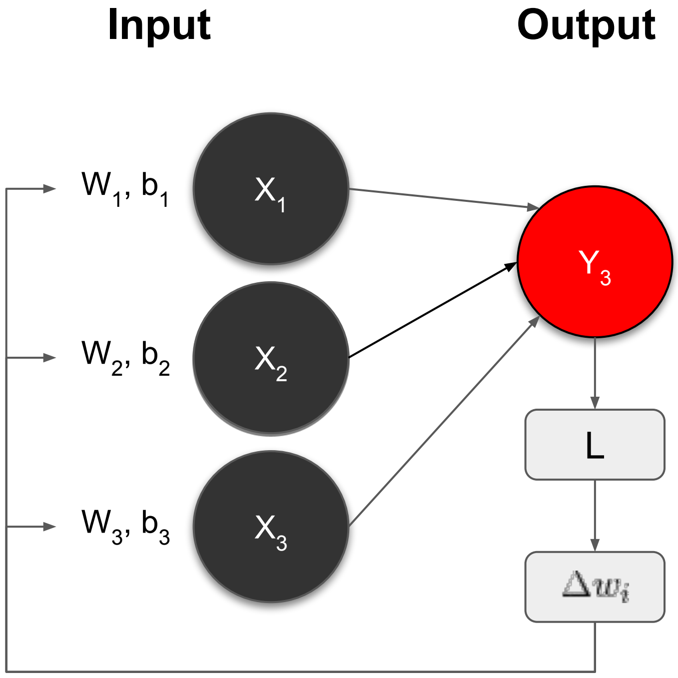
The loss function with regard to weights and biases can be defined as

$$L(\mathbf{w}, \mathbf{b}) = \frac{1}{2}\sum_i (\mathbf{Y}(\mathbf{X}, \mathbf{w}, \mathbf{b}) - \mathbf{Y}'(\mathbf{X}, \mathbf{w}, \mathbf{b}))^2$$

The weight update is computed by moving a step to the opposite direction of the cost gradient.

$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$

Iterate until L stops decreasing.



Deep Neural Network as a Universal Approximator

Universal Approximation Theorem

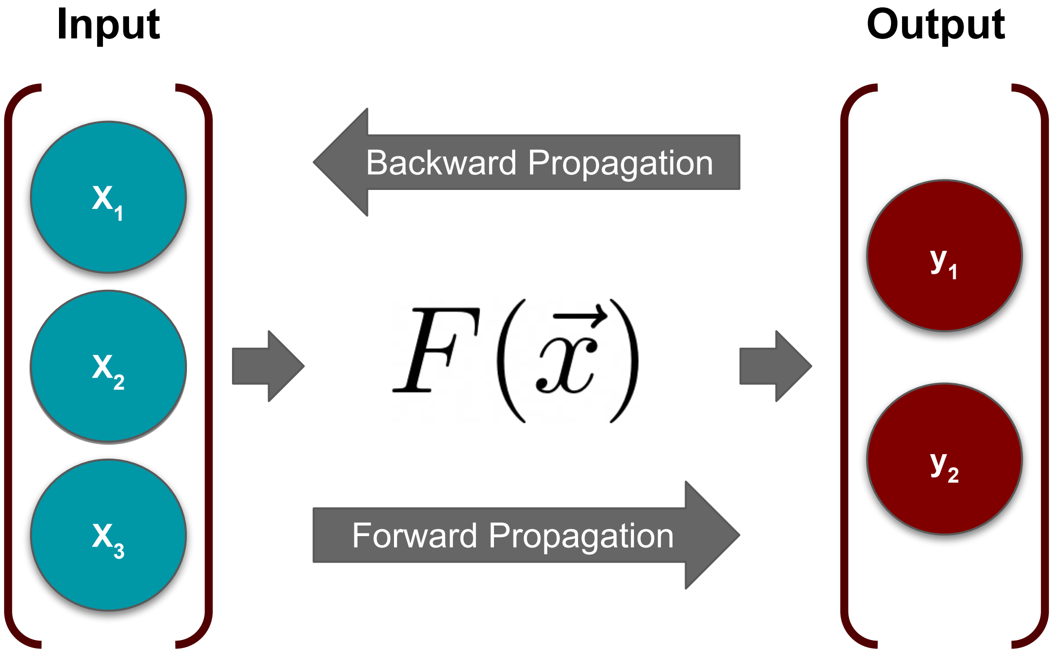
(Cybenko, 1989)

Universal approximation theorems imply that neural networks can represent a wide variety of **functions**.

