# Non-Linear Dynamics Through Linear Algebraic Lenses: Attempting to Learn the Trajectories of the Logistic Map with Artificial Neural Networks

Ritobrata Ghosh ({lastname]r@zohomail.in)

April 18, 2023

**Abstract**

To automatically learn the behavior of trajectories of a map in Non-Linear Dynamics- the Logistic Map, Deep Neural Networks have been trained. Different iterates of the Logistic Map have been generated and models have been fit to them to test the learning capabilities of Neural Networks under such scenario. This paper examines the capability of Neural Networks to learn the dynamics of a system that can be modeled with the Logistic Map.

**keywords**- Non-Linear Dynamics, Deep Learning, Artificial Neural Networks, Physics, Computational Mathematics, Logistic Map

## 1 Introduction

The study of Non-Linear Dynamics depends heavily on studying the trajectories of several maps and flows. This paper investigates the possibility of training Deep Neural Networks to learn and predict the trajectories of the Logistic Map with several iterates and different parameters. This is an investigatory study to check whether it is possible to do so.

The Logistic Map, and Non-Linear Dynamics, in general is a tool that has found its application in wide fields such as Physics [Lor63], Chemistry [CB06], Biology [MO76] [May88], Economics [HM11], and more [SS17]. In this paper, the possibility of using Deep Linear Networks to learn the dynamics of the Logistic Map is investigated.

## 2 Methodology

### 2.1 Choice of Language and Frameworks

- **Python** was chosen as the primary language for this study as it is the most widespread one to be used for tasks involving Deep Neural Networks.
- The framework for working with both 1D tensors to generate trajectories and defining and training Deep Neural Networks was chosen to be **PyTorch** [PGC+17].
- The library for plotting trajectories, which is very important in the studies involving Non-Linear Dynamics, and Chaos Theory, was chosen to be Matplotlib [Hun07].
- While these are the main components of the study, a complete list of libraries, frameworks, and other dependencies can be found in the GitHub repository of this project [Gho23a].

### 2.2 Platform, Operating System, and Software

- All codes involving tensors and Deep Learning were run on an **NVIDIA GeForce RTX 3050 Ti** graphics card with 2304 CUDA cores mounted on an Asus laptop.
- Other codes were run on **11th Gen Intel i5-11300H**.
- The OS was **Pop!_OS 22.04 LTS x86_64** with Linux Kernel version **6.2.6-76060206-generic**.
- For environment management, **conda** was used [ana20].
- For writing the code, JupyterLab Desktop, built on top of Jupyter was used.

## 2.3 The Logistics Map

### 2.3.1 The Function

The Logistic Map is defined as:

$$x_{n+1} = rx_n(1 - x_n)$$

Where $r$ is a parameter and $x_n$ is the value of $x$ in the time-step $n$, and $x_{n+1}$ is the result.

The Logistic Map is a discrete time function and differs from flows where time is continuous. It is a "difference equation".

### 2.3.2 Implementation

The Logistic Map is implemented as below:

```
def logistic_function(parameter: float, initial_value: float) -> float:
    '''
    implements one iteration of the logistic fucntion
    '''
    return parameter * initial_value * (1 - initial_value)
```

Listing 1: Implementation of Logistic Function

## 2.4 Generating Data (Logistic Map Outputs) for Training

The outputs of several iterates (first, second, and third) of the Logistic Map were stored into PyTorch tensors.

Before that, the $x$ values were chosen to be in a range.

```
# range of numbers from 0 to 4 like 0.01, 0.02...3.99, 4.00
x_0s = torch.arange(start=0, end=4, step=0.01, dtype=torch.float32, device='cuda')
```

Listing 2: Generating Inputs of Logistic Map

Note that the tensor is initialized in the GPU memory, and not ordinary RAM. And the datatype chosen is `torch.float32`, i.e. PyTorch's representation of 32-bit floating point numbers.

Then they were used as inputs to the function described in the previous Subsection.

```
# generating outputs of the first iterate of the logistic map
# logistic_function_22 is the logistic map with r = 2.2
ys = torch.func.vmap(logistic_function_22)(x_0s)
```

Listing 3: Generating Outputs of Logistic Map

`torch.func.vmap` was used for a functional and much faster calculation which is apt for this use case.

## 2.5 Generating Data (Logistic Function Trajectory) for Training

A function was defined to generate the trajectory for the Logistic Map. The data generated by this function for different values of the parameter were used to make the plots, and to generate the data used for training.

Below is the definition of the function. Taken from a Kaggle Notebook authored by the author. [Gho23b]

```
def logistic_function_trajectory(parameter: float,
                                 initial_value: float,
                                 num_iter: int) -> torch.Tensor:
    '''
    function for generating trajectory and storing them into
    PyTorch tensors.
    '''
    trajectory = torch.zeros(num_iter)
    trajectory[0] = initial_value
    for i in range(num_iter-1):
        trajectory[i+1] = logistic_function(parameter, initial_value)
        initial_value = trajectory[i+1]
    return trajectory.clone().detach()
```

Listing 4: Function for Generating Trajectories of the Logistic Map

## 2.6 DataLoading for Neural Network Training and Splitting

For points of data to be fed into a Neural Network, a proper structuring of data is needed.

And at the same time, the whole of the data won't be fed into the Neural Network for training. 20% of the data were kept away as the "test set" where the trained Neural Network would be tested.

The Neural Network would not see this data while getting trained.

The splitting was done using the `train_test_split()` method in the `model_selection` module of Scikit-learn library.

The dataloading was done using methods in the `torch.utils.data` module of PyTorch.

For full code, refer to the GitHub repository of this paper. [Gho23a]

## 2.7 Defining and Training the Neural Network

### 2.7.1 Defining the Neural Network

A Neural Network with four layers was trained from scratch for each iterate, and later, for each trajectory with different parameter values.

Here is how the Neural Network is defined:

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.layer1 = nn.Linear(input_size, 3)
        self.layer2 = nn.Linear(3, 5)
        self.layer3 = nn.Linear(5, 3)
        self.layer4 = nn.Linear(3, output_size)
    def forward(self, x):
        x = self.layer1(x)
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        x = self.layer3(x)
        x = F.relu(x)
        x = self.layer4(x)
        return x
```

Listing 5: Defining the Neural Network

The `nn.Module` is from the `torch.nn` module.

### 2.7.2 Training the Neural Network

The Neural Networks' architectures are kept the same. For some cases, the number of epochs it was trained for was changed. [RLM22] [ZLLS21]

```
# initializing a Neural Network object from the NeuralNetwork class
nn_model = NeuralNetwork(1, 1).to(device)

learning_rate = 0.001
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(nn_model.parameters(), lr=learning_rate)

num_epochs = 100
# storing the loss for each epoch
loss_hist = [0] * num_epochs

# main training loop
for epoch in trange(num_epochs):
    for x, y in train_dl:
        pred = nn_model(x)
        loss = loss_fn(pred, y)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        loss_hist[epoch] += loss.item()*y.size(0)
    loss_hist[epoch] /= len(train_dl.dataset)
```

Listing 6: Training the Neural Network

## 2.8 Predicting the $y$s from Unseen Data

The part of the data that was kept away from the training dataset is used to test the prediction of the Neural Networks.

Mean Squared Error was used for a numerical measurement of the training for each model.

Plots were drawn for each Neural Network on unseen data.

## 2.9 Maintaining Reproducibility

To ensure reproducibility, i.e. the results were the same every time the code were run, giving the same results, measures were taken.

### 2.9.1 Reproducibility while Splitting the Data

When using the `train_test_split()` method from the Scikit-learn library's `model_selection` module, an argument called `random_state` was passed to ensure that every time the code was run for each trajectory/iterate, the data would be split in the same way.

```
# generating training and test data  from the function outputs
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Listing 7: Generating the same train-test split each time.

