Comparative Analysis of Traditional Recurrent Neural Networks and Reservoir Computing Models for Predicting Complex Ball Trajectories in a 2D Environment

Minerva University

Nikita Koloskov
April 2023

Complimentary Github Repository: https://github.com/Carterochka/capstone

Title: Comparative Analysis of Traditional Recurrent Neural Networks and Reservoir Computing Models for Predicting Complex Ball Trajectories in a 2D Environment.
Author: Nikita Koloskov (Minerva University, Class of 2023).

Problem & Background:
I am exploring the use of recurrent neural networks (RNNs) to learn underlying physics in classical mechanics scenarios, specifically predicting the trajectories of single ball objects in 2D physics simulations. I am using the Phyre simulator to generate data and isolate the intuitive physics problem from contextual inference. My research is motivated by potential uses in various fields, such as robotics, the military, neuroscience, and general AI. My capstone discusses how RNNs can be leveraged to understand the underlying physics of classical mechanics scenarios by attempting to learn Newton's Laws of Motion and the Law of Conservation of Momentum. I am using the analysis of how well RNNs can predict trajectories as a proxy for speculating how accurately the models learn these laws of Physics.

Approaches:
I am using a bottom-up research approach. I started with using simple Feedforward Neural Networks to predict one time-step at a time in free-fall scenarios, using input that allows solving the same problem analytically. I am then transitioning to predicting the whole free-fall and bounce in a single-dimension scenario by using only the initial state as an input. Step by step I am increasing the complexity of the problem: by including the second coordinate, then including two other balls to the input, transitioning to scenarios with falls and collisions, and finally predicting the evolution of the entire scene with three balls. I am comparing the performance of Recurrent Neural Networks models (among which are Vanilla RNN, GRU, and LSTM) to that of Reservoir Computing (RC) models (among which are Echo State Networks and their deep variations) on each of the tasks mentioned above.

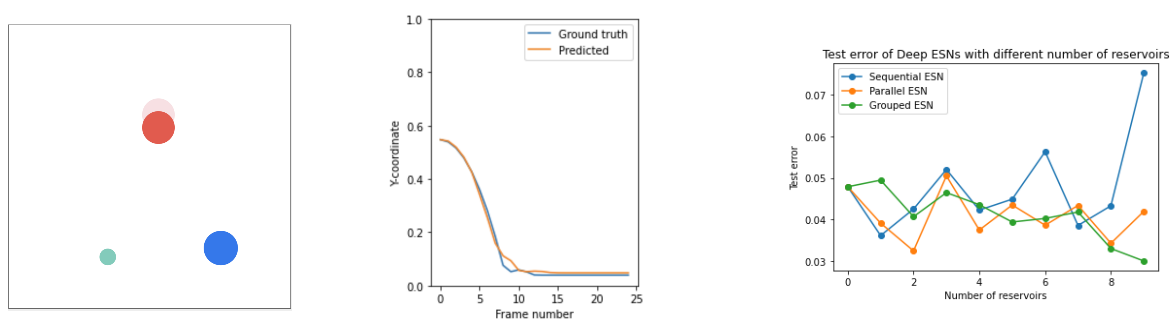


Figure 1. (left) Visual output from Phyre physics simulator; (middle) Predicted and ground truth red ball's Y-coordinate time-series in free-fall; (right) Test set performance of Deep ESN models as a function of the number of reservoirs.

Outcomes:
Traditional RNN architectures demonstrated outperformed RC models on each of the explored tasks. All models demonstrated a solid ability to predict the trajectory of the ball in the scenarios where it free-falls and bounces from the ground, but struggled to provide reasonable predictions in scenarios where it collided with other balls. Arguably, RC models demonstrate a better grasp of the underlying physics while struggling with the precision of predicted values. It especially manifests in the shape of predicted trajectories for the free-fall scenarios: RC models correctly predicted the fall strictly along a straight line, as well as the bounce once the ball hits the ground. In contrast, traditional RNNs do not grasp such aspects, but the free-fall trajectories are predicted with a much smaller error. For scenarios with collisions, both types of models fail to accurately predict the trajectories. RC models tend to fail to correctly identify collisions but somewhat reasonably predict the movement after collisions that are identified incorrectly. Traditional RNNs, in contrast, tend to fail to identify the movement correctly but are doing a better job minimizing the overall error.

Table of contents

Section 1. Introduction

In this project, I investigate the potential of recurrent neural networks (RNNs) in learning the underlying principles of classical mechanics, specifically focusing on trajectory predictions in two-dimensional (2D) physics simulations. My research is centered on predicting the trajectories of single ball objects within scenes containing no more than three ball objects, allowing for a more in-depth exploration of the subject. To generate the data, I utilize the Phyre simulator, an unlimited data source that enables me to isolate the intuitive physics problem from contextual inference.

My research is motivated by numerous potential applications across various disciplines, such as robotics, military, neuroscience, and general artificial intelligence (AI). In the field of robotics, understanding the physical dynamics of objects and predicting their trajectories can enhance the analysis of physical interventions' impacts. Similarly, in military applications, the ability to predict artillery or missile parts' trajectories post-impact could prove invaluable for devising defense tactics. Furthermore, modeling the human brain's perception of object dynamics could benefit neuroscience research and general AI tasks, including the modeling of environmental perception.

In this paper, I will discuss the implications of my research and the potential of leveraging neural networks to comprehend the underlying physics of classical mechanics scenarios. Accurate trajectory prediction requires the proper application of Newton's Laws of Motion and the Law of Conservation of Momentum. Analyzing the RNNs' performance in predicting object trajectories can serve as a proxy for determining the extent to which these models learn and apply the aforementioned laws of Physics.

Section 2. Previous Work

In this research paper, I investigate the application of Recurrent Neural Network (RNN) and Echo State Network (ESN) architectures in predicting object trajectories within simulated classical mechanics scenarios. This study builds upon prior work exploring deep learning techniques in image understanding, trajectory, and destination prediction.

Mottaghi et al. (2015) proposed a novel framework for Newtonian image understanding, analyzing static images by modeling the dynamics of objects within them. Their approach employed a Convolutional Neural Network (CNN) to learn object features and a physics-based model to predict each object's future state. Despite the model's efficacy in predicting object trajectories in images, it was limited by the reliance on only 12 modeled trajectories and did not consider object interactions beyond interactions with the ground.

In a subsequent study, Mottaghi et al. (2016) explored predicting the impact of external forces on objects in images. Their framework combined deep learning with physics-based models, learning a representation of the object and force before predicting the object's future state post-force application. The authors showed that their approach achieved state-of-the-art results on several datasets, including real-world images. However, their trajectory prediction was limited to the binary prediction about whether the object can move along a specified direction and by how much.

In Song et al.'s (2020) work on destination prediction, the authors proposed using a Deep Echo State Network (DeepESN) to predict the destination of an object based on its trajectory. DeepESN is a variant of the popular Echo State Network (ESN) that is capable of learning complex temporal dynamics. The authors showed that their model outperformed several baselines on a dataset of taxi trajectories, demonstrating its potential for applications such as traffic prediction. The moving space of the considered problem, however, is a city rather than a classical mechanics environment. So instead of being a problem of intuitive physics, it is a problem of modeling human decisions.

Finally, Wu et al. (2017) proposed a recurrent neural network (RNN) based approach for modeling trajectories. Their method uses a variant of RNN called Long Short-Term Memory

(LSTM) to capture long-term dependencies in the trajectory data. The authors evaluated their model on several datasets, including human motion tracking and vehicle navigation, and showed that it outperformed several previous methods. Despite outperforming previous methods in human motion tracking and vehicle navigation, however, their focus remained on modeling human behavior rather than intuitive physics.

In summary, previous works demonstrate deep learning techniques' potential in various tasks related to image understanding, trajectory, and destination prediction. These methods have proven accurate and applicable in real-world scenarios, such as traffic prediction and human motion tracking. My research aims to explore the capabilities of recurrent neural networks in predicting trajectories within deterministic physics-based scenarios.

Section 3. Data Preparation

To generate data for this study, I am utilizing the simulator provided by the Phyre benchmark package (Bakhtin et al., 2019). I am adapting the generated data to a format compatible with the PyTorch Deep Learning framework (Paszke et al., 2019).

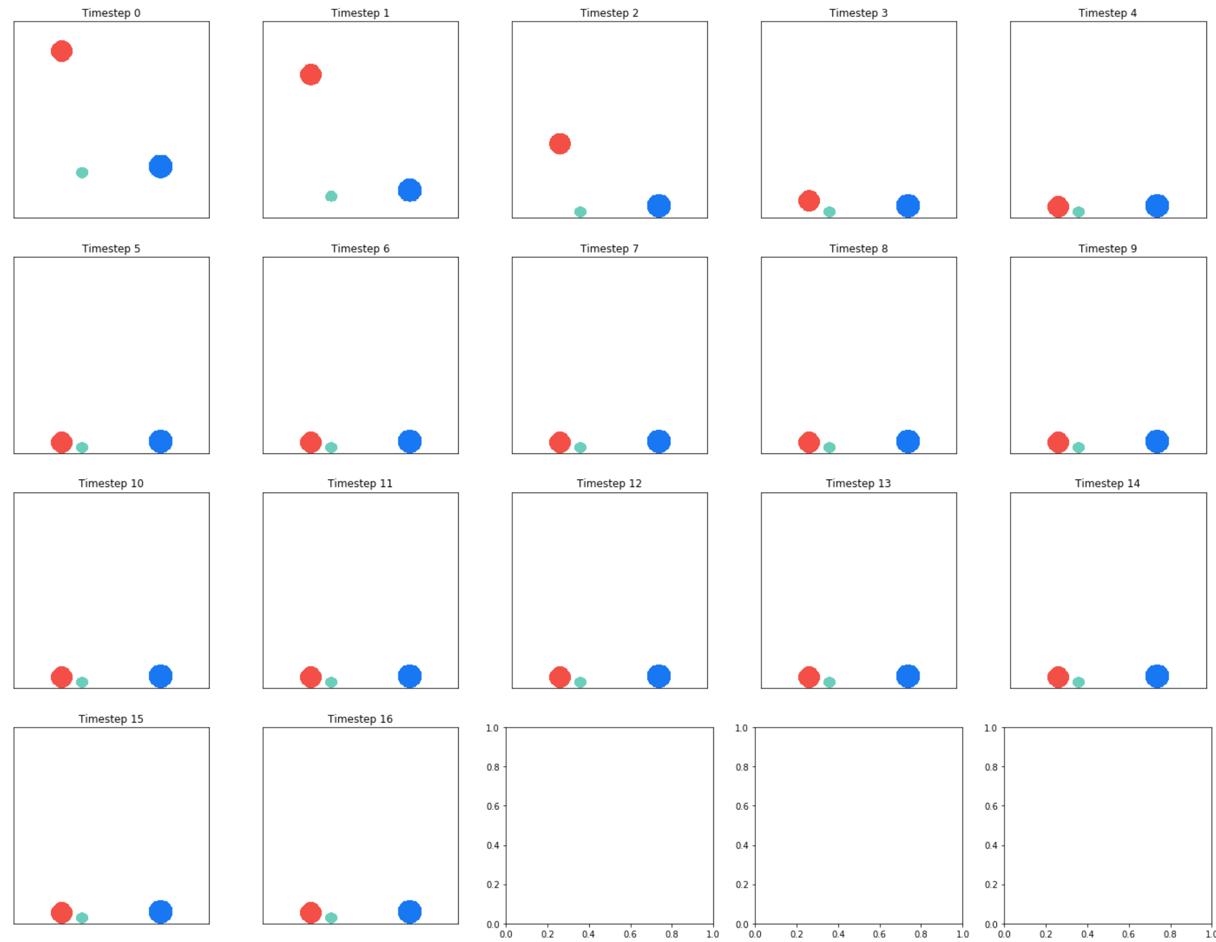
3.1. Overview of the simulator

The Phyre benchmark (Bakhtin et al., 2019) offers a two-dimensional simulated environment with Newtonian physics and a set of tasks for evaluating Intuitive Physics algorithms. Each task's initial state contains a varying number of geometric bodies, distinguishable by shape and color. Users can place one or two additional red balls in the scene, depending on the task tier, with customizable location and radius. Phyre can then simulate the scenario according to programmed physics.

Upon completing the simulation, Phyre provides two output types: visual (a set of graphical data representing the simulation state at each step) and numerical (a tensor containing information about each object in the scene at every simulation step). In the numerical output, each object is characterized by 14 values as described in Bakhtin et al. (2019) supporting Jupyter Notebook:

- x- and y-coordinates of the center of mass in pixels, divided by scene width and height, respectively;
- Object orientation (angle) in radians, divided by 2π;
- Object diameter in pixels, divided by scene width;
- One-hot encoding of object shape, in the order of "ball," "bar," "jar," "standing sticks";
- One-hot encoding of object color, in the order of "red," "green," "blue," "purple," "gray," "black."

An example of the visual output is provided in Figure 1 (a), and an example of the numerical output for the initial state of the scenario is shown in Figure 1 (b).



(a)

```
Initial featurized objects shape=(1, 2, 14) dtype=float32
[[[0.35  0.229 0.    0.059 1.    0.    0.    0.    0.    1.    0.
   0.    0.    0.   ]
  [0.75  0.261 0.    0.121 1.    0.    0.    0.    0.    0.    1.
   0.    0.    0.   ]]]
```

(b)

Figure 1. Graphical output of the sample simulation (a) and numerical output for the initial state (b) of the Phyre simulator. Retrieved from https://github.com/facebookresearch/phyre/blob/main/examples/01_phyre_intro.ipynb

Phyre provides a set of templates containing objects with different topologies. Each template contains a set of similar tasks with objects arranged differently in the initial state. While using the simulator, users have the ability to choose a template, task, action (where to put the ball(s) and how big to make them), and the simulation stride (how big is the gap between two simulation steps).

3.2. Preparation of the datasets

For my research project, I am focusing on scenarios with three balls, aiming to predict the trajectory of the red ball. I will use the simulator's numerical output, which allows me to separate the problem of learning physics from the problem of inferring information from visual data. The only information I will need to retrieve from the simulator is the x- and y-coordinates of each ball and their diameters. I will also use the visual output from the simulator, but solely for visualization purposes.

To generate data that only contains three balls in each scene, I am restricting my dataset to the Phyre template with the code name `'00000'`, tasks from which can be seen in the demo playground that supports Bakhtin et. al. (2019) paper. For each task in this template, I am randomly sampling actions – triplets of x- and y-coordinates and the diameter of the red ball. I use the simulator's functionality to filter out invalid simulations, such as when parts of the red ball fall outside the simulator space or overlap with other balls. The remaining simulations are further processed for each specific type of task explored in my research.

To make my data easily usable with the PyTorch framework, I need to wrap it using the PyTorch `Dataset` class. I created an abstract `ClassicalMechanicsDataset` class that extends the abstract PyTorch `Dataset`. This abstract class serves as an umbrella data class for all my research, and provides the following functionality:

- It defines an abstract method `generate_data()` that must be implemented by the child classes. In child classes, this method invokes the Phyre simulator for simulated scenarios, each of which will be stored in a separate file.

- This class implements a static function `train_test_split(path, test_frac)` that takes the path to raw data files and test fraction as input, and outputs train and test dataset objects. This method can create dataset objects of child classes, so it doesn't need to be reimplemented during inheritance.

- It implements the `__len__(self)` method that is required by the abstract `Dataset` class by counting the number of stored data files corresponding to the given `ClassicalMechanicsDataset` instance.

For each research task that I have in the process, I extend `ClassicalMechanicsDataset` and implement the `generate_data()` method, as well as the `__getitem__(self, idx)` method required by the abstract PyTorch `Dataset` class.

Section 4. Free-fall analysis

I begin my research by exploring how machine learning models can predict the simplest type of trajectories, such as the trajectory of an object in free-fall. While the free-fall motion can be analytically calculated using kinematic equations and does not require machine learning techniques, I introduce additional complexity by focusing on the motion of a free-falling ball that hits the ground and bounces. This task is more challenging because such a trajectory cannot be described with a single mathematical equation. Furthermore, as illustrated in Figure 2, the bounce is not fully elastic, introducing a hidden physics parameter that machine learning models will need to infer: the coefficient of restitution.

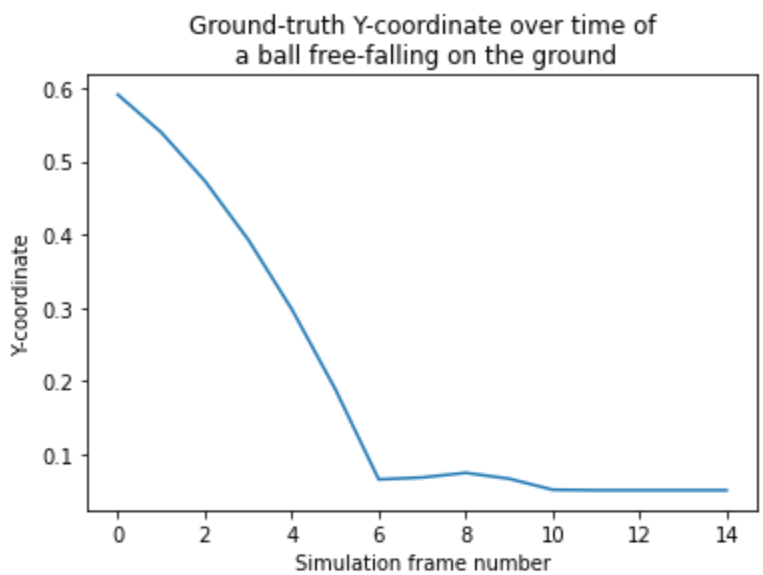


Figure 2. Simulated Y-coordinate of a ball free-falling on the ground. In case of a completely elastic collision, the ball would bounce to the initial height level.

4.1. Experiments with Feedforward Neural Networks

I begin with a naïve approach, using a feedforward neural network (FNN) as the first model to investigate patterns in the generated data. This choice is motivated by two main considerations: (1) the ability of FNNs to identify complex patterns in data, and (2) their relative ease of implementation. However, this approach is considered naïve because FNNs lack the ability to model temporal dependencies, which are essential for predicting trajectories in time series data. As a result, FNNs may struggle to capture the dynamics of the ball's motion accurately. Despite this limitation, an analysis of the results obtained with FNNs could potentially guide further research directions and serve as a reference point to compare the performance of more complex models that can better handle time-dependent data.

4.1.1. Predicting one step at a time

While I increased the complexity of the problem by adding the bounce to the ball's trajectory, I decided to start by exploring how well linear networks can learn simple kinematic equations.

Movement of the object along a straight line can be described with the following equation:

$$\vec{r} = \vec{r}_0 + \vec{v}_0 \Delta t + \frac{1}{2}\vec{a}(\Delta t)^2 \qquad (1),$$

where $\vec{r}$ and $\vec{r}_0$ represent current and initial positions respectively, $\vec{v}_0$ is initial velocity, $\vec{a}$ is acceleration, and Δt is the time of movement. In the case of the free-fall due to gravity, when the movement is strictly vertical, assuming the direction of vertical Y-axis upwards and the direction of initial velocity downwards (which is the case for the free-fall if the movement doesn't start from rest), this equation can be rewritten as

$$y = y_0 - v_0 \Delta t - \frac{1}{2} g (\Delta t)^2 \qquad (2),$$

Equation 2 gives us the relationship between initial coordinate y_0, initial vertical velocity v_0, acceleration due to gravity g, and time step Δt. We can simplify this equation in the following way:

- Given that our simulation is discrete, Δt is defined to be one simulation frame. This simplification removes time dependency from equation 2.
- Let y_0 be the coordinate of the object on the current frame, and y be the coordinate of the object on the following frame. Then, v_0 is the velocity of the object on the current frame. In such framing, we treat each frame as initial for calculating the following frame.

- Vertical velocity is the first derivative of the y-coordinate. Given the discretized time, and assuming there was some movement preceding the current frame, it can be expressed using the equation of numerical differentiation:

  $$v_0 \approx \frac{y_0 - y_{-1}}{\Delta t} = y_0 - y_{-1}.$$

  This simplifies the dependency of the following coordinate on speed in equation 2 to the dependency on coordinates on two subsequent preceding frames.

- Acceleration is the first derivative of velocity. We can use the same trick as above to get:

  $$g \approx \frac{v_0 - v_{-1}}{\Delta t} = v_0 - v_{-1}.$$

  We already expressed v_0 in terms of two subsequent coordinates. In the same way,

  $$v_{-1} \approx y_{-1} - y_{-2},$$

  which brings us to

  $$g \approx y_0 - 2y_{-1} + y_{-2}$$

  This simplifies the dependency of the following coordinate on acceleration due gravity in equation 2 to the dependency on coordinates on three preceding frames.

So equation 2 simplifies to the equation of the form

$$y \approx f(y_0, y_{-1}, y_{-2}).$$

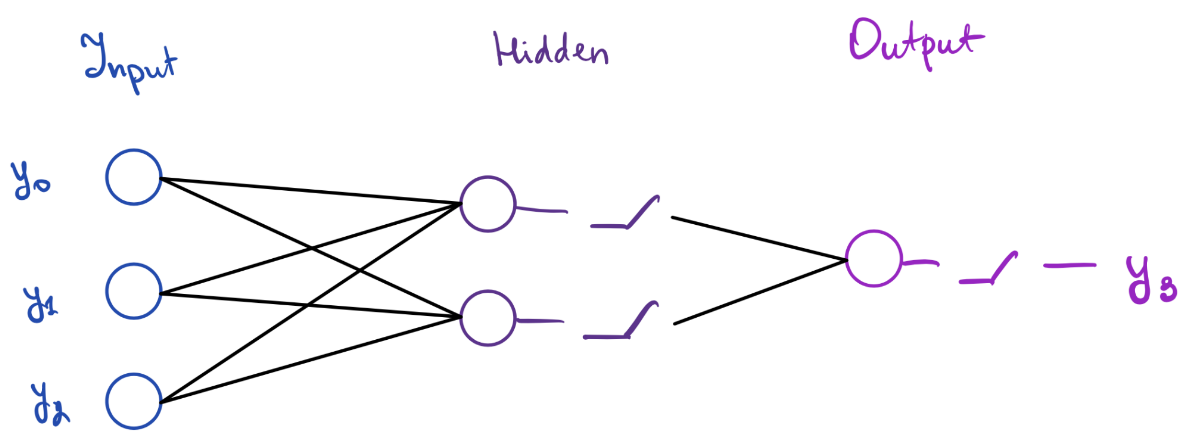
It means that knowing the coordinates on the three preceding frames is enough to predict the coordinate on the next frame. The purpose of the neural network will be to learn this function f. In fact, making the substitution of the equations above to 2 will give us the analytical form of f, so I expect the neural network to learn it without problems. As a result, I expect the linear network to make a close-to-perfect prediction of the next coordinate on a free-fall part of the ball movement and have a noticeable prediction error for the bounce part.

The neural network design can be inspired by the discussion above. The size of the input layer is 3, which corresponds to three subsequent coordinates. The hidden layer has 2 neurons, which are supposed to represent speed and acceleration. Finally, the dimensionality of the output is 1, which corresponds to the predicted following coordinate.

For this scenario, I generated a dataset of 5,360 simulations, 1,608 of which were used as a test set. To generate this dataset, I randomly sampled 10,000 actions[1]. Using Phyre functionality, I filtered the invalid simulations, and then implemented another filter that only leaves the simulations where the x-coordinate of the ball stays constant. Finally, I transformed each simulation into a set of 16 triplets of coordinates of a single free-falling ball, representing the position of the ball on the three consecutive simulation frames.

