Learning Physical Constraints with Neural Projections

Shuqi Yang, Xingzhe He, and Bo Zhu

Computer Science Department, Dartmouth College

https://y-sq.github.io/proj/neural_proj/
Background: Data-Driven Simulators

• Learning and predicting an unknown physical system

Observed data

Learn the model

An ML Model:

Input: state_i
Output: state_i+1

Predict the dynamics

State_0

Future States

Force
Background: Data-Driven Simulators

• How to describe a dynamic physical system?

System - 1

System - 2
Simulation and Learning

- Unified physics simulators in CG can inspire the design of learning algorithms to perceive physical systems

**Physics simulation:**
Systems with mathematical models
Predicting dynamics with unified models

**Physics learning:**
Systems with observation data
Predicting dynamics without known models

Use the priors from physical simulations to guide the design of network architectures

Simulation and Learning

• Mass-spring models
  • Compute the forces among each particles in the system;
  • Explicit time integration or implicit time integration.

• Interaction networks*

Simulation and Learning

• Smoothed particle hydrodynamics
  • A Lagrangian viewpoint to simulate fluids
  • Physical property approximated by weighted sum of the kernel

• Lagrangian fluid simulation*

Simulation and Learning

• PIC/FLIP
  • Eulerian-Lagrangian representation
  • Particle to grid; grid to particle

• AdvectiveNet*

Simulation and Learning

• Hamiltonian systems
  • Describe the system’s state using the position and momentum of the objects, whose evolution equation is given by the Hamilton's equations

• Hamiltonian networks*

Simulation and Learning

• Position-based dynamics
  • Model the system using the constraints $C(\cdot)$ that the position $x$ should satisfy
  • Use a projection algorithm to correct the predicted positions such that they satisfy all the constraints: $C(x) = 0$

• Constraints (Ours)

Constraints: length, angle, position, ...
Examples of Constraints

- Distance (length)
  - Distance(i, j) - constant = 0
- Angle (bending)
  - Angle(i, j, k) - constant = 0
- Shape (rigidity)
  - Shape - Initial_Shape = 0
- Non-penetration (collision)
  - Distance(i, other_objects) >= 0

\[ C(x) = 0 \]
Position-based Dynamics: Variational Perspective

\[
\min_x g(x) = \frac{1}{\Delta t^2} (x - \hat{x})^T M (x - \hat{x}) + \lambda^T C(x)
\]

- Prediction with forces
- Correction with constraints
- The correction step amounts to an energy minimization problem
Position-based Dynamics: Algorithm Overview

Algorithm 1 Position-based dynamics

1: for all vertices $i$ do
2: \hspace{1em} initialize $x_i = x_i^0$, $v_i = v_i^0$, $w_i = 1/m_i$
3: end for
4: loop
5: \hspace{1em} for all vertices $i$ do $v_i \leftarrow v_i + \Delta t w_i f_{\text{ext}}(x_i)$
6: \hspace{1em} for all vertices $i$ do $p_i \leftarrow x_i + \Delta t v_i$
7: \hspace{1em} for all vertices $i$ do genCollConstraints($x_i \rightarrow p_i$)
8: \hspace{2em} loop solverIteration times
9: \hspace{3em} projectConstraints($C_1, \ldots, C_{M+M_{\text{Coll}}}, p_1, \ldots, p_N$)
10: end loop
11: end for
12: $v_i \leftarrow (p_i - x_i)/\Delta t$
13: $x_i \leftarrow p_i$
14: end for
15: velocityUpdate($v_1, \ldots, v_N$)
16: end loop

Enforcing hard-coded constraints

Using Constraints to describe the physical systems

- Model the physical systems using its constraints: \( C(x) = 0 \)
  - \( \text{Distance}(i, j) \) - constant = 0
  - \( \text{Angle}(i, j, k) \) - constant = 0
  - \( \text{Shape} - \text{Initial}_{\text{Shape}} = 0 \)
  - \( \text{Distance}(i, \text{other} \_\text{objects}) \geq 0 \)

- Directly related to human’s perception
- A unified representation of physics
- Directly manipulate on the positions
- Inherently implicit scheme for stable prediction
Our Approach: Learning Constraints by Neural Projection

Algorithm 1 Position-based dynamics

1: for all vertices $i$ do
2:   initialize $x_i = x_i^0$, $v_i = v_i^0$, $w_i = 1/m_i$
3: end for
4: loop
5:   for all vertices $i$ do $v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i)$
6:   for all vertices $i$ do $p_i \leftarrow x_i + \Delta t v_i$
7:   for all vertices $i$ do genCollConstraints($x_i \rightarrow p_i$)
8:   loop solver $t$ times
9:     projectConstraints($C_1, \ldots, C_{M+M_{coll}} : P_1, \ldots, P_N$)
10: end loop
11: for all vertices $i$ do
12:   $v_i \leftarrow (p_i - x_i)/\Delta t$
13:   $x_i \leftarrow p_i$
14: end for
15: velocityUpdate($v_1, \ldots, v_N$)
16: end loop