### 2.9.2 Reproducibility for PyTorch

By design, PyTorch's behaviour is probabilistic. To ensure the same results for each time the code is run, manual seed was set for each iterate/trajectory, before training the model.

```
# setting manual seed for PyTorch
torch.manual_seed(0)
```

Listing 8: Maintaining reproducibility for PyTorch code

## 2.10 Details of All the Iterates and Trajectories Used

### 2.10.1 List of All Iterates and Trajectories

Below are the iterates and trajectories used in this study.

| Serial | Iterate/Trajectory | # of Iterate | Range | Step Size | $r$ | $x_0$ | # of Iteration | Nature of Dynamics |
|---|---|---|---|---|---|---|---|---|
| 1 | Iterate | 1st | 0 to 4.0 | 0.01 | 2.2 | N/A | 1 | N/A |
| 2 | Iterate | 2nd | 0 to 4.0 | 0.01 | 2.2 | N/A | 1 | N/A |
| 3 | Iterate | 3rd | 0 to 4.0 | 0.01 | 2.2 | N/A | 1 | N/A |
| 4 | Trajectory | N/A | N/A | N/A | 2.9 | 0.2 | 100 | convergent to a stable atttractor |
| 5 | Trajectory | N/A | N/A | N/A | 3.0 | 0.2 | 100 | convergent to stable attractor |
| 6 | Trajectory | N/A | N/A | N/A | 3.2 | 0.2 | 100 | 2-cycle |
| 7 | Trajectory | N/A | N/A | N/A | 3.5 | 0.2 | 100 | 4-cycle |
| 8 | Trajectory | N/A | N/A | N/A | 4.0 | 0.2 | 100 | chaotic |

Table 1: Details of all iterates and trajectories used in this paper.

### 2.10.2 Plots of All Iterates and Trajectories

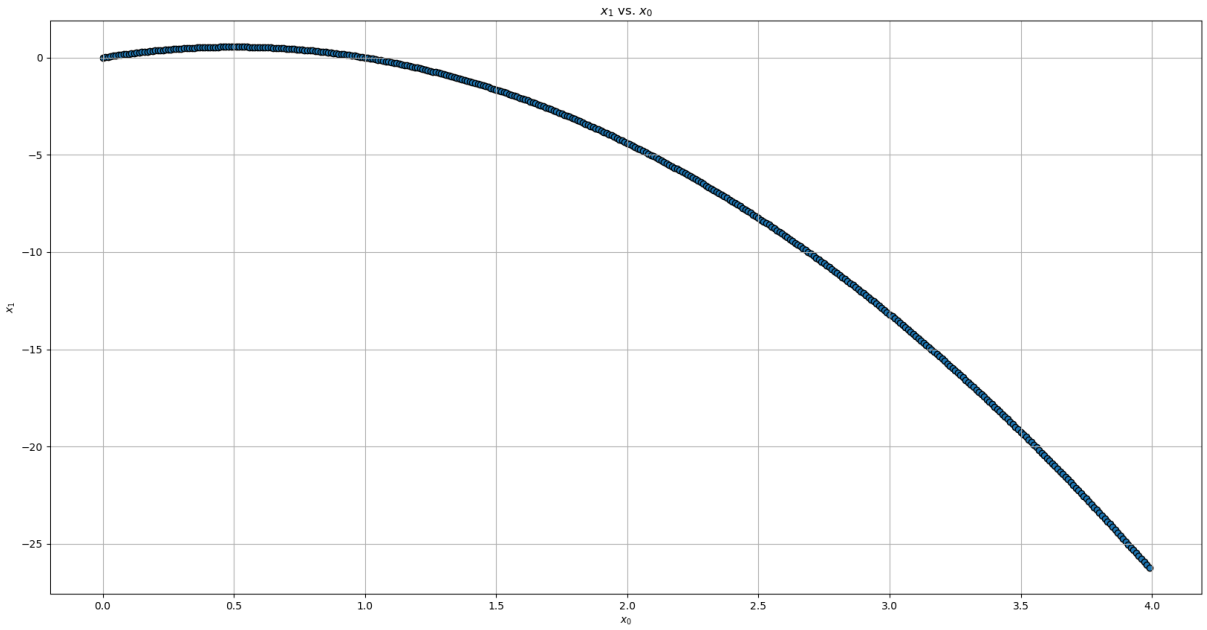



Figure 1: Serial 1.

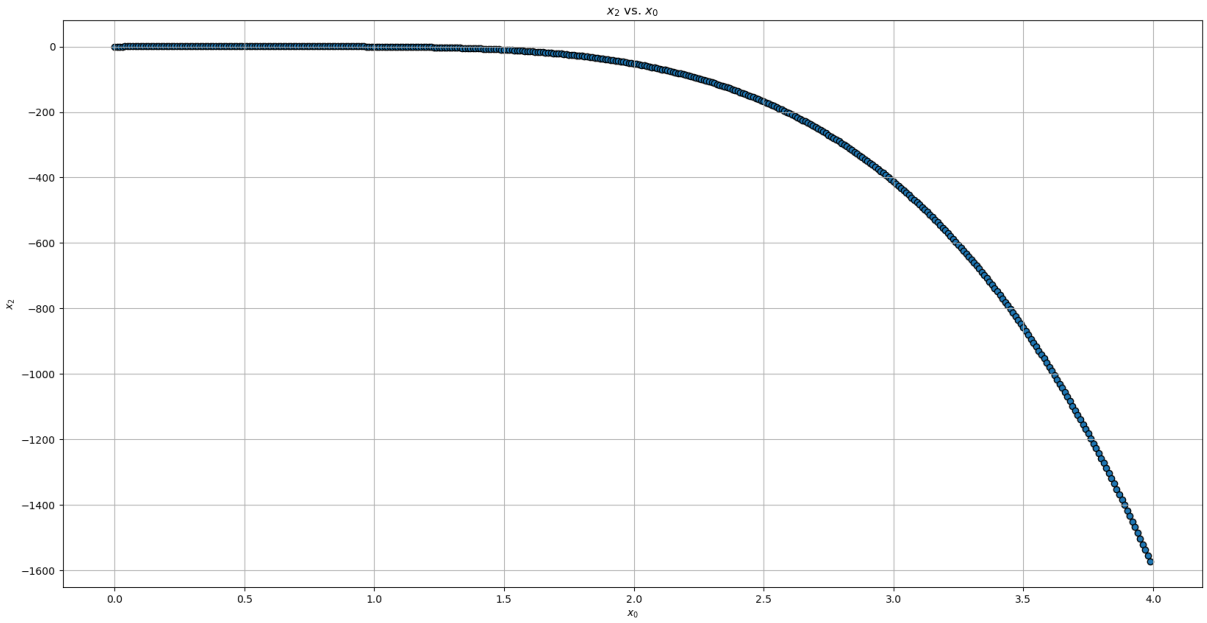



Figure 2: Serial 2.