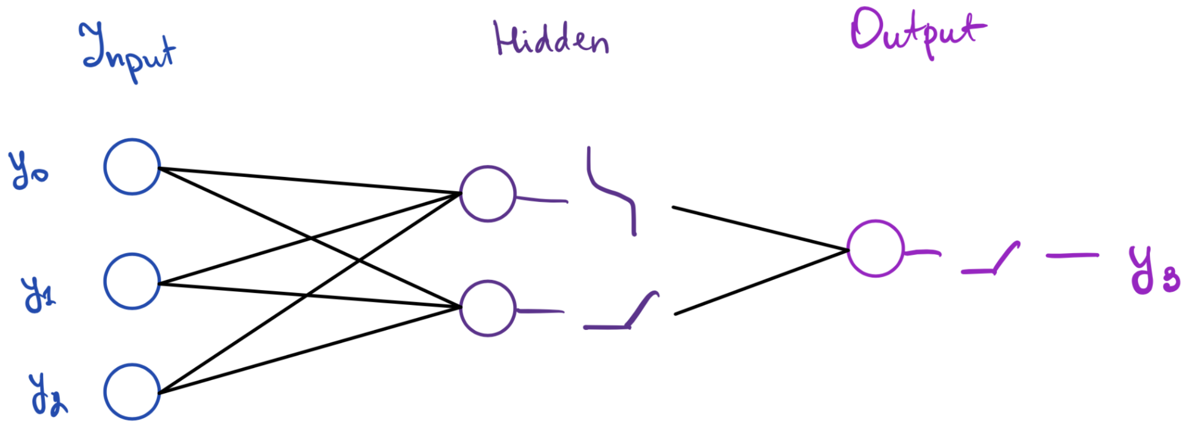
I implemented 3 different variations of this simple network that I trained and tested using the simulated data. The base model is a feedforward network with a single two-neuron hidden layer followed by the ReLU activation. The second model uses ReLU activation for one of the hidden neurons and cotangent activation for the second one to account for velocity being able to take different signs. Finally, the third one uses ReLU activations for both hidden neurons but also includes a direct connection from one of the input neurons to the output. Schematically, all three models are shown in Figure 3.

To quantify the model performance, I used Root Mean Squared Error (RMSE) loss.

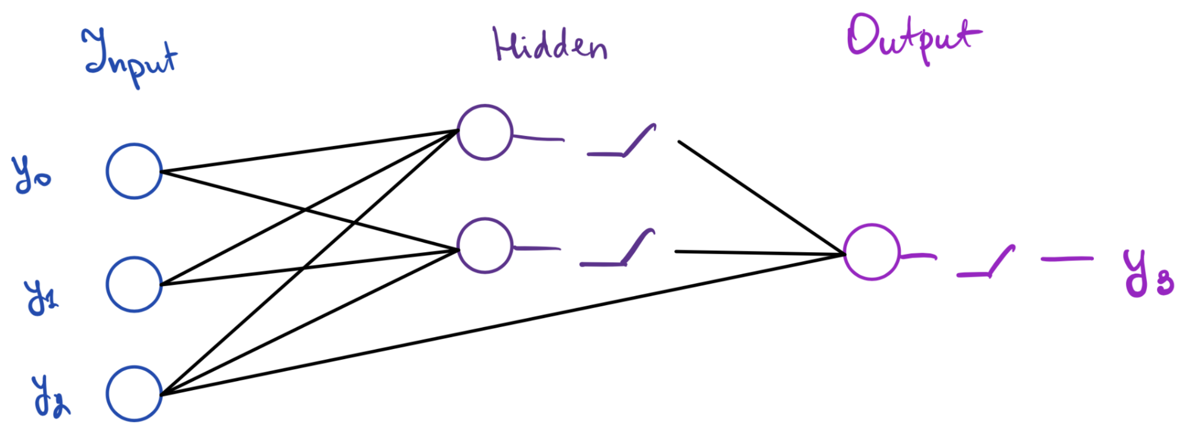


(a)

[1] Here, action is defined as a triplet of x- and y-coordinates and the diameter of the target ball.



(b)



(c)

Figure 3. Feedforward neural network designs for single-step prediction. (a) Single hidden layer design with ReLU activations; (b) single hidden layer design with ReLU and Cotan activations; (c) single hidden layer with ReLU activations and direct connection from one of the input neurons to output.

All three models demonstrate a similar level of performance, as shown in Table 1, with the second model slightly outperforming the other two. Given that all the numerical data that we get from the simulator is normalized between 0 and 1, we can interpret this RMSE value as an average percent error (relative to the scene size) in predicting a single coordinate value. The network design with ReLU and cotangent activation functions seems like the best choice out of these three models.

Model	RMSE
Dense with ReLU activations	0.01827
Dense with ReLU and Cotan activations	0.01155
Dense with a direct connection from input to output	0.01819

Table 1. Comparison of root mean square errors of each of the FNN types on one-dimensional single-frame free-fall prediction.

As expected, these models make nearly perfect predictions of coordinates during the free-fall stage of the ball's movement, while having a noticeable prediction error for the frames where the bounce is observed. It can visually be noticed in Figure 4 that demonstrates the predicted "trajectory map" of the first 10 simulations from the test set by the model with ReLU and cotangent activations. Here, by "trajectory map" I mean the target coordinates within the same simulation.

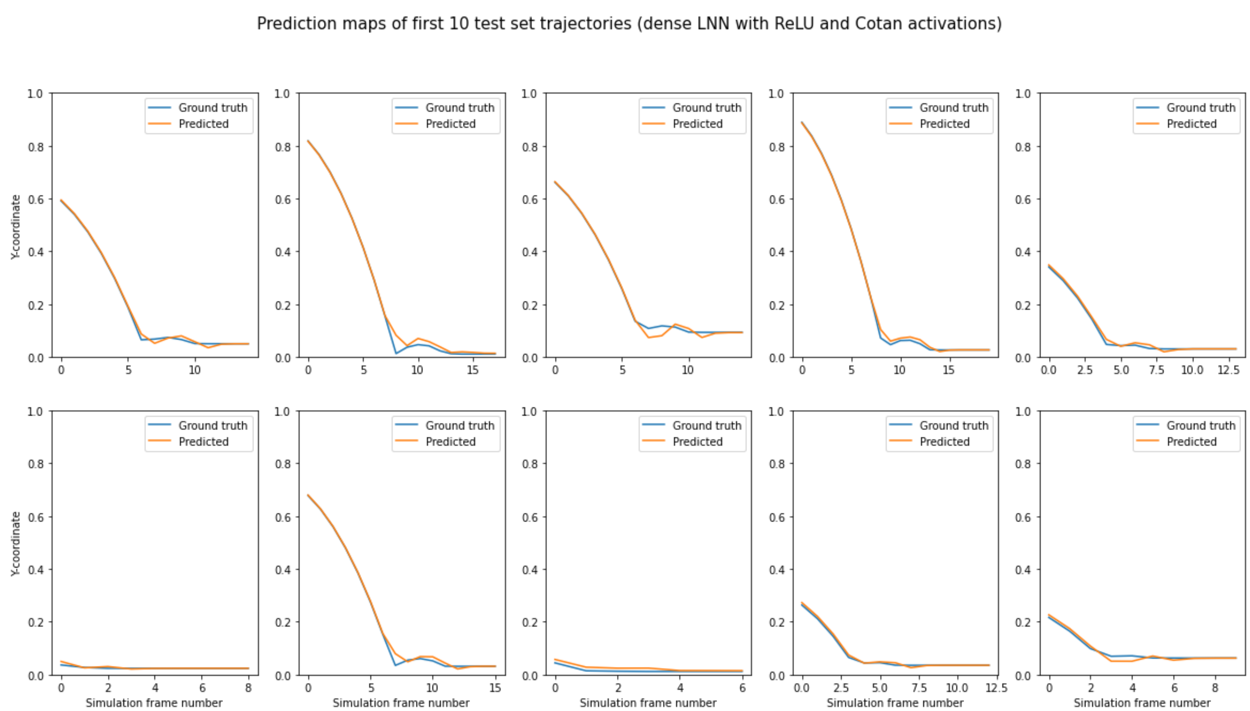


Figure 4. First 10 trajectory maps from the test set predicted by FNN with ReLU and Cotan activations.

4.1.2. Predicting the entire trajectory

While predicting the following coordinate from the three preceding is a good first research step, it doesn't have any practical value. Thus, I decided to increase the complexity of the task by bringing it closer to the goal problem - predicting the trajectory of the object from the initial state of the scene. My next step, therefore, is to train neural networks that using only the initial state will predict the entire trajectory of a free-falling ball that bounces from the ground.

I started with generating a simulated dataset, where each simulation consists of 25 frames. The y-coordinate of the ball on the first frame is separated as input to the network, while the sequence of the rest 24 coordinates is used as an output. I ended up with a dataset of 5,360 simulations, 1,072 out of which were used as a test set.

I wasn't sure if a single initial coordinate is enough information for predicting the entire trajectory. To resolve my concern, I decided to overfit my feedforward network to understand what patterns it learns from the train set, hoping that analysis of these patterns will indicate what extra data I need to include.

One of the reasons for overfitting is noise learning, which can be caused by having too large a network (Ying, 2019). I relied on this overfitting method by creating a large 5-layer linear network that takes a one-dimensional input, has four hidden layers with 256, 128, 64, and 32 neurons respectively, and outputs a vector of size 24.

After training such a model, I got an RMSE of 0.0414. The first 10 predicted trajectories with the ground truth are shown in Figure 5.

Predicted Y-coordinate time series by 5-layer linear network

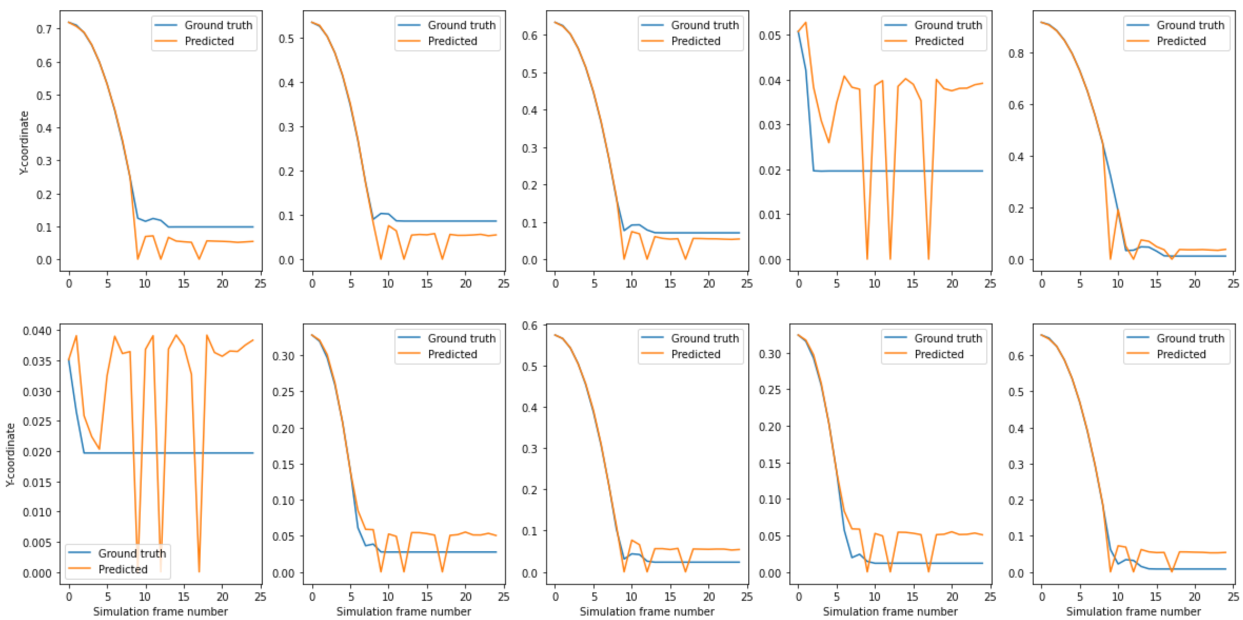


Figure 5. First 10 y-coordinate time series, ground truth, and predicted by overfitted 5-layer linear network. Plots are zoomed in to see the bounce level.

From Figure 5, which was intensionally zoomed to the scale of the whole trajectory, we can see that while the ground truth trajectory has a bounce on different levels, the model predicts it on about the same level - around 0.04-0.05 units. Seeing these results prompted me that the quantity that makes the ground truth bounce height differ is the diameter of the ball. In the absence of this information, the neural network just learns some descriptive statistics about this quantity from the training set, presumably the mean value.

To test my assumption, I modified this model to take two values as input - the initial coordinate and the ball's diameter. Testing this model on corresponding data, I received an RMSE of 0.0239 on the test set, with the first 10 predicted trajectories shown in Figure 6.

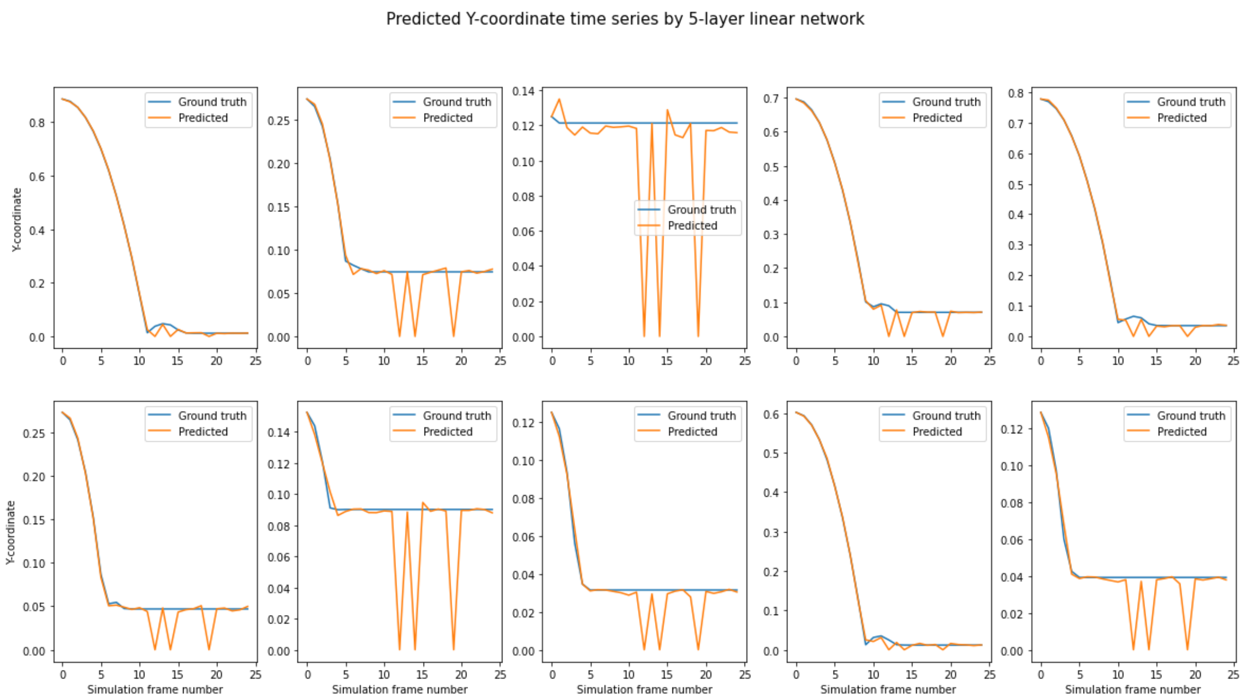


Figure 6. The first 10 trajectories from the test set, are ground truth and predicted by a 5-layer linear model that takes initial coordinate and ball diameter as input. Plots scaled to see the bounce level.

From these results, we can see that the model now predicts the bounce level correctly. The only remaining unusual pattern that we see on the plot is noise, which is not surprising given that the model is overfitted due to the large network size (Ying, 2019).

These findings conclude my exploration of feedforward neural networks. As the next steps, I will explore how recurrent neural networks will perform on the same kind of tasks.

4.2. Experiments with Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are well-suited for trajectory prediction problems because they are designed to learn from sequence data (Song et. al., 2020). RNNs can make predictions based on data from previous time steps, making them ideal for trajectory prediction problems where the output of one step is used as input for the next step. RNNs are also capable of capturing long-term dependencies, which is important for predicting trajectories that span longer periods of time.

4.2.1. Traditional Recurrent Neural Networks

When it comes to predicting trajectories, there are a few neural network models that are worth considering. Vanilla RNNs are a straightforward option that is good at learning patterns over time. However, they can struggle with capturing long-term dependencies due to the vanishing gradient problem, when the gradients can become very small as they are backpropagated through time (Pascanu et. al., 2013). This can make it difficult for the network to learn long-term dependencies because the gradient signal becomes too weak to update the weights of the network effectively. GRUs are a more lightweight alternative to LSTMs and can use gating mechanisms to avoid the vanishing gradient problem. LSTMs are great at capturing long-term dependencies thanks to their memory cells and gates that selectively remember or forget information. All three models have proven to be effective for trajectory prediction tasks in areas such as robotics, autonomous driving, and video analysis. Ultimately, the choice of model depends on the specific task and the properties of the input data.

4.2.1.1. Architectures and Results

To choose an optimal RNN design, I needed to find an optimal set of hyperparameters. The hyperparameters, the optimal values of which I am aiming to find, are the number of recurrent layers, the number of hidden neurons within each layer, and the dropout rate. My goal is to optimize the test set error. Using Grid Search as an optimization method, I found that the optimal test performance is achieved with a single recurrent layer that consists of 32 hidden neurons and a zero dropout rate.

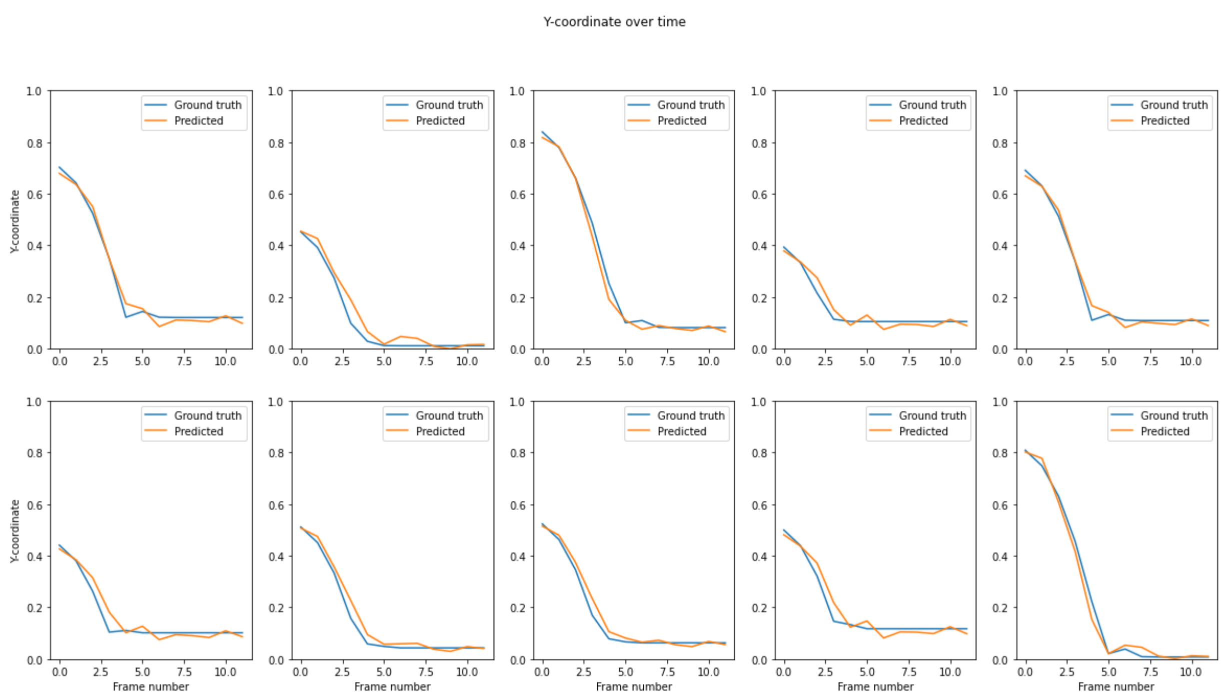
For the reason described in section 4.2.1.2. below, I had to make a slight modification of the model for testing as compared to the training. During the training stage, each of the models (Vanilla RNN, GRU, and LSTM) only consists of an input layer followed by a recurrent layer and the linear output layer. When the model is run on the test data, I added the post-processing function that bounds the outputs of the neural network between 0 and 1. The rationale behind this post-processing lies in all the data being bound between 0 and 1, so any output of the model that exceeds these bounds will not make sense. Mathematically, this post-processing function can be expressed as

$$f(x) = \max(0, \min(1, x))$$

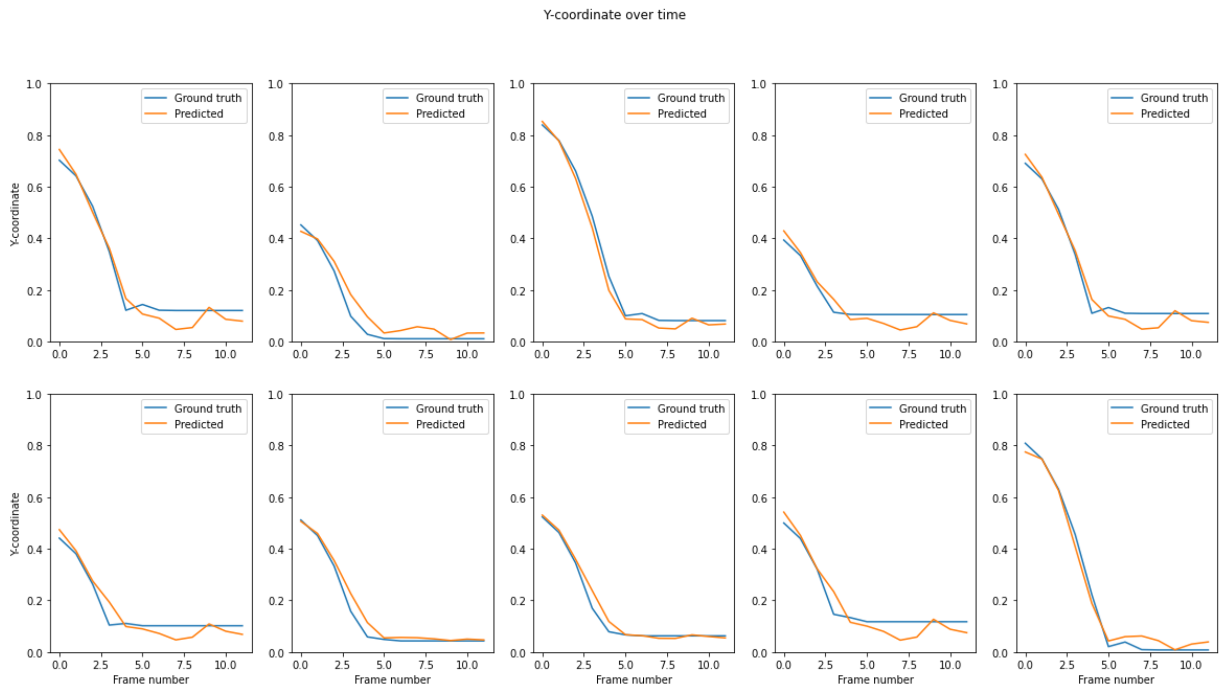
The results of running the chosen recurrent neural networks on one-dimensional free-fall data are given in Table 2. Surprisingly, the best performance out of the three models was demonstrated by the Vanilla RNN. The first 10 predicted Y-coordinate series by each of the models are shown in Figure 7.

