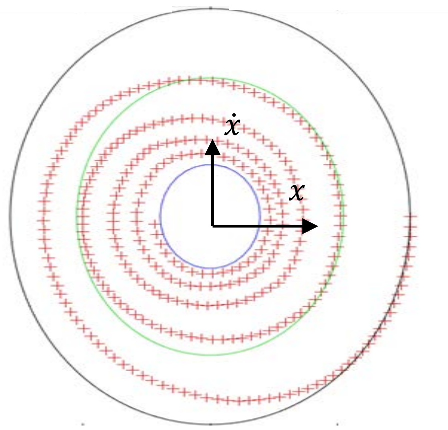
Physics-Informed Machine Learning

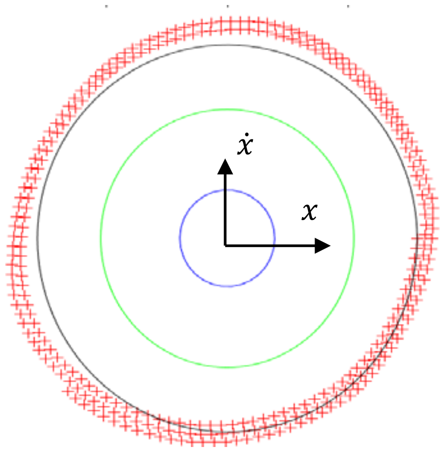
Using the laws of nature to improve generalized deep learning models

Traditional ML



vs.

Physics-Informed ML



Stanford University

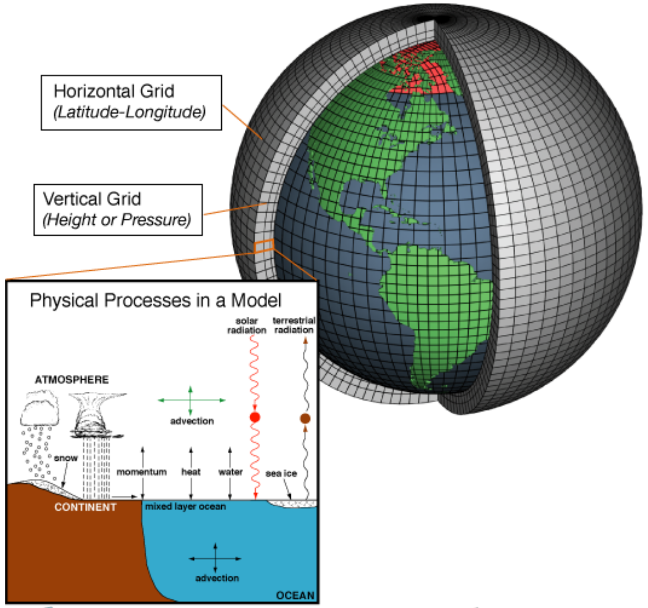
Dr. Sam Raymond | Stanford University

MathWorks

Introduction - Fusing Data and Simulation

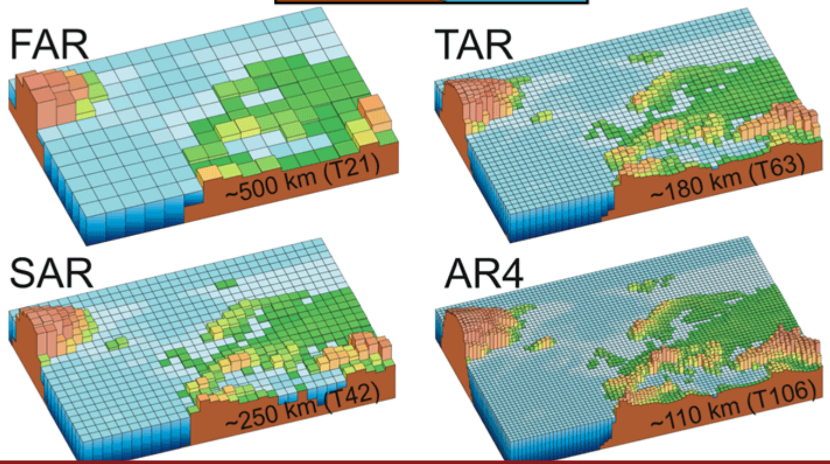
Climate Models

- Attention is now being drawn to using ML with scientific data for better predictive power
- ML can be used as simulators to generate a prediction of the updated state of a system
- This can be useful in cases like climate models where simulations are expensive and slow
- However, these ML models are trained only to minimize the error between data and predictions, no physical information is used to inform this training
- This lack of physics-informed machine learning can limit the extrapolation efficacy
- Adding some physics into the training can aid this workflow for more robust predictions