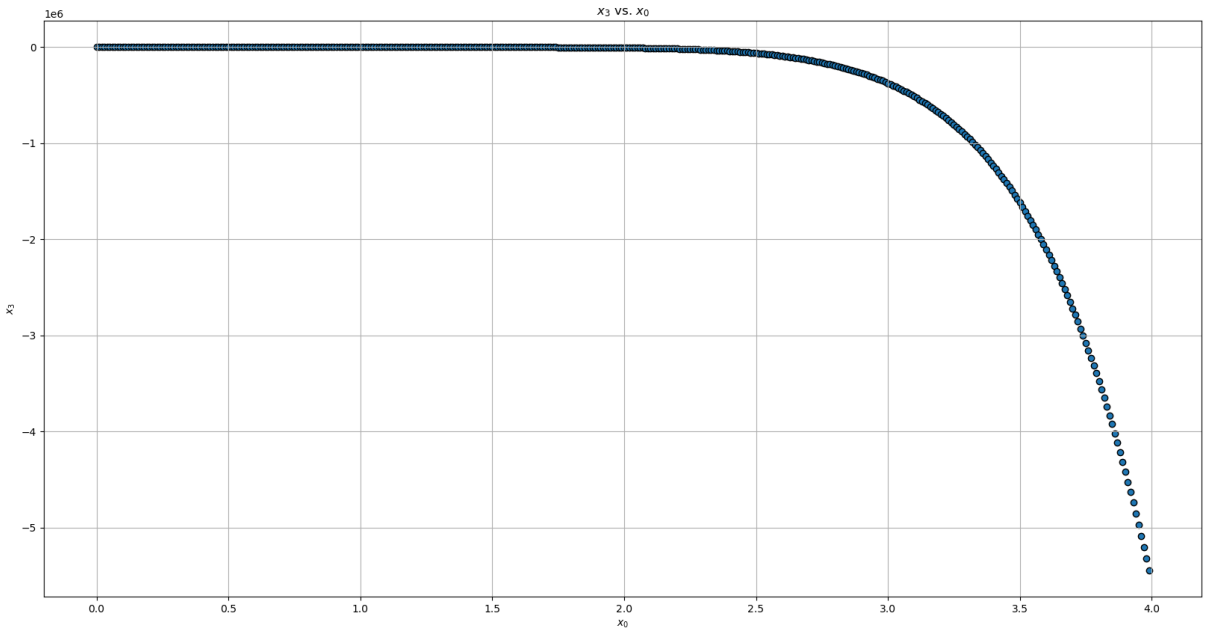



Figure 3: Serial 3.

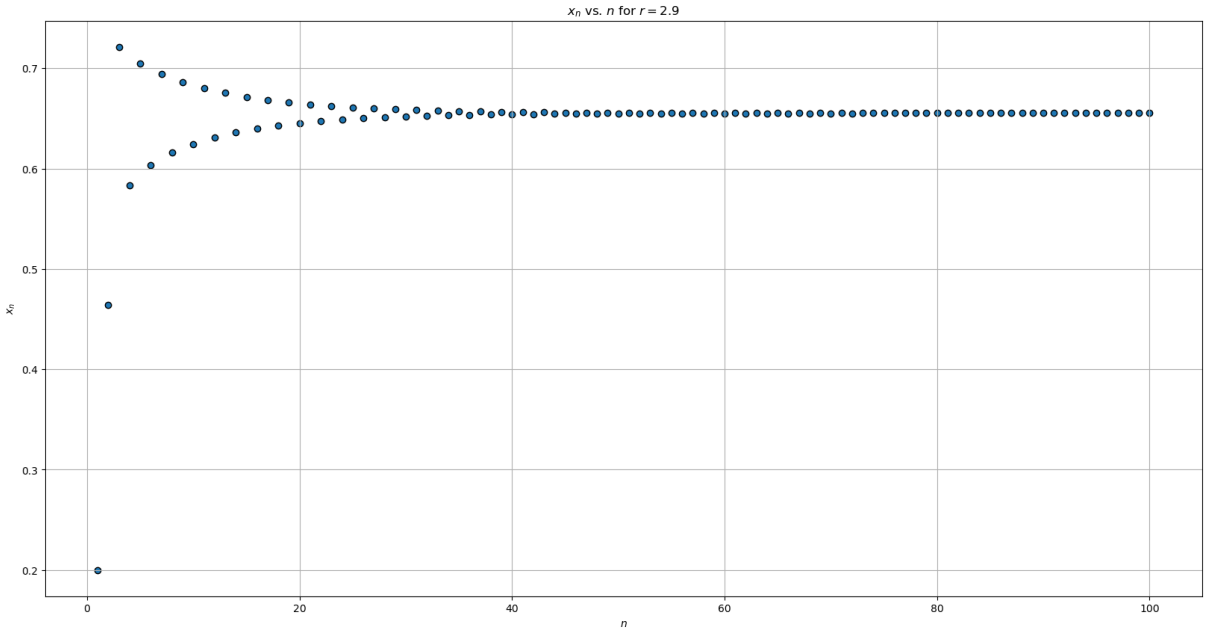



Figure 4: Serial 4.

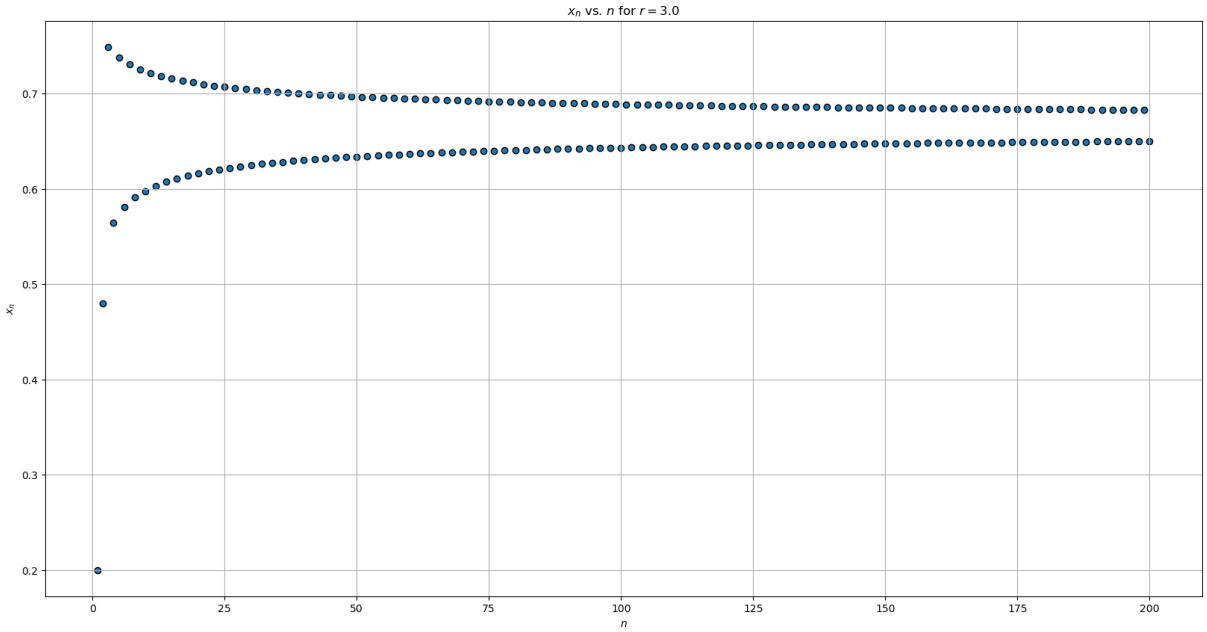



Figure 5: Serial 5.

