Symplectic Recurrent Neural Networks

A recipe for learning Hamiltonian dynamics from data

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky & Léon Bottou
Hamiltonian systems

A lot of physical systems are Hamiltonian:

\[ \dot{p} = -\frac{\partial H}{\partial q}(p, q), \quad \dot{q} = \frac{\partial H}{\partial p}(p, q), \]

- \( p \): momentum variables (of dimension \( d \))
- \( q \): position variables (also of dimension \( d \))
- \( H \): Hamiltonian function / total energy

Often, the Hamiltonian is separable:

\[ H(p, q) = K(p) + V(q) \]

- \( K \): kinetic energy
- \( V \): potential energy

In the separable case, the equations become:

\[ \dot{p} = -V'(q), \quad \dot{q} = K'(p) \]
Learning Hamiltonian systems from data

**Question:** If $H$ is unknown to us, can we learn it from observations of the system?

**Relevant to:**
1. Inverse problems
2. Intuitive physics
3. Unsupervised / self-supervised learning

**Data:** Trajectories in $p$ and $q$ (i.e. time-series data)

**Test:** Predict the unobserved evolutions of the system

Want to handle complex, noisy and stiff Hamiltonian systems

Our proposal - **Symplectic Recurrent Neural Networks (SRNN)**

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky & Léon Bottou
Step 1

**Parameterize the unknown Hamiltonian function by a neural network – “H-NET”**

We set \( H_\theta(p, q) = K_{\theta_1}(p) + V_{\theta_2}(q) \)

with each of \( K \) and \( V \) parametrized by a Multi-Layer Perceptron (MLP)

Hence the parametrized equation: \( \dot{p} = -V'_{\theta_2}(q), \quad \dot{q} = K'_{\theta_1}(p) \)

**Note:** \( V'_{\theta_2}(q), \quad K'_{\theta_1}(p) \) can be computed via auto-differentiation

Resembles a Hamiltonian Neural Network (Greydanus et al., 2019)

But we do not need the supervision on \( \dot{p} \) and \( \dot{q} \)

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky & Léon Bottou
Step 2

**Numerically integrate this NN-parametrized ODE to generate a trajectory**

We propose to use a **symplectic integrator**, such as the **leapfrog** integrator:

\[
\begin{align*}
  p_{n+1/2} &= p_n - \frac{1}{2} \Delta t \, V'(q_n) \\
  q_{n+1} &= q_n + \Delta t \, K'(p_{n+1/2}) \\
  p_{n+1} &= p_{n+1/2} - \frac{1}{2} \Delta t V'(q_{n+1})
\end{align*}
\]

Why symplectic integrators?

Desirable conservation and stability properties when applied to Hamiltonian systems.
Step 3

Compute a loss between the observed and the generated time-series

We use mean squared error (MSE), corresponding to maximum likelihood inference with Gaussian likelihood function

Step 4

Use SGD to optimize the NN’s parameters

The generation of the trajectory from the NN-parametrized Hamiltonian is differentiable

Can back-propagate through the numerical integrator

At test time:

Numerically integrate the NN-parametrized equation with the optimized parameters

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky & Léon Bottou
Numerical experiments

Spring-chain system: A chain of 20 masses connected by springs

Three-body system: 3 masses in 2D interacting via gravity

Heavy-billiard system: A billiard within a 2D box under gravity and perfect rebound

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky & Léon Bottou
Results

1. Built-in **inductive bias** of a Hamiltonian

2. **Symplectic integration** significantly improves accuracy

3. **Recurrent training** gives robustness to observation noise

Table 1: Testing errors of different models on the spring-chain system

<table>
<thead>
<tr>
<th>Model</th>
<th>Integrator (tr)</th>
<th>Integrator (te)</th>
<th>Error mean</th>
<th>Error std</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-step O-NET</td>
<td>Euler</td>
<td>Euler</td>
<td>6.93</td>
<td>1.22</td>
</tr>
<tr>
<td></td>
<td>Euler</td>
<td>Leapfrog</td>
<td>5.87</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td>Leapfrog</td>
<td>Leapfrog</td>
<td>7.28</td>
<td>1.48</td>
</tr>
<tr>
<td>H-NET</td>
<td>Euler</td>
<td>Euler</td>
<td>7.24</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>Euler</td>
<td>Leapfrog</td>
<td>3.32</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>Leapfrog</td>
<td>Leapfrog</td>
<td>3.36</td>
<td>0.67</td>
</tr>
<tr>
<td>recurrent O-NET</td>
<td>Euler</td>
<td>Euler</td>
<td>2.88</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>Euler</td>
<td>Leapfrog</td>
<td>4.12</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>Leapfrog</td>
<td>Leapfrog</td>
<td>3.34</td>
<td>0.86</td>
</tr>
<tr>
<td>H-NET</td>
<td>Euler</td>
<td>Euler</td>
<td>7.58</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>Euler</td>
<td>Leapfrog</td>
<td>5.26</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>Leapfrog</td>
<td>Leapfrog</td>
<td>2.37</td>
<td>0.87</td>
</tr>
<tr>
<td>Vanilla RNN</td>
<td>N/A</td>
<td>N/A</td>
<td>4.80</td>
<td>0.82</td>
</tr>
<tr>
<td>LSTM</td>
<td>N/A</td>
<td>N/A</td>
<td>5.95</td>
<td>1.05</td>
</tr>
</tbody>
</table>
Bonus features of SRNN

1. Further improved by initial state optimization (ISO) for noisy data

Typically, ISO is known to be difficult as an optimization problem.

But we show theoretically that performing ISO on SRNN is nearly convex thanks to symplecticness.

![Figure 1: True testing trajectory versus predictions made by different models for the spring-chain system](image)
Bonus features of SRNN

2. A trained SRNN could even outperform simulating the exact equations with the same time-step size (!)

We think the models can learn to compensate for the numerical discretization error

*Figure 2: True testing trajectory versus predictions made by SRNN (on the left) and simulating the exact equation with the same time-step size (on the right)*
3. We augment SRNN to learn perfect rebound, an example of *stiff* systems.

Adding the perfect-rebound module:

\[
[p_{t+\alpha\Delta t}^{\text{pre}}, q_{t+\alpha\Delta t}^{\text{pre}}] \xleftarrow{\text{leapfrog}} \frac{1}{\alpha\Delta t} [p_t, q_t]
\]

\[
p_{t+\alpha\Delta t}^{\text{post}} = p_{t+\alpha\Delta t}^{\text{pre}} - 2(p_{t+\alpha\Delta t}^{\text{pre}} \cdot n)n
\]

\[
[p_{t+\Delta t}, q_{t+\Delta t}] \xleftarrow{\text{leapfrog}} \frac{1}{(1-\alpha)\Delta t} [p_{t+\alpha\Delta t}^{\text{post}}, q_{t+\alpha\Delta t}^{\text{post}}]
\]

*Figure 3: True testing trajectory versus predictions made by SRNN for the heavy-billiard system*
Conclusions

SRNN can learn the dynamics of complex, noisy and stiff Hamiltonian systems from trajectory data.

Extra finding: the possibility of improving numerical integrators with learning

Related works


Thanks!