Pinkus Theorem

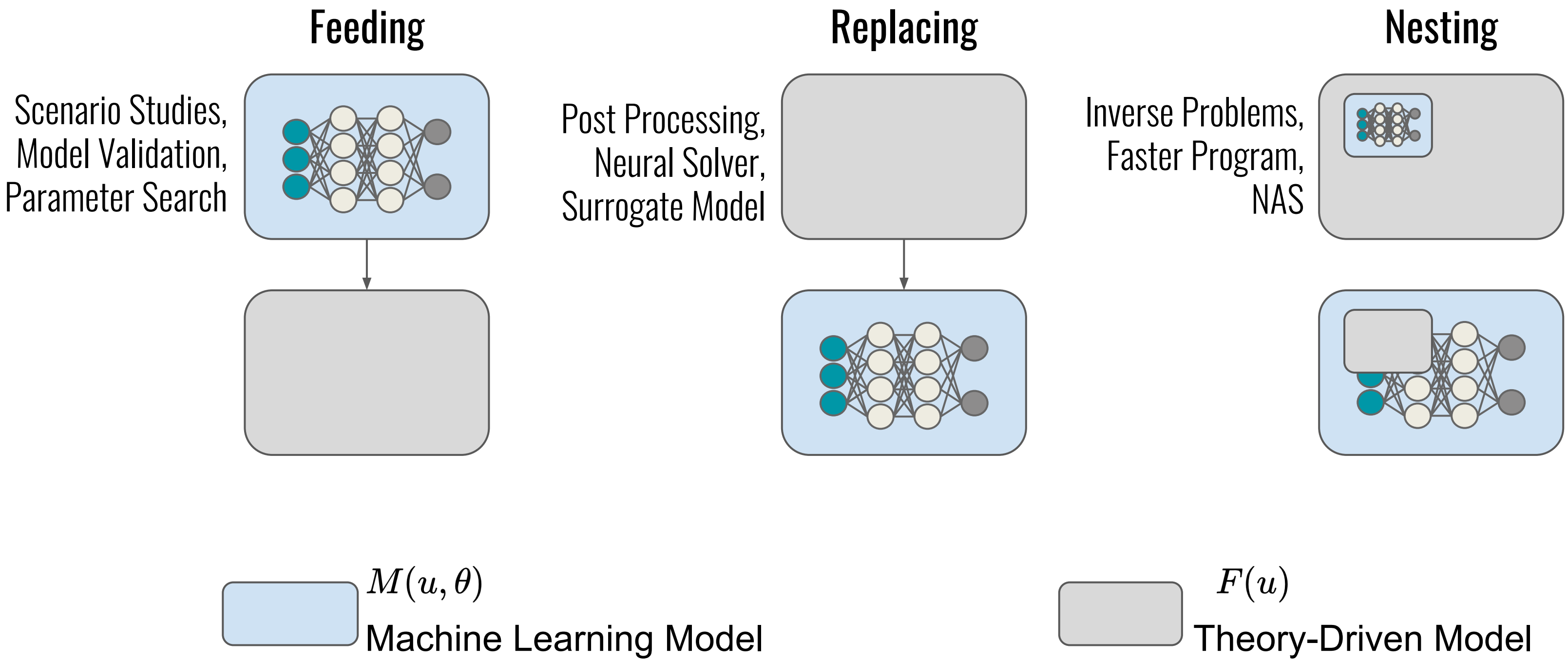
(Pinkus, 1999)

Pinkus theorems imply that neural networks can represent **directives of a function** simultaneously.

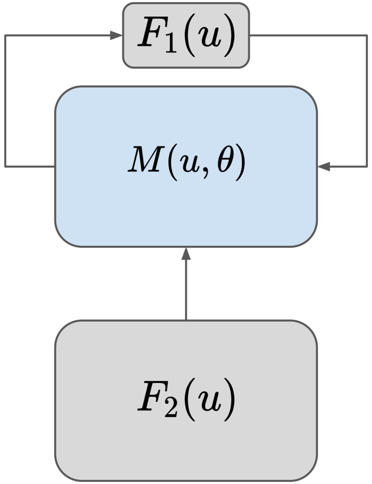


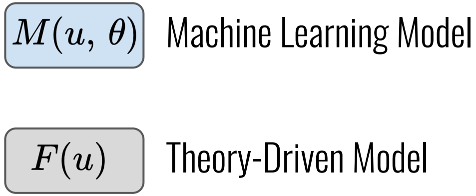
- **Training:** given **input** and **output**, find best-fit ***F***
- **Inference:** given **input** and ***F***, predict **output**

SciML - Best of the Two Worlds



Sample Workflow of a SciML Application





Nested Function

$$Y(u,\,\theta)\;=\;F_2(M(F1(M(u,\,\theta)),\theta))$$

Loss Function

$$\boxed{L(\theta)=\tfrac{1}{2}\sum_i\left(Y(u,\,\theta)\;-\;Y'(u,\theta)\right)^2+L_{I/BC/\ldots}(\theta)}$$

