Training Neural Networks on the data from: "On the parameter combinations…"

I examined the reduced-MSP model from the above paper and gathered data from Runge-Kutta integrations of the equations, while varying the parameters that are incorporated into the equations of the systems. Then I trained two types of Neural Networks based on these data and evaluated their accuracy.

Experiment: Varying the parameters of the system

- First, I imported the necessary libraries, determined the parameters of the system based on the ones the paper uses ($k_{f,1} = 0.71$, $k_{f,2} = 0.97$, $k_{r,1} = 19$, $k_{r,2} = 7000$, $k_{cat,1} = 6700$, $k_{cat,2} = 10000$) made the function for the derivatives of the system, the functions for the Runge-Kutta and Euler integration maps and the one that computes the trajectory using these, including the parameters of the system as an input to the functions.
- Next, I took a sample of 2000 random sets of values for the parameters, with deviation of -75% to -100% from the ones given by the paper (this is where the steady state starts to change), and stored the trajectories (from Runge-Kutta) using these starting values. From these trajectories, I then made the training and test data, taking the points of 80% of the trajectories as training and 20% as testing. 20% of the testing points are used as validation dataset for the training.
- Then, I trained a Neural Network to learn the time-1 map of the system, and consists of the input and output layers (input is a point in the S0, S1, S2 3D space and the parameters of the system and output is the point after time dt) and 2 hidden fully connected layers with 15 nodes each (after normalizing the input values to the same magnitude). An overview of the model is shown to the right:

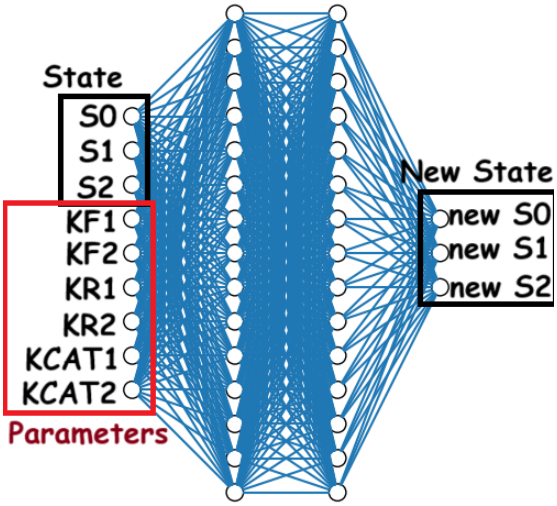

Figure 1: Diagram of the structure of the time-1 map Neural Network

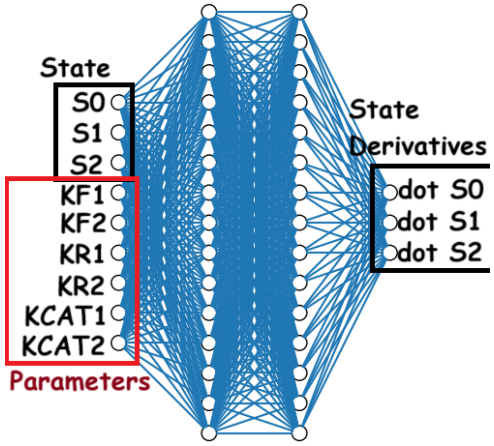

Figure 2: Diagram of the structure of the derivative Neural Network

- I also trained a model that predicts the derivatives of the values at a certain point in the S0, S1, S2 space (the RHS of the differential equation of the system). Its structure is the same as the previous model, but the output contains the derivatives instead of the values of the map. The training data are also based on the trajectories as before. An overview of the model:

- Then I plotted the trajectories of the two different models, along with the ones with the Runge-Kutta method, with 6 sets of randomly selected parameters of the test data, in order to evaluate their accuracy. Dotted line is the derivative Neural Network trajectory, solid line the Runge-Kutta, and dashed line the time-1 map Neural Network. Also, green line is S0, blue line S1, and red line S2.

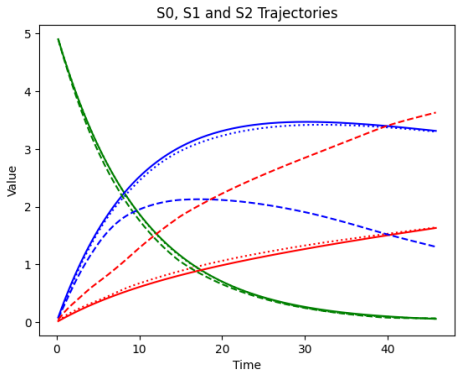

Figure 3: S0, S1 and S2 trajectories, given the 1st randomly selected set of test parameters and the same start state, calculated using the time-1 map model, the derivative model, and the Runge-Kutta method.

