Physics-Informed Machine Learning
Using the laws of nature to improve generalized deep learning models

Traditional ML vs. Physics-Informed ML

Dr. Sam Raymond | Stanford University
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

Neural network \( \tilde{\delta}_x = f_{\theta}(\tilde{\eta}, w_1) \), trained to minimize loss \( L \propto (\delta_x - \tilde{\delta}_x)^2 \).
Introduction – Physics Informed Machine Learning

Physics-Informed Neural Networks


Learning equations


Synthetic data for inverse-design

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

Dataset consists of pendulum trajectories released from different heights.

**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

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

\[ \text{Loss}_{\text{mse}} = |Y - T|^2 \]
Results – Conventional Deep Learning

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

\[ \theta^t = 0 = \frac{\pi}{2} \]
\[ \dot{\theta}^t = 0 = 0 \]

\[ \theta^t = 0 = \frac{\pi}{3} \]
\[ \dot{\theta}^t = 0 = 0 \]

\[ \theta^t = 0 = \frac{\pi}{6} \]
\[ \dot{\theta}^t = 0 = 0 \]
Solution – Modified Loss Function

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

$$\text{Loss} = f(Y - T)$$

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, $E$:

$$\text{Loss} = f(Y - T, E_Y - E_T)$$

$$\text{Loss}_{\text{mse}} = |Y - T|^2$$

$$\text{Loss}_{\text{PIML}} = (1 - \lambda)\text{Loss}_{\text{mse}} + \lambda |E_Y - E_T|^2$$
Tools Used

Improving Generalizability of Neural Networks with Physics-Based Loss Functions

Lessons from Flex: with lots of data the normal net is better for long and short extrapolations. But when the dataset is small the point net is better.

Step 0: The trajectory of a simple pendulum

The governing equation for a simple pendulum as shown in the figure below can be expressed as a function of the angular position, $\theta(t)$, and the angular acceleration, $\ddot{\theta}(t)$:

$$\ddot{\theta}(t) = \frac{g}{l} \sin(\theta(t))$$

The solution to this equation can be found as the analytical harmonic equation $\theta(t) = A \sin(\omega t)$ with $\omega = \sqrt{\frac{g}{l}}$ being the natural frequency of the pendulum.

Therefore, for different values of $\omega$, the pendulum has a defined, fixed trajectory in phase space, with the natural frequency acting as the energy of the particular phase trajectory.

Step 1: Learning the trajectory of a simple pendulum

Now that we have a simulator that can produce the data, we want to build a neural network that can predict the next position and velocity of the pendulum when given the current position and velocity of the pendulum.

If we take the first solution that we made on our dataset, we have two signs, angular position and velocity, that show a pattern that emerges over time. We will cut this data into the form $X_{1:n}$ for training validation testing.

```matlab
if ~retrain_regenerate
    [theta_sol,t1]=ode45(@(t,y)ode_func(y,t,phi0,omg0),t1span,thet0);
    theta_sol_test=[theta_sol(1:end-1,:),theta_sol(1:end,:)];
end
```

Live scripts

Deep Learning Toolbox

Now that we have split the original dataset, we still need to split this again to differentiate between the input and the target to the neural network.

For this approach we will use each timestep as the input and the next timestep as the output. To do this we just need to offer the input and target arrays by one so that the two arrays are aligned and are of the same size. We will need to do this for the training, validation, and testing sets.

```matlab
retain_regenerate=true;
```
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:

\[
Loss_{\text{PIML}} = (1 - \lambda) Loss_{\text{mse}} + \lambda |E_Y - E_T|^2
\]
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...

MATLAB-powered PIML-based design tools in biomedical engineering

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