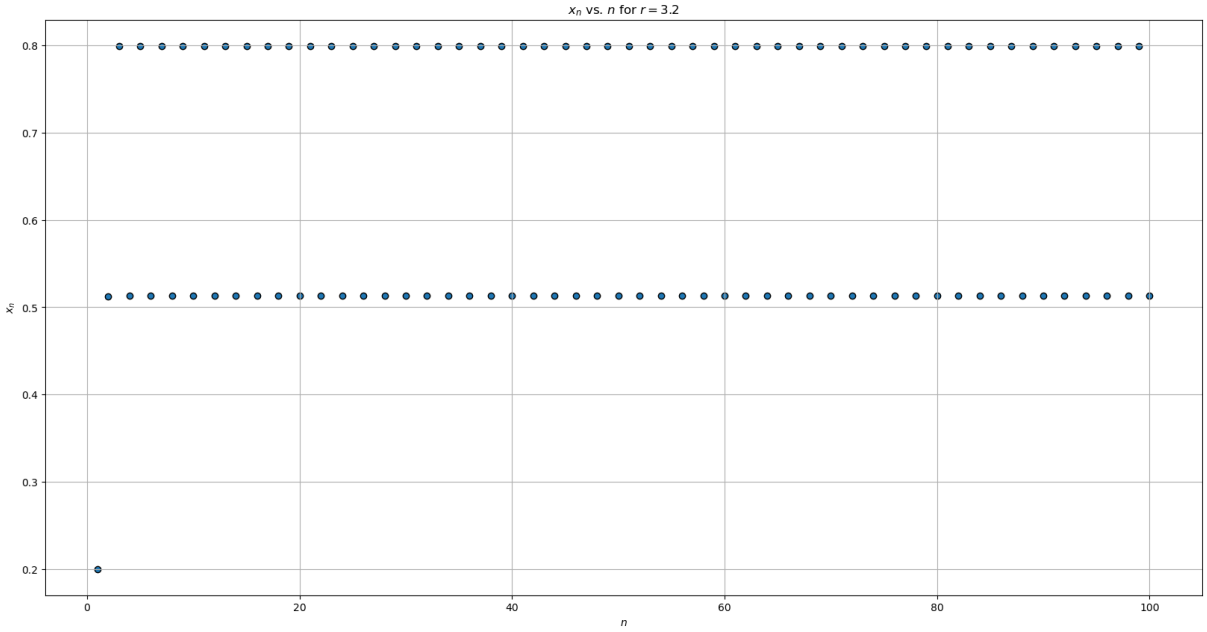



Figure 6: Serial 6.

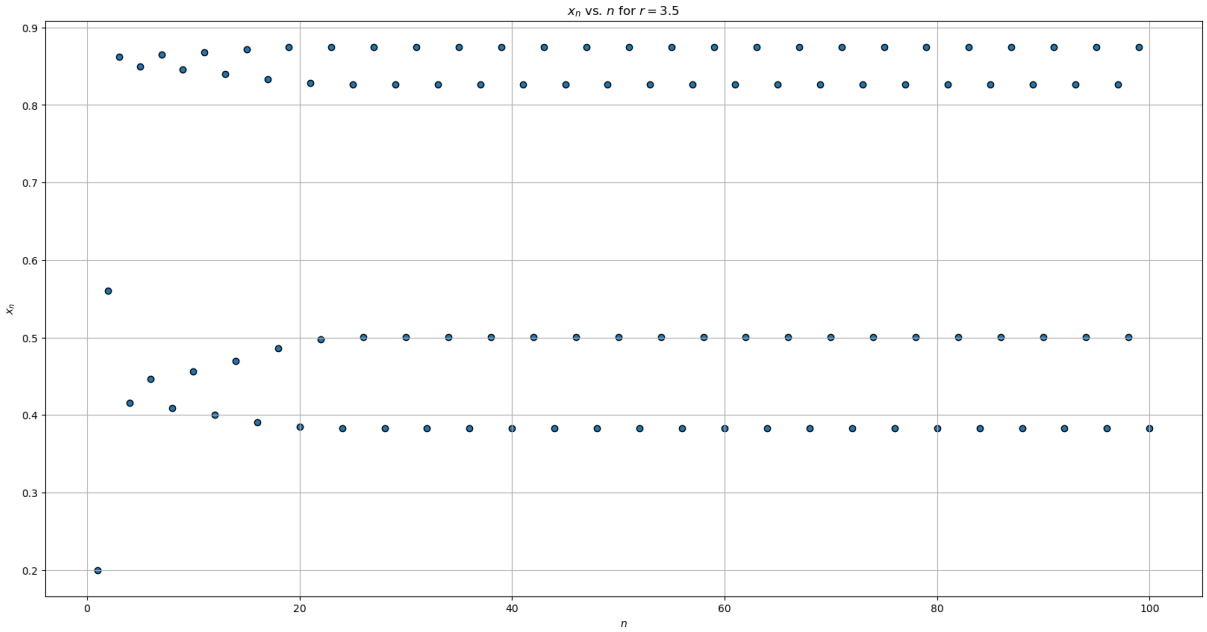



Figure 7: Serial 7.

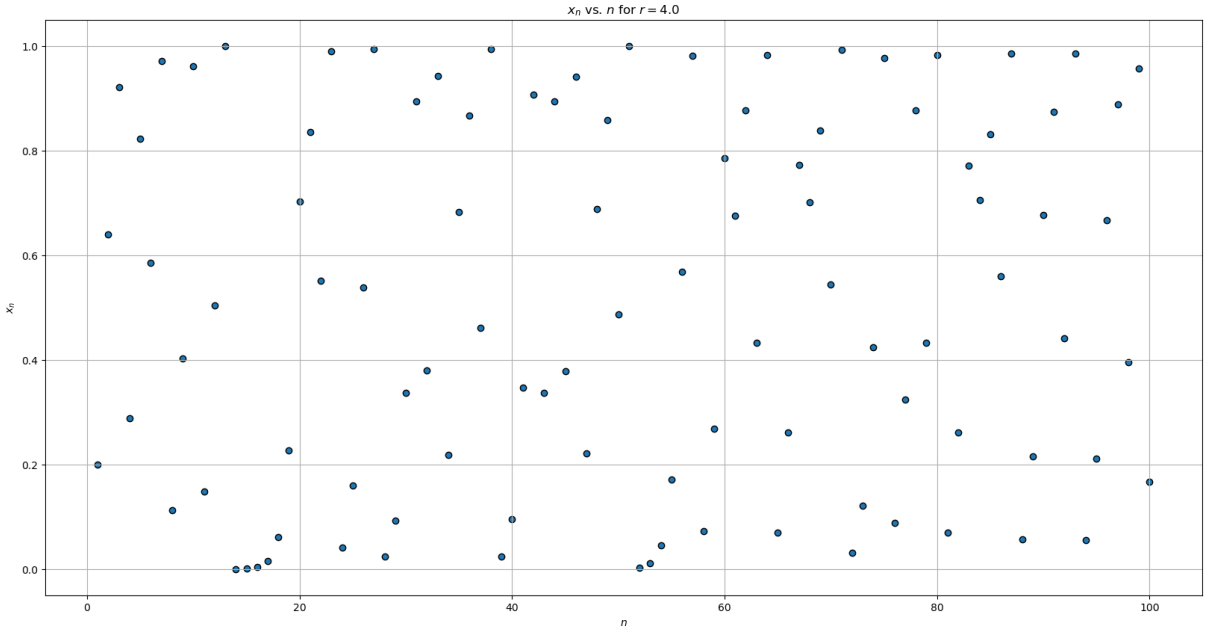



Figure 8: Serial 8.

# 3 Results

## 3.1 Mean Squared Error Measures

| Model | MSE |
|---|---|
| 1st iterates of Logistic Map | 0.08851087093353271 |
| 2nd iterates | 546.01953125 |
| 3rd iterates | 1750122758144.0 |
| Trajectory for $r = 2.9$ | 0.0105267483741045 |
| $r = 3.0$ | 0.0009091303800232708 |
| $r = 3.2$ | 0.03018713928759098 |
| $r = 3.5$ | 0.05465976521372795 |
| $r = 4.0$ | 0.12474610656499863 |

Table 2: Mean Squared Errors for Different Models.

## 3.2 Plots of Correlation between Outputs of the Iterates vs. Predictions of the NN

Plots of outputs of iterates and predictions of the respective Neural Network on the same points (always from the unseen data) are plotted for the first, second, and the third iterates of the Logistic Map. The plots follow.

