Lagrangian Neural Networks

Machine Learning Principles and Applications for Physics

Mathis Batard-Rinaudo, Noé Daniel, Richard Pateau

Project defense

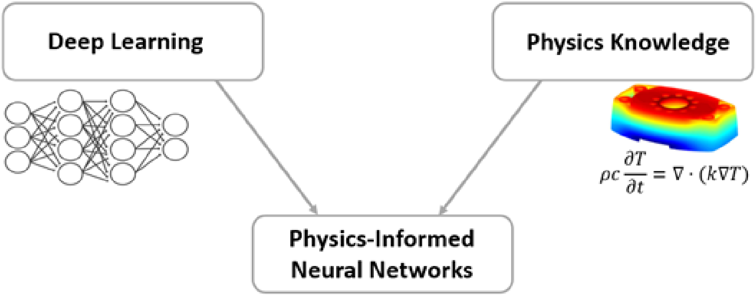
Département de Physique
École normale supérieure

21 January 2025

Why ? Objectives.

What are we trying to do ?

- Enhance basic NN for physical problems **with prior physics knowledge**.
- Keep the physics in the predictions (conserved quantity, time fidelity...)

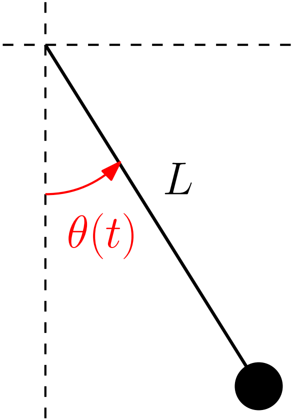


A first example : the simple pendulum

- Pretty simple ODE, easy for small angles

$$\ddot{\theta} + \omega_0^2 \sin\theta = 0 \qquad \omega_0 = \sqrt{\frac{g}{L}} \tag{1}$$

- Conservative system : $\partial_t E = 0$
- Response of the system is mainly cos and sin
  $\longrightarrow$ easy to fit for a baseline NN.



Create a trajectory with a NN ?

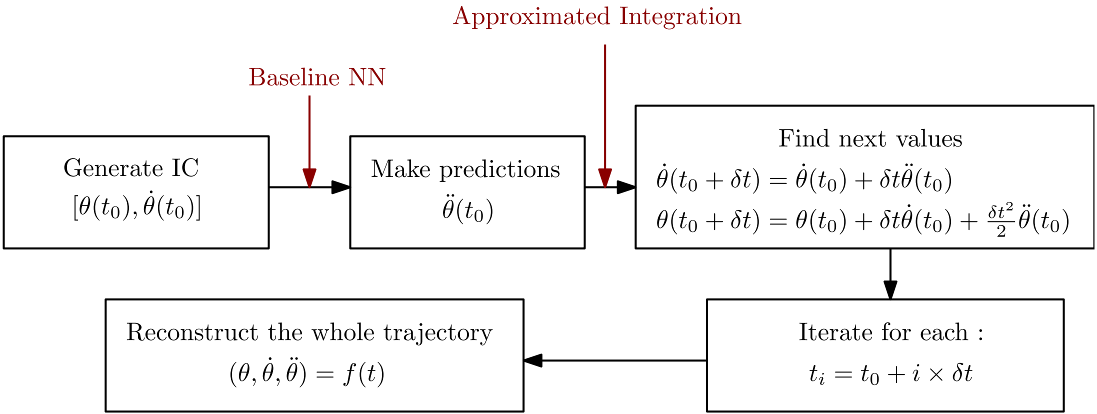


Figure: Steps to recreate a trajectory starting from initial conditions

A first solution : a baseline NN

Objective

Give $(\theta(t_i), \dot{\theta}(t_i))$ and predict $\ddot{\theta}(t_i)$.

