Lagrangian Neural Networks (LNNs)

Lagrangian Neural Networks (LNNs) are designed to learn physical dynamics by preserving the structure of the Lagrangian formulation of classical mechanics. Below is the mathematical formulation and examples for a block sliding on a slope and a double pendulum.

1. Lagrangian Mechanics

The Lagrangian L is defined as:

$$L = T - V$$

where:

- T: Kinetic energy
- V: Potential energy

The equations of motion are derived using the Euler-Lagrange equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0$$

where:

- q_i: Generalized coordinates
- $\dot{q}_i$: Generalized velocities

In LNNs, the Lagrangian L is modeled as a neural network $L_\theta(q, \dot{q})$, parameterized by weights θ. The equations of motion are implicitly learned by minimizing the error between the predicted and observed trajectories.

2. Example: Block Sliding on a Slope

System Setup

- Let q be the position of the block along the slope.
- The slope has an angle α with the horizontal.
- The block's mass is m, and gravity g acts downward.

Kinetic Energy

$$T = \frac{1}{2}m\dot{q}^2$$

Potential Energy

$$V = -mgq\sin\alpha$$

Lagrangian

$$L = T - V = \frac{1}{2}m\dot{q}^2 + mgq\sin\alpha$$

Euler-Lagrange Equation

$$\frac{d}{dt}\left(m\dot{q}\right) + mg\sin\alpha = 0$$

This simplifies to:

$$\ddot{q} = -g\sin\alpha$$

LNN Formulation

- Inputs to the neural network: $q, \dot{q}$
- Outputs: $L_\theta(q, \dot{q})$
- Loss function: Mean squared error between predicted accelerations $\ddot{q}_{\text{pred}}$ and actual accelerations $\ddot{q}$.

3. Example: Double Pendulum

System Setup

- Two masses m_1, m_2 attached by rigid, massless rods of lengths l_1, l_2, and angles θ_1, θ_2 with the vertical.
- Generalized coordinates: $q = [\theta_1, \theta_2]$

Kinetic Energy

$$T = \frac{1}{2}m_1(l_1\dot{\theta}_1)^2 + \frac{1}{2}m_2\left[(l_1\dot{\theta}_1)^2 + (l_2\dot{\theta}_2)^2 + 2l_1l_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1 - \theta_2)\right]$$

Potential Energy

$$V = -m_1gl_1\cos\theta_1 - m_2g(l_1\cos\theta_1 + l_2\cos\theta_2)$$

Lagrangian

$$L = T - V$$

Euler-Lagrange Equations

For θ_1:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) - \frac{\partial L}{\partial \theta_1} = 0$$

For θ_2:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) - \frac{\partial L}{\partial \theta_2} = 0$$

LNN Formulation

- Inputs to the neural network: $q = [\theta_1, \theta_2], \dot{q} = [\dot{\theta}_1, \dot{\theta}_2]$
- Outputs: $L_\theta(q, \dot{q})$
- Loss function: Minimize the discrepancy between the predicted and true trajectories.

4. General Implementation

1. **Define the Lagrangian**:
   - $L_\theta(q, \dot{q})$ modeled by a neural network.
2. **Compute Gradients**:
   - Use automatic differentiation to compute $\frac{\partial L}{\partial q}$, $\frac{\partial L}{\partial \dot{q}}$, and their time derivatives.
3. **Euler-Lagrange Equation**:
   - Enforce the dynamics through the loss function.
4. **Train the Model**:
   - Input observed data (positions and velocities) and minimize the loss.

This framework can generalize to other systems while ensuring physical consistency.