Model Type	RMSE
Vanilla RNN	0.0278
GRU	0.0304
LSTM	0.0314

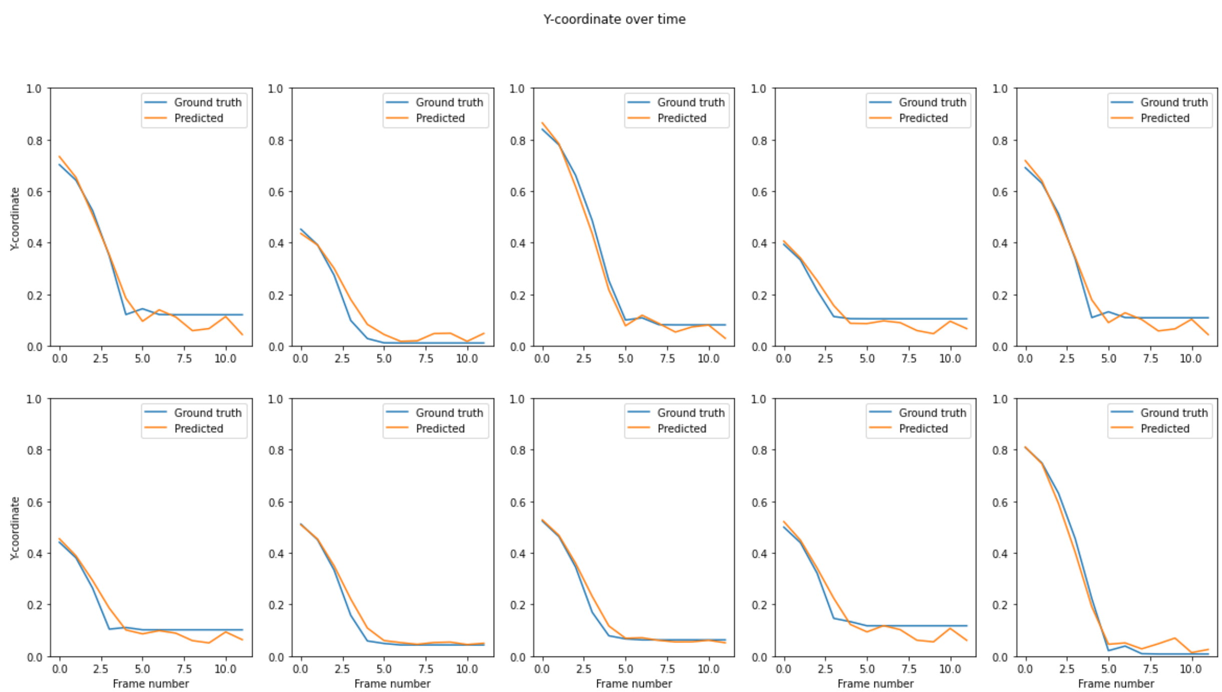
Table 2. Comparison of root mean square errors of each of the RNN types on one-dimensional free-fall prediction.



(a)



(b)



(c)

Figure 7. The first 10 predicted trajectories from the test set. (a) Vanilla RNN; (b) GRU; (c) LSTM.

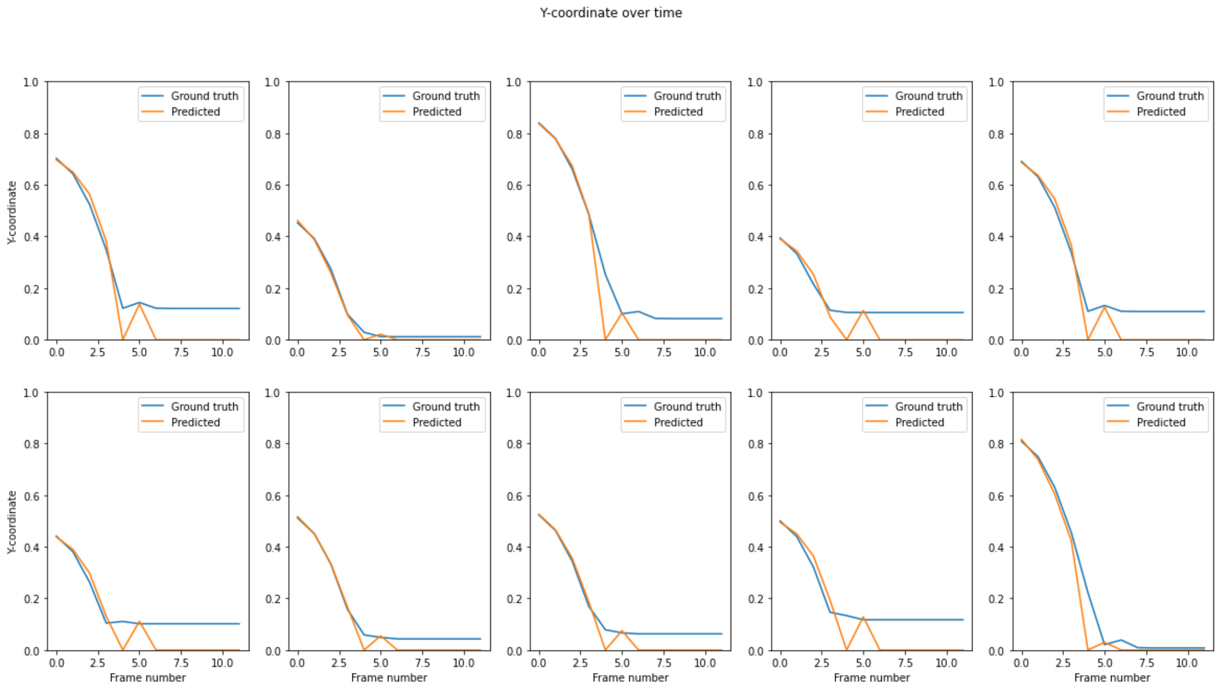
4.2.1.2. Non-convergent models with ReLU layer

One of the discoveries that I had while experimenting with recurrent neural networks is that adding a ReLU activation for the final layer can noticeably decrease the model's performance. My rationale behind adding this function lies in the fact that my data is bound between 0 and 1. Adding ReLU to the final layer would prevent the model from predicting the negative values, which in theory was supposed to improve the model performance. In reality, however, the performance of each type of model decreased. Quantification of this decrease is given in Table 3, while the first 10 trajectories from the test set are shown in Figure 8. By looking at these trajectories, we can see that the models are still predicting reasonable trajectories; however, these predictions have a noise similar to that produced by overfitted feedforward neural network.

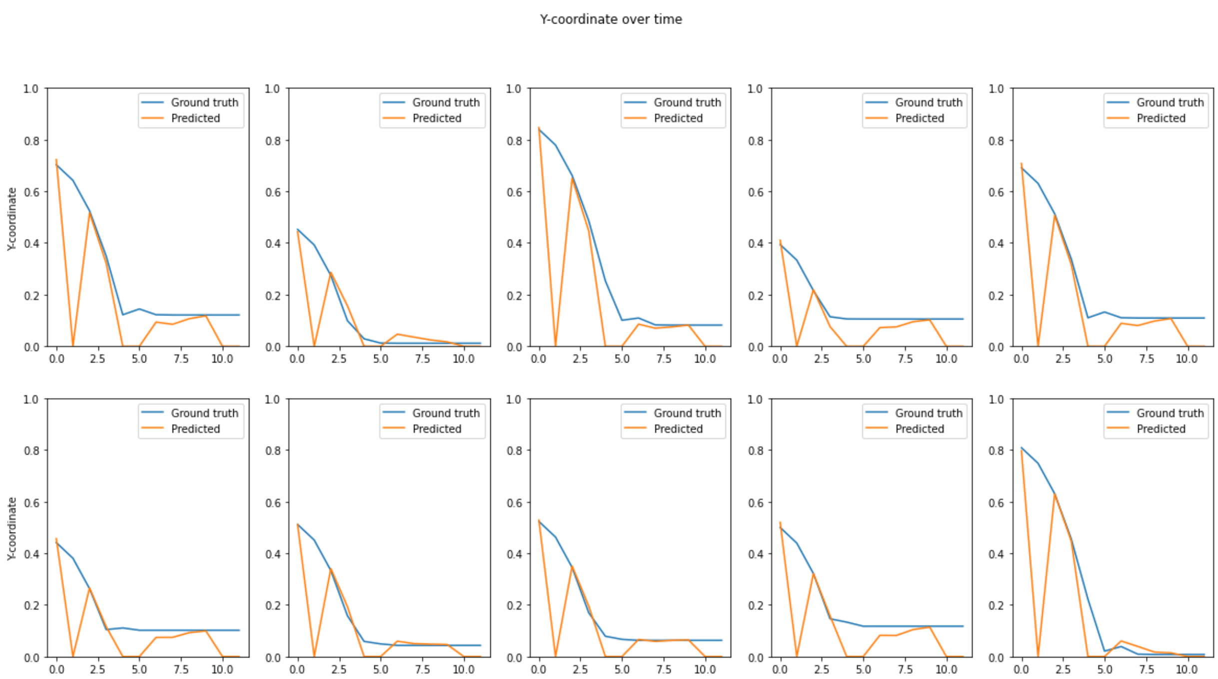
There are several reasons why this might happen. Adding a ReLU activation after the recurrent layer effectively sets the output of that layer to 0 for the negative outputs. Pascanu et. al. (2013) suggests that this causes a vanishing gradient problem, which makes it different to learn long-term dependencies. Similarly, Zaremba et. al. (2015) claims that learning long-term dependencies becomes complicated in such scenarios because the hidden states of the RNN become sparse.

Model Type	**RMSE**
Vanilla RNN with ReLU	0.1375
GRU with ReLU	0.1143
LSTM with ReLU	0.1869

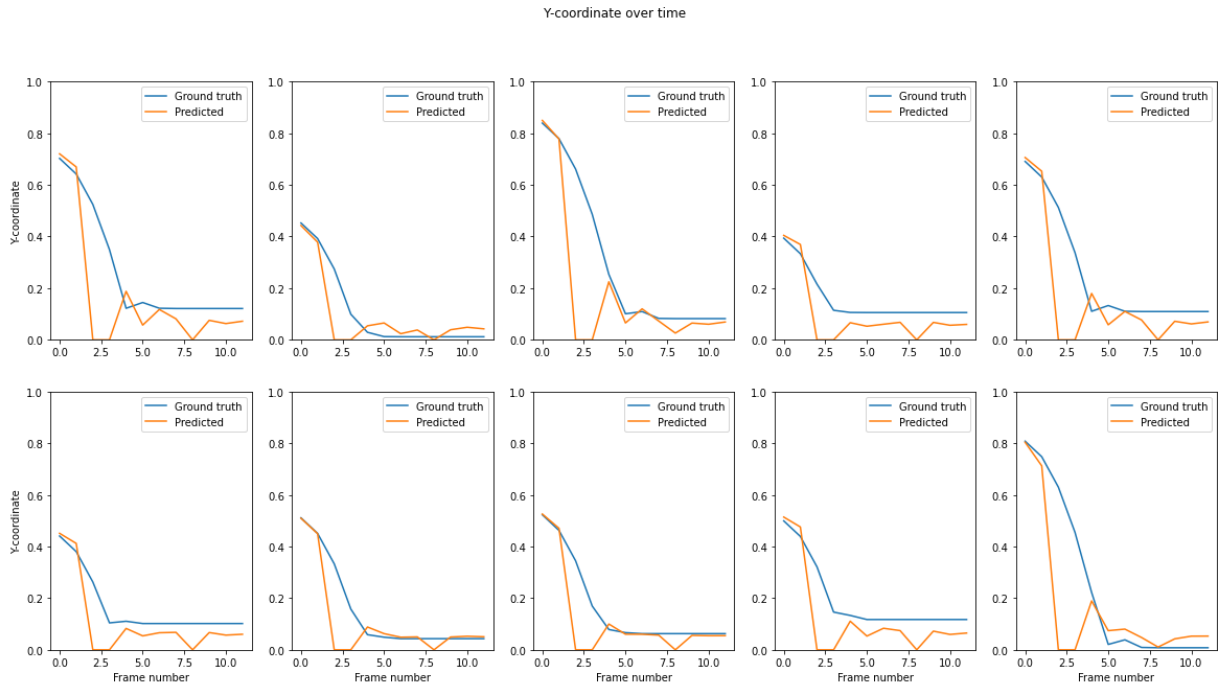
Table 3. Comparison of root mean square errors of each of the RNN models with ReLU layers on one-dimensional free-fall prediction.


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(a)


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(b)



(c)

Figure 8. The first 10 predicted trajectories from the test set. (a) Vanilla RNN with ReLU; (b) GRU with ReLU; (c) LSTM with ReLU.

4.2.2. Reservoir Computing Models

While the traditional recurrent neural networks demonstrated good performance, just exploring them is not compelling research on RNNs' ability to predict trajectories. In general, if the training data is long, it becomes challenging to train RNNs due to the vanishing and exploding gradients problems (Bengio et. al., 1994). Presumably, this is what happens when I add a final ReLU layer to RNN models.

The problem of training RNNs using long data sequences is solved by the Reservoir Computing (RC) paradigm that is being actively developed over the past decade (Song et. al., 2020). Echo State Network (ESN), the most widely known RC model, "has been recognized as the most efficient network structure for training RNNs" (Malik et. al., 2016). It has also been proven to be effective for chaos prediction (Wolchover, 2018), which makes it a perfect choice for trajectory prediction problems. This is why I decided to explore various ESN architectures, including deep ESN models, for trajectory prediction tasks.

A classical Echo State Machine consists of the following components (Malik et. al., 2016):

- Input layer, neurons of which are randomly connected with the reservoir neurons.
- Reservoir - a recurrent neural network with randomly (usually not sparsely) interconnected neurons. The connectivity of neurons within the reservoir is usually quite small (around 10%). The weights of the reservoir, as well as the weights between the input and the reservoir layers, are fixed and not trainable.
- Output (readout) layer - a linear layer densely connected with the reservoir. The weights between the reservoir and the output layer are trainable and are trained in a single epoch. Typically, Ridge Regression is used as a readout layer as it is optimal for highly correlated inputs, as it happens in the reservoir, neurons of which receive responses from each other (Lukosevicius, 2012).

A schematic representation of the Echo State Network with 4-dimensional input, the reservoir of size 7, and one-dimensional output is shown in Figure 9 (Verzelli et. al., 2019).

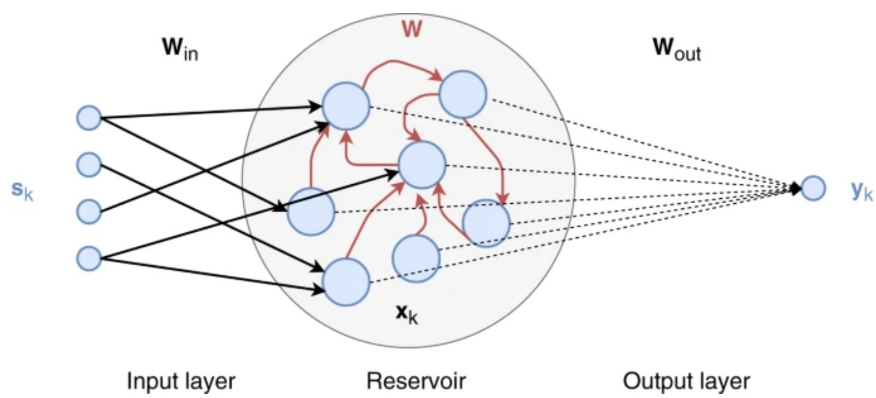


Figure 9. Echo State Network diagram with 4-dimensional input, the reservoir of size 7, and one-dimensional output. Retrieved from "Echo State Networks with Self-Normalizing Activations on the Hyper-Sphere" (2019) by Verzelli, P., Alippi, C., and Livi, L.

4.2.2.1. Architectures and Results

I started with testing out how different Echo State Network architectures will perform on predicting one-dimensional free-fall with a bounce. To implement the models, I used a Reservoirpy framework created by Trouvain et. al. (2020).

To test the ability of ESNs to predict trajectories, I implemented several different architectures. I started with a simple single reservoir Echo State Network, the number of hyperparameters of which allows to run Grid Search to find their optimal set. I found out that the optimal hyperparameters of ESN with Ridge Regression Readout for free-fall prediction are:

- Reservoir size of 70 neurons;
- Leaking rate of 0.7;
- Spectral radius of 0.95;
- Ridge coefficient of 0.01.

A number of papers, among which are papers by Malik et. al. (2016) and Song et. al. (2020), suggest that Deep Echo State Networks show better performance when compared to classical ESNs. Conducting a Grid Search for deep architectures would take an exponentially longer time compared to the classical ESN due to the increased number of parameters. Specifically, even adding a single extra reservoir adds 3 more hyperparameters (reservoir size, leaking rate, and spectral radius of the new reservoir) that, ideally, need to be optimized in addition to those of the first reservoir. While I don't have enough computational resources to run a Grid Search for hyperparameter optimization in Deep ESNs, I decided to take the optimal set that I found for the simple ESN and use it while creating Deep ESN models. The deep models that I tried for the free-fall prediction are

- Sequential ESN, inspired by Multilayered ESN presented by Malik et. al. (2016). I modified the model presented in this paper by adding a readout layer between each pair of reservoirs. The rationale behind it is that stacking several reservoirs without the readout layer is not different in principle from having a single reservoir with a larger size and modified connectivity;
- Parallel ESN, suggested by Song et. al. (2020);
- Grouped ESN, suggested by Song et. al. (2020).

All four types of ESN (classical plus three deep architectures) are schematically shown in Figure 10.

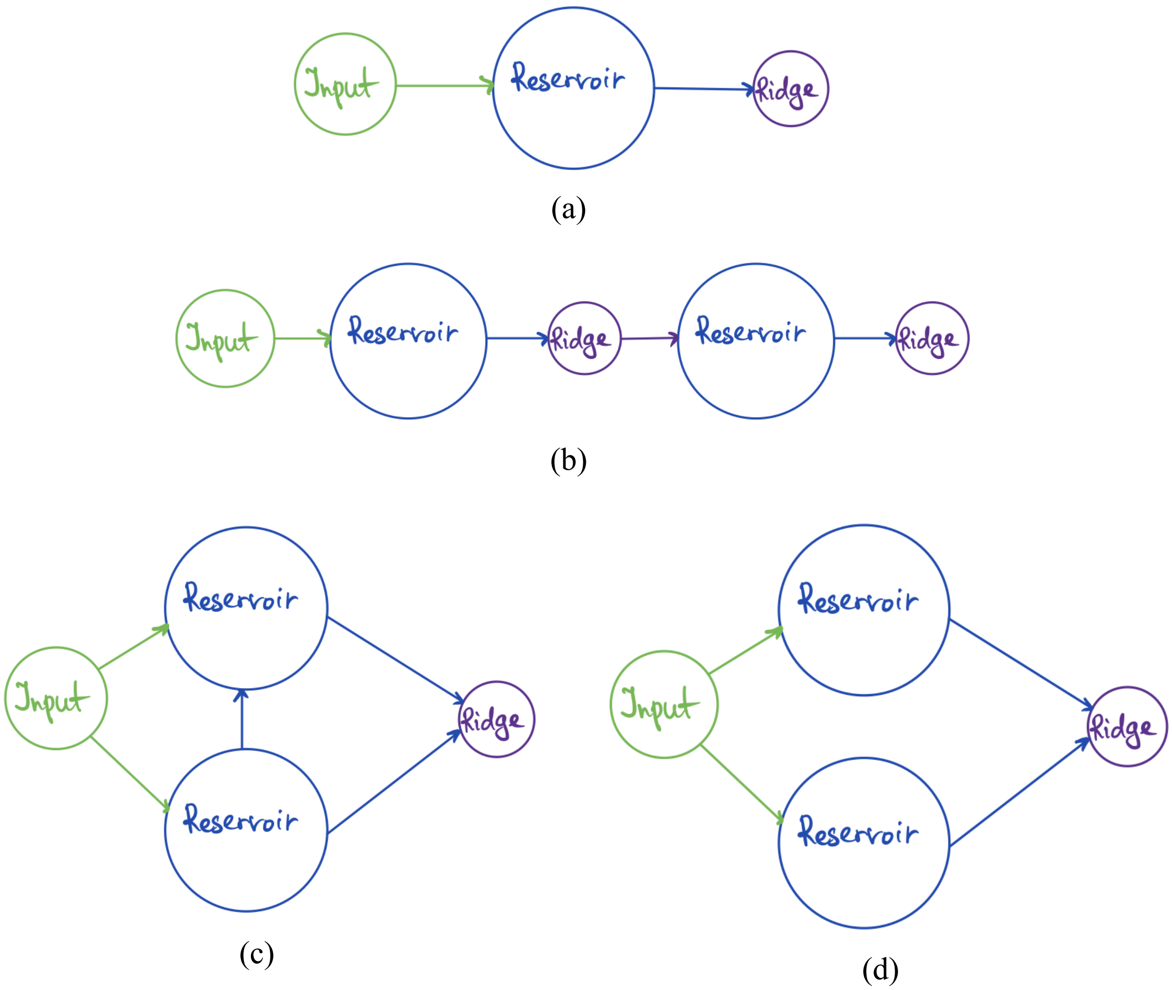


Figure 10. Implemented Echo State Network Architectures. (a) Classical ESN; (b) Sequential ESN; (c) Parallel ESN; (d) Grouped ESN.

It is easy to note that if any of the deep architectures will consist of a single reservoir, it will correspond to a simple ESN. To test the performance of these models, I trained a simple ESN as well as all the deep models with a number of reservoirs between 2 and 10. The performance of each of all of these models on the test set is shown in Figure 11, where the first point in every plot corresponds to the performance of a simple ESN.

