Double Pendulum Forecasting with Artificial Neural Networks

Theory of the Artificial Neural Networks, Time Series Analysis and Double Pendulum

Juan José Romero Cruz

August, 2023

A humble attempt to provide a theoretical support for a naive and beautiful experiment

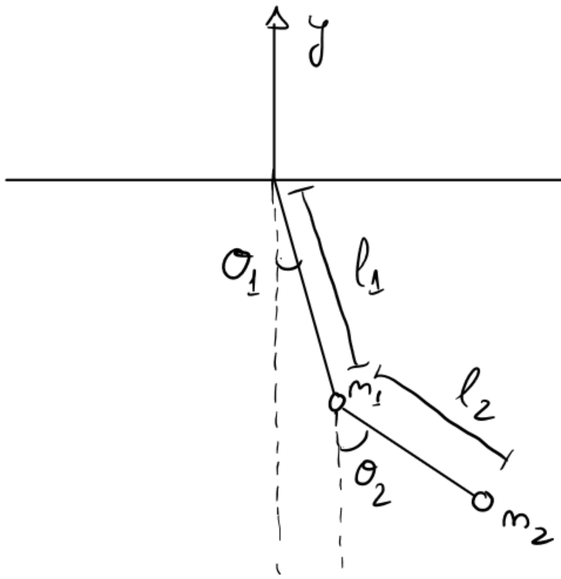
Contents

1 Introduction

This is a little attempt of proposing something such groungbreaking such as the Artificial Neural Networks (ANN) for something which is very important too, which are the dynamical systems, saw as time series.

2 The double pendulum

Firstly, It is important to introduce our problem in a purely mechanical point of view, me have the following problem:



As we can see in this gorgeus picture, we have a mass, m_1, hanging by a thread of lenght l_1 and, at the same time, a mass m_2 hangs by another thread of lenght l_2, which is sticked to the mass m_1. We can use cartesian coordinates to obtain the equations but it is better using polar coordinates since we have, as constrain, that the radius is fixed by l_1 and l_2. So, we only have to use the variables θ_1 and θ_2.

I won't write here all the process that has brought me to the set of differential equations but I'll describe the process, which is based in Lagrangian Mechanics.

The two both masses are influenced by gravity, so the Lagrangian is:

$$\begin{aligned} &L(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}) = T(\dot{\theta_1}, \dot{\theta_2}) - V(\theta_1, \theta_2) \\ &T(\dot{\theta_1}, \dot{\theta_2}) = \frac{1}{2} m_1 (\dot{r_1})^2 + \frac{1}{2} m_2 (\dot{r_2})^2 \\ &V(\theta_1, \theta_2) = m_1 g y_1 + m_2 g y_2 \end{aligned} \tag{2.1}$$

I have found the position vectors of the two both masses, r_1 and r_2, which are:

$$\begin{aligned} &\vec{r}_1 = l1(\sin\theta_1, -\cos\theta_1) \\ &\vec{r}_2 = \vec{r}_1 + l2(\sin\theta_1, -\cos\theta_2) \end{aligned} \tag{2.2}$$

Therefore, the final lagrangian is:

$$\begin{aligned} &L(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}) = \frac{1}{2} m_1 (l_1 \dot{\theta_1})^2 + \frac{1}{2} m_2 (l_1 \dot{\theta_1})^2 + \frac{1}{2} m_2 (l_2 \dot{\theta_2})^2 + \\ &m_2 l_1 l_2 \dot{\theta_1} \dot{\theta_2} \cos(\theta_1 - \theta_2) + m_1 g l_1 \cos\theta_1 + m_2 g l_2 \cos\theta_2 + m_2 g l_1 \cos\theta_1 \end{aligned} \tag{2.3}$$

And now, we can use the Euler-Lagrange equations, which are:

$$\frac{d}{dt}[\frac{\partial L}{\partial \dot{q}_i}] - \frac{\partial L}{\partial q_i} = 0 \tag{2.4}$$

With $q_i = \theta_1, \theta_2$. Finally, after differentiate a lot (wuhuuu), the movement equation for θ_1 is:

$$l_1\ddot{\theta_1} + \mu_2 l_2 \ddot{\theta_2}\cos(\theta_1 - \theta_2) + \mu_2 l_2 (\dot{\theta_2})^2 \sin(\theta_1 - \theta_2) - g\mu_1 \sin\theta_1 - g\mu_2 \sin\theta_1 = 0 \tag{2.5}$$

Where $\mu_1 = \frac{m_1}{m_1+m_2}$, $\mu_2 = \frac{m_2}{m_1+m_2}$. And the movement equation for θ_2 is:

$$l_2\ddot{\theta_2} + l_1\ddot{\theta_1}\cos(\theta_1 - \theta_2) - l_1(\dot{\theta_1})^2 \sin(\theta_1 - \theta_2) - g\sin\theta_2 = 0 \tag{2.6}$$

As we can see, these equations are coupled, this means that $\ddot{\theta_1}$ and $\ddot{\theta_2}$ are in the to both equations. If these equations were discoupled, each second derivative would be in one of the equations. This program solves the equations once they are discoupled, but I won't write this.

