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 200 random 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 plotted the trajectories (from Runge-Kutta) using these starting values. The relevant plots are presented below:

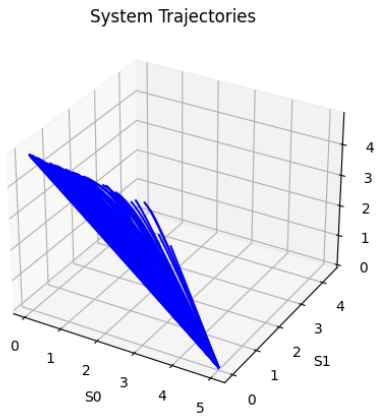


Figure 1: Set of Trajectories in 3D

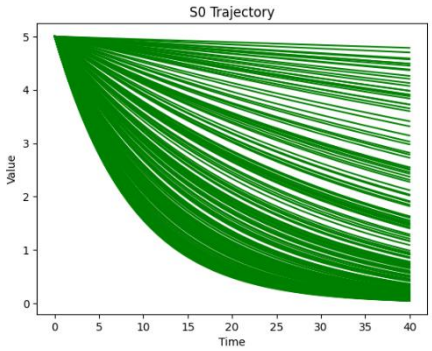


Figure 2: Set of S0 Trajectories

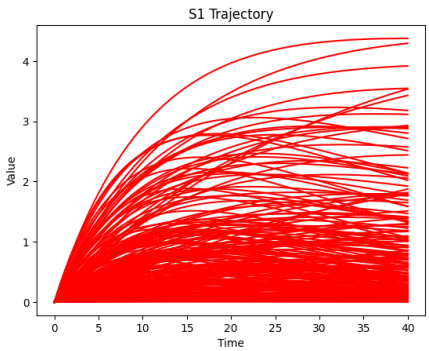


Figure 3: Set of S1 Trajectories

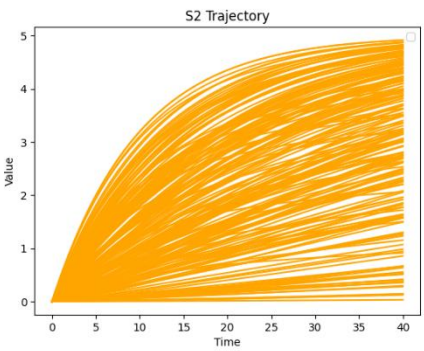
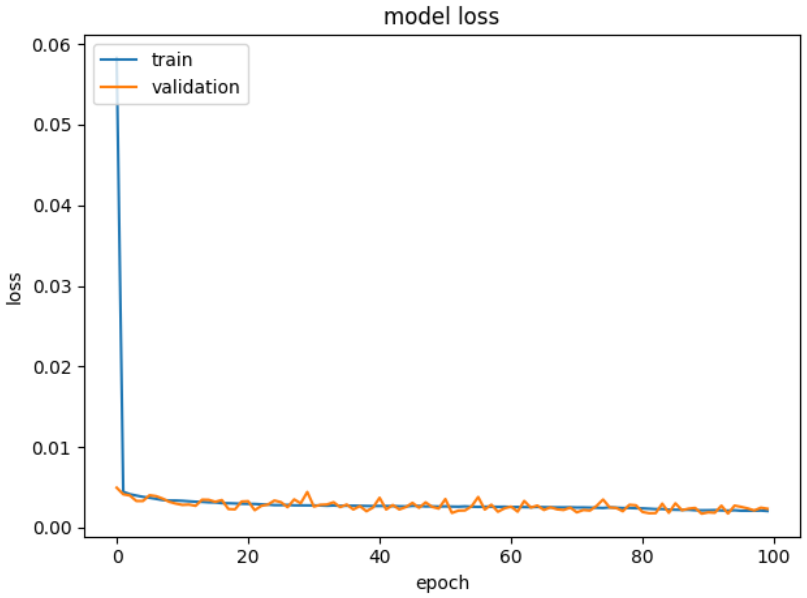


Figure 4: Set of S2 Trajectories

- From the same trajectories, but with 2000 parameter values, I also made the training and test data, taking 80% of the 400000 points (200 steps with 2000 choices for the parameters) 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 a layer that normalizes the input values to the same magnitude)
- The results of the training are shown below:




```
Evaluate on training data
2500/2500 [==============================] - 4s 1ms/step - loss: 0.0023
training loss: 0.0023277720902115107
Evaluate on test data
625/625 [==============================] - 1s 1ms/step - loss: 0.0023
test loss: 0.0023304636124521494
```