Training process

- Choose $y_0 = (\theta_0, \dot{\theta}_0)$ and generate associated pendulum trajectory.
- Choose a time interval δt and generate $(\theta(t = \delta t), \dot{\theta}(t = \delta t))$ and $(\ddot{\theta}(t = \delta t))$.
  - Integrate (1) between 0 and δt to generate $\theta(\delta t), \dot{\theta}(\delta t)$ and use (1) to obtain $\ddot{\theta}(\delta t)$.
- Repeat this procedure N times for random y_0's.
- Generate training data
  - x_{train} : all the couples $(\theta, \dot{\theta})$ generated.
  - y_{train} : all the values $(\ddot{\theta})$ generated.

$$x_{\text{train}} = \left\{(\theta^1, \dot{\theta}^1), \dots, (\theta^N, \dot{\theta}^N)\right\}$$

$$y_{\text{train}} = \left\{\ddot{\theta}^1, \dots, \ddot{\theta}^N\right\}$$

The NN structure

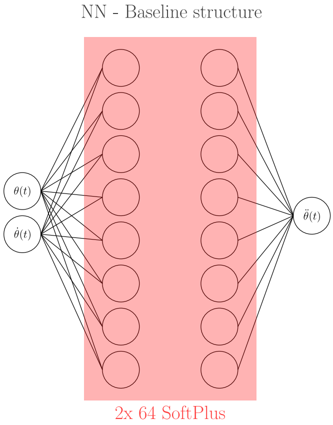


Figure: Structure of the Baseline used for Simple Pendulum prediction

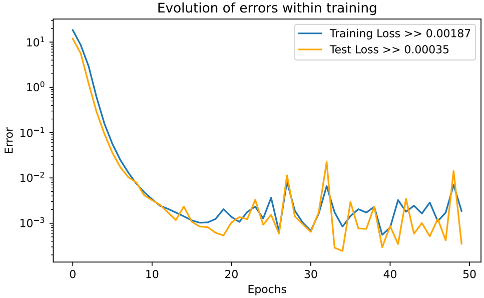
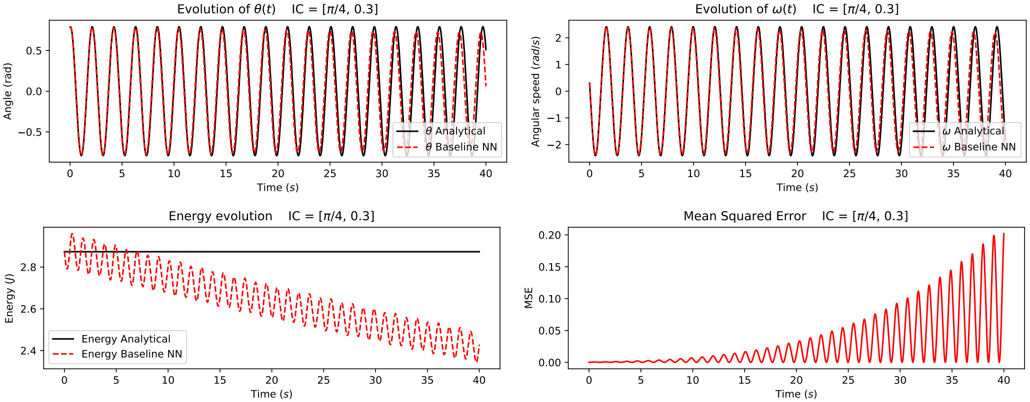


Figure: Training of the Baseline NN

Results

Initial conditions : $(\theta(t=0), \omega(t=0)) = (\pi/4, 0.3)$

Trajectory reconstruction time $\sim$ 2 min for $\mathrm{d}t = 0.002$ s and $N_{\text{steps}} = 2000$.



Trajectories are well reproduced but **energy is not conserved !**

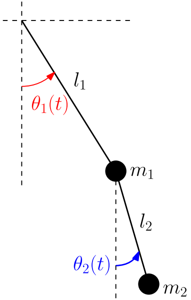
A more chaotic system ?