Climate modeling

- Huge domains
- Billions of variables
- Massive time scales

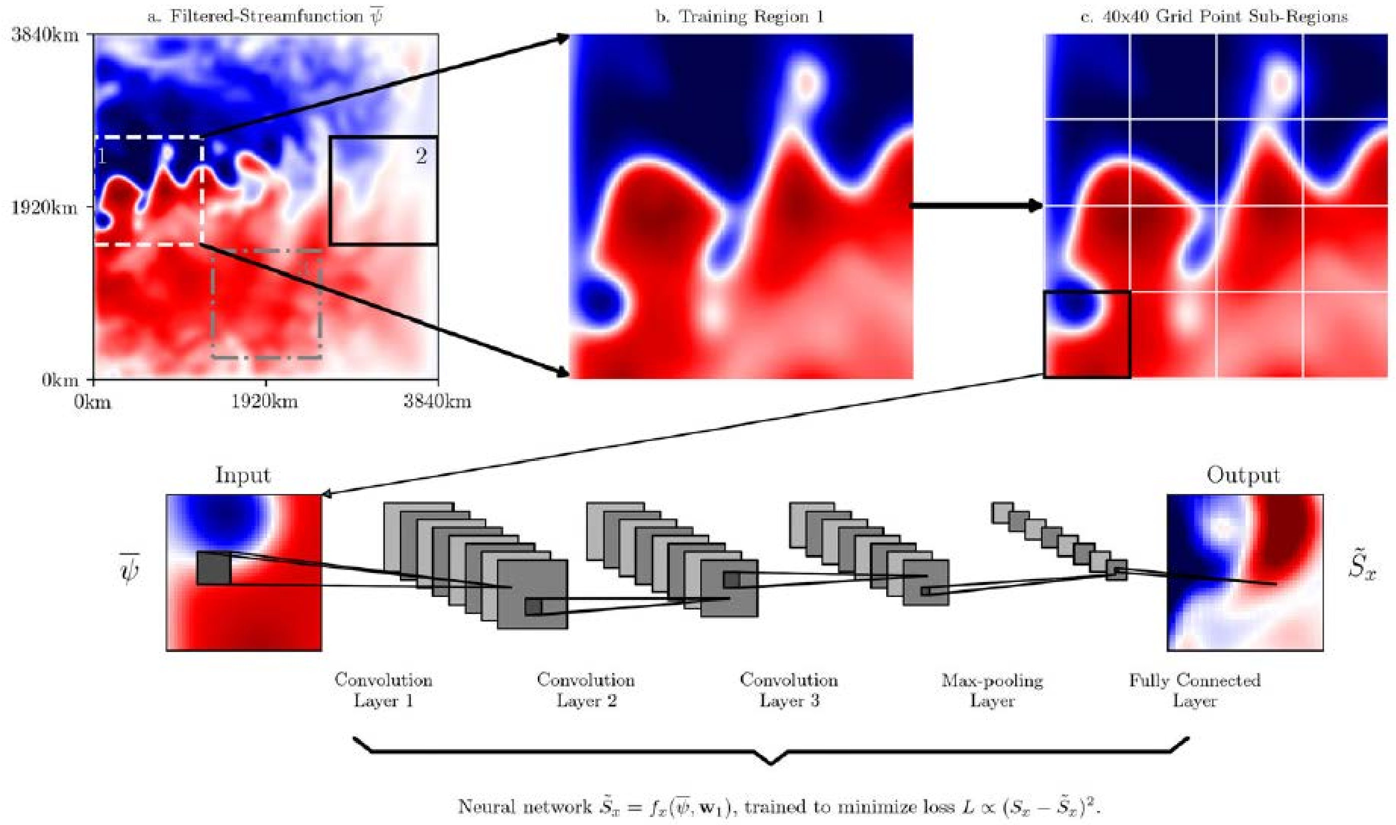


Introduction – Fusing Data and Simulation

Typically, these models involving solving complex CFD systems with interacting scales is extremely costly.

A new approach:

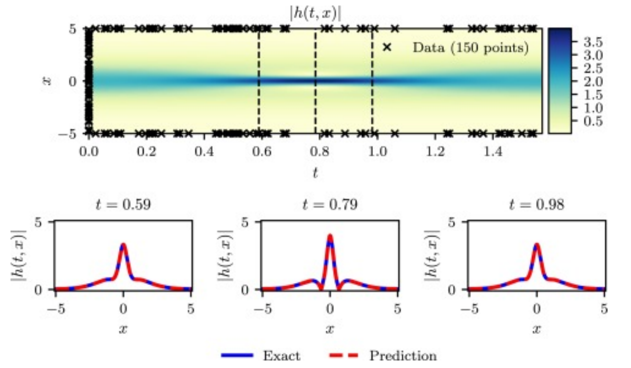
- Deep Learning-powered simulations
- Using data to power more efficient solutions
- Learn on one type of weather to help predict new weather patterns