- Then, I plotted the trajectory of the Runge-Kutta method and the one predicted from the time-1 map model, with -85% of the base parameters and the initial starting values, to evaluate its accuracy. Blue is the Runge-Kutta, and red the Neural Network:

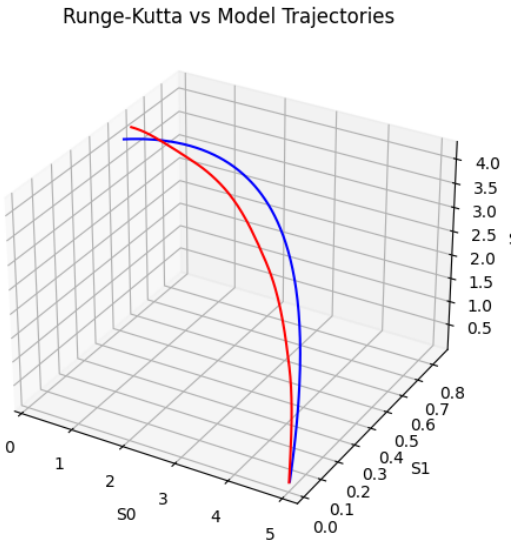


Figure 5: Trajectories in 3D

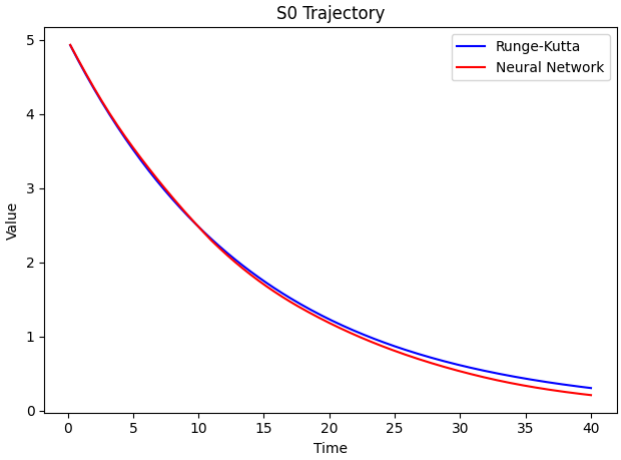


Figure 6: S0 Trajectories

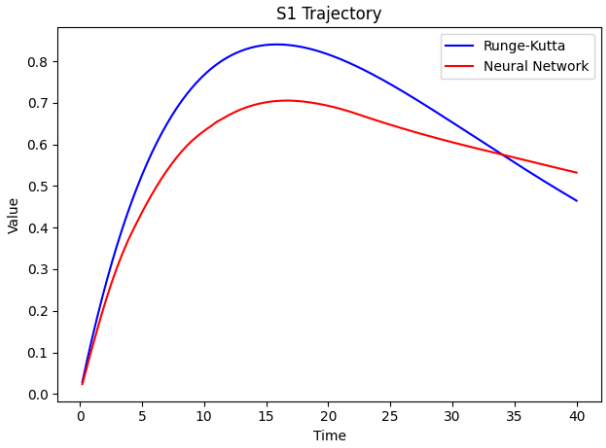


Figure 7: S1 Trajectories

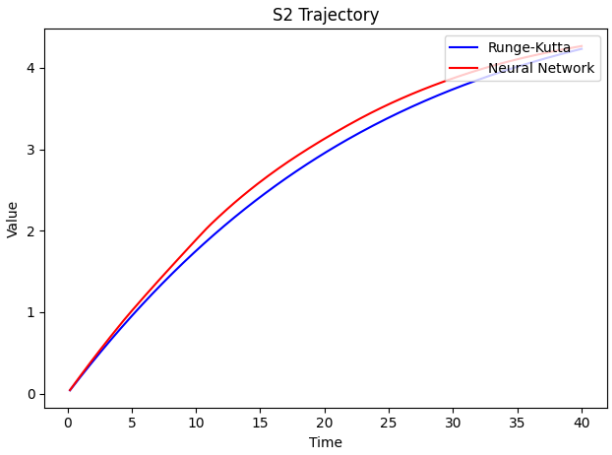


Figure 8: S2 Trajectories

- I also plotted the Runge-Kutta and time-1 map model trajectories with the same starting value but with 10 different values for the parameters, sampled randomly with -75% to -100% deviation from the base parameters:

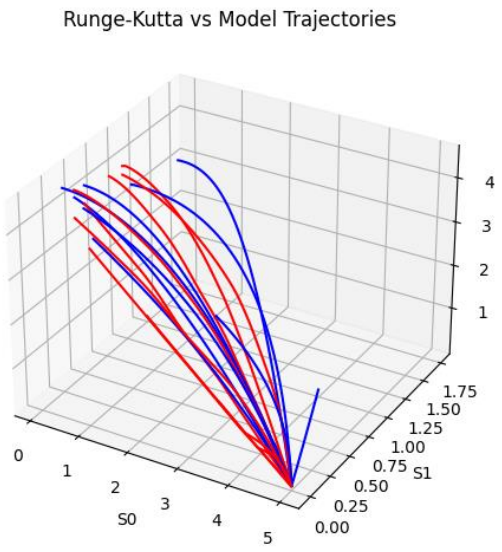


Figure 9: Set of trajectories in 3D

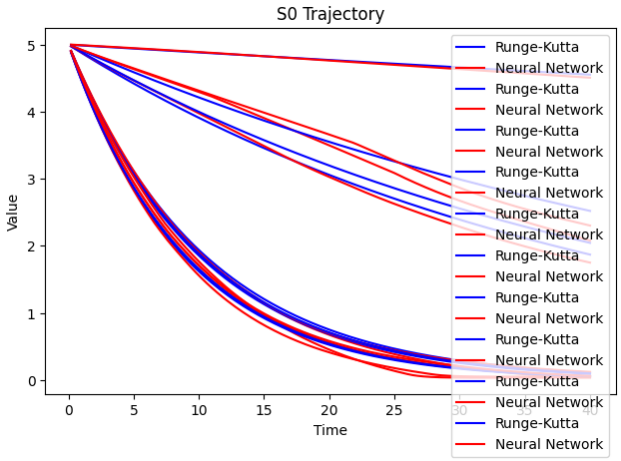


Figure 10: Set of S0 Trajectories

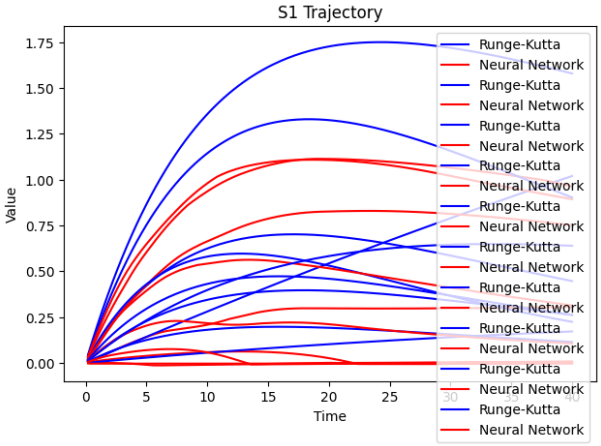


Figure 11: Set of S1 Trajectories

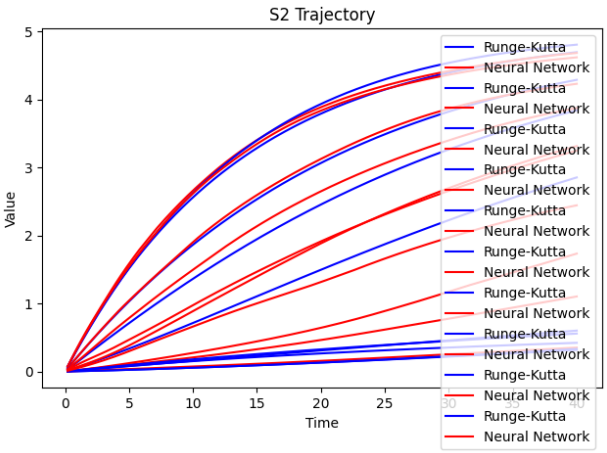
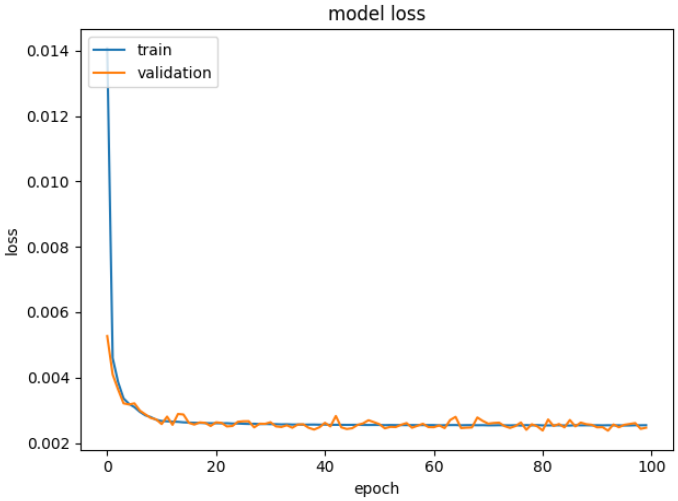