Question

We've seen that the NN works well on a simple harmonic system. **Would it work as well for more chaotic systems ?**

Try on a new system : the double pendulum.

$$T = \frac{1}{2} m_1 (l_1 \omega_1)^2 + \frac{1}{2} m_2 \left((l_1 \omega_1)^2 + (l_2 \omega_2)^2 + 2 l_1 l_2 \omega_1 \omega_2 \cos(\theta_1 - \theta_2) \right)$$
$$V = -m_1 g l_1 \cos\theta_1 - m2 g l_2 (l_1 \cos(\theta_1) + l_2 \cos\theta_2)$$
$$\mathcal{L} = T - V$$



$$\frac{\partial L}{\partial q} - \frac{\mathrm{d}}{\mathrm{dt}} \left(\frac{\partial L}{\partial \dot{q}} \right) = 0 \quad \text{(Euler-Lagrange)}$$

Analytical computation

$$\alpha_1(\theta_1,\theta_2) := \frac{l_2}{l_1}\left(\frac{m_2}{m_1+m_2}\right)\cos(\theta_1-\theta_2) \qquad \alpha_2(\theta_1,\theta_2) := \frac{l_1}{l_2}\cos(\theta_1-\theta_2)$$

$$f_1(\theta_1,\theta_2,\dot\theta_1,\dot\theta_2) := -\frac{l_2}{l_1}\left(\frac{m_2}{m_1+m_2}\right)\dot\theta_2^2\sin(\theta_1-\theta_2) - \frac{g}{l_1}\sin\theta_1$$

$$f_2(\theta_1,\theta_2,\dot\theta_1,\dot\theta_2) := \frac{l_1}{l_2}\dot\theta_1^2\sin(\theta_1-\theta_2) - \frac{g}{l_2}\sin\theta_2$$

$$g_1 := \frac{f_1-\alpha_1 f_2}{1-\alpha_1\alpha_2} \qquad g_2 := \frac{-\alpha_2 f_1 + f_2}{1-\alpha_1\alpha_2}$$

Using Euler-Lagrange :

$$\frac{d}{dt}\begin{pmatrix}\theta_1\\\theta_2\\\omega_1\\\omega_2\end{pmatrix} = \begin{pmatrix}\omega_1\\\omega_2\\g_1(\theta_1,\theta_2,\omega_1,\omega_2)\\g_2(\theta_1,\theta_2,\omega_1,\omega_2)\end{pmatrix} \longleftarrow \text{Necessary to create training data}$$

A new paradigm : training a Lagrangian

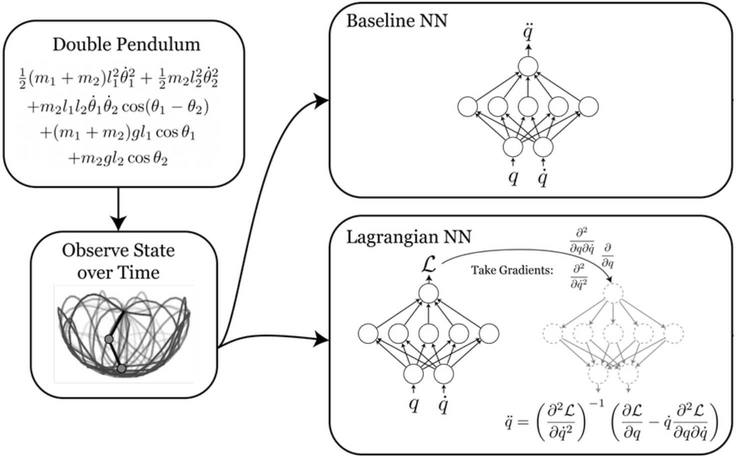


Figure: Training method of a LNN vs Baseline NN

Lagrangian Neural Network

How to learn Lagrangians in Machine Learning ?

1. Obtain data from a physical system
2. Find the Lagrangian parametrised by a NN
3. Apply the Euler-Lagrange constraint
4. Backpropagate through the constraint to train a parametric model that approximates the true Lagrangian to optimize the weight of the parametric model

How to apply the Euler Lagrange Constraint ?