Introduction – Physics Informed Machine Learning

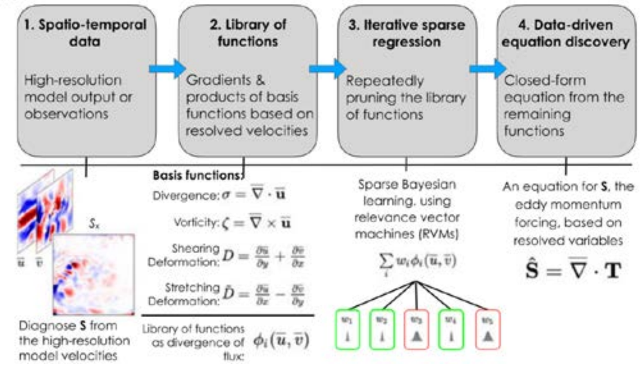
Physics-Informed Neural Networks

M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, Volume 378, 2019. https://doi.org/10.1016/j.jcp.2018.10.045



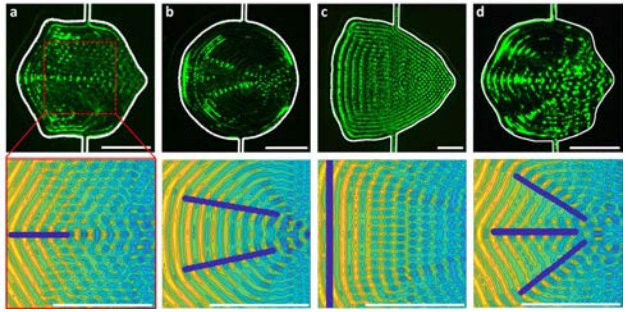
Learning equations

Zanna, L., Bolton, T. (2020). Data-driven equation discovery of ocean mesoscale closures. *Geophysical Research Letters*, 47, e2020GL088376. https://doi.org/10.1029/2020GL088376



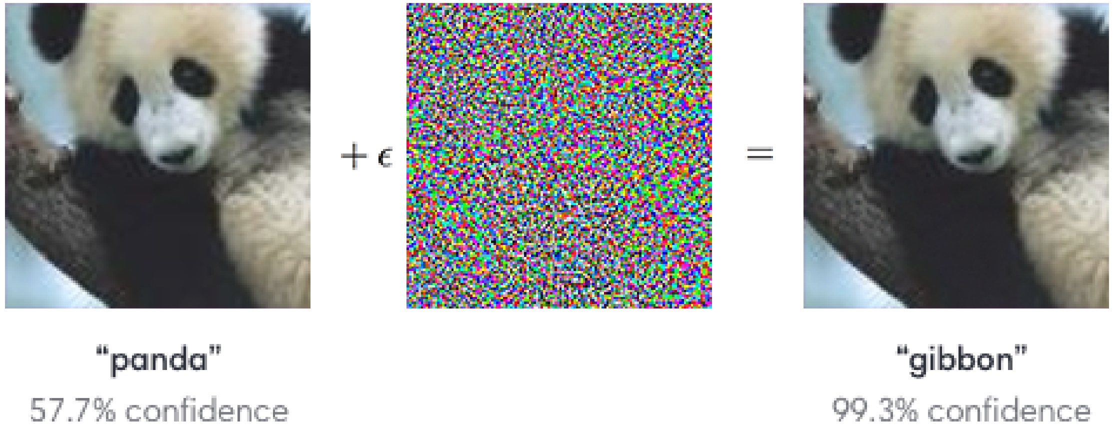
Synthetic data for inverse-design

Raymond, S.J., Collins, D.J., O'Rorke, R. *et al.* A deep learning approach for designed diffraction-based acoustic patterning in microchannels. *Sci Rep* **10,** 8745 (2020). https://doi.org/10.1038/s41598-020-65453-8



How well can we trust ML-infused physics?

- A system cannot be well predicted by simply learning the data for a physical system.
- This system obeys laws that simply fitting data may not learn
- When using this model to extrapolate, this can lead to divergences
- This may be mediated by adding some existing knowledge to the training process



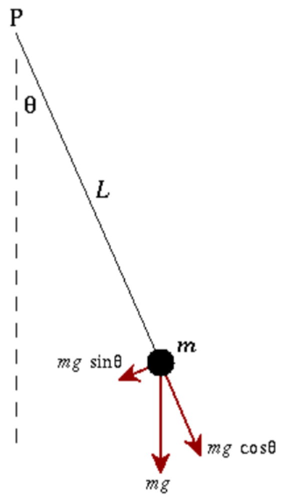
Conventional neural networks can be brittle and vulnerable to *adversarial* attacks.

A simpler example – A pendulum

Complex systems like climate models can be represented by much simpler models.

These models are typically mechanical systems where kinetic and potential energy oscillate. Pendulums and other oscillators are used often in physics to understand more complex processes.

Here we will focus on the prediction of the motion of a pendulum released from different heights.