Finally, the Hamiltonian is:

$$\begin{aligned} &H(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}) = \frac{1}{2}(m_1 + m_2)(l_1)^2(\dot{\theta_1})^2 + \frac{1}{2}m_2(l_2)^2(\dot{\theta_2})^2 + \\ &m_2 l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2) + m_1 g l_1 cos\theta_1 + m_2 g l_2 \cos\theta_2 + m_2 g l_1 \cos\theta_1 \end{aligned} \tag{2.7}$$

And, the system's energy is:

$$\begin{aligned} &E(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}) = \frac{1}{2}m_1(l_1\dot{\theta_1})^2 + \frac{1}{2}m_2(l_1\dot{\theta_1})^2 + \frac{1}{2}m_2(l_2\dot{\theta_2})^2 + \\ &m_2 l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2) - m_1 g l_1 \cos\theta_1 - m_2 g l_2 \cos\theta_2 - m_2 g l_1 \cos\theta_1 \end{aligned} \tag{2.8}$$

This is the problem we are going to tackle up: we want a model that wouldn't care about the dynamics of this problem, this model will only see the pendulum's evolution a lot of times.

3 Using the ANNs for the double pendulum forecasting

We want to know the solution of the system: $x'(t) = f(t, x)$ given a certain time t. Our model will give us a solution, $x(t + \Delta t)$ given $x(t)$. By this way, we will be able to give to the model the previous solution x(t+Δt) in order to obtain the solution $x(t + 2\Delta t)$. By induction we can say that we could calculate x(t+(n+1) Δt) by using as input x(t + n Δn).

As a note, it could have been a good idea having done a dimensionality reduction by using an "autoencoder type structure".

3.1 Train

What we are going to do is something called "self-supervised learning", which is like supervised learning, but the labeled dataset is generated by the computer itself.

Firtsly, remember that we are treatening the following dynamical system:

$$\begin{aligned} &\dot{x}(t) = f(t, x) \\ &\dot{x}, x \in \mathbb{R}^n, t \in [0, T] \end{aligned} \tag{3.1}$$

Therefore, the train follows like this:

1 We resolve the system by using numeric resolution tecniques such as Runge-Kutta algorithm n times.In each time we resolve this, we give a initial conditions, $(\theta_1(0), \theta_2(0), \dot{\theta}_1(0), \dot{\theta}_2(0))$ by sampling them from a continous uniform distribution $U[0, \pi]$. After solving these equations, we will have a dataset with the variables $t - x(t)$.

2 From this dataset, we will take the data from the 0-th value to the $n - 1$-th value and generate a dataset with this, this dataset will be the values for the supervised learning. And then we will do the same but taking from the 1-th value to the n-th value, this dataset will be the labels for the supervised learning.

3 From each new dataset we have generated, we will take the 90 per cent for the training data, and the rest as the testing data.

4 Train the ANN model.

In this case, the model will be fitted with 10 epochs, using the testing data as the validation data, the model will be evaluated with the testing data as well.

At this point, we will split this project in two possible models: the one that counts on the velocities and the one who does not. The reason why is that I didn't know if was a good idea to include the velocities, because the angular speed, $\dot{\theta}_i$ has a functional relation with the angle of deviation, θ_i. This means that the correlation between these variables should be very strong (then I studied the correlation and it turned on that it isn't that strong) so it could be better to discard the velocities. Finally, I decided to try the to both models.

In the case in whcih I have included the angular speeds, the model of neural network used is the one called "modelo_definitivo1.h5". And, in the case in which I haven't included the angular speeds, the model of neural network used is the one called ”modelo_1.h5”

The hyperparameters and other caracteristics of the models for the problem counting in the velocities and counting out them are:

Hyperparameters	**Model with velocities**	**Model without velocities**
Number of hidden layers	9	5
Learning rate	0.001791702	0.00124108
Dropout rate	0.5	0.5
m (number of times we resolve the ODE's)	3500	1000
time we simulate the system	10 s	10 s
Number of neurons/hidden layer	400 (1-7) and 500 (8-9)	380
Activation function	SELU (without dropout after each hidden layer)	SELU (with dropout after each hidden layer)
Kernel initializer	Le Cunn Normal	Le Cunn Normal
Optimizer	Sthocastic Gradient Descent	Adam
Loss function	Mean Squared Error	Mean Squared Error

Once the model is trained, the last task to do is to forecast the systems's dynamics. Firtsly, we generate new initial conditions and solve numerically the differential equations, and then I though in using two methods:

1. **Method 1 (”Stepper” or ”Step by Step”)**. Consists in generate randomly the initial conditions. With them, we obtain the variables at a time Δt, and then we iterate until we reach time T (the time that the simulation lasts).

2. **Method 2** Consists in take the dataset generated as solution of the ODEs, split it in two parts, and use the firts part of this dataset to create a new second half by using the ANN model.

It turned on that the most accurate method was the second one, so all the animations and the exploratory data analysis have been done after use this method.

3.2 Testing the model

In order to see if the model is accurate with the dynamics of the system, I have done two tasks:

1. Calculate the diference between the second half forecasted by solving numerically the ODEs and the second half forecasted by using the ANN. It is a kind of absloute error.

$$MEA(t) = |x_{ODE}(t) - x_{ANN}(t)| \tag{3.2}$$

   Where $x_{ODE}(t)$ is the solution given by solving numerically the EDO's and $x_{ANN}(t)$ is the solution given by the ANN.

2. Generate an animation of the pendulum, where the first half of the animation is the movement of the pendulum predicted by the ODE's and the second is the second half predicted by the ANN. This gives a intuition of the accuracy of the model.

References

[1] N. Kutz,
Neural Networks for Dynamical Systems, See the link to the video here

[2] A. Géron,
Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow, (2019)

[3] P.I. Viñuela & I. M. Galván,
Redes de neuronas artificiales. Un enfoque prático, (2004)

[4] Solving and animating double pendulum with Matplotlib