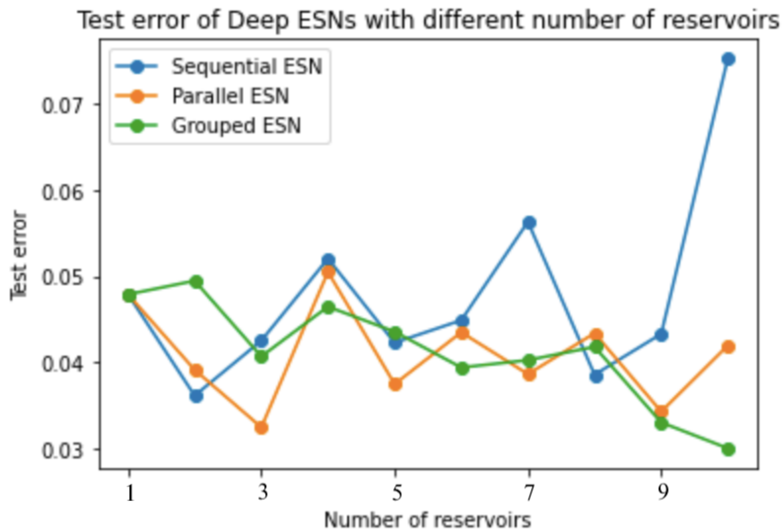
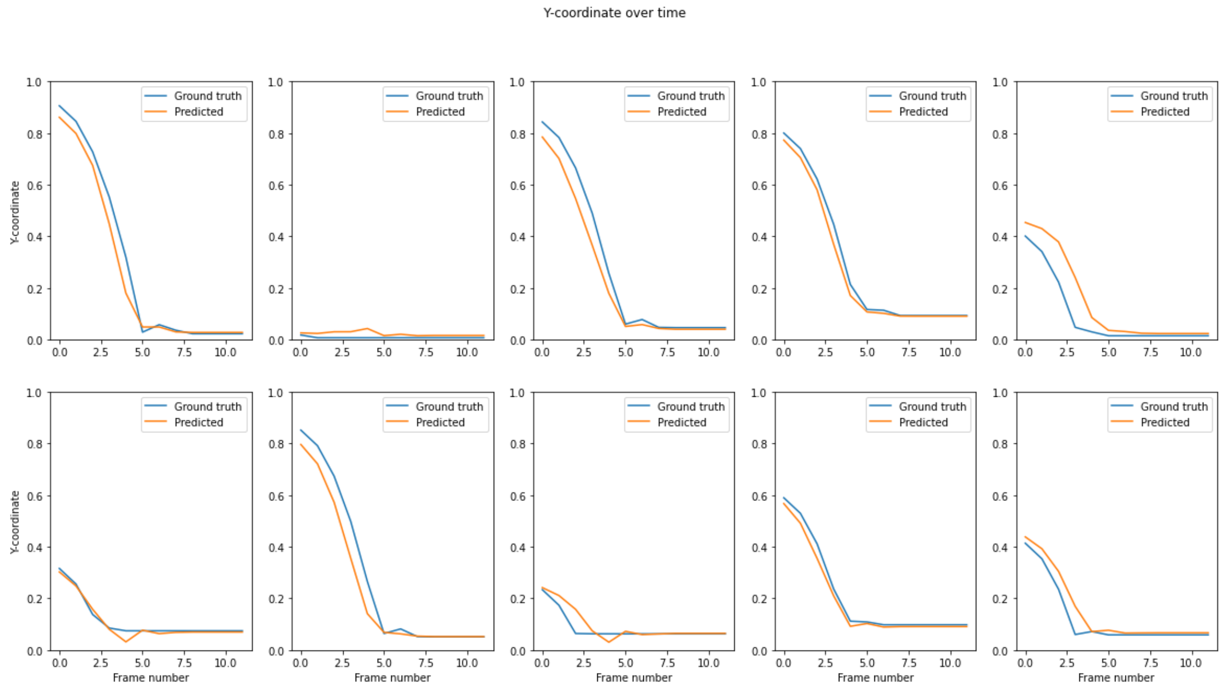
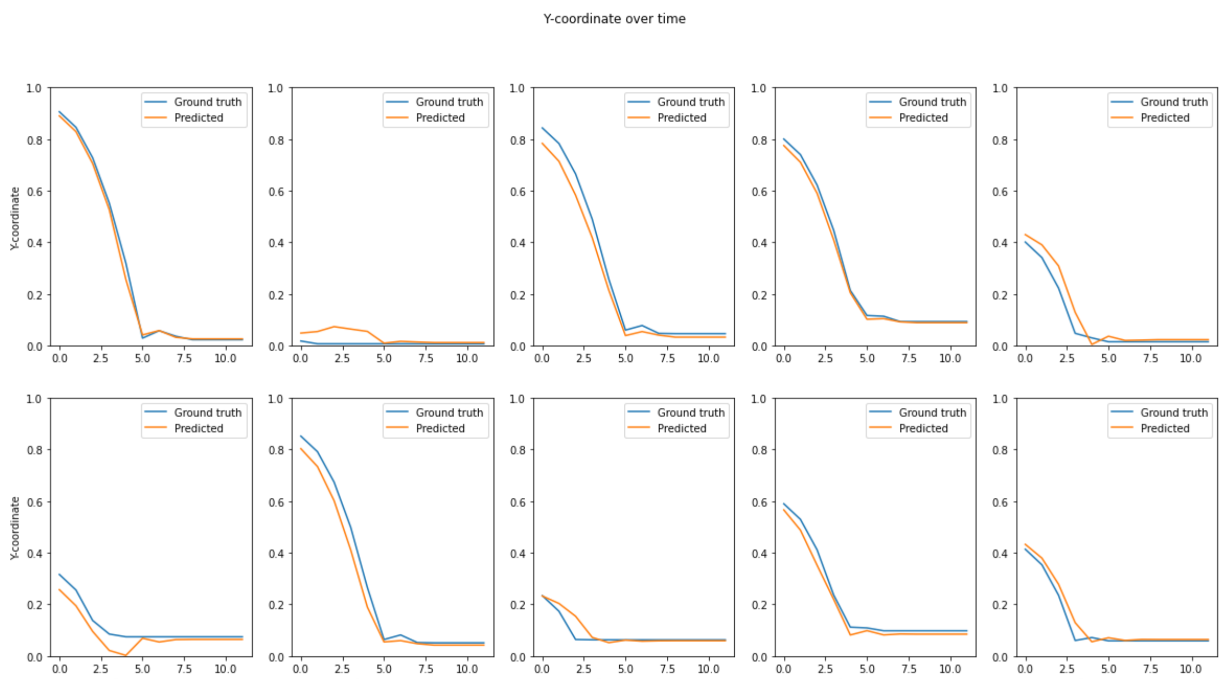


Figure 11. Performance of the Echo State Networks on the test set as a function of the number of reservoirs.

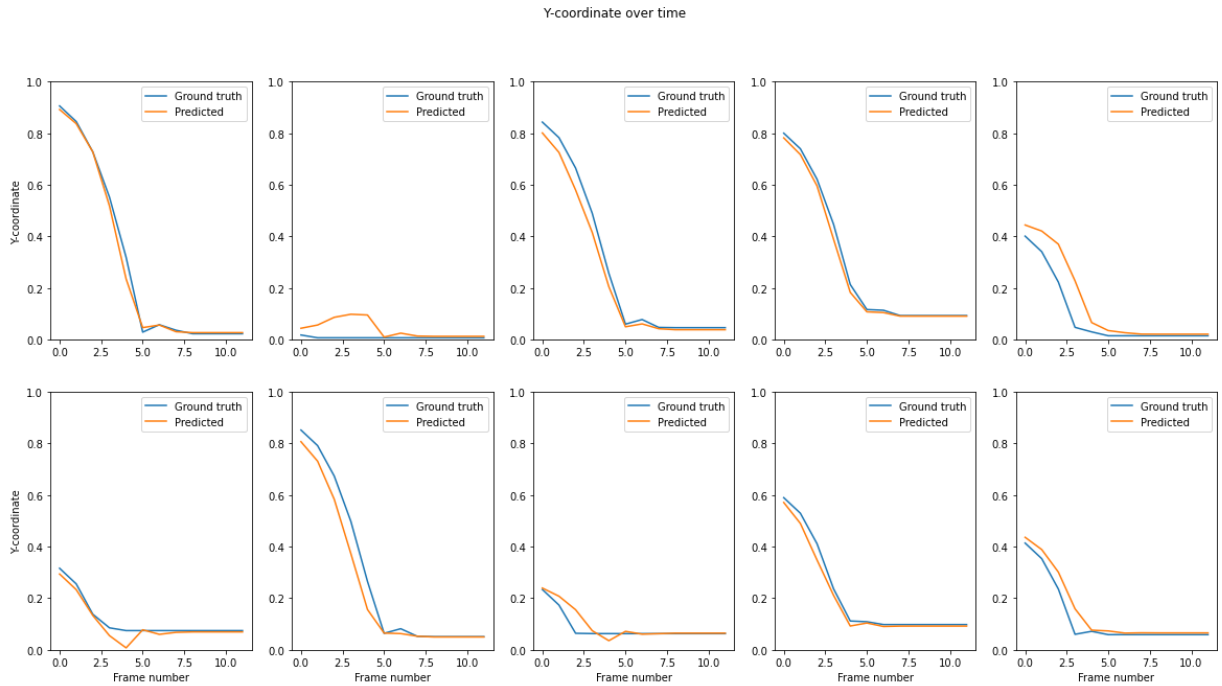
We can see that, perhaps with the exception of Grouped ESN, increasing the number of reservoirs doesn't improve the test set performance. Moreover, we can see that all of these models perform slightly worse when compared to the traditional recurrent neural networks. The first 10 test set trajectories for simple ESN and each of the deep ESN with two reservoirs are shown in Figure 12.


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

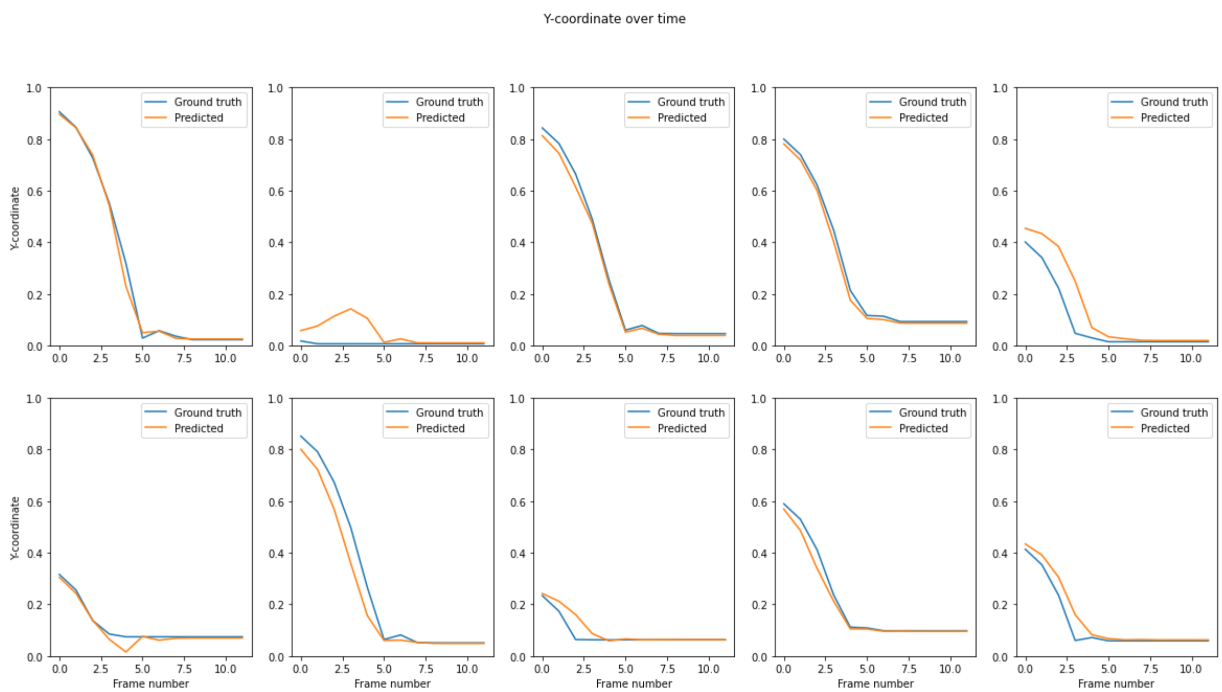
(a)


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(b)



(c)



(d)

Figure 12. First 10 test set predicted trajectories by Echo State Networks. (a) Simple ESN; (b) Sequential ESN; (c) Parallel ESN; (d) Grouped ESN.

4.2.2.2. ReLU layer experiment

After noticing that adding the ReLU layer prevents the training of traditional RNNs to converge properly, I decided to check if adding this layer also affects the performance of Echo State Networks. To do it, I decided to run an experiment on two ESN models, the only difference between which is the presence or absence of the final ReLU layer. My null hypothesis is that the mean test errors will be the same for both models, with an alternative hypothesis that the mean error will be less for the model *without* the final ReLU layer. I decided to make a decision on whether to reject the null hypothesis based on the t-test with a 0.05 significance level.

To perform the test, I generated two relatively small datasets (1,110 free-fall simulations). Then, I made 80 random train-test splits with a test fraction of 0.2 (222 simulations), and for each of these splits, I trained both models and saved the resulting test loss. A sample size of 80 for each model is chosen for Central Limit Theorem to hold properly. Once training and calculating test loss is complete for each split and each model, I run the difference of means test. I got a p-value of 0.22 on the one-tailed test, which is insignificant on the chosen significance level. This means that the test failed to reject the null hypothesis, or, in other words, the presence of the final ReLU layer doesn't make the performance of the model worse. The histogram of the test losses is given in Figure 13.

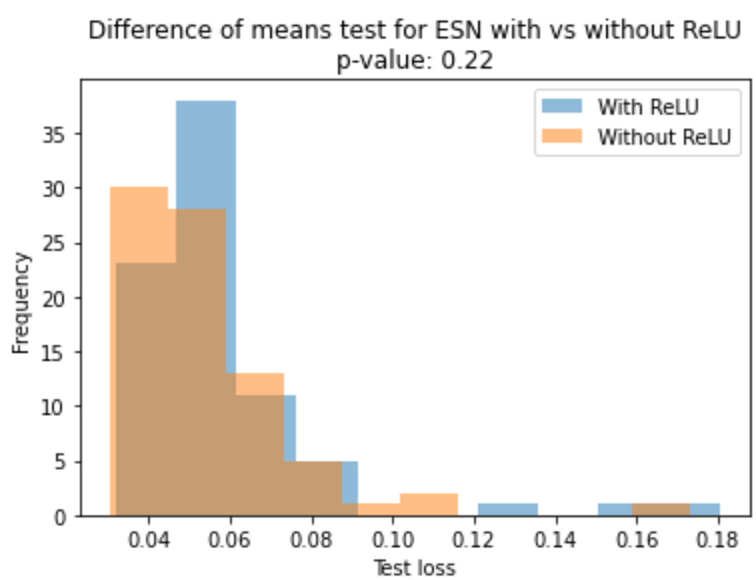


Figure 13. Histogram of test losses for ESNs with and without final ReLU layer.

4.2.3. Generalizing free-fall scenarios

Now that the one-dimensional free-fall is explored, there are a few more intermediate steps that need to be done to come closer to the trajectory prediction tasks in scenarios with collisions.

4.2.3.1. Free-fall in two dimensions

Up to this point, I was using a single Y-coordinate, knowing that X-coordinate stays constant during the free-fall. Now I want to test if my models will be able to infer that this coordinate is constant.

For this purpose, I generated a new dataset that includes X-coordinate in the output. I simulated 52,810 scenarios, 10,562 out of which were used as a test set.

Running a Grid Search for two-dimensional free-fall prediction resulted in the same optimal set of hyperparameters for Echo State Network models as for the one-dimensional scenarios. For traditional Recurrent Neural Networks, while a single recurrent layer was still found to be optimal, the number of hidden layers needs to be increased to 64.

While it is incorrect to compare the errors between one and two-dimensional free-fall predictions *to make any inference*, I was expecting the error to rise due to the increased output dimension to have an approximately matching error per predicted value. However, an interesting finding is that for all types of models that I tried, the test error noticeably decreased, although the number of parameters that need to be predicted increased two times.

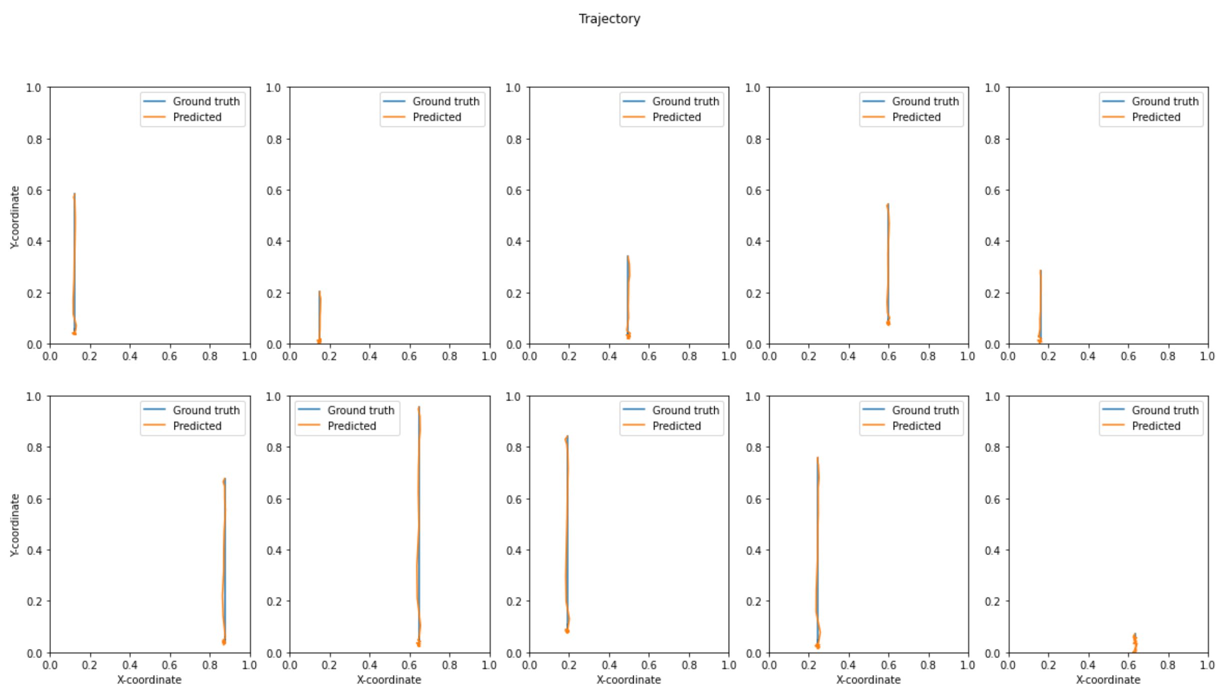
The test set error for each of the models is given in Table 4. The first 10 predicted y-coordinate sequences and full trajectories from the test set are shown in Figure 14.

Model	**RMSE**
Vanilla RNN	0.0182
GRU	0.0160
LSTM	0.0164
ESN	0.0320

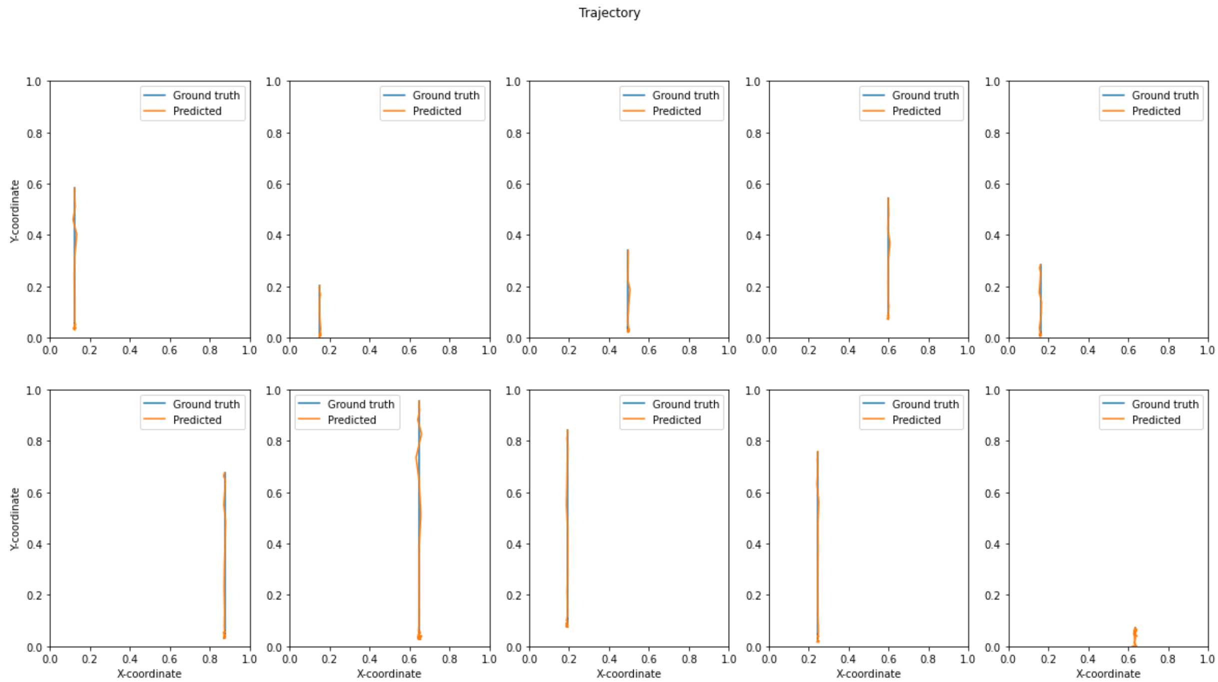
Sequential ESN	0.0413
Parallel ESN	0.0314
Grouped ESN	0.0314

Table 4. Root mean square error of each model on the test set for two-dimensional free-fall prediction.

We can see that the Reservoir Computing models perform noticeably worse compared to the traditional RNNs. From Figure 14, however, we can notice a significant difference in the predicted trajectories: while traditional RNNs do a good job predicting the free-fall trajectories very closely, we can still see oscillations in predicted values for both coordinates. Meanwhile, ESNs do a worse job resembling the trajectories closely; however, they perfectly learn that free-fall is happening along a straight line.

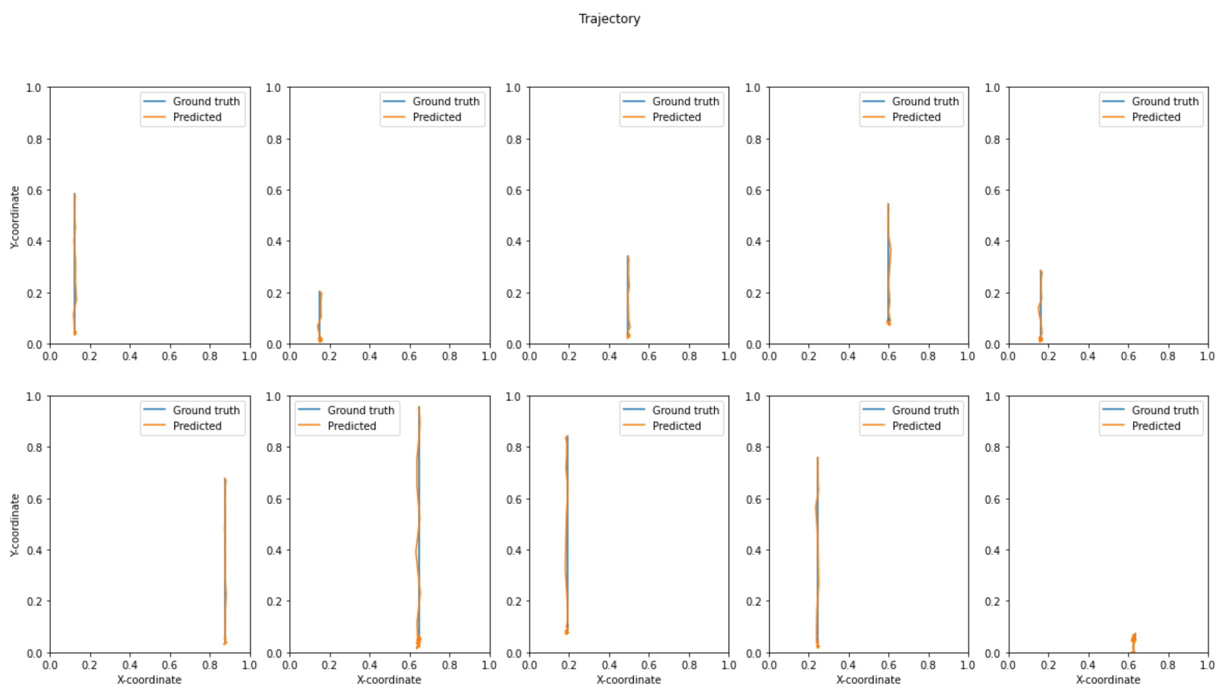


(a)



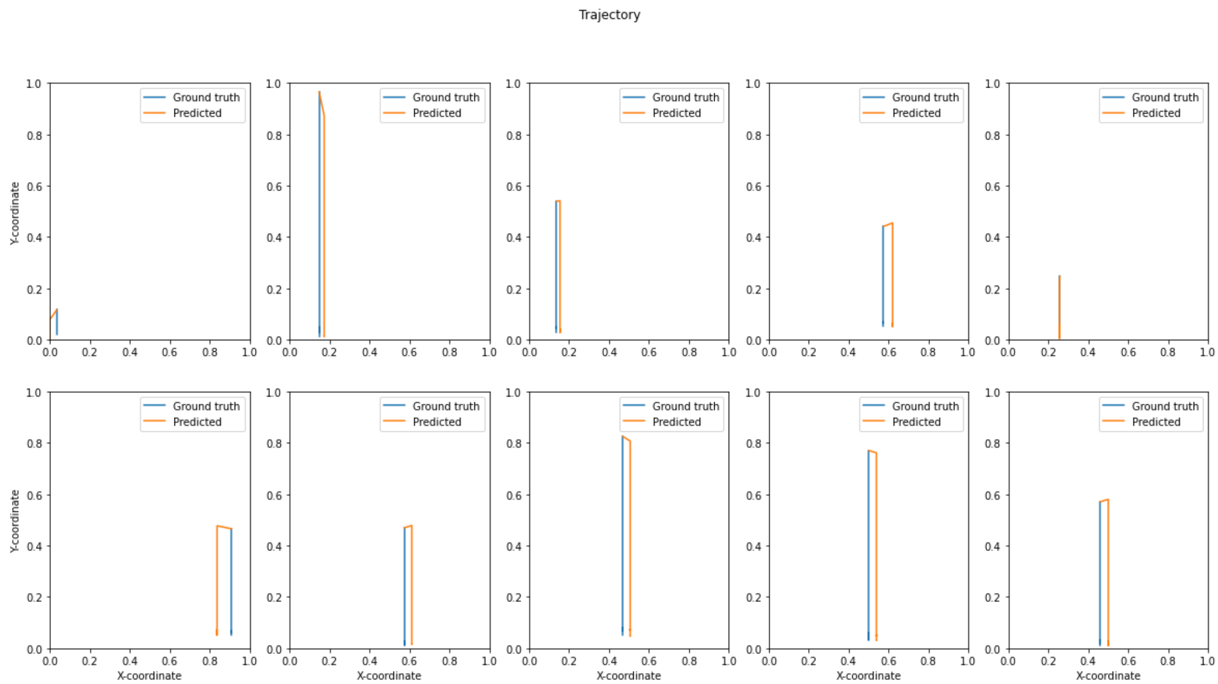
Trajectory
Ground truth
Predicted
Y-coordinate
X-coordinate