Gradient Descent

$$\Delta\theta=-\alpha\frac{\partial L}{\partial\theta}$$

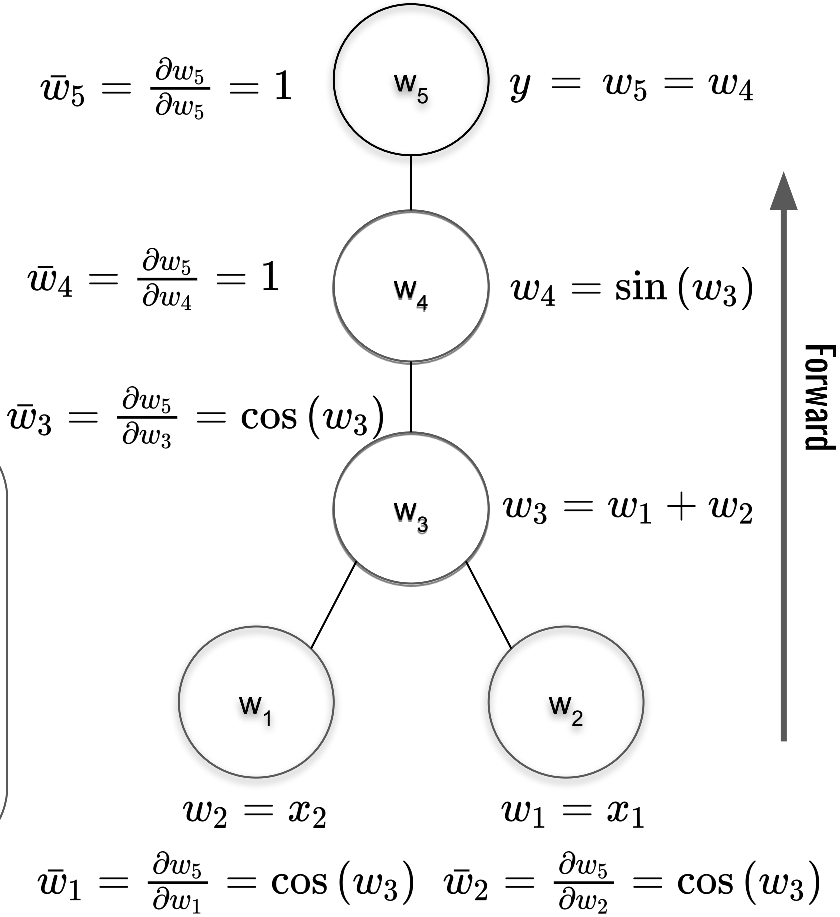
Automatic Differentiation

An autodiff system converts the program into a sequence of primitive operations to compute derivatives.