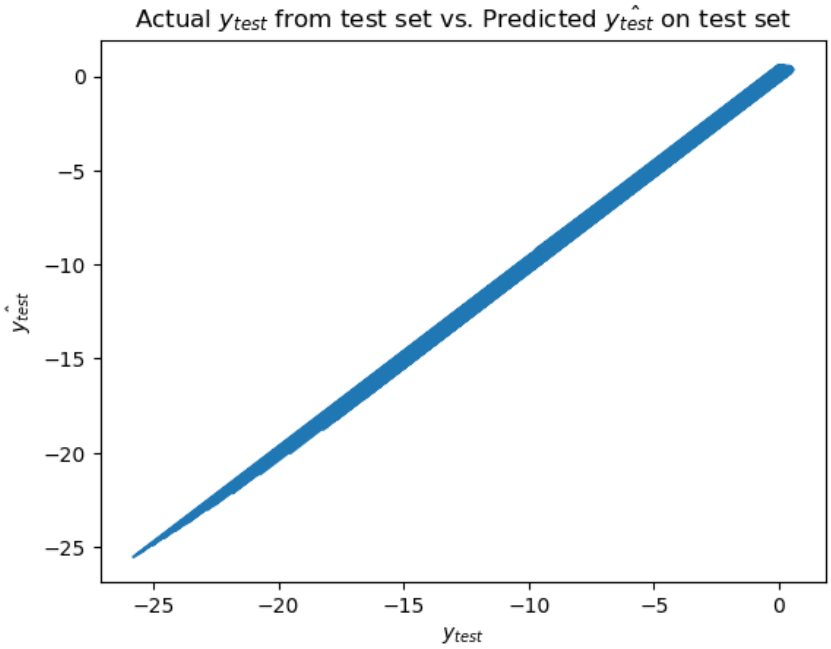



Figure 9: Serial 1.

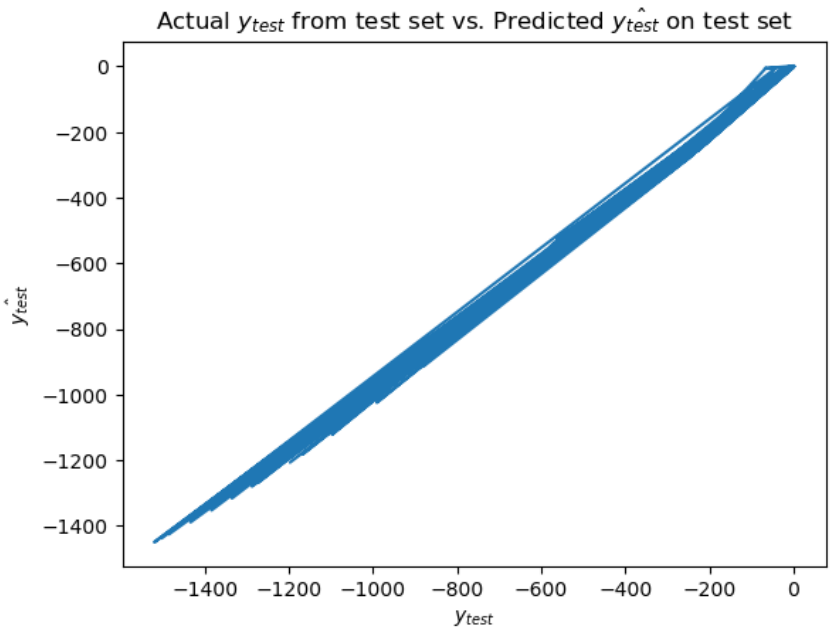



Figure 10: Serial 2.

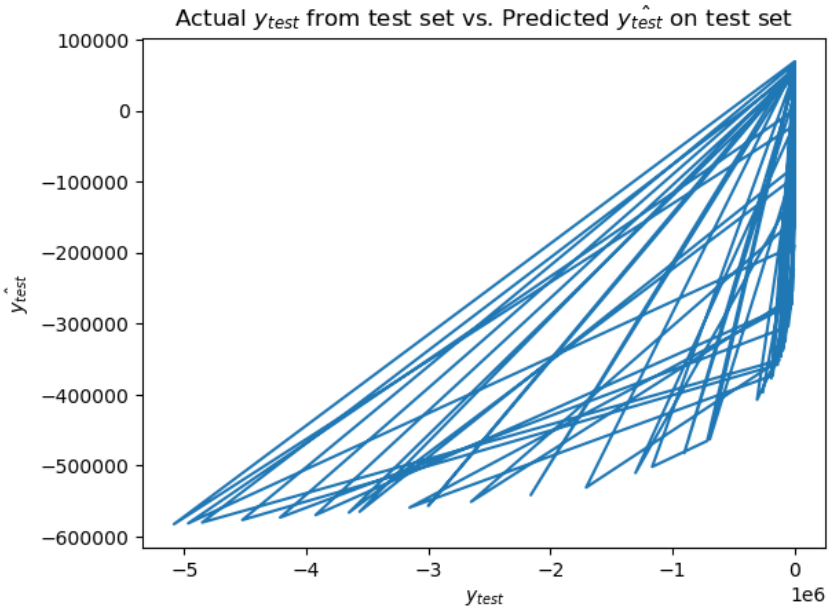



Figure 11: Serial 3.

### 3.3 Plots of The NNs' Predicted Outputs vs. True Outputs

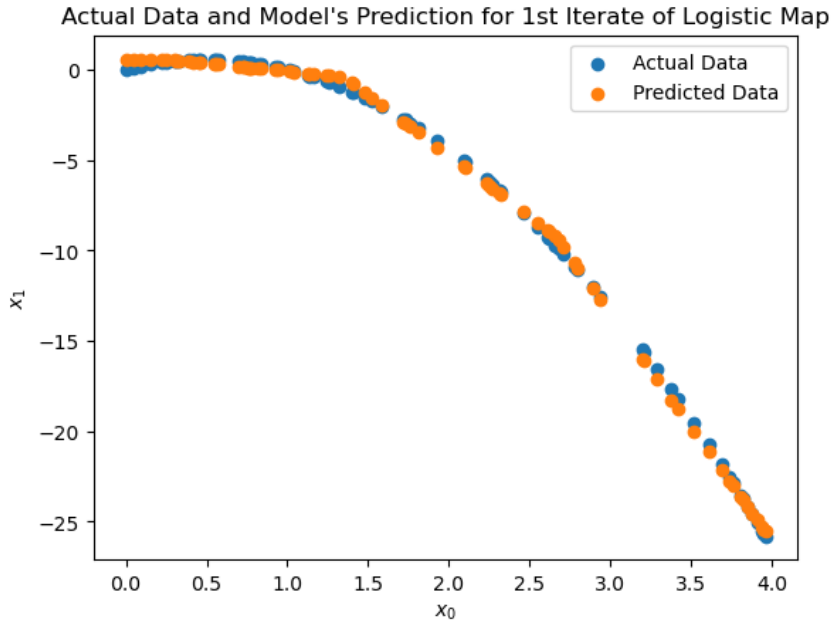



Figure 12: Serial 1.

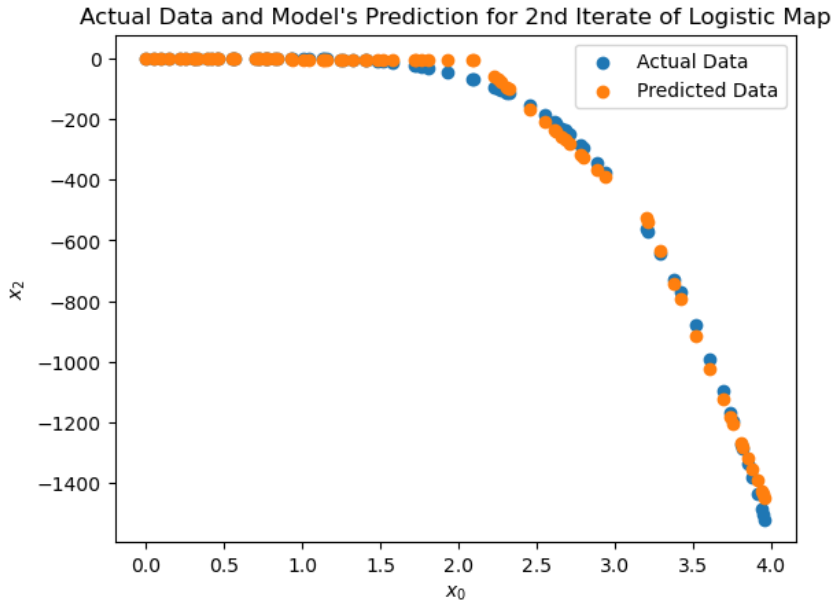



Figure 13: Serial 2.