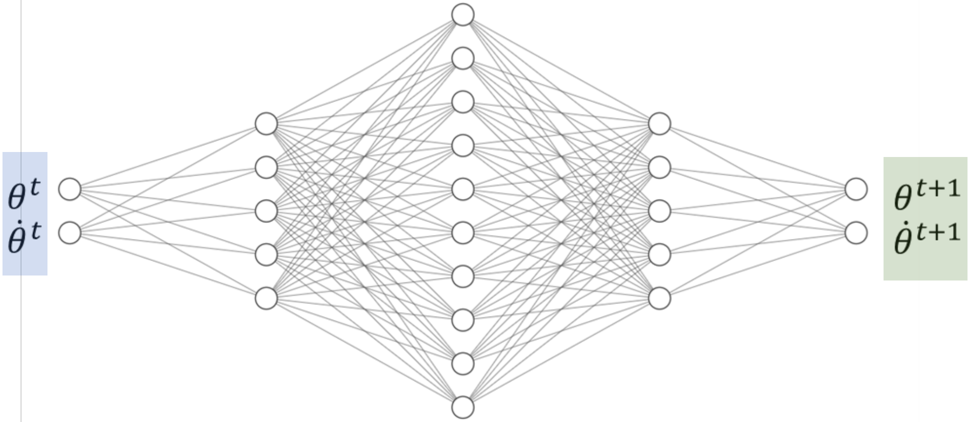
Equation of motion

$$\ddot{\theta}(t) + \frac{g}{L}\sin\theta(t) = 0$$

Energy of the system

$$E = \frac{1}{2}m\dot{\theta}^2 + mg(1 - \cos(\theta))$$

Results – Conventional Deep Learning



Input is angular position and velocity

Output is angular position and velocity

Network is acting like a time integrator

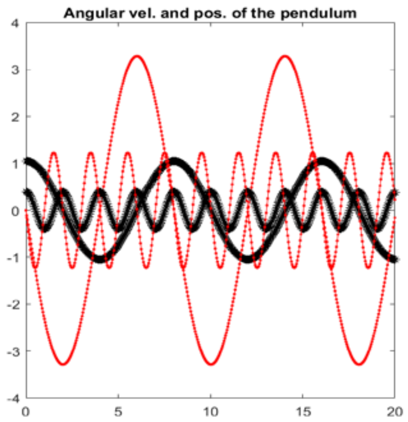
Network Architecture

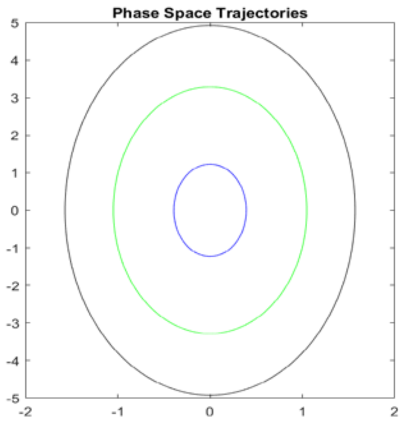
Input layer: 2 input neurons for angle and angular velocity

Internal layers: 3 fully connected hidden layers with ReLU activation of the form : 50,100,50

Output Layer: 2 output neurons for angle and angular velocity

Dataset consists of pendulum trajectories released from different heights.

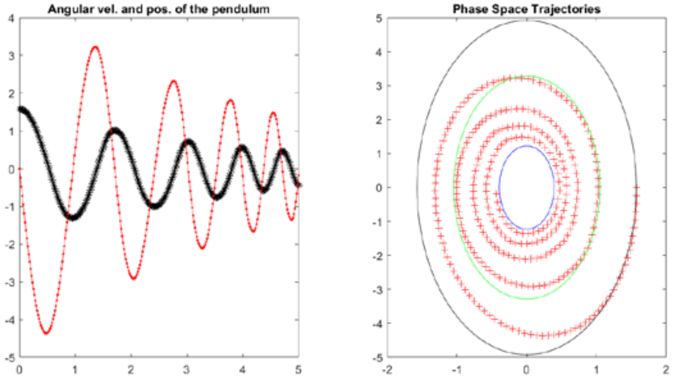




Training with an ADAM optimizer and the commonly-used mean squared error loss function:

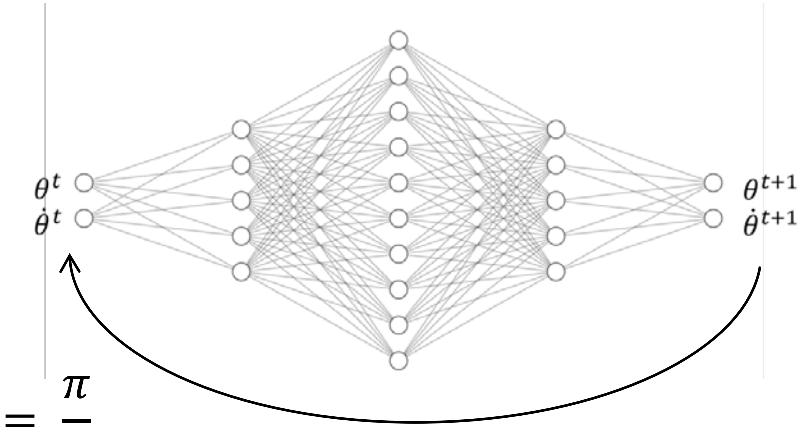
$$Loss_{mse} = |Y - T|^2$$

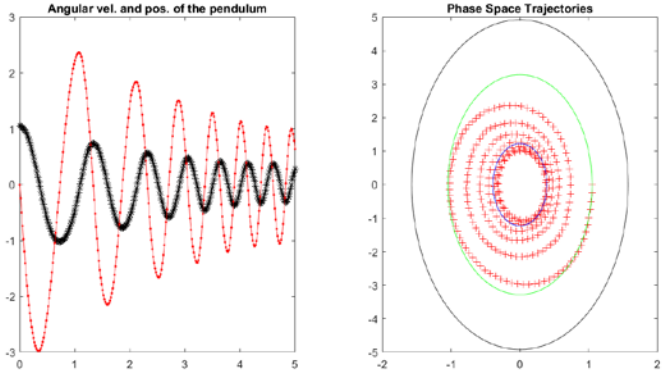
Results – Conventional Deep Learning



$$\theta^{t=0} = \frac{\pi}{2}$$

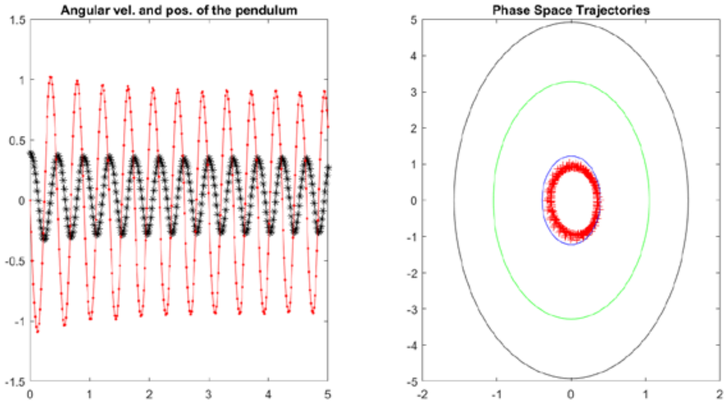
$$\dot{\theta}^{t=0} = 0$$





$$\theta^{t=0} = \frac{\pi}{3}$$

$$\dot{\theta}^{t=0} = 0$$



$$\theta^{t=0} = \frac{\pi}{6}$$

$$\dot{\theta}^{t=0} = 0$$

Predicted results decay from desired trajectory to a dominant (minimal) energy trajectory

Solution – Modified Loss Function

$$Loss_{mse} = |Y - T|^2$$

$$Loss_{PIML} = (1 - \lambda)Loss_{mse} + \lambda|E_Y - E_T|^2|$$

The traditional loss function seeks to minimize the predicted value (Y) from the target/true value (T):

$$Loss = f(Y - T)$$

However, we want to ensure that there is a notion that other factors are important, such as the conservation of energy. In a mechanical system like the pendulum the energy (E) can be expressed as a sum of the kinetic and potential terms:

$$E = \frac{1}{2}m\dot{\theta}^2 + mg(1 - \cos(\theta))$$

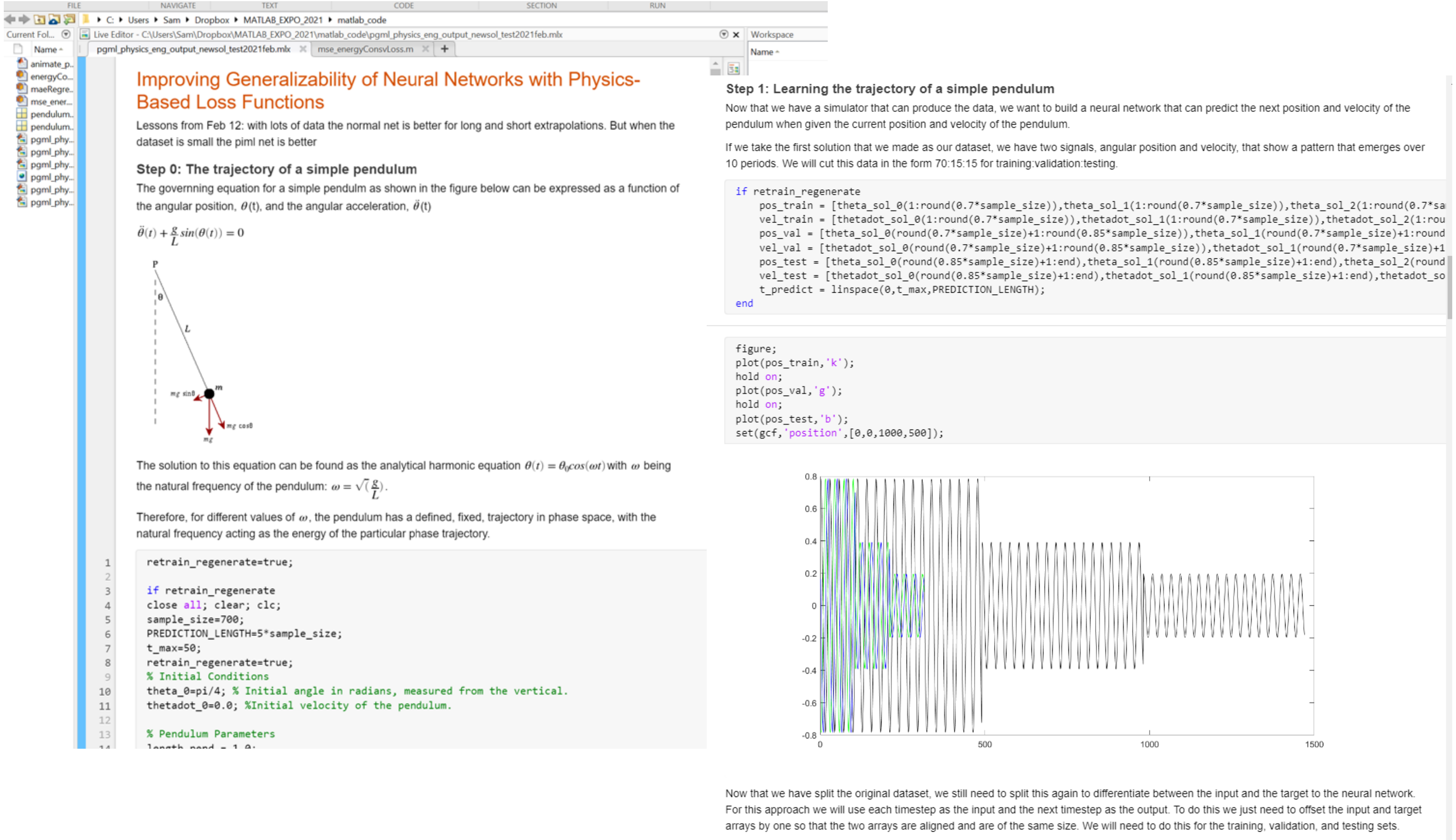
The new loss, therefore, is a function of the values Y,T and the energy, E:

$$Loss = f(Y - T, E_Y - E_T)$$

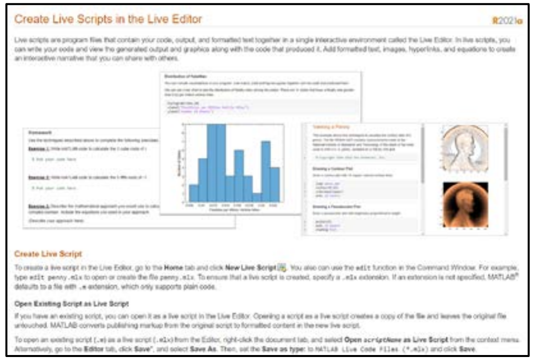
```
layers = [...
        sequenceInputLayer(2)
        fullyConnectedLayer(50);
        reluLayer;
        fullyConnectedLayer(100);
        reluLayer;
        fullyConnectedLayer(50);
        reluLayer;
        fullyConnectedLayer(2);
        mse_energyConsvLoss('dE');
    ];
```

```
classdef energyConsvLoss < nnet.layer.RegressionLayer

    properties
        % (Optional) Layer properties.

        % Layer properties go here.
    end

    methods
        function layer = energyConsvLoss(name)
            % Creates a physics-based loss function
            % that includes the conservation of Energy

            % Layer constructor function goes here.
            layer.Name = name;
            layer.Description = 'MAE + Energy Conservation loss';
        end

        function loss = forwardLoss(layer, Y, T)
R = size(Y,3);

            meanAbsoluteError = sum(abs(Y-T),3)/R;

            % Take mean over mini-batch.
            N = size(Y,4);
            MAEloss = sum(meanAbsoluteError)/N;

%             % Layer forward loss function goes here.
            energy_system_pred = -9.81.*cos(Y(1,:)) + (1/2).*(Y(2,:).^2);
            pred_energy = sum(energy_system_pred)/N;
            energy_system_true = -9.81.*cos(T(1,:)) + (1/2).*(T(2,:).^2);
            true_energy = sum(energy_system_true)/N;



            e_factor = 1.0e-5;%e-8;%2*N;
            L = 1;
            energy_loss = e_factor*(abs(true_energy - pred_energy));
            loss_mag = sqrt(energy_loss^2 + MAEloss^2);
            loss = L*MAEloss*MAEloss/loss_mag + (1-
L)*energy_loss*energy_loss/loss_mag;

        end

end
end
```