(b)

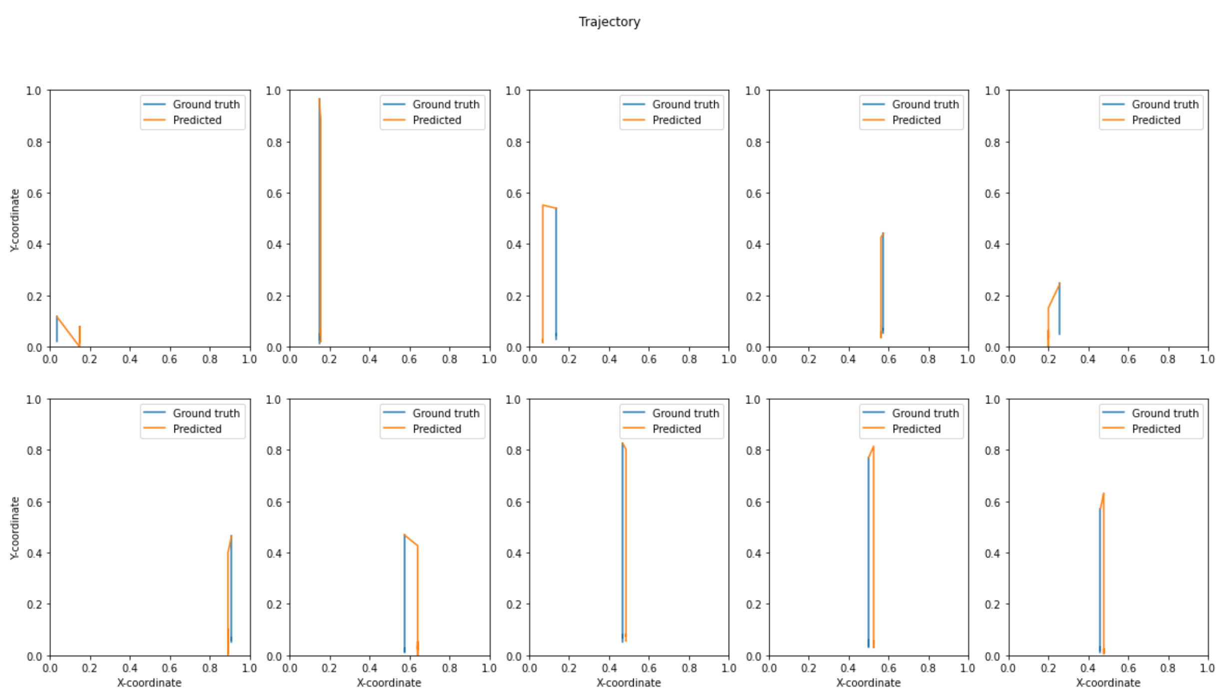


Trajectory
Ground truth
Predicted
Y-coordinate
X-coordinate

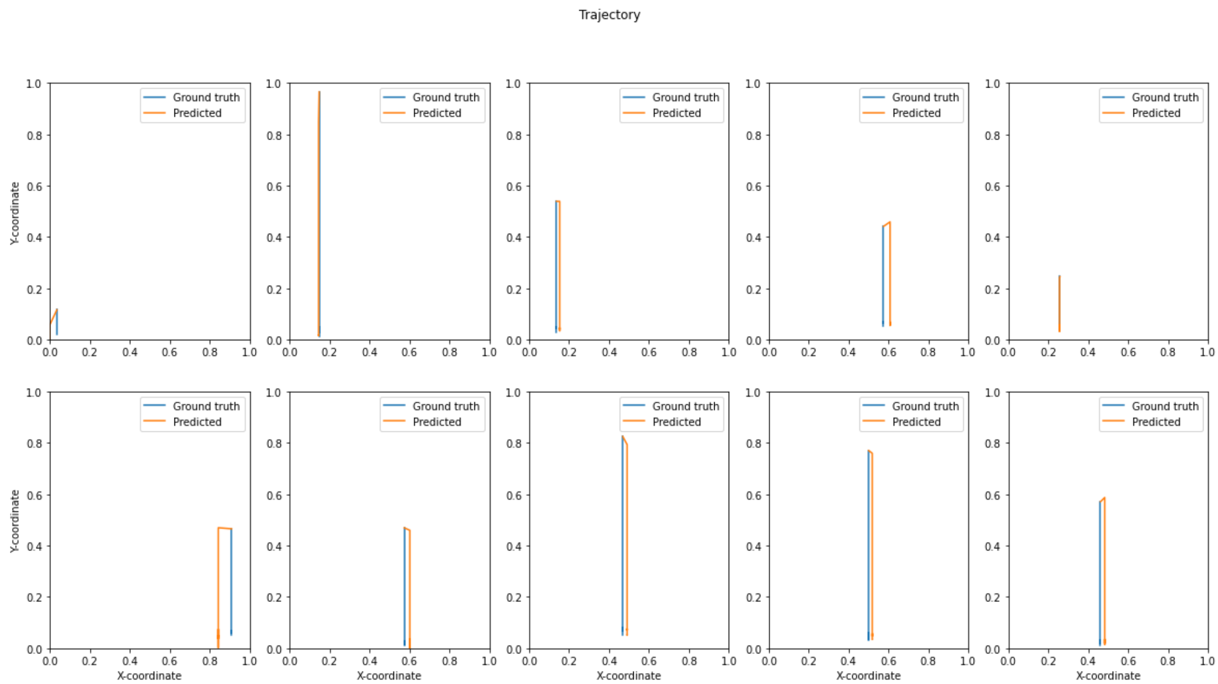
(c)


Trajectory
Ground truth
Predicted
Y-coordinate
X-coordinate

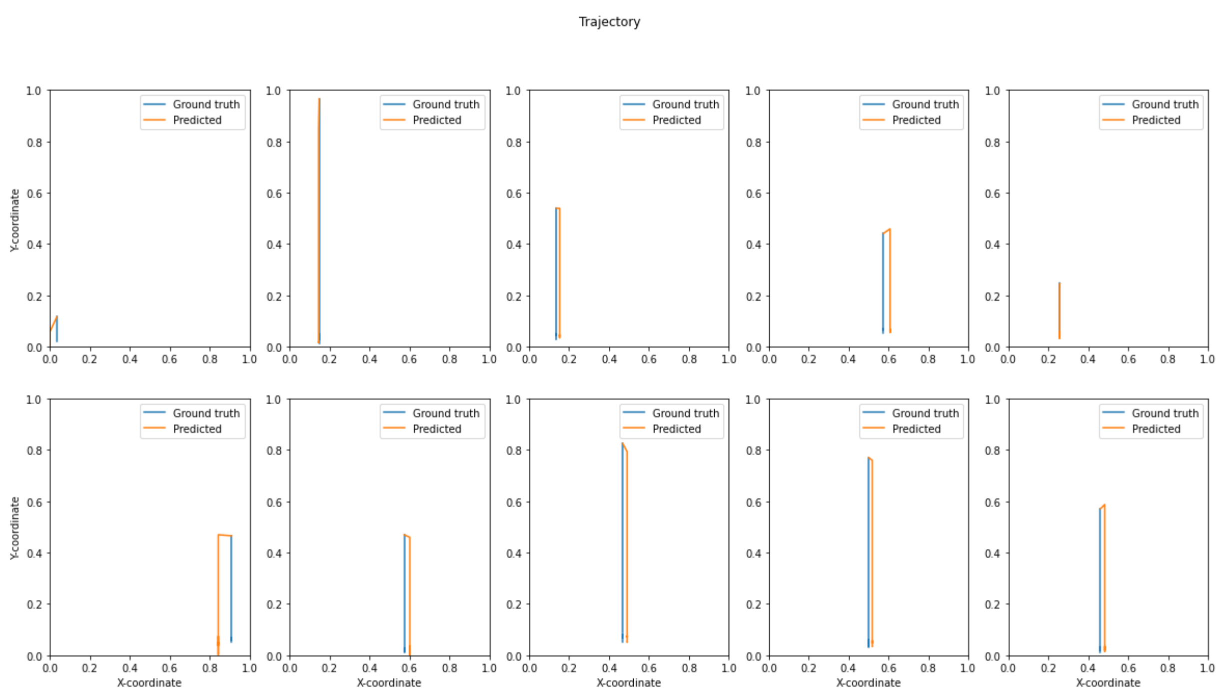
(d)


Trajectory
Ground truth
Predicted
Y-coordinate
X-coordinate

(e)



(f)



(g)

Figure 14. Predicted two-dimensional free-fall trajectories by each of the models. (a) Vanilla RNN; (b) GRU; (c) LSTM; (d) ESN; (e) Sequential ESN; (f) Parallel ESN; (g) Grouped ESN.

4.2.3.2. Scenarios with multiple balls

The last layer of complexity that I can add to predicting the free-fall trajectory of the ball is adding information about other balls on the scene to the input.

Table 5 shows the free-fall prediction RMSE value of each of the models on the test set. As we can see, adding extra information to the input did not affect the performance of the traditional models, while the performance of Reservoir Computing models actually improved. My hypothesis is that the extra input serves as the extra variable for regularization in Ridge Regression, which positively affects the model performance.

Model	**RMSE**
Vanilla RNN	0.0194
GRU	0.0165
LSTM	0.0158
ESN	0.0251
Sequential ESN	0.0374
Parallel ESN	0.0257
Grouped ESN	0.0257

Table 5. RMSE of predictions on the test set for free-fall scenarios with multiple balls.

4.3. Intermediate Conclusion

In this section, I investigated the performance of various Recurrent Neural Network architectures in predicting the trajectory of a ball undergoing free fall and bouncing off the ground. A thorough comparison between traditional RNNs and Reservoir Computing models revealed that traditional RNNs outperform their counterparts in minimizing prediction errors. However, a closer examination of the visualization of the models' predictions suggested that RC models may have a better grasp of the underlying physics of free-fall motion.

A prime example of this is demonstrated in the two-dimensional free-fall prediction task. While traditional RNNs achieve lower error rates, they struggle to capture the fact that free-fall occurs strictly along a straight line. In contrast, RC models successfully learn this pattern but fail to accurately determine the specific x-coordinate along which the free-fall occurs.

In conclusion, this section highlights the strengths and weaknesses of both traditional RNNs and Reservoir Computing models in the context of trajectory prediction for free-falling objects. While traditional RNNs excel at minimizing prediction errors, RC models demonstrate a better understanding of the underlying motion patterns, despite their shortcomings in accurately predicting specific trajectories. This insight may guide future research on developing hybrid or improved models that can leverage the strengths of both architectures for more accurate and physically plausible trajectory predictions.

Section 5. Movement with collisions

In this section, I will address the challenges associated with the increased complexity of scenarios involving three balls falling and colliding with each other. The larger datasets required for these scenarios lead to longer training times, making hyperparameter optimization more difficult. To tackle these challenges, I will discuss two aspects of data engineering that need to be reconsidered: limiting the number of free-fall scenarios and optimizing the use of data for hyperparameter optimization and model training.

5.1. Optimizing data engineering

While my focus now shifts from free-fall scenarios to those involving collisions, randomly sampling actions will still include a significant number of scenarios where the target ball is still in free fall. To ensure that my dataset adequately represents the more complex collision scenarios, I will need to limit the fraction of free-fall scenarios in the dataset. I will discuss the method that I used to limit the number of free-fall scenarios in section 5.1.1.

As the dataset size increases and training times become longer, it is crucial to use the available data efficiently for hyperparameter optimization and model training. In section 5.1.2 I will discuss how I changed my approach to hyperparameter optimization to deal with a large dataset.

5.1.1. Limiting the fraction of free-fall scenarios

In the current approach to generating datasets, as discussed in section 3.2, random actions are sampled, resulting in a high fraction of free-fall scenarios even when not constrained to them. This can lead to biased model predictions towards free-fall, and thus, the number of free-fall scenarios in the dataset must be constrained. Since Phyre doesn't provide an API for implementing such constraints, I will propose an alternative approach.

Let's define:

- t as the total number of valid simulations;
- f as the number of free-fall scenarios among t;
- ϵ as the desired fraction of free-fall scenarios;

- r as the number of free-fall simulations that need to be removed from the pool.

We can detect the free-fall simulations as those where the x-coordinate of the red ball stays constant.

To limit the fraction of the free-fall simulations to ϵ, I need to remove r free-fall simulations such that the following equation holds:

$$\frac{f-r}{t-r} = \epsilon$$

From this equation I can express the number r of the simulations that need to be removed.

$$r = \frac{f-\epsilon t}{1+\epsilon}$$

Thus, the strategy is to simulate some number of scenarios, labeling the free-fall simulations in some special way. Once the whole dataset is generated, randomly sample r free-fall simulations, according to the chosen fraction ϵ, and remove them from the dataset. This will result in a dataset with a more balanced representation of free-fall and collision scenarios, allowing the model to better generalize to the task of predicting trajectories in scenarios with collisions.

5.1.2. Changing the data usage for training vs. hyperparameter optimization

In scenarios where the ball's movement is not constrained in 1 dimension and interacts with other balls, the complexity increases, necessitating more data for training machine learning models to predict such movements. However, more data leads to longer training times and exponentially longer Grid Search computations for hyperparameter tuning.

To address this issue, I will generate two separate datasets for training and hyperparameter tuning, with their size being the only difference. For hyperparameter tuning, I will create a relatively small dataset. It is acknowledged that the smaller dataset might result in inadequate training of the models, leading to a suboptimal final performance. However, the assumption is that despite each model being undertrained, their relative performance when trained on the smaller dataset should be preserved compared to the larger dataset.

For hyperparameter tuning, I created two datasets (for predicting single-ball trajectory and evolution of the entire scene, as discussed in sections 5.2 and 5.3, respectively) containing 18,186 simulations each, 3,637 of which were used as hold-out sets. Once the optimal set of hyperparameters is found for each model using this dataset, I can use them to train the models using bigger datasets. The bigger datasets contain 274,056 simulations each, 54,811 of which were used for testing.

5.2. Predicting single ball trajectory

I started with running the Grid-Search for each of the traditional model types. The search space that I had was the following:

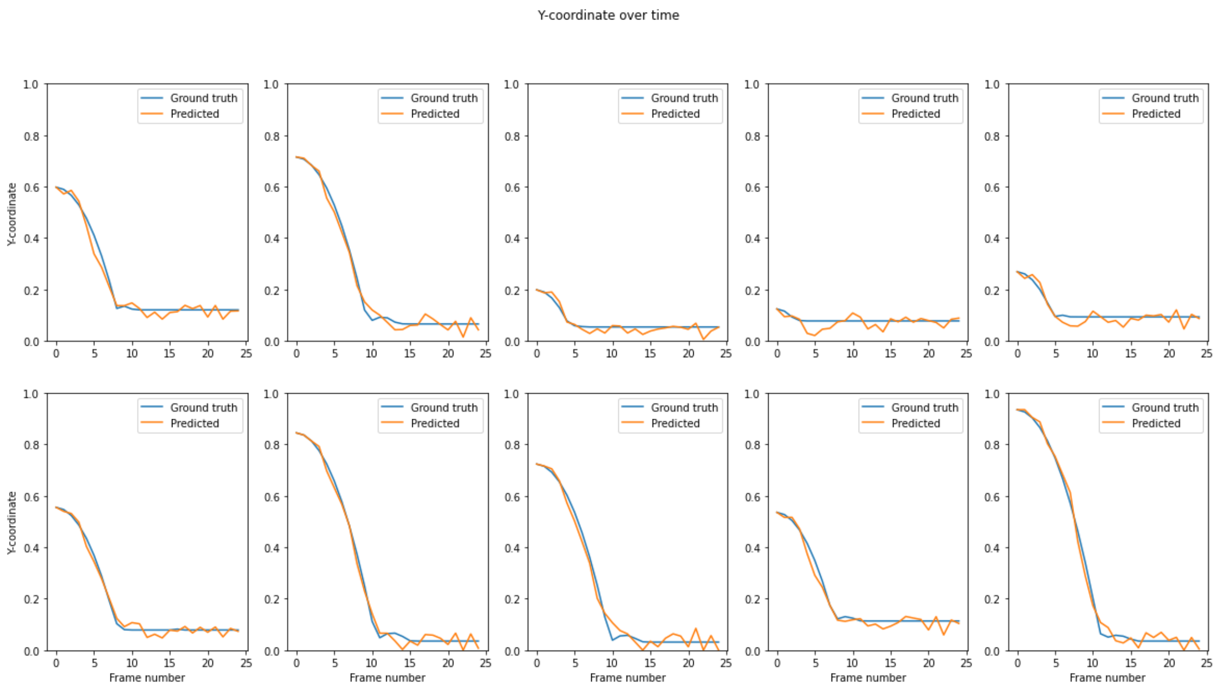
- Number of hidden neurons: 64, 96, 128, 192, 258;
- Number of recurrent layers: 1, 2, 3;
- Dropout rate (for the number of layers greater than 1): 0, 0.1.

For all three models, the optimal set of hyperparameters found with Grid Search is 96 neurons, 1 recurrent layer, and 0 dropout rate. The test set RMSE of traditional RNN models is shown in Table 6, which indicates a threefold decrease in performance compared to predicting solely free-fall with similar input.

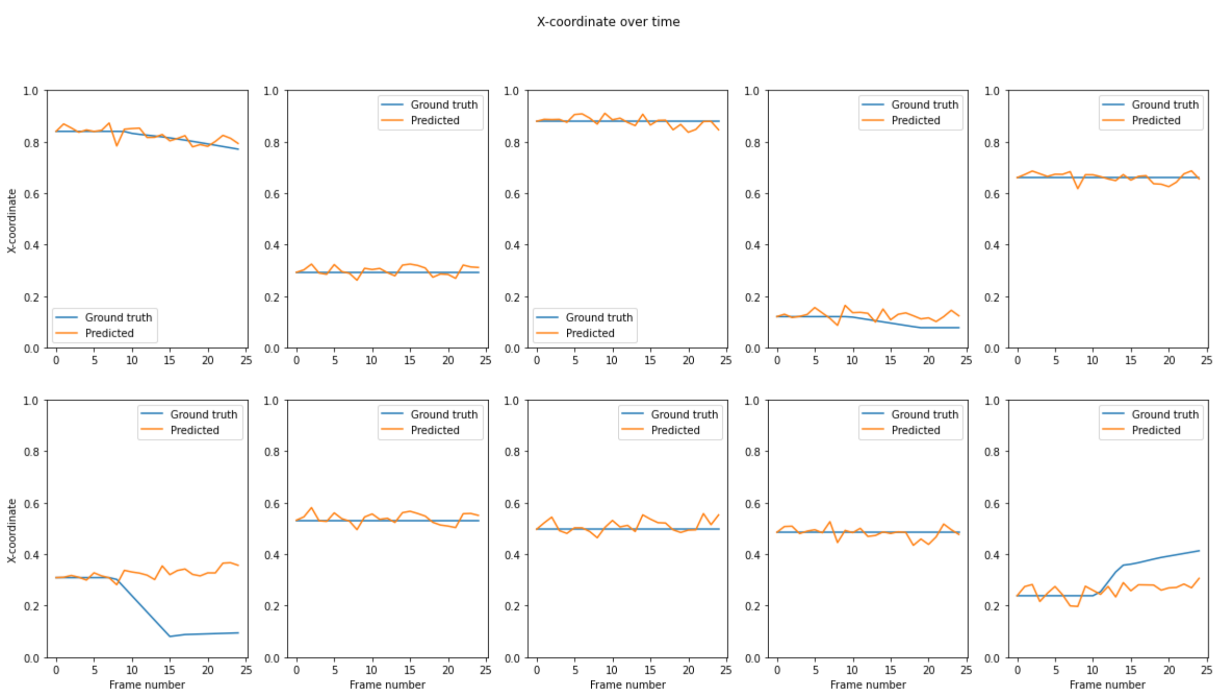
Model	**RMSE**
Vanilla RNN	0.0467
GRU	0.0487
LSTM	0.0426

Table 6. RMSE of traditional models on predicting trajectories with collisions from the test set.

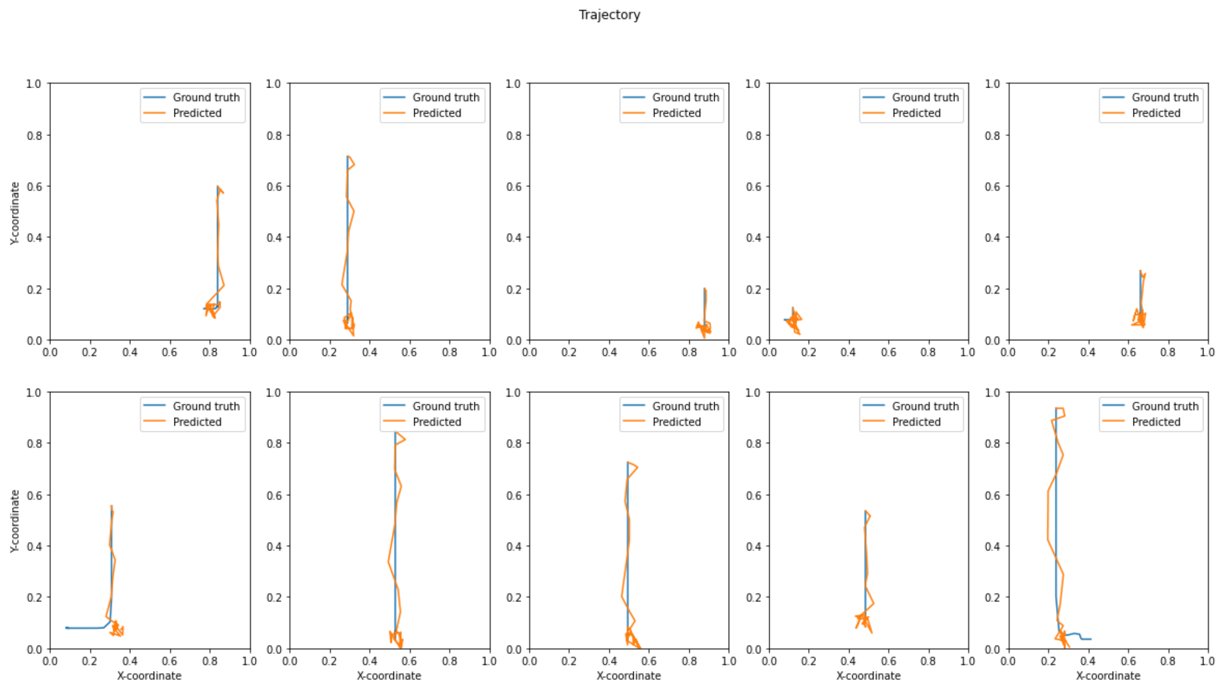
Figures 15-17 illustrate the predicted trajectories by each traditional model, along with the predicted x-coordinate and y-coordinate time series. Each coordinate prediction exhibits significant noise, and this noise persists even when increasing the dataset size threefold (from 91,256 simulations that I used originally to 274,056 simulations that were used to produce the reported results). Among the three models, LSTM produces the least noisy predictions, though they are still not highly accurate. While the models correctly predict the object's downward motion, they often fail to accurately infer collisions and post-collision movements.