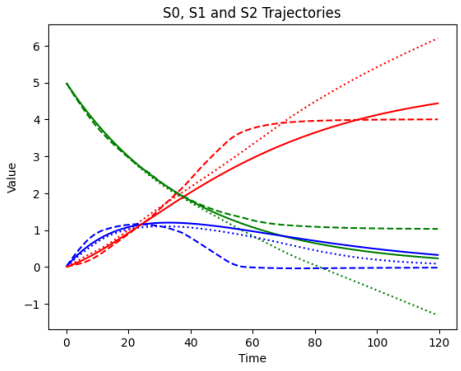

Figure 4: S0, S1 and S2 trajectories, given the 2nd randomly selected set of test parameters.

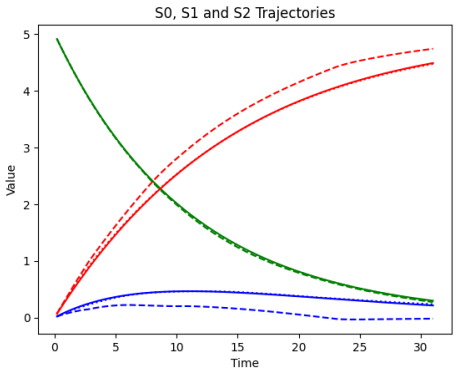

Figure 5: S0, S1 and S2 trajectories, given the 3rd randomly selected set of test parameters.

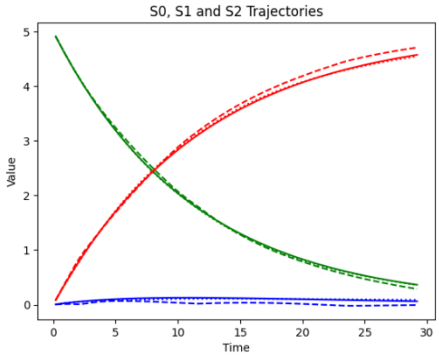

Figure 6: S0, S1 and S2 trajectories, given the 4th randomly selected set of test parameters.

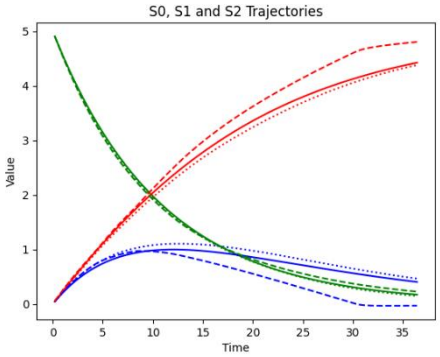

Figure 7: S0, S1 and S2 trajectories, given the 5th randomly selected set of test parameters.

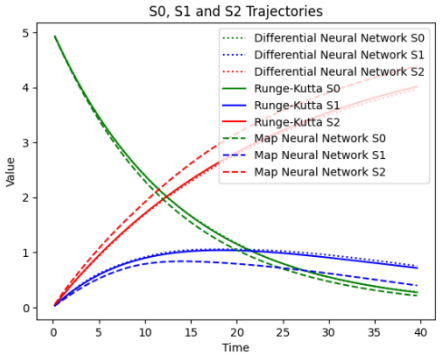

Figure 8: S0, S1 and S2 trajectories, given the 6th randomly selected set of test parameters.

- In all the trajectories, I stopped time-integrating when the system reached a steady state, which was detected when the norm of the difference of the new state with the previous state was less than 0.2% of the norm of the initial state. Then I calculated the norm of the difference of the steady states computed with the time-1 map and the derivative Neural Network, with the steady state computed with the Runge-Kutta method. The results are shown below (for the trajectories plotted above):

	# of set of randomly selected test parameters					
	1st	2nd	3rd	4th	5th	6th
Differential NN Error in Steady State	0.017	2.35	0.03	0.037	0.08	0.063
Time-1 Map NN Error in Steady State	2.83	0.97	0.34	0.17	0.58	0.5

Table 1: Errors in steady state for the time-1 map and the derivative Neural Network, with 6 sets of randomly selected test parameters.

Resources

- The python notebook with which I made the plots and trained the models is: parameters_paper_parameters.ipynb
- The models (in .tf format) are: models.zip