Figure 12: Set of S2 Trajectories

- Then I 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. The training results are shown below:




```
Evaluate on training data
  30/2500 [..............................] - ETA: 4s - loss: 0.0024
2023-08-08 17:12:02.955358: W tensorflow/tsl/framework/cpu_allocator_imp
2500/2500 [==============================] - 4s 1ms/step - loss: 0.0025
training loss: 0.0024633295834064484
Evaluate on test data
625/625 [==============================] - 1s 1ms/step - loss: 0.0025
test loss: 0.002465428551658988
```

- Then I plotted the trajectories of the two different models, along with the ones with the Runge-Kutta method, in order to evaluate their accuracy. Green is the derivative Neural Network trajectory, blue the Runge-Kutta, and red the time-1 map Neural Network:

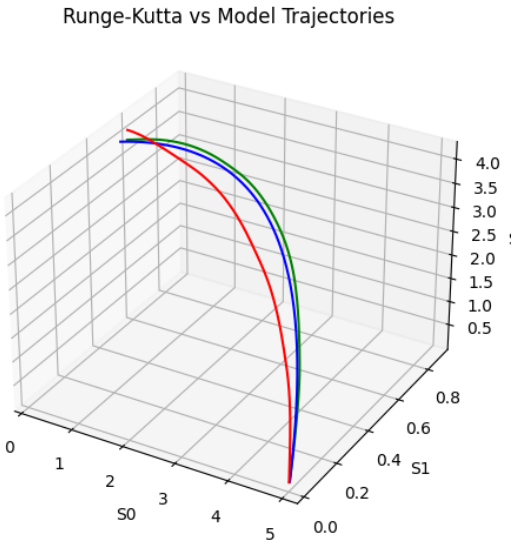


Figure 13: Trajectories in 3D

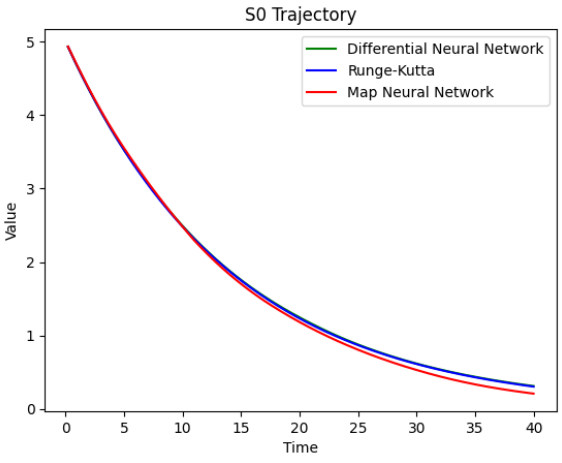


Figure 14: S0 Trajectories

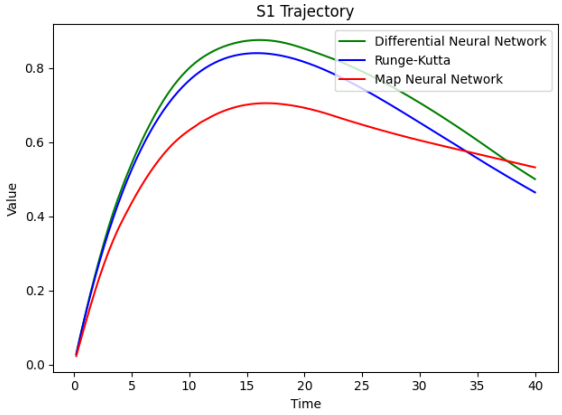


Figure 15: S1 Trajectories

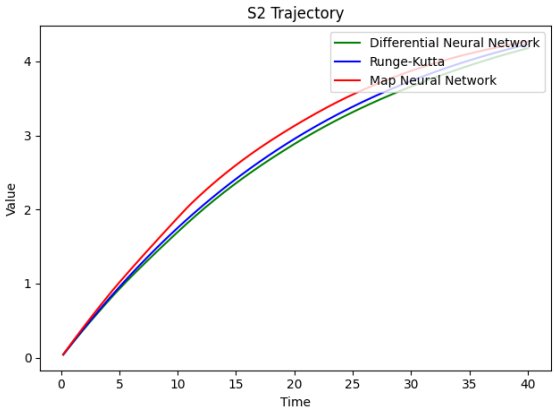


Figure 16: S2 Trajectories

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