Pablo Lahmann - 19-947-035
AI in the Sciences and Engineering
June 26, 2024

Project 3: Neural differential equations, model discovery, and transfer learning

DUE: June 26th.

Question 1. Controlling an Inverted Pendulum

Task 1.1: Produce plot of $x(t), \theta(t)$. Use a standard ODE solver. For this Task I used the diffeqsolve method from the diffrax library. The solver Tsit5 proved to give satisfying results. Implementing this task was generally quite straightforward. First I define the functions which describe the environment, i.e. the force function and then the pandulum dynamics. The latter of which describes the differential equations to be solved. Here you can find a plot of the resulting solution.

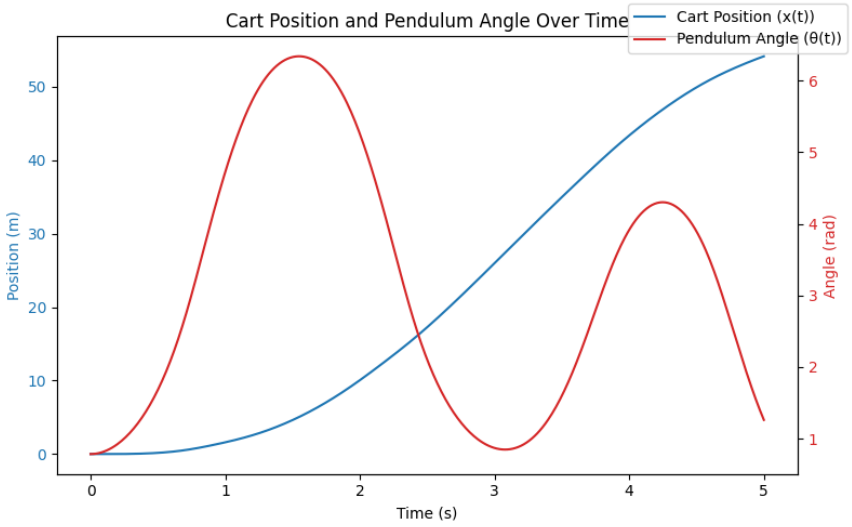

FIGURE 1. Caption of your plot

Task 1.2: For this task I decided to use a fully connected NN with one hidden layer of size 32. The NN normalizes its input to make training easier. During training, in each epoch, I pass the NN model, which learns a function F. This F is then passed together with the same pendulum environment function from task 1.1 into the ode solver odeint from the library torchdiffeq. This solver turned out to work better for me. Using the same solver from 1.1 would have needed a lot of debugging. To ensure the NN learns to keep the pendulum at an angle close to 0, I used 4 different terms for my loss. 1. a term for keeping the positional value of the cart low, ensuring that the cart doesn't run off into infinity. 2. a term for keeping the angle itself low. 3. a term for keeping the angles derivative low as well, for smoother behavior. I received best results on the relevant last quarter of the time interval, when applying the penalties 2 and 3 on the NN for the entire second half of the time frame (as opposed to only penalizing the last quarter, like suggested in the task description. 4. an additional term, more heavily penalizing the angle during the last half second, to be close to zero. This last penalty I introduced, to encourage the NN to not loose the balance that was won during the balancing process. Naturally I achieved best results when weighting the penalty 2 the most, among all four penalty terms. Lastly, during training I initially used very fine spacing of the time values. It turned out that using 100 linearly spaced points the the time interval $[0, 5]$ turned out to also give very good results, while considerably speeding up the training process (compared to using e.g. 5000 points). Here you can see the relevant plots for this task:

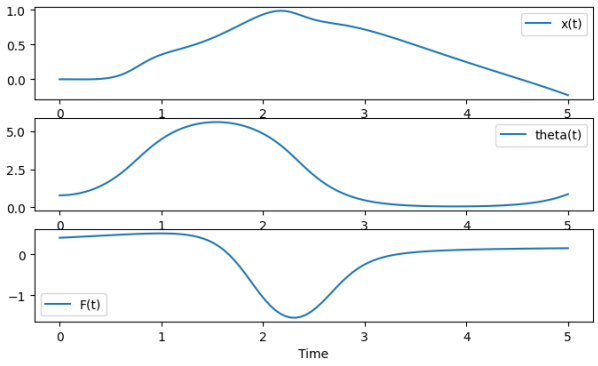

FIGURE 2. Caption of your plot

Question 2. PDE Discovery

For this task I essentially always used the same approach. Though equation 3 needed an adaptation to that approach. Thus the following description mostly applies to equations 1 and 2. For approximating the partial derivatives I tried several approaches, of which the finite difference method turned out to work best. This works, since after brief inspection of the data, I could see that the points are always evenly spaced on a grid. Using a NN would typically not give too good results. To not be overwhelmed with possible partial derivatives, I did not include too many complex combinations, but decided to opt for less complexity. This typically allows for better interpretation of the result. This still seemed to achieve quite good results. As suggested in the provided paper, I used ridge regression for optimizing a sparse vector of coefficients. For better readability of the output, I also include a threshold, where all coefficients with lower absolute value are truncated to zero. Further, I use the $L0$ norm for the ridge regression, to encourage sparsity of the solution vector (this is suggested in the paper). Lastly I use the adam optimizer to descend to a good solution. This results in the following two possible PDEs to describe the provided data.

Equation 1: $u_t = -0.1u_{xx} - uu_x + 0.01u^2u_x$
Equation 2: $u_t = -0.7u_x - 1.05uu_x + 1.79u^2u_x$

Both Equation 1 and 2 could be solved with essentially the same methodology.

Though for solving Equation 3, a little more work was necessary. Given that I am now working with a higher dimensional domain and solution space. Thus I decided to approach this again, completely from zero. Generally, I achieved the most satisfying results using the finite difference method for approximating the partial derivatives, even though it turned out to be quite tedious to compute. Essentially, I did this analogously to the first part of this task. Like hinted in the task description, I assumed in equation (7) that both u_t and ξ are matrices with two columns. Thereby learning both PDEs simultaneously. Furthermore, the space of possible PDE terms to use for this problem is much larger than for the previous equations. Thus, for my solution library, from which terms to choose, I decided to restrict myself to very simple terms. Using anything more than ca. 22 features would not compile. This is why I decided to disregard quadratic terms, aswell as third derivatives. Again, an advantage of this approach is, that the final result is much more easily interpretable, compared to using a very large solution library. Like in the previous parts of this task, I used the ridge regression method from the paper, which regularizes the coefficients using the $L0$ norm. This encourages sparsity in the solution vector. Thus receiving the following result.

Equation 3:
$u_t = 0.1u^2 + 1.01v^2 + 0.57uv + 0.37u_x + 0.05v_x + 0.01u_y + 0.19u_{yy} + 0.73uv_x + 0.99vv_y + 0.04v_xu_y$

$v_t = 2.02u + 0.12u^2 + 0.52v^2 + 0.57uv + 0.32u_x + 0.55v_x + 0.12u_y + 0.78u_{yy} + 0.13v_y + 0.68uv_x + 0.02uv_y + 0.01vv_x + 1.1vu_y + 0.98vv_y$