HAMILTONIAN NEURAL NETWORK IN MATLAB

George Papazafeiropoulos

1. Overview

This MATLAB script demonstrates a simple implementation of a Hamiltonian Neural Network (HNN) trained using the backpropagation algorithm. Hamiltonian Neural Networks enable the use of Neural Networks under the law of conservation of energy or other invariants. In Hamiltonian mechanics, the motion of an object is described in terms of its position q and momentum p. For more than one degrees of freedom, these two quantities become vector-like as $\mathbf{q} = (q_1, q_2, \dots, q_n)$ and $\mathbf{p} = (p_1, p_2, \dots, p_n)$. Taken together, they offer a complete description of the system. The Hamiltonian is a scalar function $H(\mathbf{q}, \mathbf{p})$ for which:

$$\frac{d\boldsymbol{q}}{dt} = \frac{\partial H}{\partial \boldsymbol{p}}, \frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \mathbf{q}}$$

The time evolution of the system happens towards the direction $\left(\frac{\partial H}{\partial \mathbf{p}}, -\frac{\partial H}{\partial \mathbf{q}}\right)$ which is called the "symplectic gradient" and leads to constant Hamiltonian function (i.e. total energy), contrary to the gradient of $H(\mathbf{q}, \mathbf{p})$, [i.e. $\left(\frac{\partial H}{\partial \mathbf{p}}, \frac{\partial H}{\partial \mathbf{q}}\right)$], which would change the Hamiltonian function (total energy) as quickly as possible. The neural network in this script does not fit directly against the target quantities $\left(\frac{d\mathbf{q}}{dt}, \frac{d\mathbf{p}}{dt}\right)$ as this is the way of an ordinary ANN. Instead, the neural network in this code learns and respects exact conservation laws in an unsupervised manner, by predicting $H(\mathbf{q}, \mathbf{p})$ instead of $\left(\frac{d\mathbf{q}}{dt}, \frac{d\mathbf{p}}{dt}\right)$. After the forward operation of the neural network for predicting $H(\mathbf{q}, \mathbf{p})$, the following loss is evaluated:

$$L_{HNN} = \left|\frac{\partial H}{\partial \mathbf{p}} - \frac{d\mathbf{q}}{dt}\right| + \left|\frac{\partial H}{\partial \mathbf{q}} + \frac{d\mathbf{p}}{dt}\right|$$

In this code, the task of modeling the dynamics of the frictionless mass-spring system ("Task 1: Ideal Mass-Spring" in [1]) is solved. See reference [1] for more details.
The main script serves as a reference implementation to help users understand how HNNs can be implemented and modify it for more advanced applications.
No MATLAB toolbox is required to run this code, which is particularly useful for educational HNN prototypes or if you want fine-grained control over weight updates, learning rate, activation functions, etc. No dependencies on MATLAB's Deep Learning Toolbox or any other toolboxes exist, therefore it can run on *any* MATLAB version. It is transparent and easy to extend (ideal for HNN research and learning). The local functions used in this script are listed alphabetically at the end of the main script.
NOTE: The results produced by the code may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. For consistency purposes and without loss of generality, the random number generator is appropriately initialized at the beginning of the code. If this option is removed, consider running the example a few times and compare the average outcome.

2. Features

The following features are used in the Matlab script:
- The HNN contains two hidden layers.
- It uses tanh activation function in the two hidden layers of the ANN involved.
- Gradient-based weight updates for training of the ANN involved is used.
- The training data are generated using an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair, through an Ordinary Differential Equation (ODE) solver.
- The following adjustable parameters for the generation of the training data are included:
  - Time span (start and end times of simulation for the generation of q, p data).
  - Refinement of time steps involved in the simulation
  - Degree of noise added to the q, p data
  - Number of trajectories of the undamped oscillator which are included in the training data
  - Tolerance of the ODE solver used for the simulation.
- The following adjustable neural network parameters are included:
  - Hidden layers' size of HNN.
  - Maximum number of training epochs.
  - Batch size (i.e. the data size per epoch used for training of the GAN)
  - Learning rate.

3. How to use

3.1. Execution
Open demo.m in Matlab and run it (press F5)

3.2. Customization
Users can modify the following parameters related to the generation of the training data:

```
t_span = [0 3];             % start and end times of simulation
timescale = 10;             % refinement of time steps
noise_std = 0.01;           % noise in q, p data
numTrajectories = 50;       % number of independent trajectories
rtol = 1e-10;               % solver tolerance
```

Users can modify the following parameters related to the Hamiltonian neural network:

```
n_hidden = [50,45];    % Size of hidden layers (two hidden layers)
n_epochs = 10000;      % Number of epochs
n_batch = 750;         % Batch size
learning_rate = 2e-2;  % Learning rate
```

4. Output

4.1. Plot of loss history
It plots the loss value as it evolves during training of the HNN

4.2. Plot of the Hamiltonian function value
It plots the value of the Hamiltonian function for various testing data which are generated after the training of the HNN is completed

5. Future improvements

The code can be improved in various ways including, but not limited to:
- Model Architecture. Experiment with alternate model architectures for the HNN, such as more or fewer nodes, layers, and alternate activation functions such as leaky ReLU.

• Data Scaling. Experiment with types of scaling of training data.
• Alternative invariant quantities or differential equations for dynamics of the model under consideration.

6. Licensing

Keywords: Hamiltonian Network, Neural Network, Backpropagation, Feedforward, Total energy, Conservation, Invariant, Dynamics, Perceptron, Gradient, Training.

REFERENCES

[1] Sam Greydanus, Misko Dzamba, Jason Yosinski, Hamiltonian Neural Network, arXiv:1906.01563v1 [cs.NE] 4 Jun 2019. 1906.01563v1.pdf (arxiv.org).

ABOUT THE AUTHOR

Google Scholar profile	https://scholar.google.com/citations?hl=en&user=JXRPD7UAAAAJ
RG profile	https://www.researchgate.net/profile/George_Papazafeiropoulos2
LinkedIn profile	https://www.linkedin.com/in/george-papazafeiropoulos-8b6b6025/
MathWorks profile	https://www.mathworks.com/matlabcentral/profile/authors/2438703-george-papazafeiropoulos
Facebook profile	https://www.facebook.com/george.papazafeiropoulos.5
e-mail	gpapazafeiropoulos@yahoo.gr