Tools Used



Live scripts



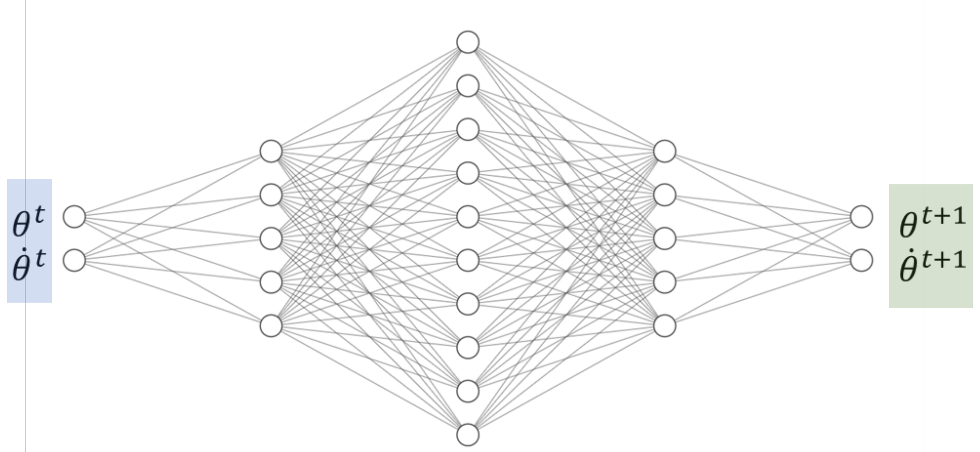
Deep Learning Toolbox

Results – PIML Deep Learning

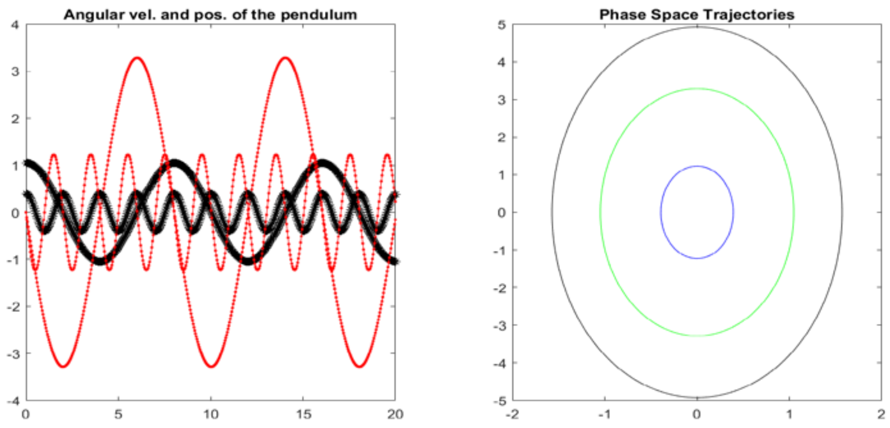
Input is position and velocity

Output is position and velocity

Network is acting like a time integrator



Dataset same as before.

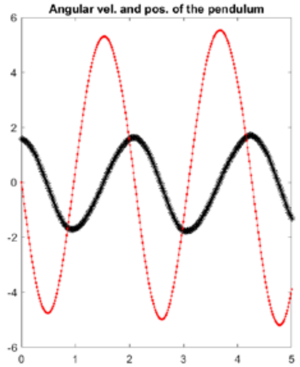


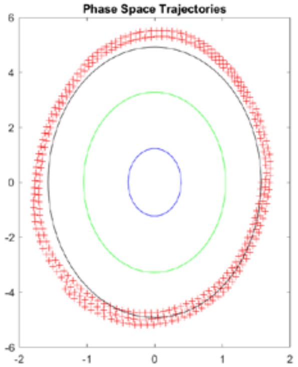
Training with the new PIML loss function:

$$\cancel{Loss_{mse} = |Y - T|^2}$$

$$Loss_{PIML} = (1-\lambda)Loss_{mse} + \lambda|E_Y - E_T|^2|$$

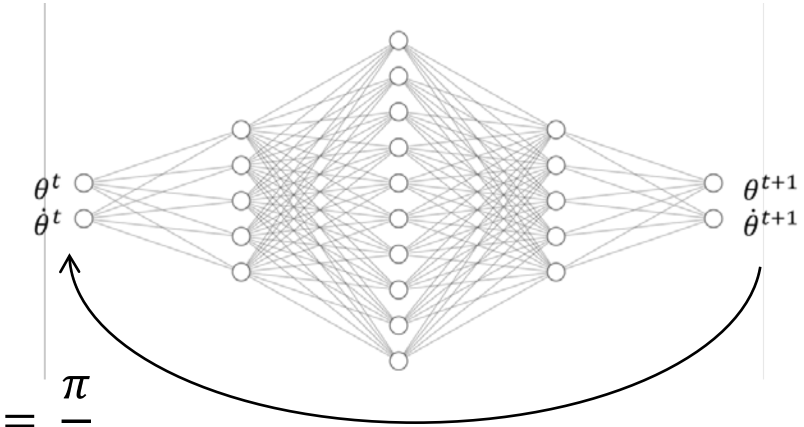
Results – PIML Deep Learning

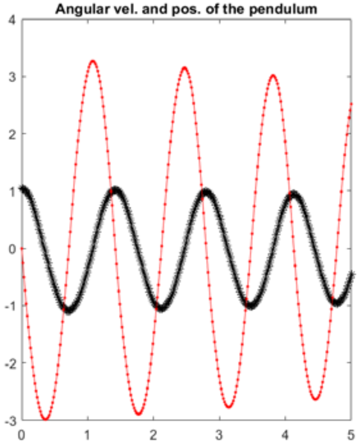


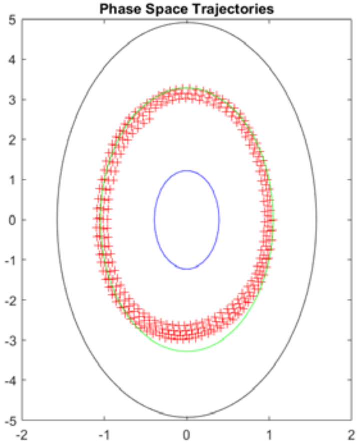


$$\theta^{t=0} = \frac{\pi}{2}$$

$$\dot{\theta}^{t=0} = 0$$

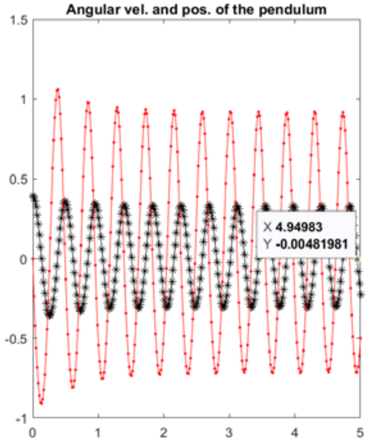


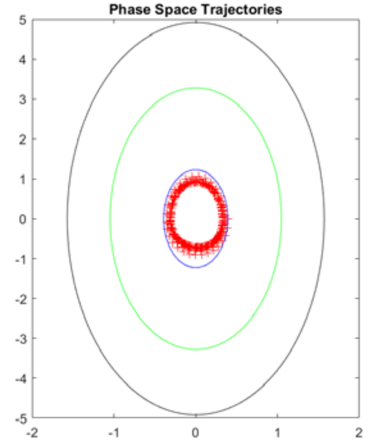




$$\theta^{t=0} = \frac{\pi}{3}$$

$$\dot{\theta}^{t=0} = 0$$





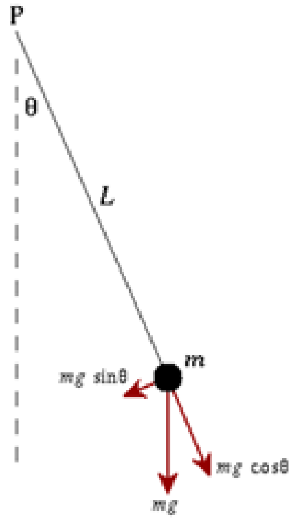
$$\theta^{t=0} = \frac{\pi}{6}$$

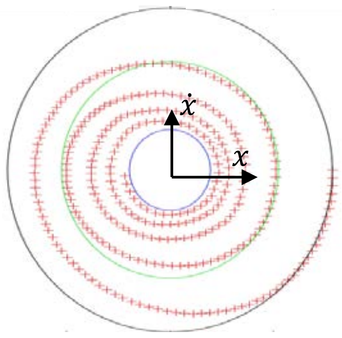
$$\dot{\theta}^{t=0} = 0$$

Predicted results no longer decay to the smallest value. Energy is maintained for the prediction cycle

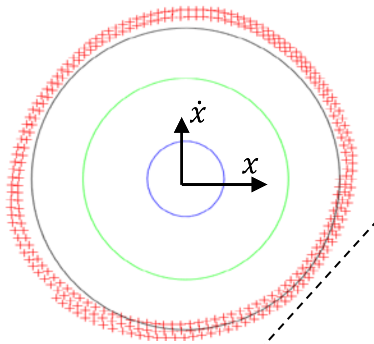
Takeaways - The Future of PIML

Physics-Informed Machine Learning is still very young but full of potential…



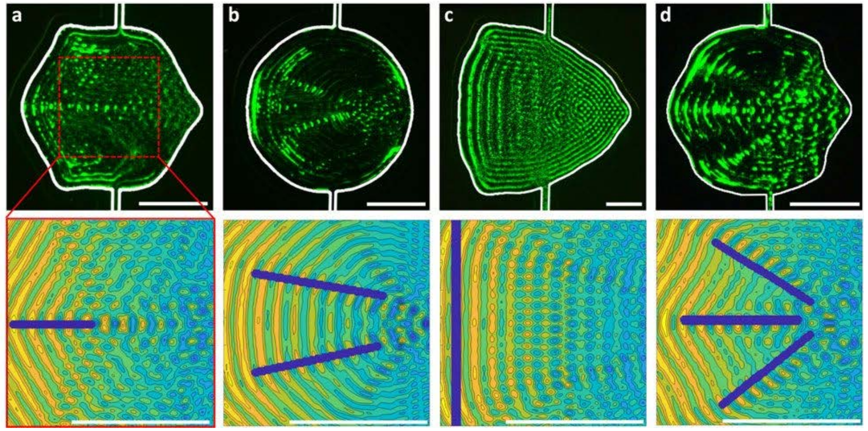


vs.



PIML-based loss functions for more robust predictions in hybrid (simulation/data) systems

MATLAB-powered PIML-based design tools in biomedical engineering



Read More

Stanford University

Thank you!

Dr. Sam Raymond | Stanford University | sjray@stanford.edu