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

$$\frac{d}{dt}\nabla_{\dot{\mathbf{q}}} L - \nabla_{\mathbf{q}} L = 0$$

$$\frac{d}{dt}\nabla_{\dot{\mathbf{q}}} L = \left(\frac{d}{dt}\nabla_{\dot{\mathbf{q}}} L\right) = \left(\nabla_{\dot{\mathbf{q}}}\nabla_{\dot{\mathbf{q}}} L\right)\dot{\mathbf{q}} + \left(\nabla_{\mathbf{q}}\nabla_{\dot{\mathbf{q}}} L\right)\ddot{\mathbf{q}}$$

$$\left(\nabla_{\dot{\mathbf{q}}}\nabla_{\dot{\mathbf{q}}} L\right)\ddot{\mathbf{q}} + \left(\nabla_{\mathbf{q}}\nabla_{\dot{\mathbf{q}}} L\right)\dot{\mathbf{q}} = \nabla_{\mathbf{q}} L$$

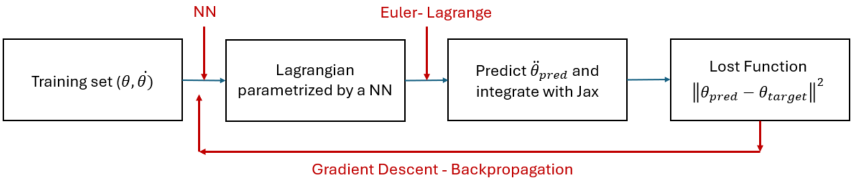
$$\ddot{\mathbf{q}} = \left(\nabla_{\dot{\mathbf{q}}}\nabla_{\dot{\mathbf{q}}}^{T} L\right)^{-1}\left[\nabla_{\mathbf{q}} L - \left(\nabla_{\mathbf{q}}\nabla_{\dot{\mathbf{q}}}^{T} L\right)\dot{\mathbf{q}}\right] \tag{2}$$

Equation (2) easy to implement in `Jax`.

What is Particular in the LNN ?- The Lost function

Baseline : trained with the classical MSE : $\|\ddot{\theta}_{\text{pred}} - \ddot{\theta}_{\text{target}}\|^2$

LNN : We are looking for the Lagrangian, but we don't have any target value.



2x 128 SoftPlus for LNN

What's the training data ?

Sample from a unique **chaotic trajectory** !

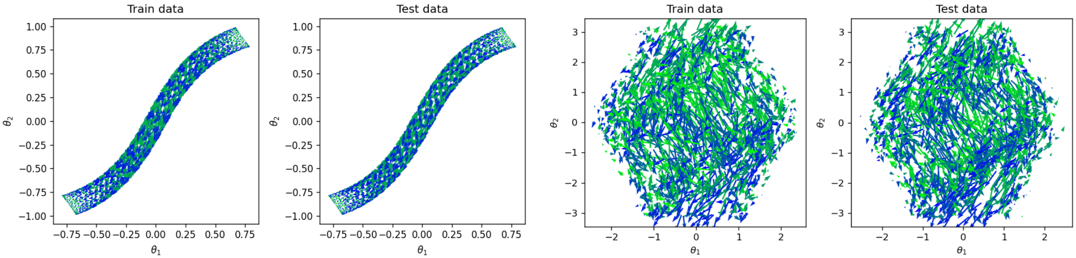


Figure: *Left.* Smooth training trajectory. *Right.* Chaotic training trajectory.

Animation - Double pendulum.