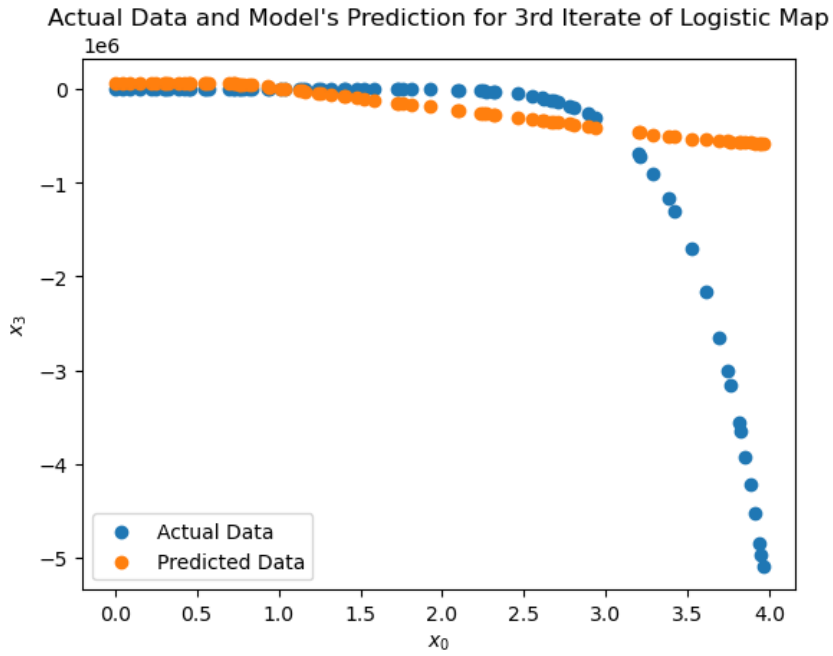



Figure 14: Serial 3.

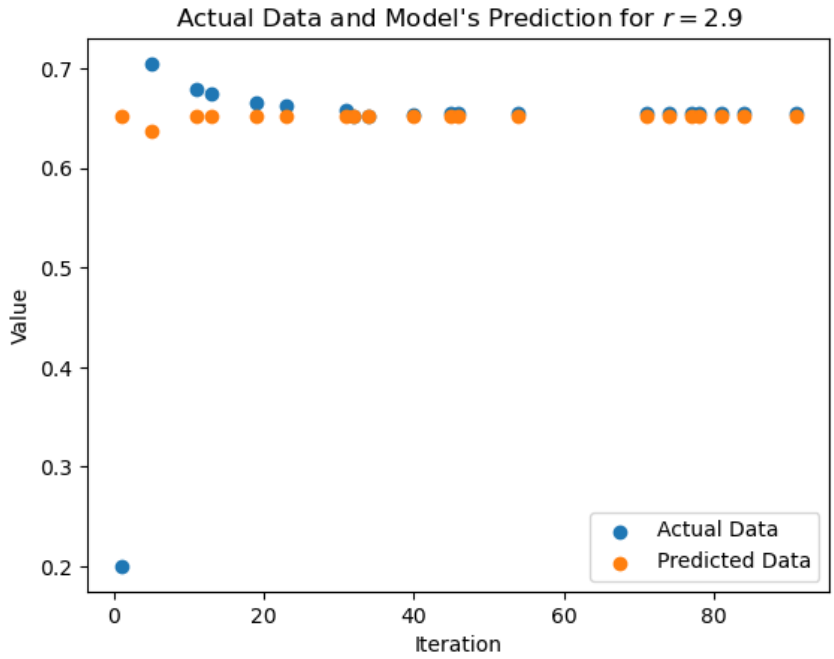



Figure 15: Serial 4.

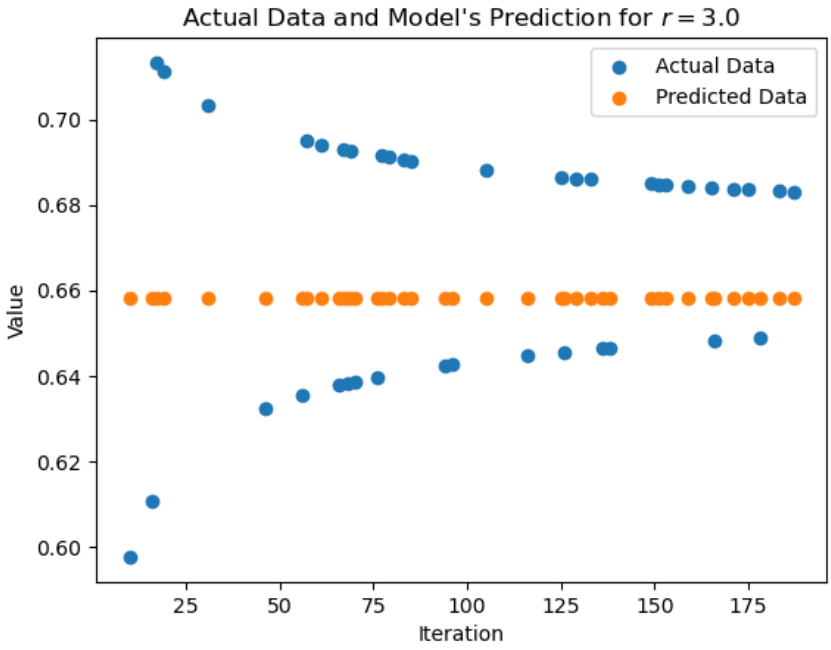



Figure 16: Serial 5.

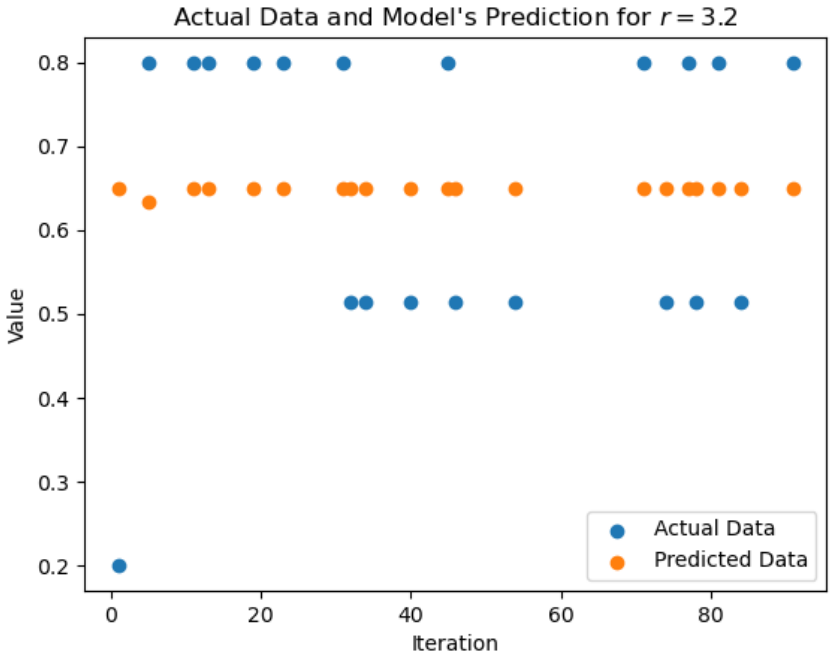



Figure 17: Serial 6.

