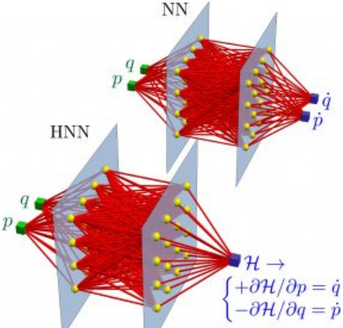
# Hamiltonian Neural Networks

Anil Radhakrishnan

University of Illinois

ML Journal Club, May 7 2020



# Motivation

- Invariant quantities have served as powerful tools for understanding the behavior of a system.
- While commonly used by physicists these rules can be difficult to infer directly from data.
- Using principles from Hamiltonian Mechanics can allow a neural network to better infer these rules .

# Brief Review of Hamiltonian Mechanics

Defining a scalar function, $\mathcal{H}(\mathbf{q}, \mathbf{p})$ such that:

$$\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \tag{1}$$

Hence moving along $\mathbf{S}_{\mathcal{H}} = \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\right)$ gives the time evolution:

$$(\mathbf{q}_1, \mathbf{p}_1) = (\mathbf{q}_0, \mathbf{p}_0) + \int_{t_0}^{t_1} \mathbf{S}(\mathbf{q}, \mathbf{p}) \; dt \tag{2}$$

# Implementing in a neural networks

Let the hamiltonian be parametrized with neural network parameters $\theta$.

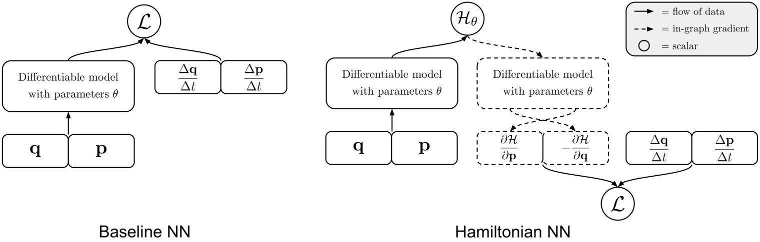
$$\frac{d\mathbf{q}}{dt} - \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} = 0, \quad \frac{d\mathbf{p}}{dt} + \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} = 0 \tag{3}$$

We can use this to create the following minimization objective:

$$\operatorname*{argmin}_\theta \left\| \frac{d\mathbf{q}}{dt} - \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} \right\|^2 + \left\| \frac{d\mathbf{p}}{dt} + \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} \right\|^2 \tag{4}$$

# Implementing in a neural networks

- This expression is similar to $\mathcal{L}_2$ loss from supervised learning.
- Instead of the standard $\left\|y - f_\theta(x)\right\|^2$ form, it takes $\left\|y - \frac{\partial f_\theta(x)}{\partial x}\right\|^2$



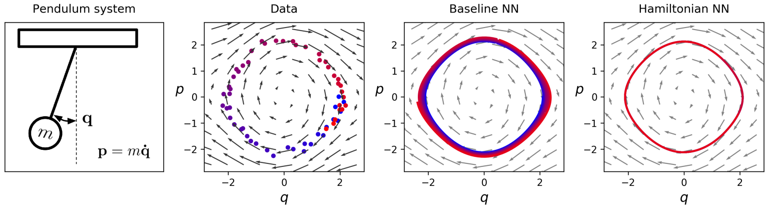

In essence, we optimize the gradient of a neural network

# What this looks like in code

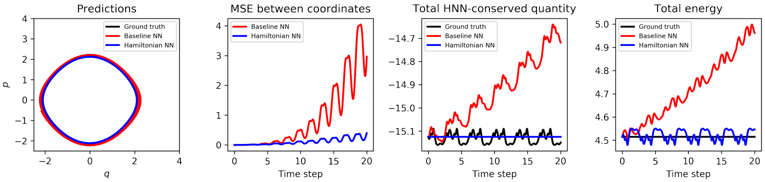
```
nn_model = MLP(...) # multilayer perceptron
model = HNN(..., differentiable_model = nn_model)
''' Time derivative calculated from helmholtz decomposition
of the gradient of the forward'''
dxdt_hat = model.time_derivative(x)
loss = L2_loss(dxdt, dxdt_hat)
```

we use $\mathcal{L}_2$ on the derivative to compute our loss

# Results




Simulating a pendulum



# Results

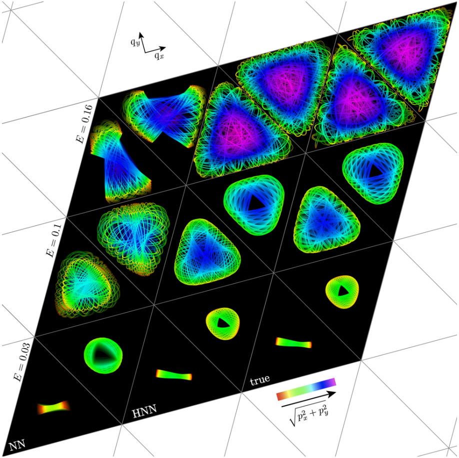



**Figure:** Comparing HNN with NN for chaotic orbits

# Other ideas and implementations

- Lagrangian neural networks
- Training on latent vectors
- Hamiltonian based inference network
- Using HNNs for normalizing flows

# Concluding remarks

- Using Hamiltonian mechanics allows us to use the symmetries of a system
- The basic principle lies in optimizing the gradient of the Neural Network
- HNN perform much better that baseline NN in systems with conservation laws
- Perfect reversibility: HNN perform well for chaotic systems

Questions?