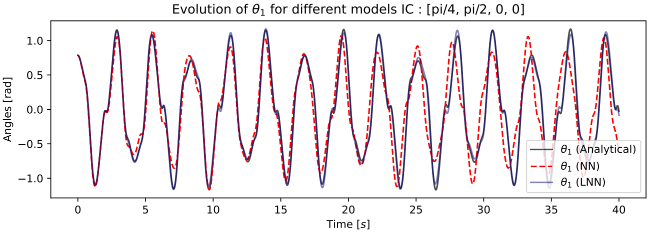
Comparaison Baseline vs LNN

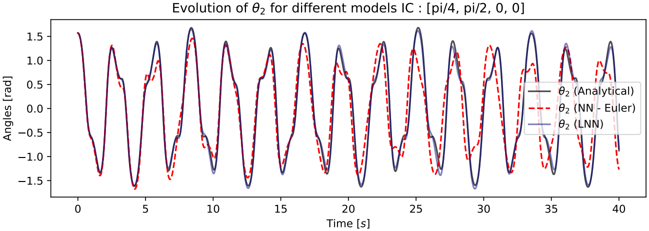
<u>Initial conditions</u> :

$$\theta_1(t=0) = \pi/4$$
$$\theta_2(t=0) = \pi/2$$
$$\omega_1(t=0) = 0$$
$$\omega_2(t=0) = 0$$

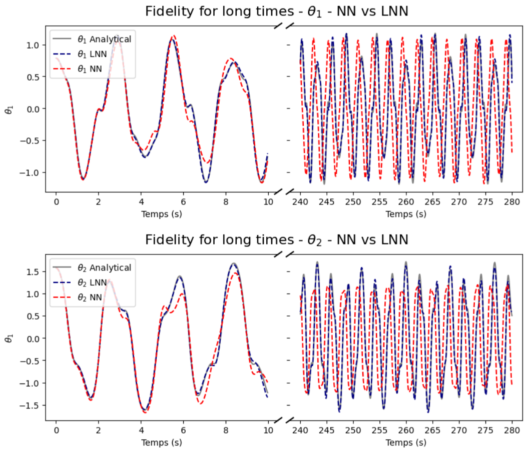
- Reconstruction time for Baseline $\sim$ 1 min 30s.
- Reconstruction time for LNN $\sim$ 3s

Both have a good accuracy but LNN has more fidelity

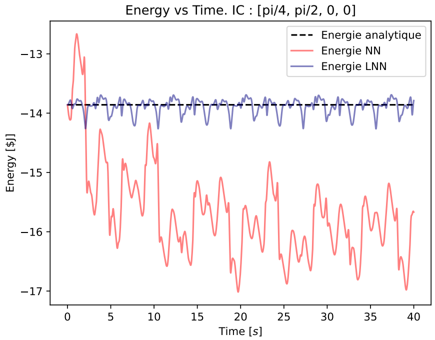




Checking accuracy of the NNs for smooth trajs



(a) Trajectories over long time durations



(b) Evolution of the system's energy wrt t

Baseline does not conserve E but LNN does !

Quantification of the observed dephasing

Mean Phase Shift algorithm

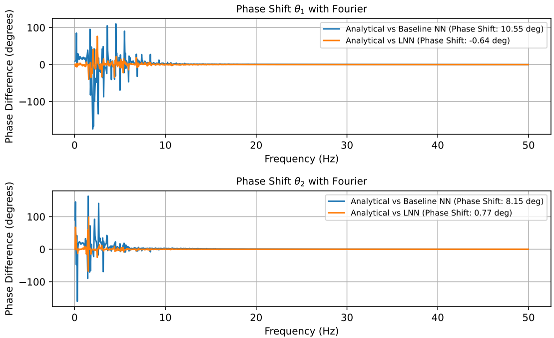
Analytical signal
$\theta_1(t) := e(t)$

Test signal
$\theta_{1,model} := s(t)$

$$E[k] = \sum_n^{N-1} e[n]e^{-j\frac{2\pi kn}{N}}$$
$$S[k] = \sum_n^{N-1} s[n]e^{-j\frac{2\pi kn}{N}}$$
$$k = 0, 1, \ldots, N-1$$

$$\phi_e(E[k]) = \arg(E[k])$$
$$\phi_s(S[k]) = \arg(S[k])$$
$$\Delta\phi(k) = \phi_e(E[k]) - \phi_s(S[k])$$