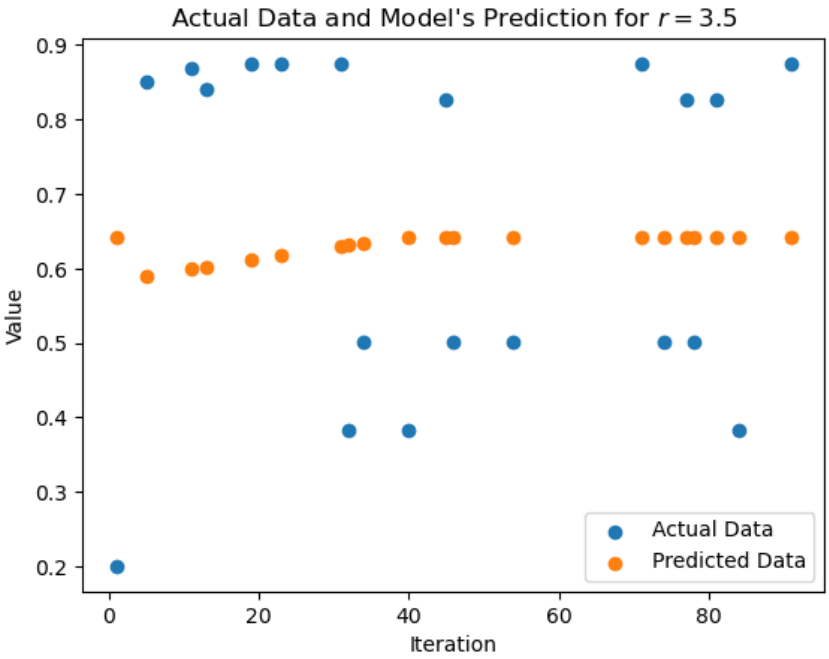



Figure 18: Serial 7.

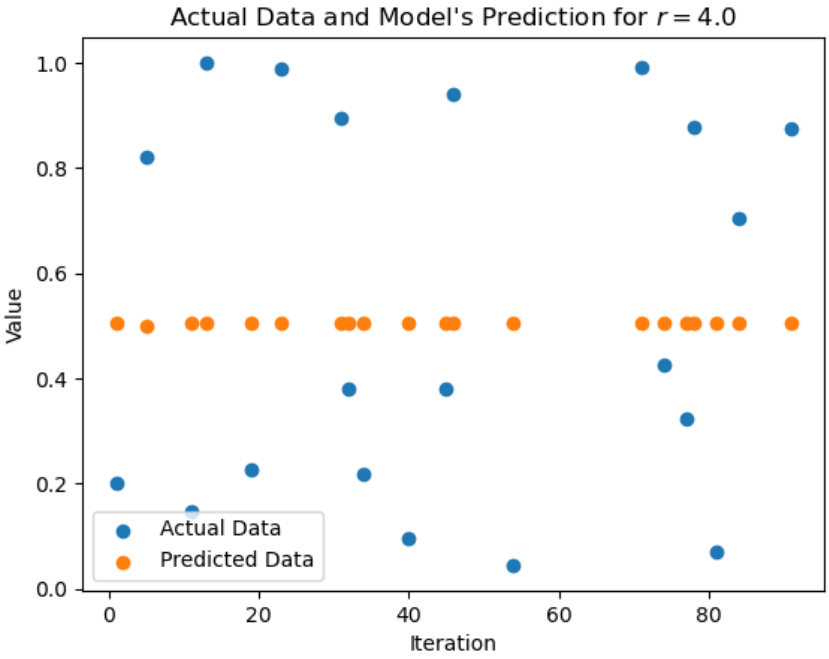
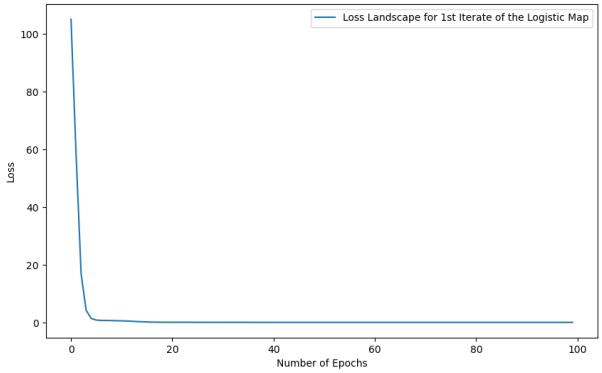
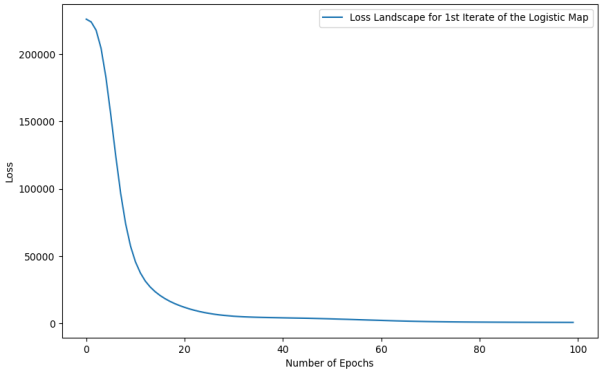



Figure 19: Serial 8.

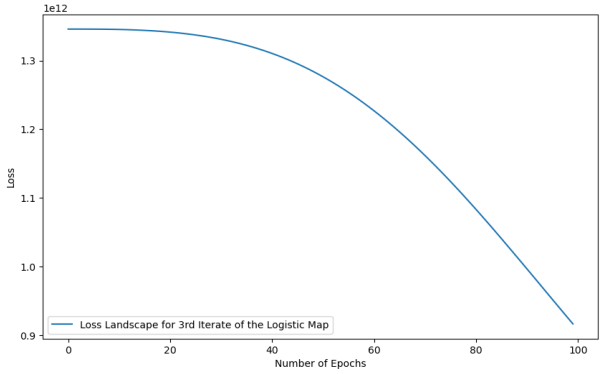
## 3.4 Plots of the Loss Landscape of The Neural Network Models