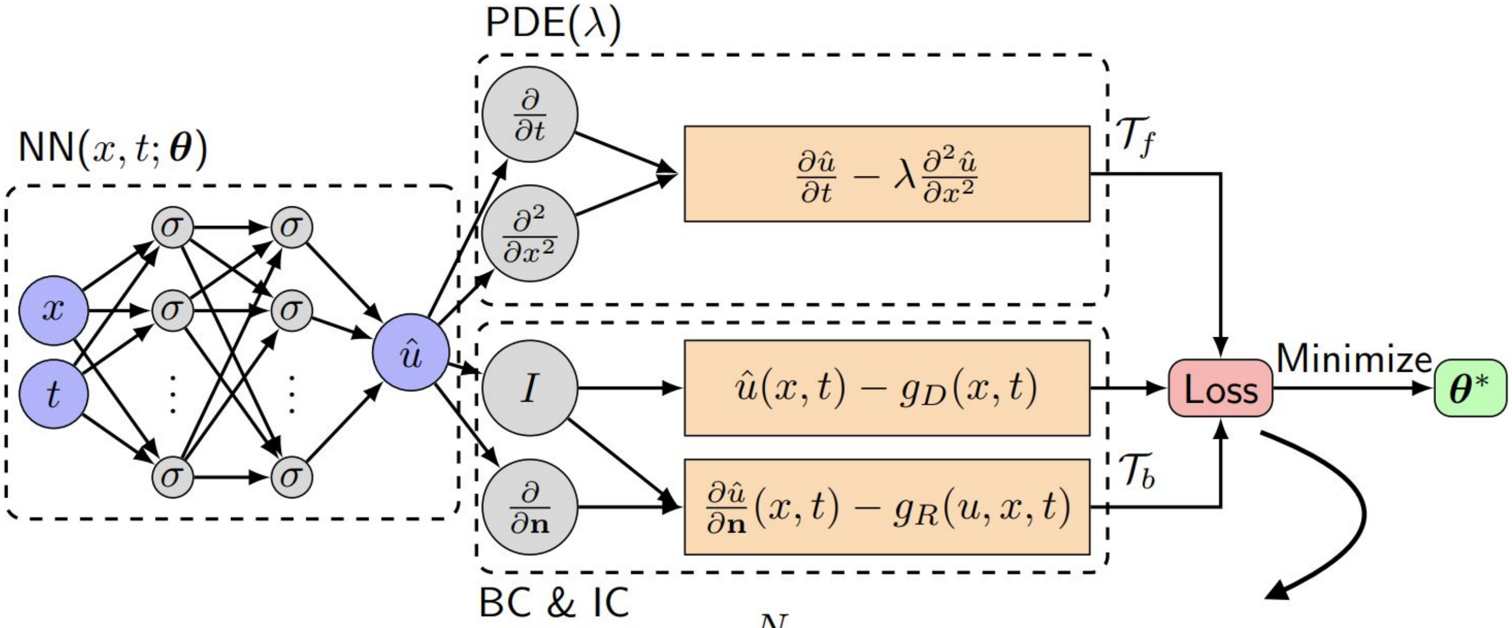
Given $$y(x_1, x_2) = \sin(x_1 + x_2)$$

Numerical:
(Finite Difference) $$\frac{\Delta y}{\Delta x_1} = \frac{\sin(x_1+h+x_2)-sin(x_1+x_2)}{h}$$

Symbolic: $$y'(x_1, x_2) = \cos(x_1 + x_2)$$



Physics-Informed Neural Networks (PINNs)



$$\mathcal{L}(\theta) := \underbrace{\frac{1}{N_u} \sum_{i=1}^{N_u} [\boldsymbol{u}_i - f_\theta(\boldsymbol{x}_i)]^2}_{\text{Data fit}} + \underbrace{\frac{1}{\lambda} \mathcal{R}[f_\theta(\boldsymbol{x})]}_{\text{Physics regularization}}$$

Lu Lu, Xuhui Meng, Zhiping Mao, George E. Karniadakis, DeepXDE: A deep learning library for solving differential equations (https://arxiv.org/pdf/1907.04502.pdf)

Hands-on Session

Getting Started with NeuralPDE.jl

Install Julia Packages

```
# Set Julia Depot path under $SCRATCH
$export JULIA_DEPOT_PATH=$SCRATCH/.julia

# start Julia
$cd $SCRATCH/julia-1.5.4/bin; ./julia

# type ']' to open Pkg REPL
julia>]
(@v1.5) pkg> add Plots, NeuralPDE, Flux, ModelingToolkit, GalacticOptim, Optim,
DiffEqFlux

# type 'Backspace' to get back to REPL and paste the code directly into the shell.
julia> using NeuralPDE, Flux, ModelingToolkit, GalacticOptim, Optim, DiffEqFlux
```

ODE with a 3rd-Order Derivative

$$\frac{\partial^3 u(x)}{\partial x^3} = \cos(\pi x)$$
$$u(0) = 0$$
$$u(1) = \cos(\pi)$$
$$\frac{\partial u(0)}{\partial x} = 1$$
$$x \in [0, 1]$$

```julia
using NeuralPDE, Flux, ModelingToolkit, GalacticOptim, Optim, DiffEqFlux

@parameters x
@variables u(..)

Dxxx = Differential(x)^3
Dx = Differential(x)
# ODE
eq = Dxxx(u(x)) ~ cos(pi*x)

# Initial and boundary conditions
bcs = [u(0.) ~ 0.0,
       u(1.) ~ cos(pi),
       Dx(u(1.)) ~ 1.0]

# Space and time domains
domains = [x ∈ IntervalDomain(0.0,1.0)]

# Neural network
chain = FastChain(FastDense(1,8,Flux.σ),FastDense(8,1))

discretization = PhysicsInformedNN(chain, QuasiRandomTraining(20))
pde_system = PDESystem(eq,bcs,domains,[x],[u])
prob = discretize(pde_system,discretization)

cb = function (p,l)
    println("Current loss is: $l")
    return false
end

res = GalacticOptim.solve(prob, ADAM(0.01); cb = cb, maxiters=2000)
phi = discretization.phi
```

```julia
using UnicodePlots

analytic_sol_func(x) =
(π*x*(-x+(π^2)*(2*x-3)+1)-sin(π*x))/(π^3)

dx = 0.05
xs =
[domain.domain.lower:dx/10:domain.domain.upp
er for domain in domains][1]
u_real  = [analytic_sol_func(x) for x in xs]
u_predict  = [first(phi(x,res.minimizer))
for x in xs]

x_plot = collect(xs)
plot(x_plot ,u_real,title = "real")
plot!(x_plot ,u_predict,title = "predict")
```

[ODE with a 3rd-Order Derivative · NeuralPDE.jl]()