Mean Phase Shift
Average on k
$$\overline{\Delta\phi(k)}$$



$$\Delta\phi_{\theta_1,\text{Baseline}} = 10.55^\circ$$
$$\Delta\phi_{\theta_1,\text{LNN}} = -0.64^\circ$$
$$\Delta\phi_{\theta_2,\text{Baseline}} = 8.15^\circ$$
$$\Delta\phi_{\theta_2,\text{LNN}} = 0.77^\circ$$

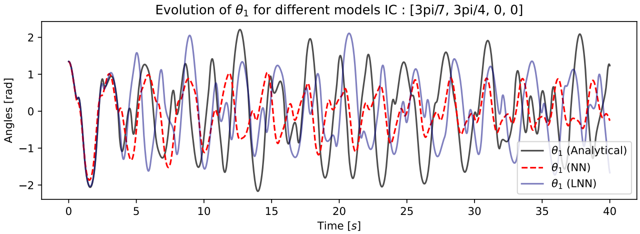
LNN beats Baseline NN

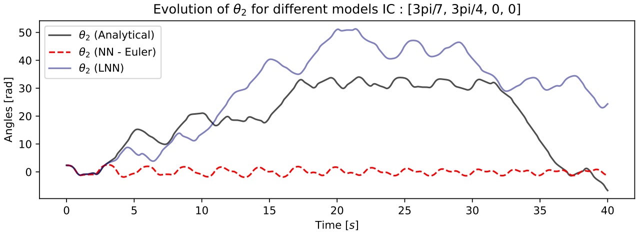
A more chaotic IC

For smooth IC, both behave well !

Initial conditions :

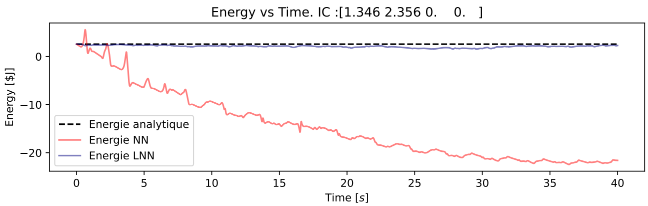
$$\theta_1(t=0) = 3\pi/7$$
$$\theta_2(t=0) = 3\pi/4$$
$$\omega_1(t=0) = 0$$
$$\omega_2(t=0) = 0$$

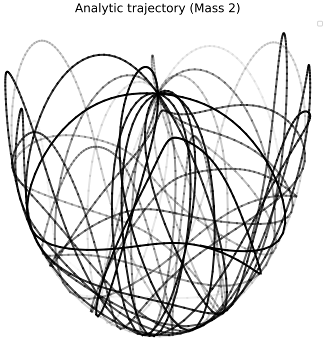


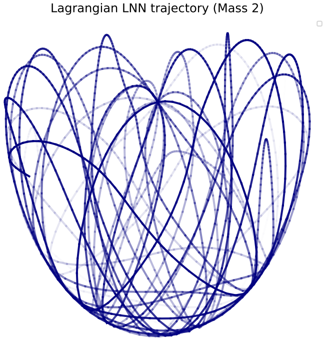


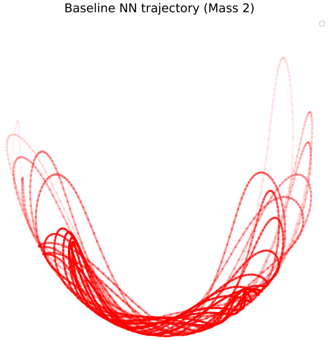
Both have a bad accuracy but LNN has more fidelity.