(a)

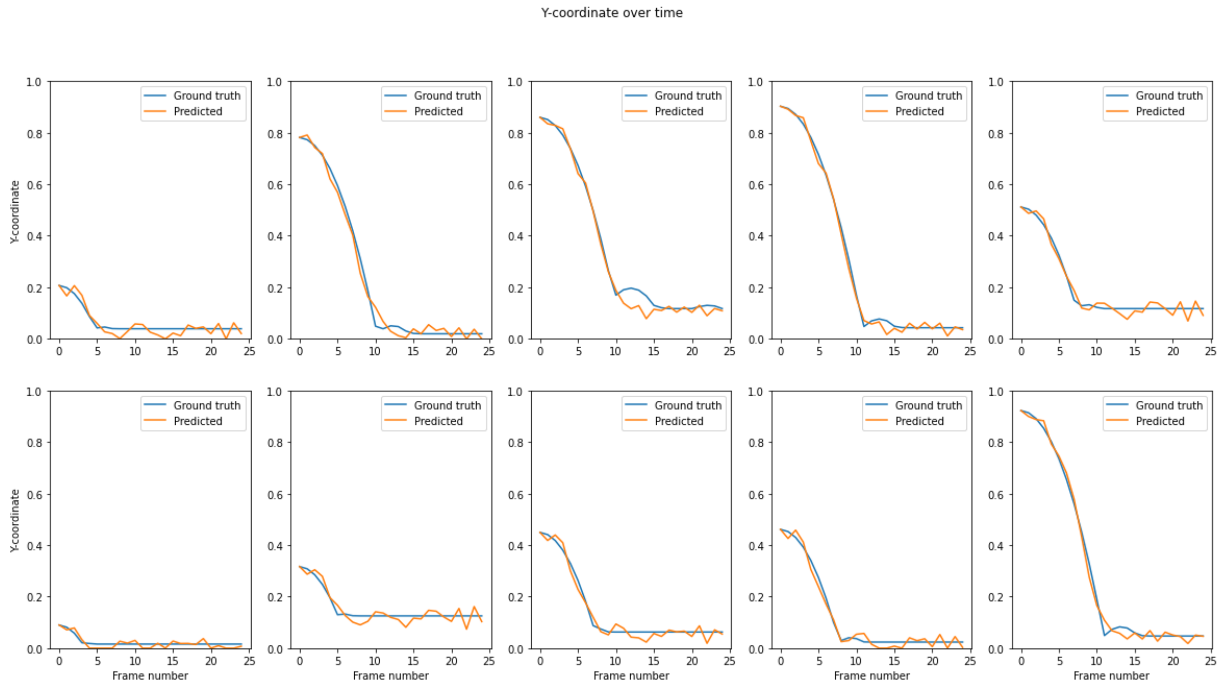


(b)

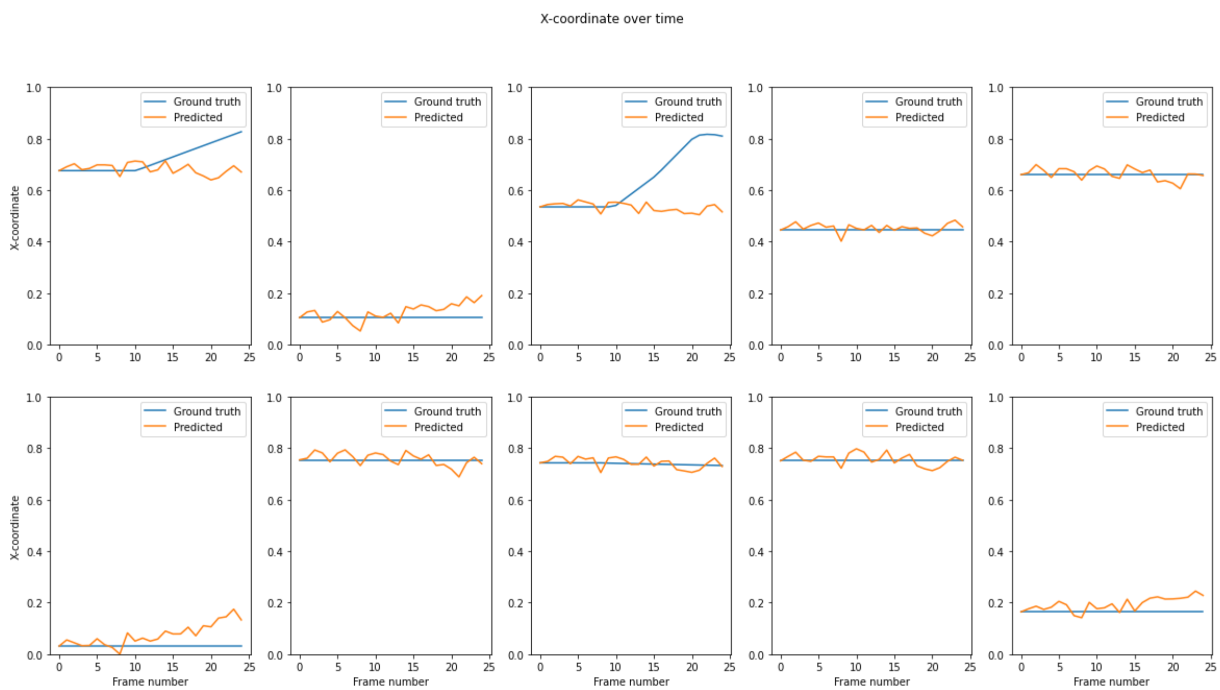


(c)

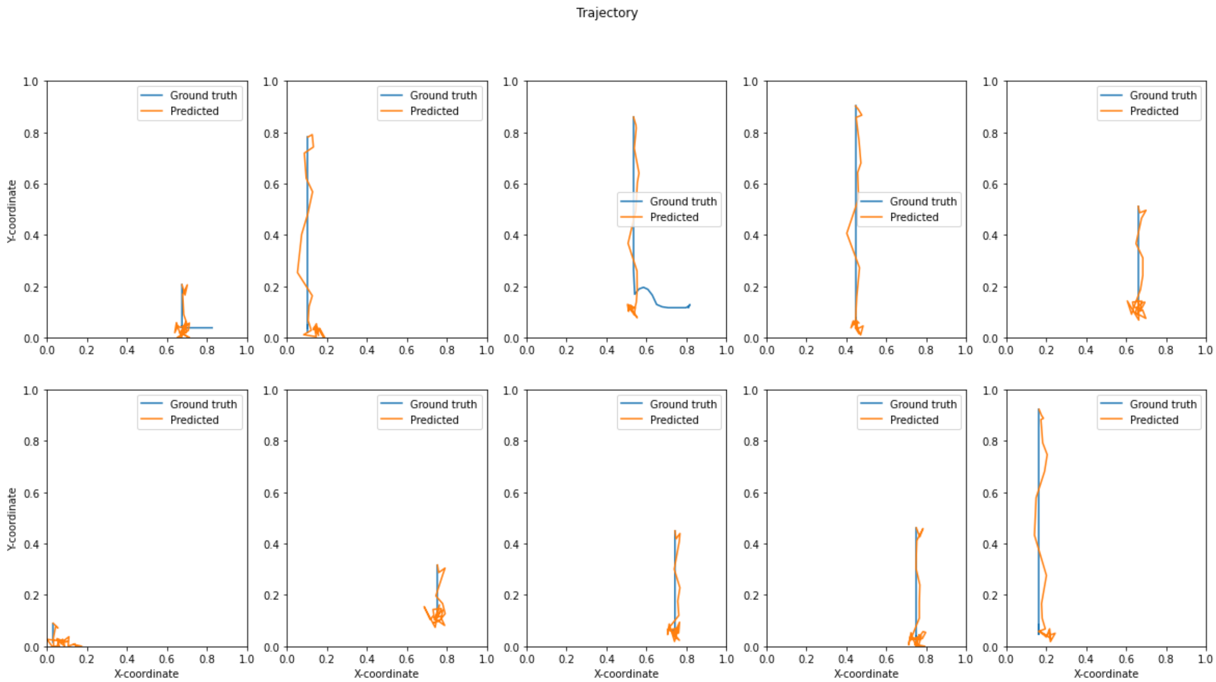
Figure 15. The first 10 test-set predictions of Vanilla RNN on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(a)

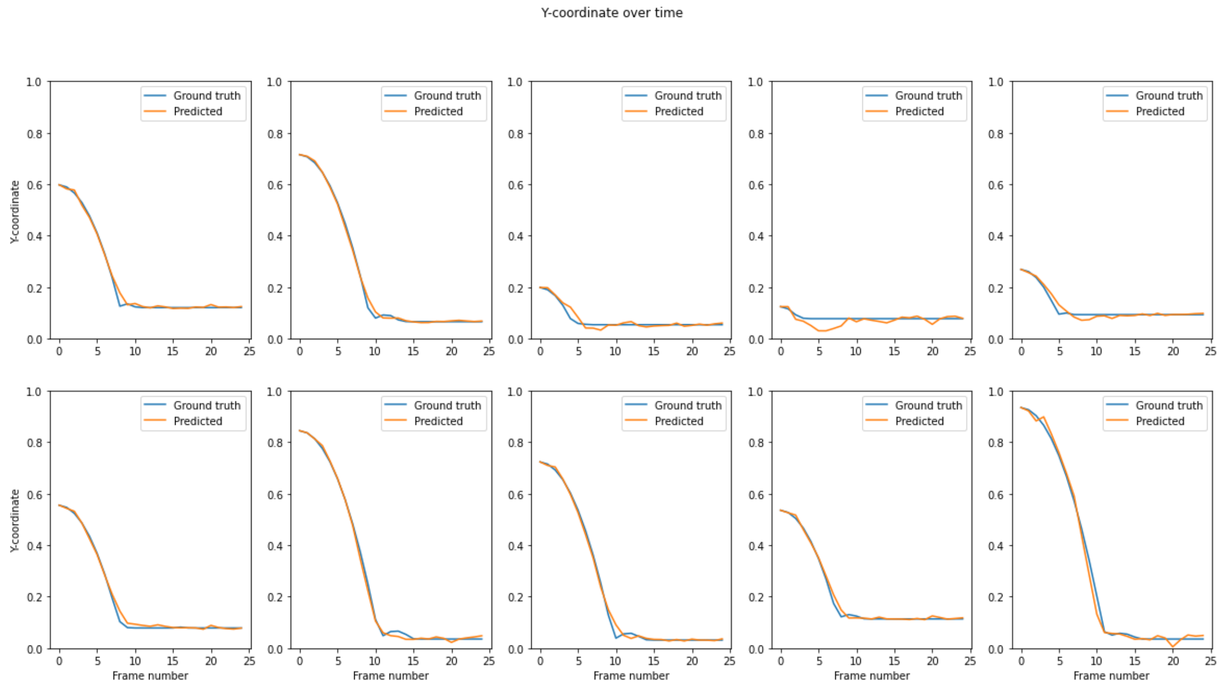

X-coordinate over time
Ground truth
Predicted
X-coordinate
Frame number

(b)

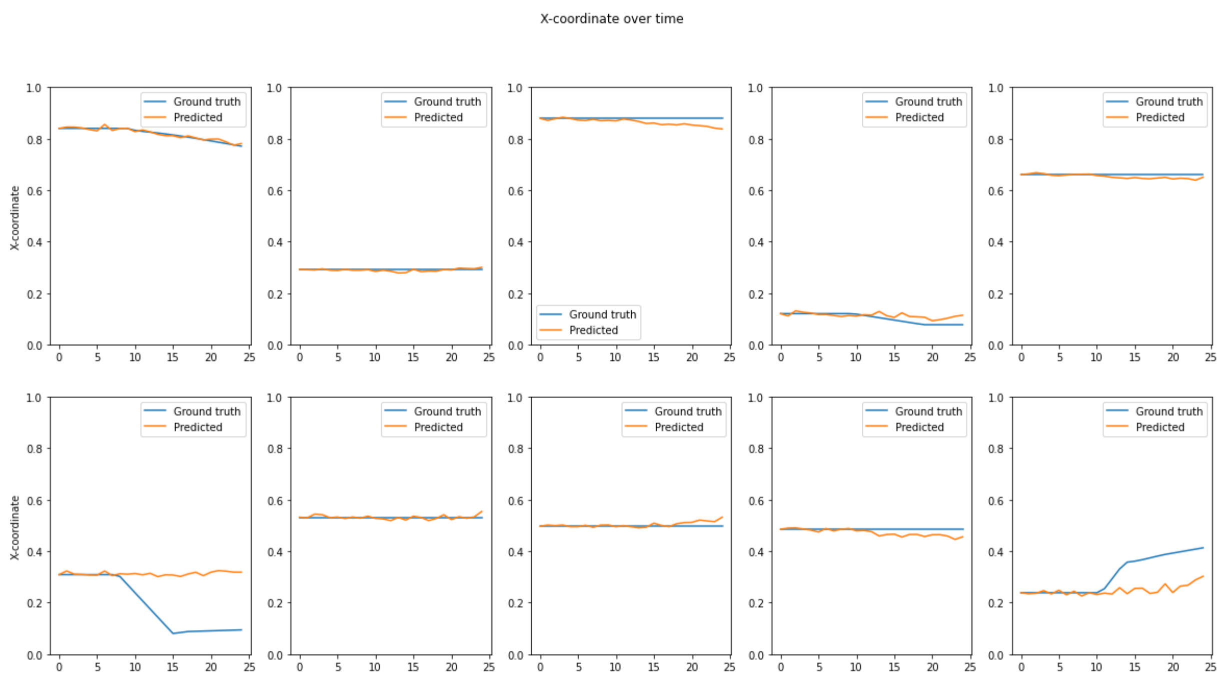


(c)

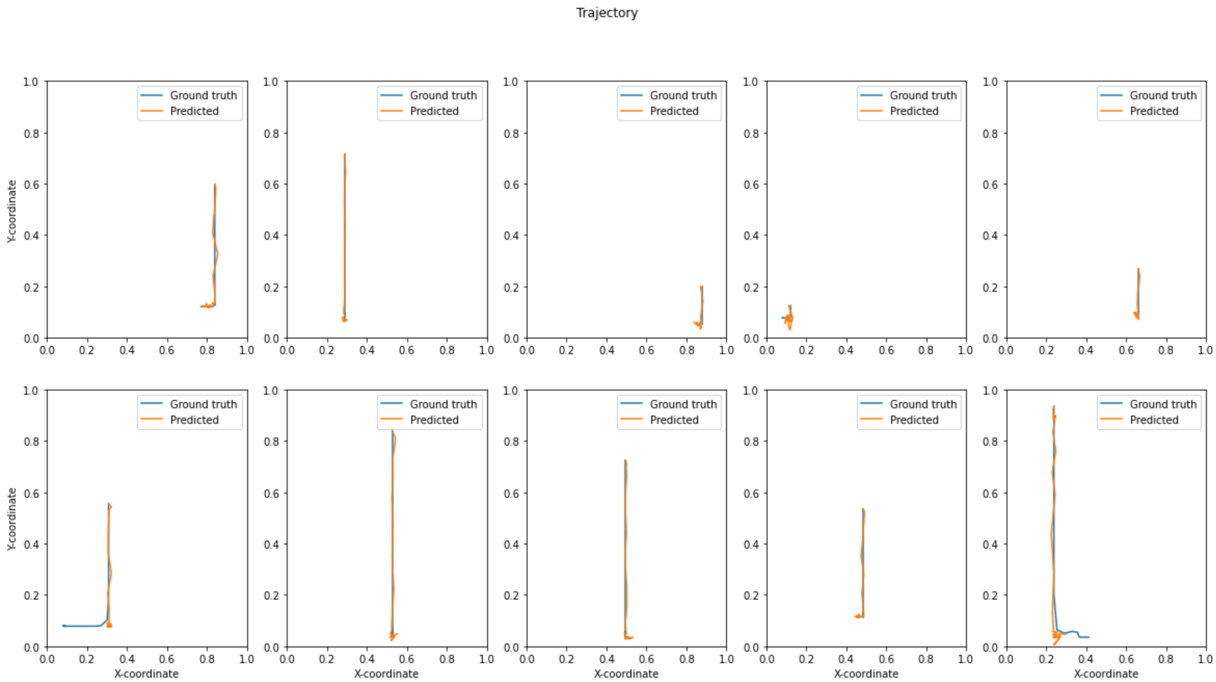
Figure 16. The first 10 test-set predictions of GRU on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(a)


X-coordinate over time
Ground truth
Predicted
X-coordinate
Frame number

(b)



(c)

Figure 17. The first 10 test-set predictions of LSTM on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.

For the Reservoir Computing (RC) models, the Grid Search was performed using the following search space:

- Reservoir size: 150, 200, 250 neurons;
- Leaking rate: 0.7, 0.9;
- Spectral radius: 0.1, 0.3, 0.5;
- Ridge parameter: 0.01, 0.1.

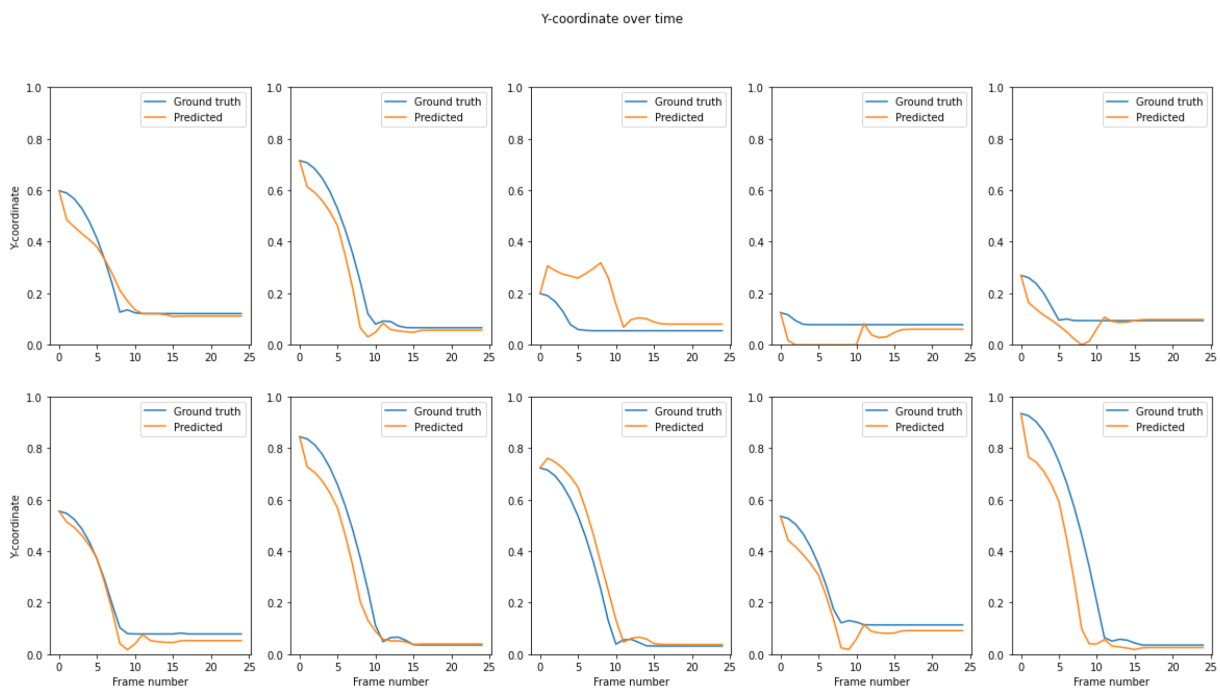
The optimal set of hyperparameters found by Grid Search includes 200 neurons in the reservoir, a leaking rate of 0.9, a spectral radius of 0.1, and a Ridge parameter of 0.1.

I noticed that the RC models exhibit less noise in their predictions compared to traditional RNNs. However, they often incorrectly predict that the object moves upward before falling, and they also struggle to identify collisions and predict post-collision movements. The test set RMSE of RC models is shown in Table 7, which reveals that traditional RNNs provide closer predictions.

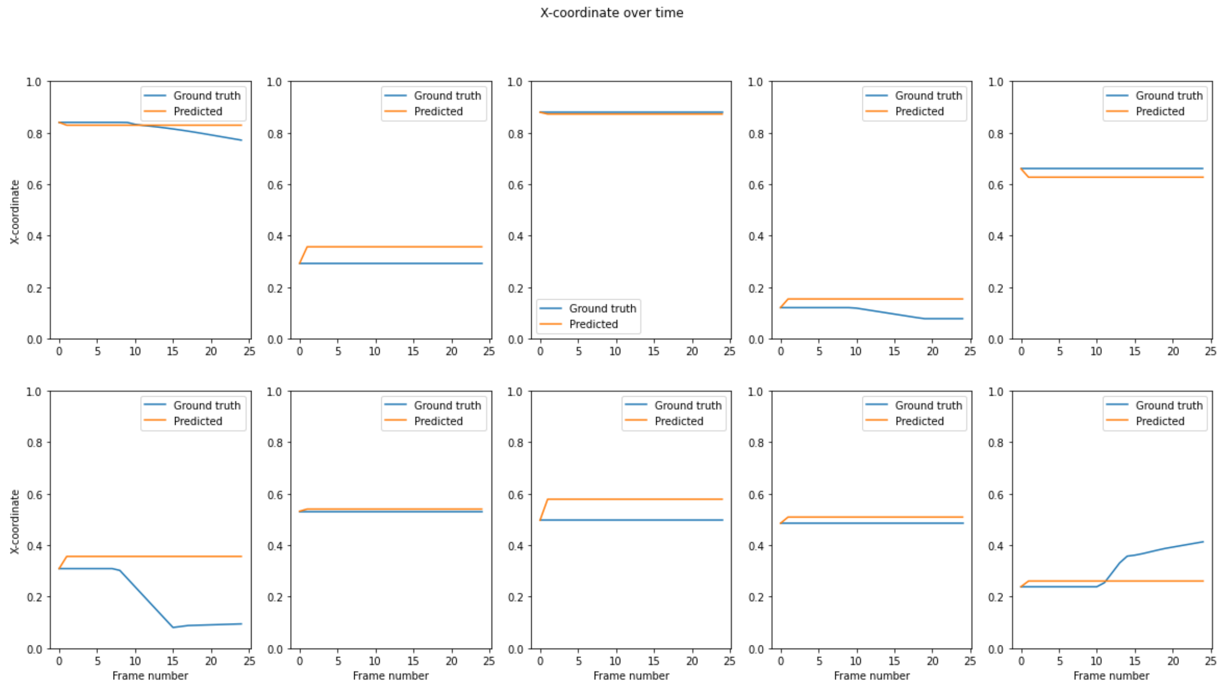
Figures 18-21 display the visualized predicted trajectories, where some of the RC models' predictions deviate significantly from the ground truth.

Model	RMSE
ESN	0.0716
Sequential ESN	0.0978
Parallel ESN	0.0741
Grouped ESN	0.1233

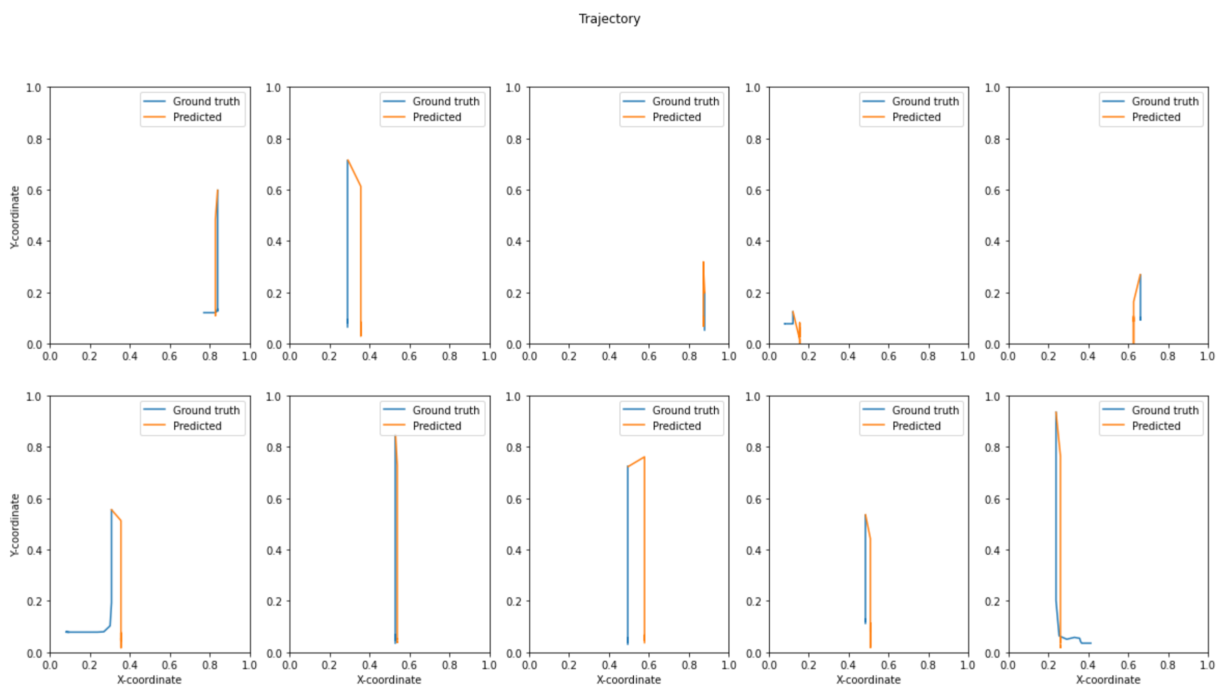
Table 6. RMSE of RC models on predicting trajectories with collisions from the test set.