Unknown constraints:
Replace this module with an NN

Our Approach: Learning Constraints by Neural Projection

- Using a lightweight neural network to represent the constraints $C(\cdot)$
- Iteratively solve the projection that moves the input positions $\hat{x}$ to a state where $C(x)=0$ holds.

**Algorithm 1: Projection Unit**

| Input: Constraint $C_{net}(\cdot)$, positions $\hat{x}$.
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $\tilde{x}^1 = \hat{x}$;</td>
</tr>
<tr>
<td>2 for $i = 1 \rightarrow N$ do</td>
</tr>
<tr>
<td>3 $\lambda = C_{net}(\tilde{x}^i)/</td>
</tr>
<tr>
<td>4 $\delta\tilde{x} = -\lambda \nabla C_{net}(\tilde{x}^i)$;</td>
</tr>
<tr>
<td>5 $\tilde{x}^{i+1} = \tilde{x}^i + \delta\tilde{x}$;</td>
</tr>
<tr>
<td>6 end</td>
</tr>
</tbody>
</table>

**Output:** Projected positions $x$

Gradients calculated by auto differentiation.
Our Approach: Learning Constraints by Neural Projection

- **Prediction:**
  - Advance the position of each particle with given external forces

- **Correction:**
  - A black-box NN module to enforce constraints
  - Multiple loops of projection iterations
  - Update the particle positions to the places where the black-box constraints are satisfied

- **Velocity update:**
  - Based on the positions in time $n$ and $n+1$

Loss is computed using the difference between the predicted and the groundtruth (groundtruth data generated using physical simulation)
Our Approach: Multi-Group Representation

- **Overlapping Groups:**

- **Sub-Networks:**

   ```plaintext
   Algorithm 2: Multi-Group Projection
   Input: NNs $C_{net_1}(\cdot), \ldots, C_{net_M}(\cdot)$,
   Group of positions $\hat{x}_1, \ldots, \hat{x}_M$.
   1. $\hat{x}^1 = \hat{x}$;
   2. for $i = 1 \rightarrow N$ do
   3.     for $j = 1 \rightarrow M$ do
   4.         $\bar{x}^{i+1}_j = \text{Project}(C_{net_j}, \hat{x}^i_j)$;
   5.     end
   6.     Synchronizing $\bar{x}^{i+1}$ among groups;
   7. end
   Output: Projected positions $x$
   ```
Animations of the predicted results

- A rigid body rotating with the external forces that sum to 0.
Animations of the predicted results

• A rope with stretching and bending.

[Graphs and diagrams showing the predicted results of a rope with stretching and bending.]
Animations of the predicted results

• The articulated body

[Learning physical constraints with neural projections]
Animations of the predicted results

- Rigid body collisions

[Graphics showing simulations and data plots]
Discussions

- The learned constraints:
  - The value has a physical meanings; Future work to better separate each constraint.

- The iterative projection:
  - A projection process is also used in other applications.
  - Fixed point problems.

- Network architectures for more types of systems.
Take-home message

• Intersection between physics simulation and physics learning

• Use the priors from physical simulations to guide the design of network architectures

• Specific network architectures target at embedding specific types of priors
  - Two additional examples
More physics learning

• Soft Multicopter Control using Neural Dynamics Identification


https://corlconf.github.io/corl2020/paper_396/
More physics learning

• Nonseparable Symplectic Neural Networks

* Shiyin Xiong, Yunjin Tong, Xingzhe He, Cheng Yang, Shuqi Yang, Bo Zhu. Nonseparable Symplectic Neural Networks. International Conference on Learning Representations (ICLR 2021)
https://shiyingxiong.github.io/proj/NSSNN/NSSNN
More work in our lab: bridging physics simulation and machine learning

- Simulation: turbulent flows, vortex dynamics, bubbles, surface-tension-dominant contact
- Learning: solid systems, fluid systems, soft-bodied multicopter control, point cloud processing

Paper references: https://www.cs.dartmouth.edu/~bozhu/
Thank you!

For more information:
- https://www.cs.dartmouth.edu/~bozhu/
- https://y-sq.github.io/
- https://www.youtube.com/playlist?list=PLPkJxkrZZDviP3XG9mJNBHBLBmlm