LNN is always more "physical"









Quick Check with litterature

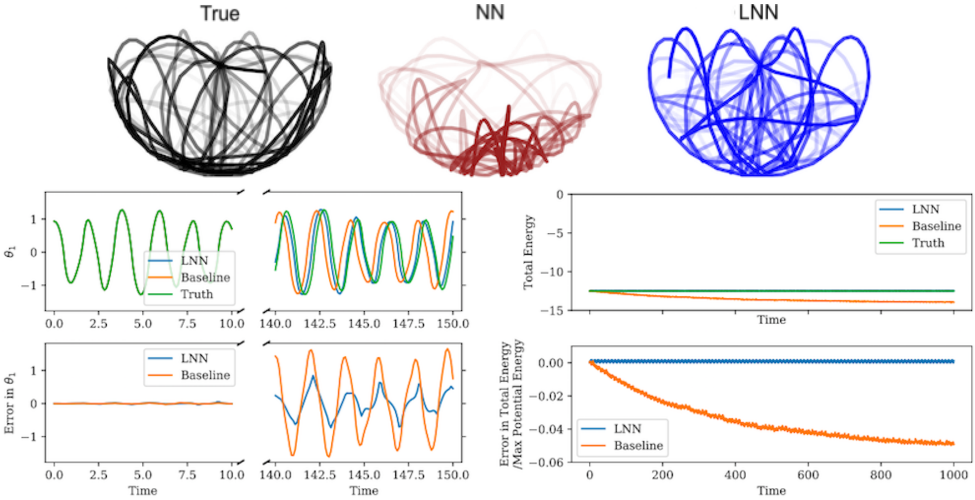
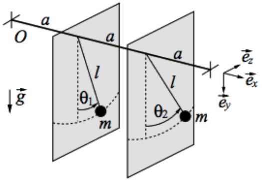


Figure: Results of Sam Greydanus, Miles Cranmer and Stephan Hoyer

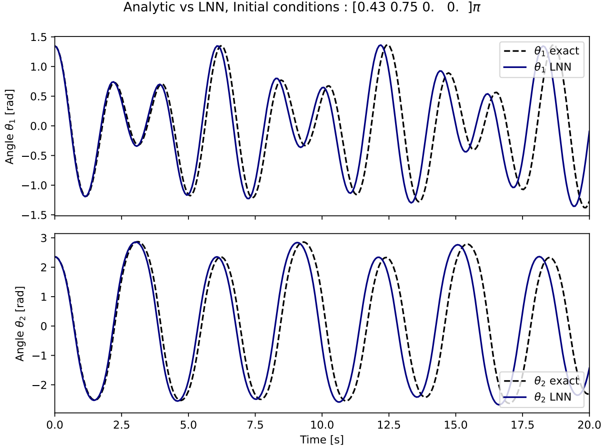
Another system: 2 coupled pendulums



New Lagrangian:

$$\mathcal{L} = \frac{1}{2} ml^2(\dot{\theta_1}^2 + \dot{\theta_2}^2) + mgl(\cos(\theta_1) + \cos(\theta_2)) - \frac{1}{2} k(\theta_1 - \theta_2)^2$$

Comparison of LNN and analytical trajectories:

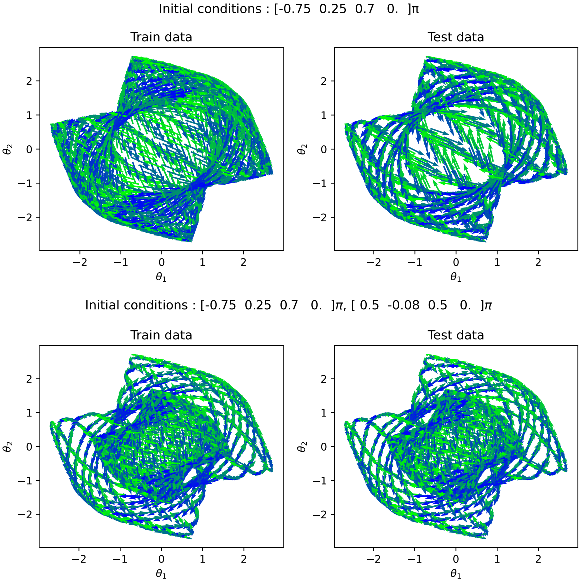


Accuracy is much harder to obtain there is no chaotic solution.