(a)

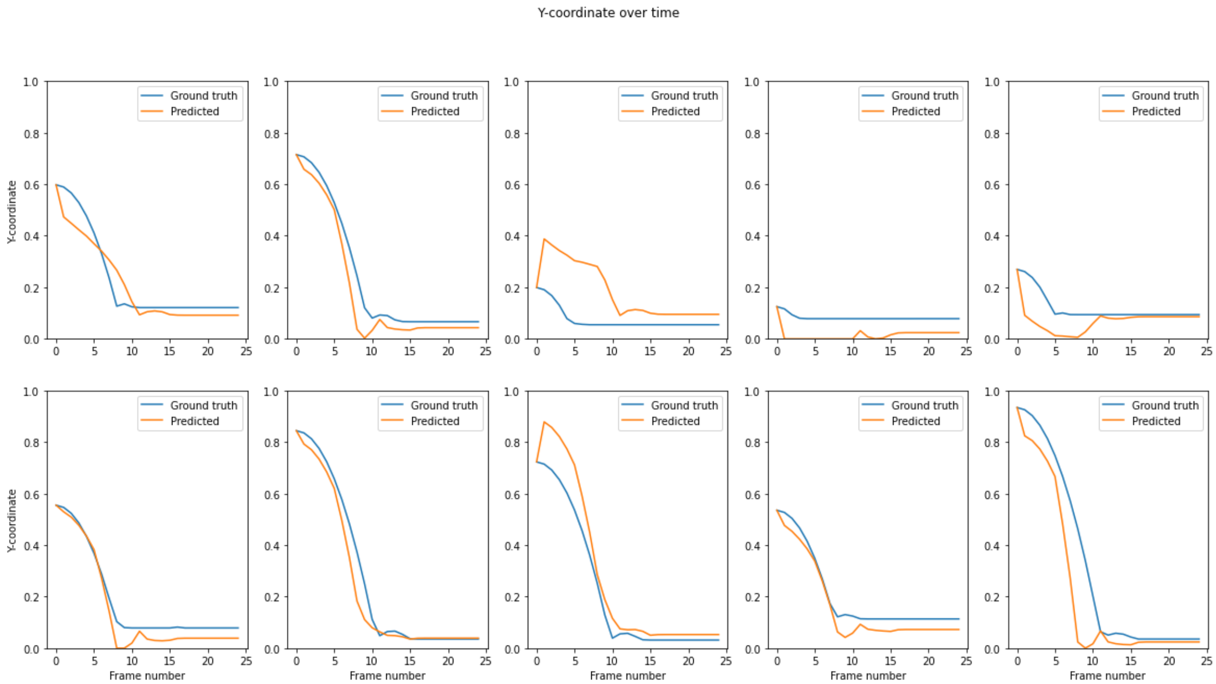


(b)

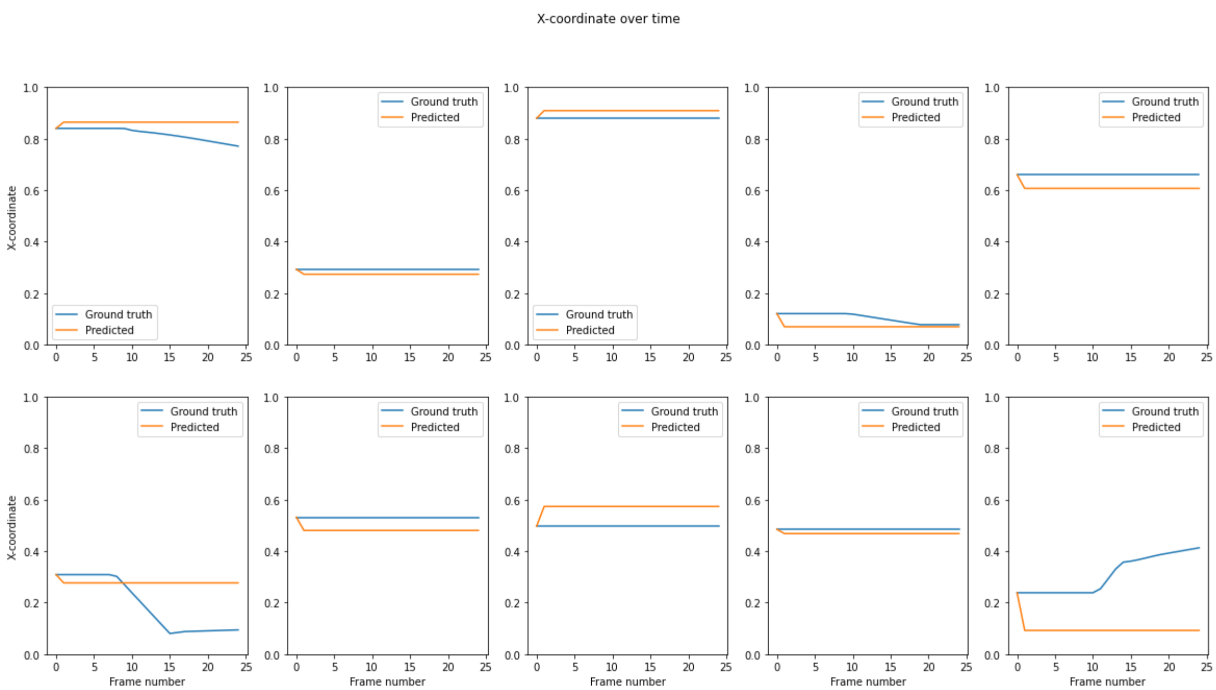


(c)

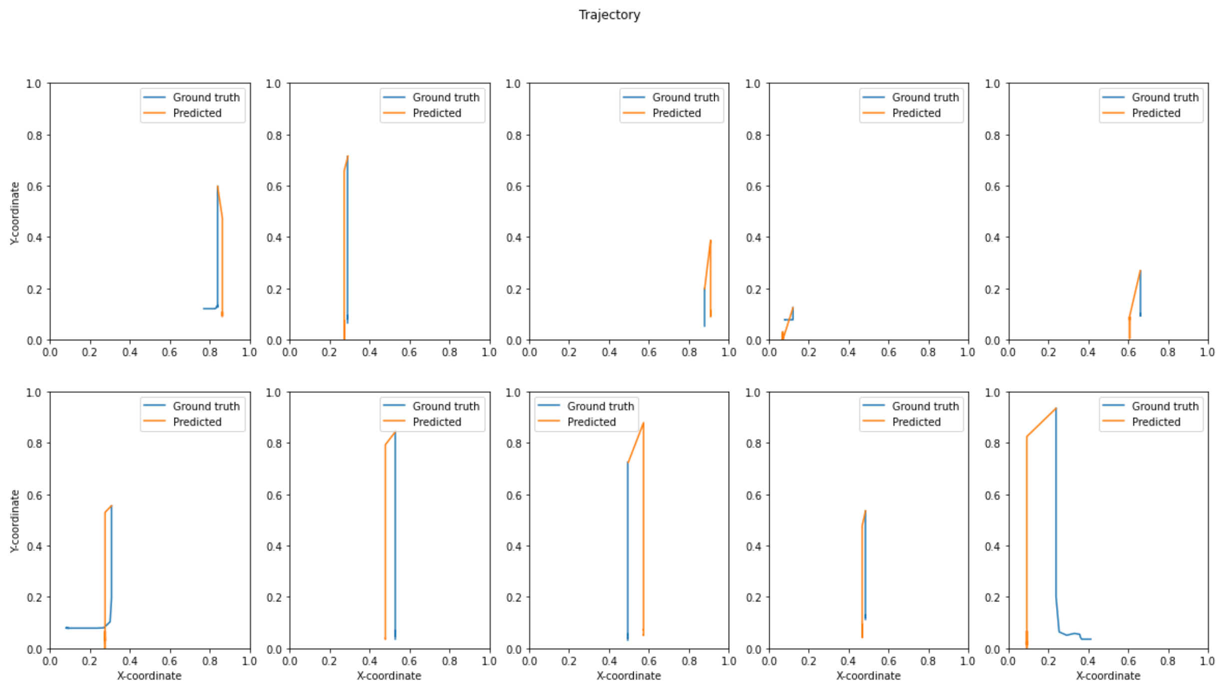
Figure 18. The first 10 test-set predictions of ESN on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.


Y-coordinate over time
Y-coordinate
Frame number
Ground truth
Predicted

(a)

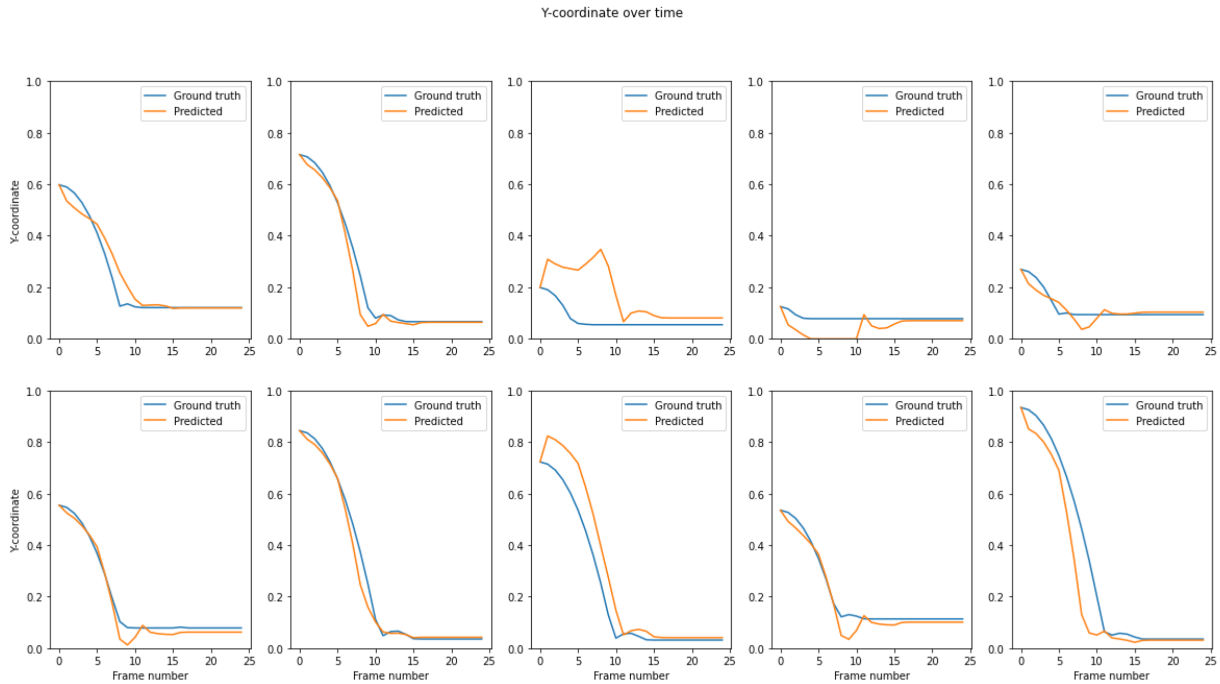

X-coordinate over time
X-coordinate
Frame number
Ground truth
Predicted

(b)

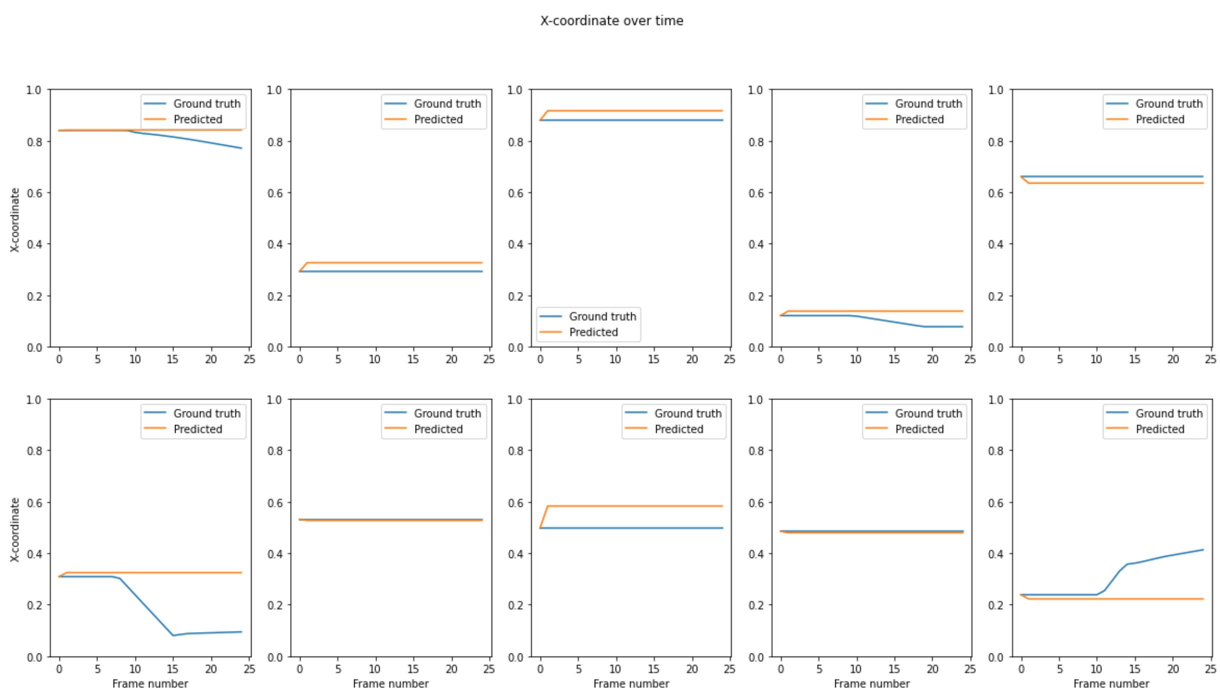


(c)

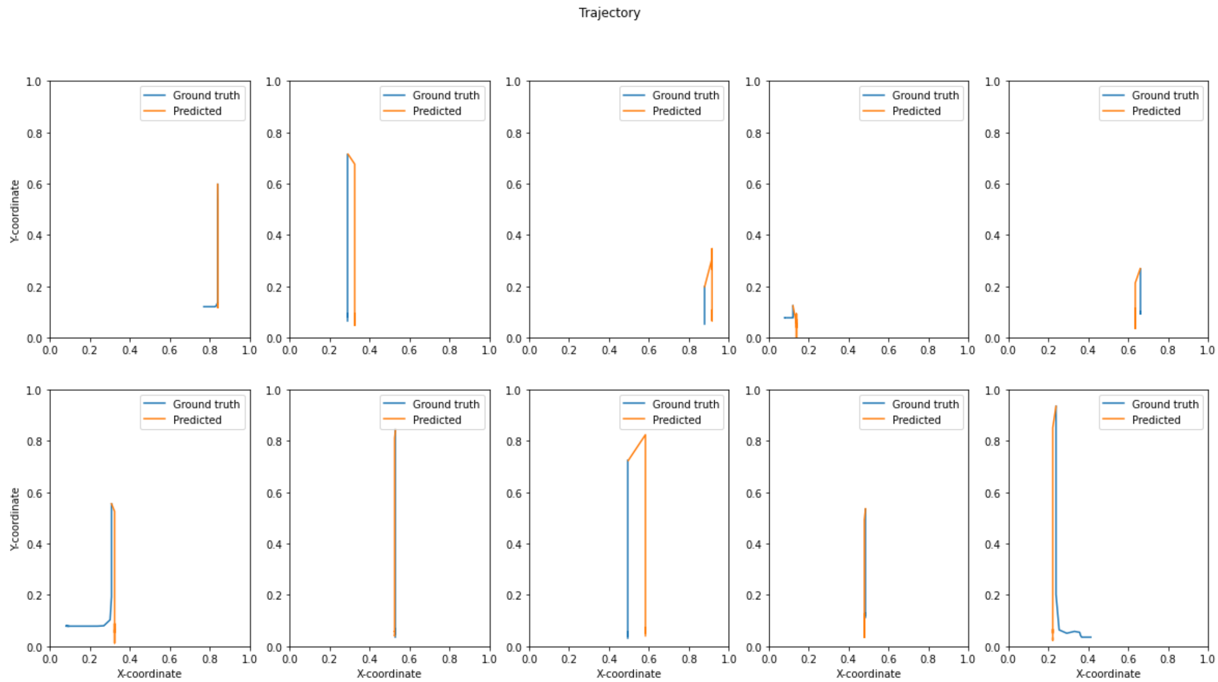
Figure 19. The first 10 test-set predictions of Sequential ESN on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(a)

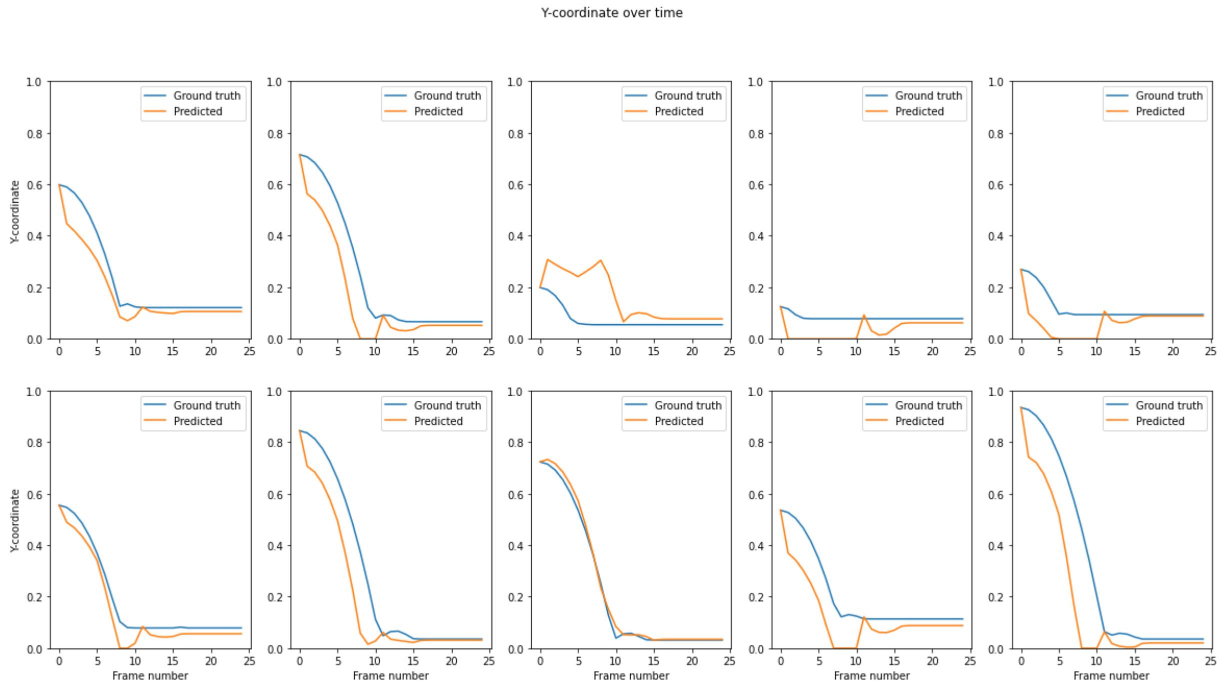

X-coordinate over time
Ground truth
Predicted
X-coordinate
Frame number

(b)

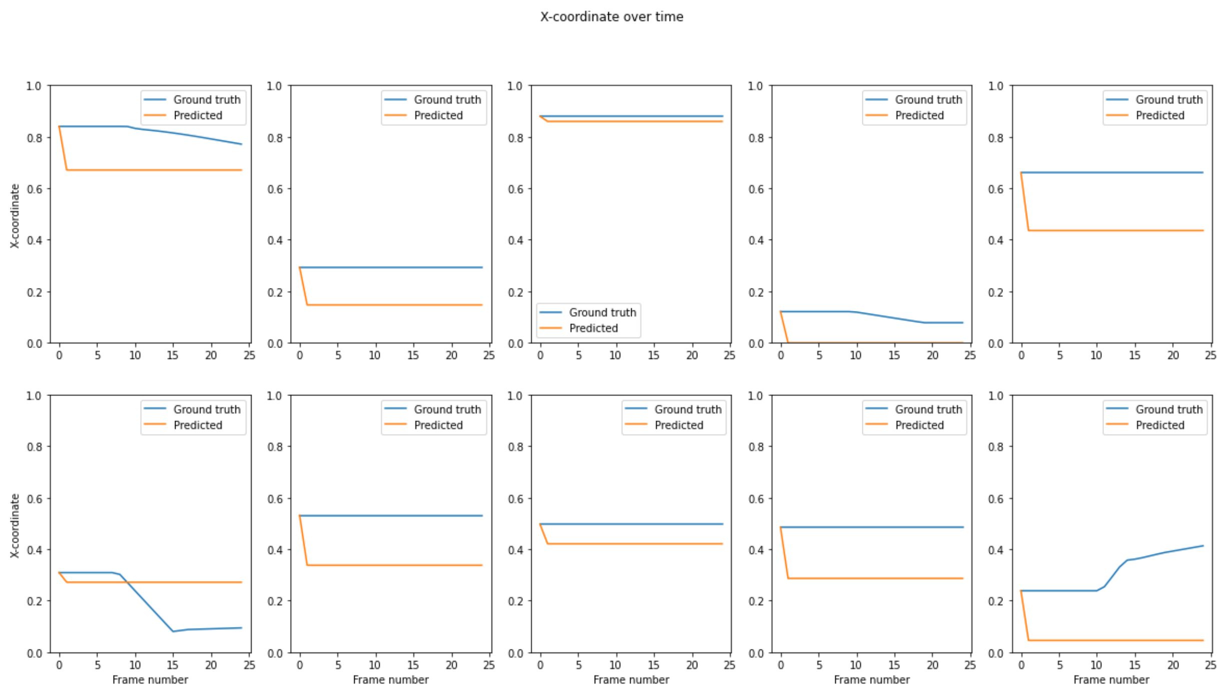


(c)

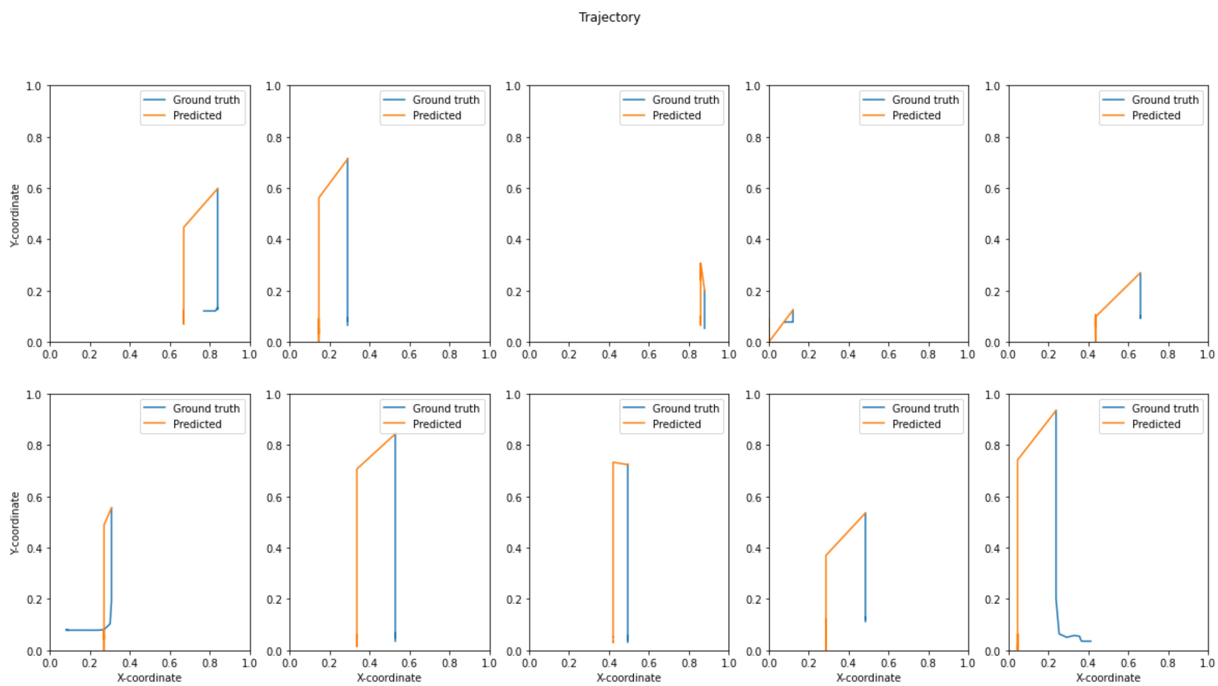
Figure 20. The first 10 test-set predictions of Parallel ESN on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.