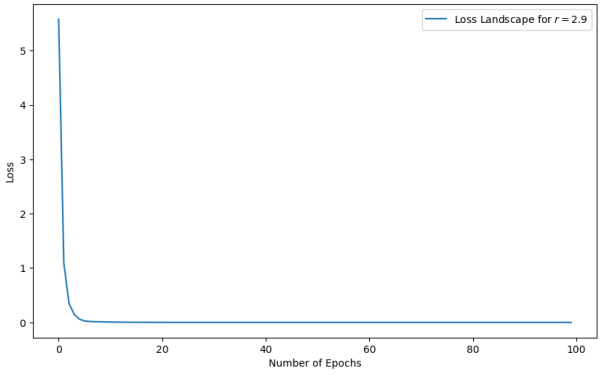

(a) Serial 1



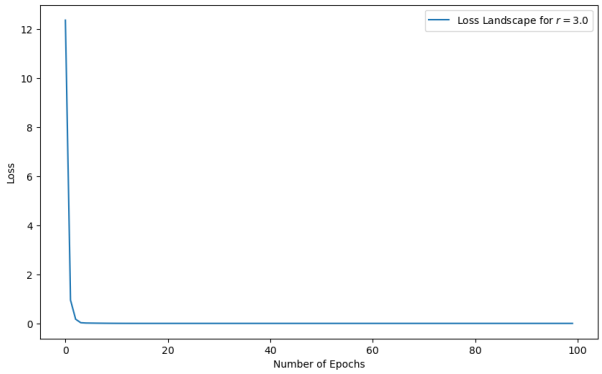

(b) Serial 2



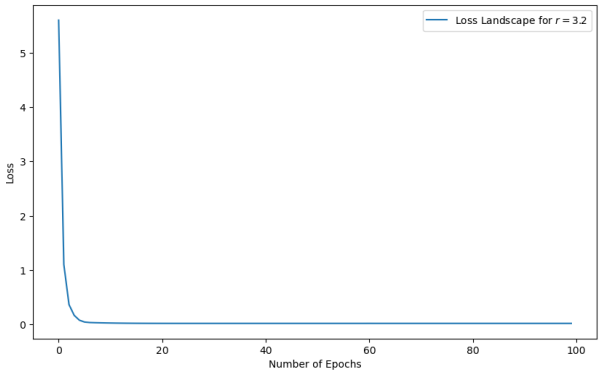

(c) Serial 3



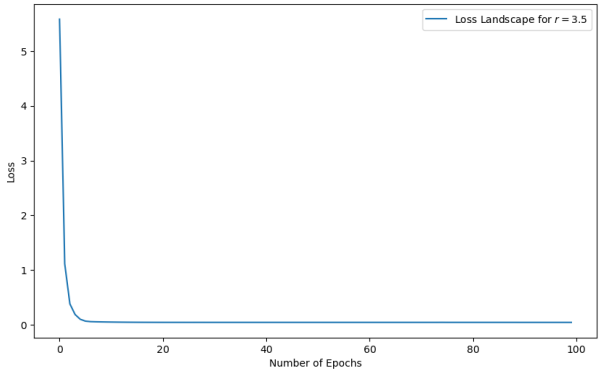

(d) Serial 4



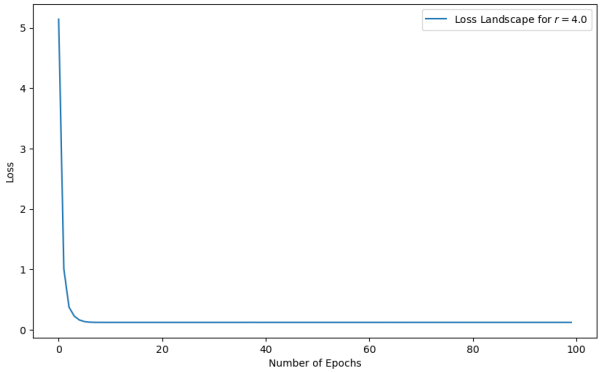

(e) Serial 5




(f) Serial 6




(g) Serial 7




(h) Serial 8

Figure 20: Loss Landscape of the Training of the Neural Networks

# 4 Observation and Future Directions

## 4.1 Observations

- The Neural Networks, in each case, quickly train and, in each case, the Loss goes down very quickly.
- The chosen metric, which, in this case, is the same as the Loss function, is giving desired results with the 1st and 2nd iterates, but not so with the third iterates (all when tested on the unseen data).
- The Loss goes very low for all the trajectories, but predictions by the Neural Network does not *follow* the trajectories.
- for the 1st, 2nd iterates, the correlations between the actual $y$s from the test set, and the predictions by the Neural Networks $\hat{y}$s for the same set of $x$s are very high, but for the 3rd iterate, it is not the case.
- For the 1st and 2nd iterates, the tail of the data is not sparse, and are packed with points, but for the 3rd iterate, the data points are sparse. So, the NN was unable to fit the curve properly.
- For the trajectories, even though the predictions by the Neural Networks do not follow the trajectory, the metric shows really good results.
- The above observation raises the question that whether for real-world scenarios, path predicted by a Neural Network would be the optimal to take under dynamics that show the properties of 2-cycle, 4-cycle, 8-cycle, or even chaotic.
- The Loss function, MSE, is not very apt if the desire is to model the trajectories of 2-cycle, 4-cycle, or chaotic dynamical systems. But it is, as expected, good at being close to the values of the trajectories.
- The Neural Network training, i.e. the update of weights, is driven by the Loss function- which the Neural Network tries to minimize. While the Loss function chosen here- the Mean Squared Error loss- is apt for some of the iterates, but not for all, and most trajectories.

## 4.2 Future Directions

- A Neural Network with different architecture- different number of layers, different number of neurons for each layer should be tried to see if NNs with different architecture perform better on this task.
- Several hyperparameters such as the learning rate, and the choice of optimizers, as well as number of epochs the NNs are trained for should be altered and studied to see if performance improves.
- **The choice of Loss function** is very significant while working with such data. Different Loss functions, especially some designed for this specific purpose should be used and studied to check if the NNs predict better points that closely learn the trajectory, and not merely the value. This is an interesting direction of research.
- While trying to learn the trajectories, although the Neural Networks failed to learn the trajectories, the metric, MSE, gave very satisfying results. So, it is seen that NNs are successful to learn the average *values* of the outputs. So, **it demands further research that whether we can use NNs in the real world scenarios** where we expect Non-Linear Dynamical behaviour, as it will be presumably very good for predicting the values, even if it fails to predict the trajectories. Datasets with real world data where similar dynamics were recognized- should be studied and used to train Neural Networks to see if they can predict useful values- as compared to ground truth as seen in the real-world.
- Different metrics should also be tried to test performance.

# 5 Conclusion

This study was a very successful initial study to test the viability of using Artificial Neural Networks to learn and predict the dynamics of a system. The insights gained from this study will be instrumental in future studies in this line. Although Artificial Neural Networks could not predict different cycles out-of-the box, investigation into a newer loss function seems like a positive direction. Although it cannot be

said with certainty that under chaotic dynamics in the real world, sometimes it would be the best policy to stick to the predictions of an NN, but the experiments strongly point at that direction. It also points towards two very interesting directions for future research.

# 6 Resources and Acknowledgements

## 6.1 Resources

- Bradley, Liz: Nonlinear Dynamics: Mathematical and Computational Approaches via Complexity Explorer, Santa Fe Institute
- Ghosh, Ritobrata: Learning Logistic Map with Deep Neural Networks - the official GitHub code repository for this paper.
- Ghosh Ritobrata: Non Linear Dynamics Trajectories with Python, via Kaggle
- Gleick, J. (2008). Chaos: Making a new science. Penguin.
- Strogatz, Steven: MAE5790: Non-Linear Dynamics and Chaos, Cornell University, via YouTube
- The PyTorch website
- Zubin, Gleb: Solving sklearn datasets with PyTorch, via Kaggle

## 6.2 Acknowledgements


- I would like to acknowledge the role of **Mr. Rajesh Nag**, the guide of this paper for the guidance and freedom he has given me. I would like to further thank all the teaching staff of M.Sc.(CS) stream of Techno College Hooghly, and all the non-teaching staff of the college for providing me with the environment required to undertake this project.
- I would like to thank my family and friends for providing support while working on this project.


# References


[ana20] Anaconda software distribution, 2020.

[CB06] Jeffrey A Cramer and Karl S Booksh. Chaos theory in chemistry and chemometrics: a review. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 20(11-12):447–454, 2006.

[Gho23a] Ritobrata Ghosh. Learning Logistic Map with Deep Neural Networks, 4 2023.

[Gho23b] Ritobrata Ghosh. Non linear dynamics trajectories with python, feb 2023.

[HM11] Andrew G Haldane and Robert M May. Systemic risk in banking ecosystems. *Nature*, 469(7330):351–355, 2011.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[Lor63] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[May88] Robert M May. How many species are there on earth? *Science*, 241(4872):1441–1449, 1988.

[MO76] Robert M May and George F Oster. Bifurcations and dynamic complexity in simple ecological models. *The American Naturalist*, 110(974):573–599, 1976.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[RLM22] Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing, Birmingham, UK, 2022.

[SS17] Christos H Skiadas and Charilaos Skiadas. *Handbook of applications of chaos theory*. crc Press, 2017.

[ZLLS21] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.