Animation - Torsion

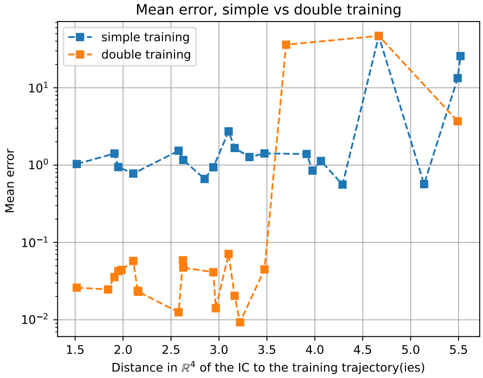
Improving the performance: 2 training trajectories

- Keep same number of total samples for training trajectories
- Goal: train the LNN with a larger coverage of the phase space to improve its adaptability
- Comparison of 1 vs 2 training trajectories in the phase space: (top: 1 traj, bottom: 2 traj.)



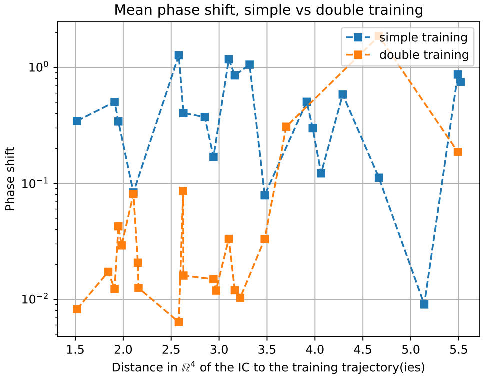
Performance improvement

Mean error:



$$\langle|\theta_{1,an} - \theta_{1,LNN}|\rangle + \langle|\theta_{2,an} - \theta_{2,LNN}|\rangle$$

Phase shift:



When the IC is far from what the LNN has learnt, the performance is poor **regardless of the number of training trajectories**.

Conclusion

- Classical NN are already a good starting point to reconstruct physical trajectories and behaviors of systems.
  BUT they don't act physically !
- NN in difficulty when faced to **chaotic** and **complex** systems.

- A promising solution : train a **physical descriptor of the system** !
- Much more accuracy
- Conservation of physical quantities
- Long lasting fidelity & strong adaptibility

Bibliographie

- Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, Shirley Ho *LAGRANGIAN NEURAL NETWORKS*, arXiv:2003.04630
- Marc Finzi, Ke Alexander Wang, Andrew Gordon Wilson *Simplifying Hamiltonian and Lagrangian Neural Networks via Explicit Constraints*, arXiv:2010.13581

Backpropagation

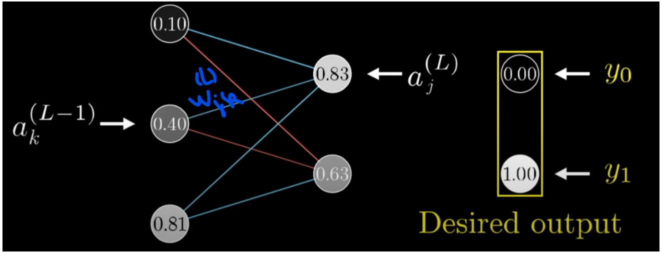


Figure: Architecture of a neural network

We want to compute :

$$\frac{\partial C}{\partial \omega_{jk}} = \frac{\partial C}{\partial a_k^{(L)}} \frac{\partial a_k^{(L)}}{\partial z_k} \frac{\partial z_k}{\partial \omega_{jk}}$$

$a_j^{(L)} = \sigma(z_j)$, with σ a non linear fonction

$z_j = \sum_k w_{jk} a_k^{(L)} + b_k$ and $C = \sum_j (a_j^L - y_j)^2$, the cost function