Y-coordinate over time
Ground truth
Predicted
Y-coordinate
Frame number

(a)


X-coordinate over time
Ground truth
Predicted
X-coordinate
Frame number

(b)



(c)

Figure 21. The first 10 test-set predictions of Grouped ESN on scenarios with collisions. (a) Y-coordinate time series; (b) X-coordinate time series; (c) Full trajectory.

In conclusion, both traditional RNNs and RC models face challenges in predicting trajectories with collisions. While traditional RNNs generate closer predictions, both model types struggle to accurately predict collisions and post-collision movements. Further research is needed to improve the performance of these models in complex scenarios involving collisions.

5.3. Predicting the evolution of the entire scene

As an extension task, I trained the models with the goal to predict the movement of all three balls simultaneously, instead of focusing on just one ball. This involves predicting 144 values (x- and y-coordinates of 3 balls on 24 simulation frames) instead of 48 values (x- and y-coordinates of only a single ball on 24 simulation frames).

For the traditional RNNs, I started with the Grid Search again, using the same search space as before. The optimal hyperparameters were found to be 128 hidden neurons, 2 recurrent layers, and 0 dropout rate for LSTM and GRU, and 3 recurrent layers, 128 hidden neurons, and 0 dropout rate for Vanilla RNN.

The test set RMSE for the traditional models is given in Table 7. Surprisingly, the RMSE was found to be approximately the same as for the single-ball trajectory prediction with collisions, even though the number of predicted values increased threefold. This means the error per predicted value was three times smaller. However, the visual assessment of the results did not show significant improvements; the models still failed to accurately predict the trajectories of the balls. Figures 22-24 show the predicted and ground truth trajectories of each ball and the entire scene evolution done by traditional RNN models on the test set.

Model	**RMSE**
Vanilla RNN	0.0494
GRU	0.0466
LSTM	0.0460

Table 7. RMSE of traditional models on predicting the scene evolution using the data from the test set.

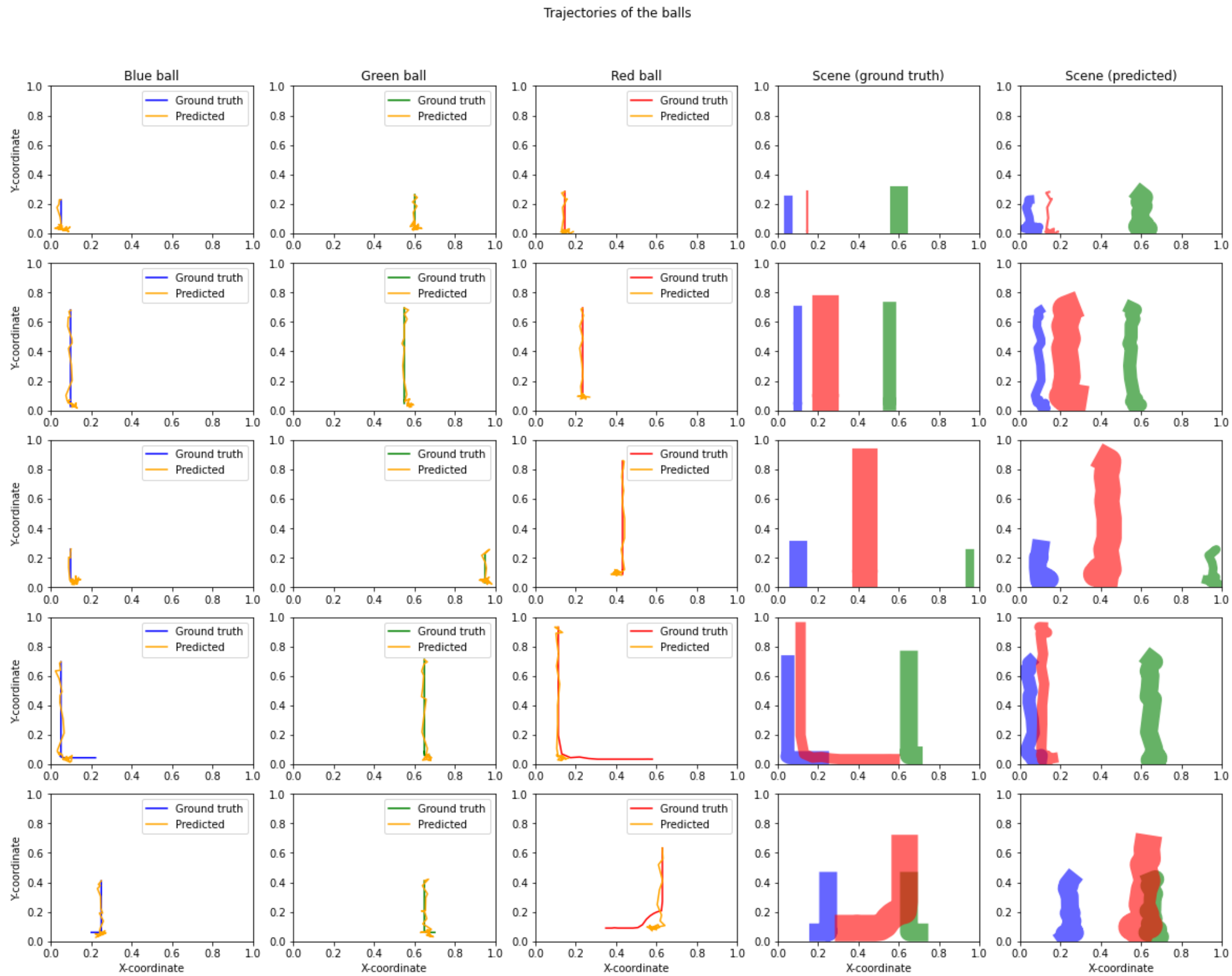


Figure 22. First 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by Vanilla RNN on the test set.

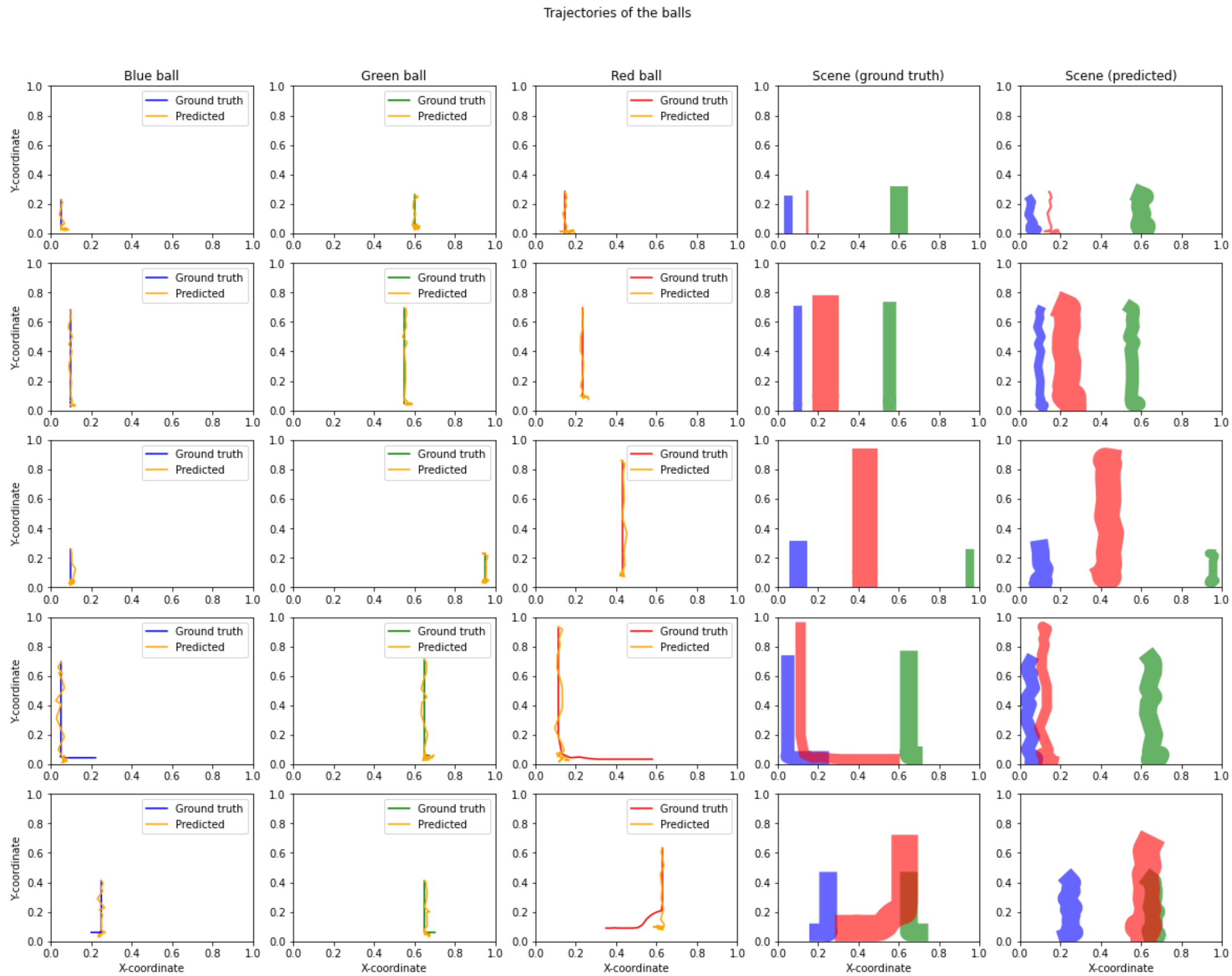


Figure 23. First 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by GRU on the test set.

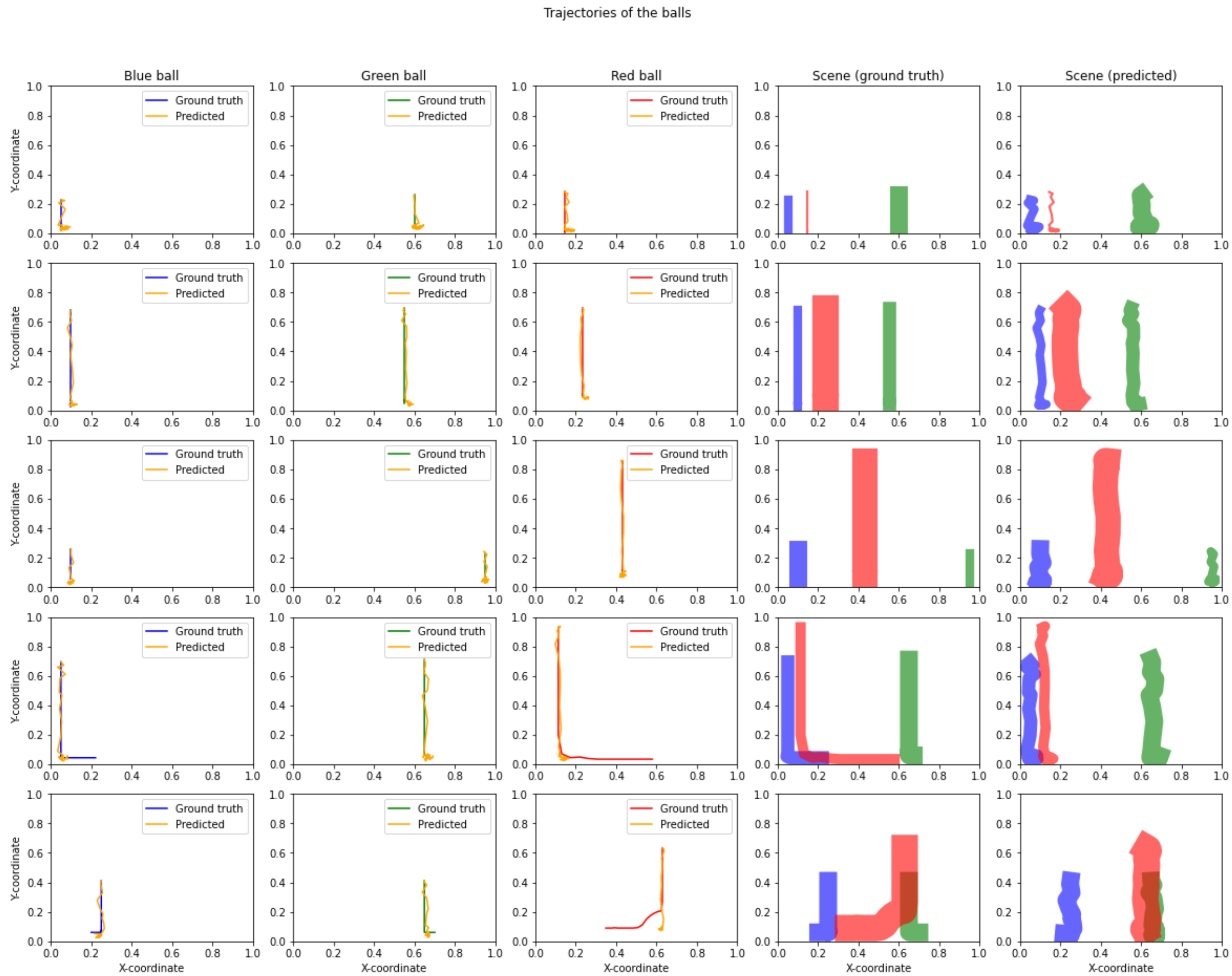


Figure 24. First 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by LSTM on the test set.

For the RC models, I changed the search space to

- Reservoir size: 200, 250, 300, 350
- Leaking rate: 0.7, 0.9
- Spectral radius: 0.1, 0.3, 0.5
- Ridge parameter: 0.01, 0.1

This change was motivated by my assumption that a more complicated task would require a larger reservoir size. The optimal set of hyperparameters that was found by Grid Search is 300 neurons in a reservoir, a leaking rate of 0.9, a spectral radius of 0.3, and a Ridge parameter of 0.1.

The test set prediction error for each RC model is shown in Table 8. These errors were significantly larger than those of the traditional RNN models for the same task. However, the visual assessment revealed an advantage of the RC models: for incorrectly predicted collisions, they still reasonably predicted the movement of the balls caused by such collisions. This effect was most prominent in the visualizations of the predicted scene evolutions done by the Parallel ESN model. Figures 25-28 show the ground truth and predicted trajectories of each ball, along with the ground truth and predicted scene evolution made by each RC model using the test set.

Model	**RMSE**
ESN	0.1867
Sequential ESN	0.2160
Parallel ESN	0.2176
Grouped ESN	0.1898

Table 8. RMSE of RC on predicting the scene evolution using the data from the test set.

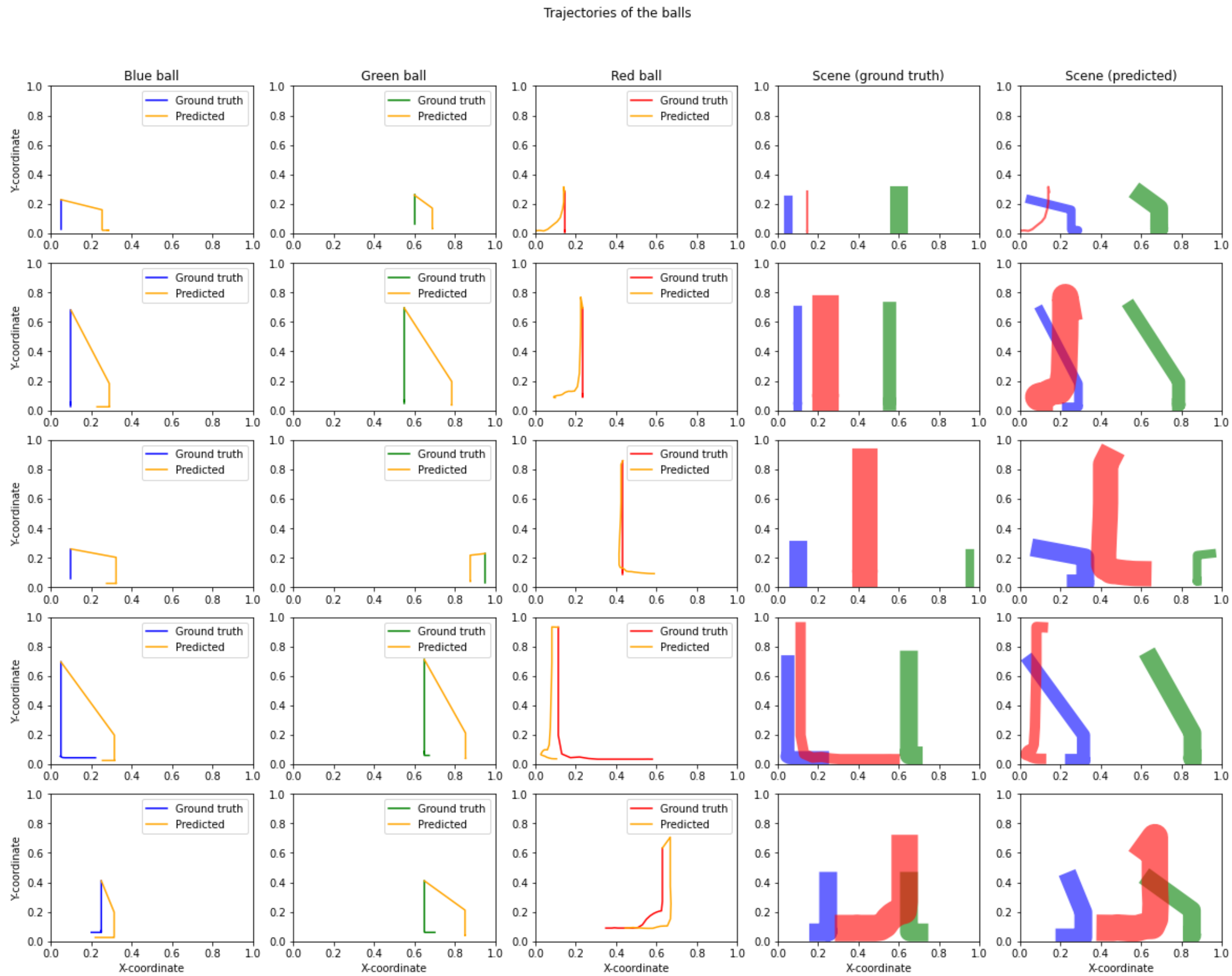


Figure 25. The first 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by simple ESN on the test set.

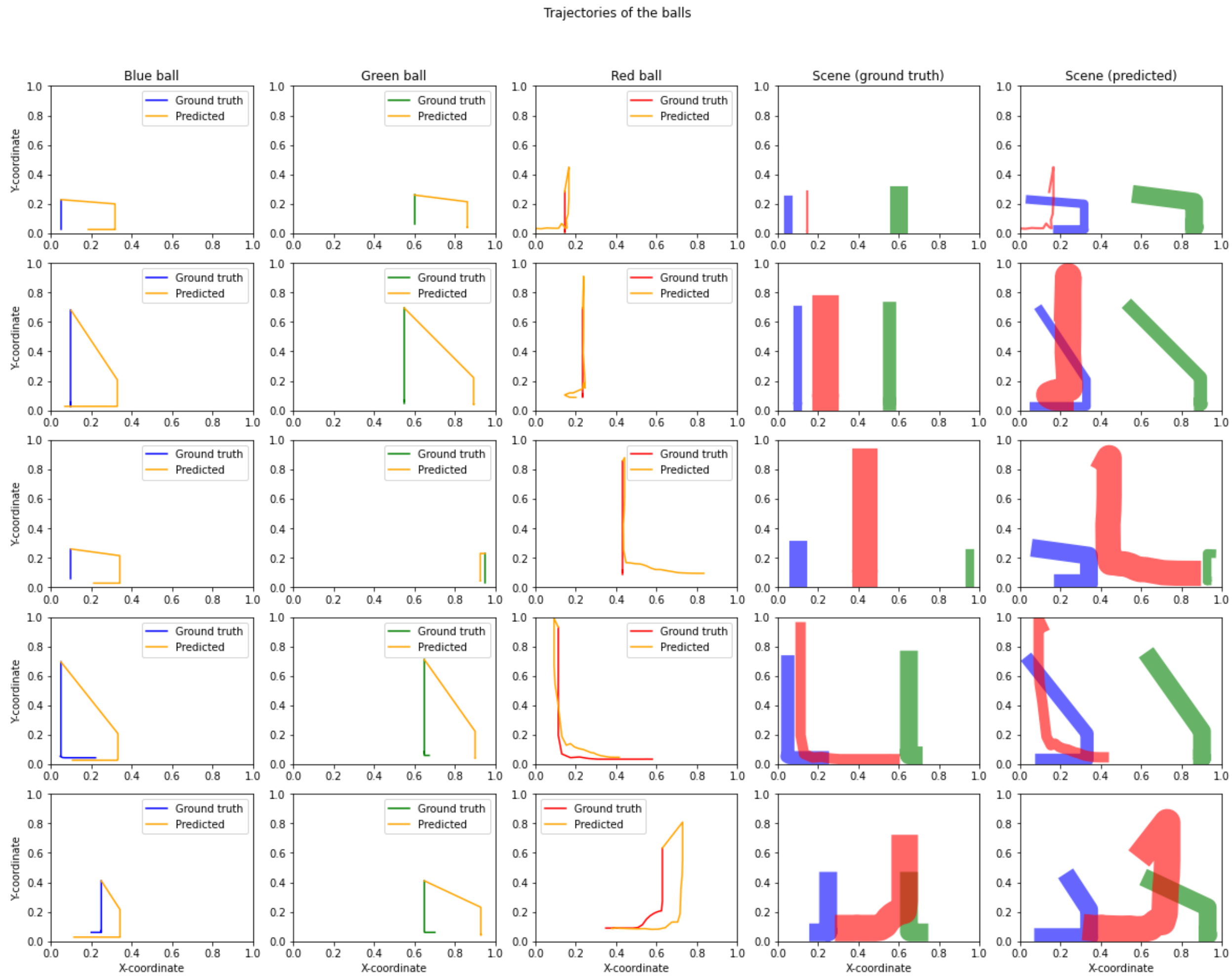


Figure 26. The first 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by Sequential ESN on the test set.

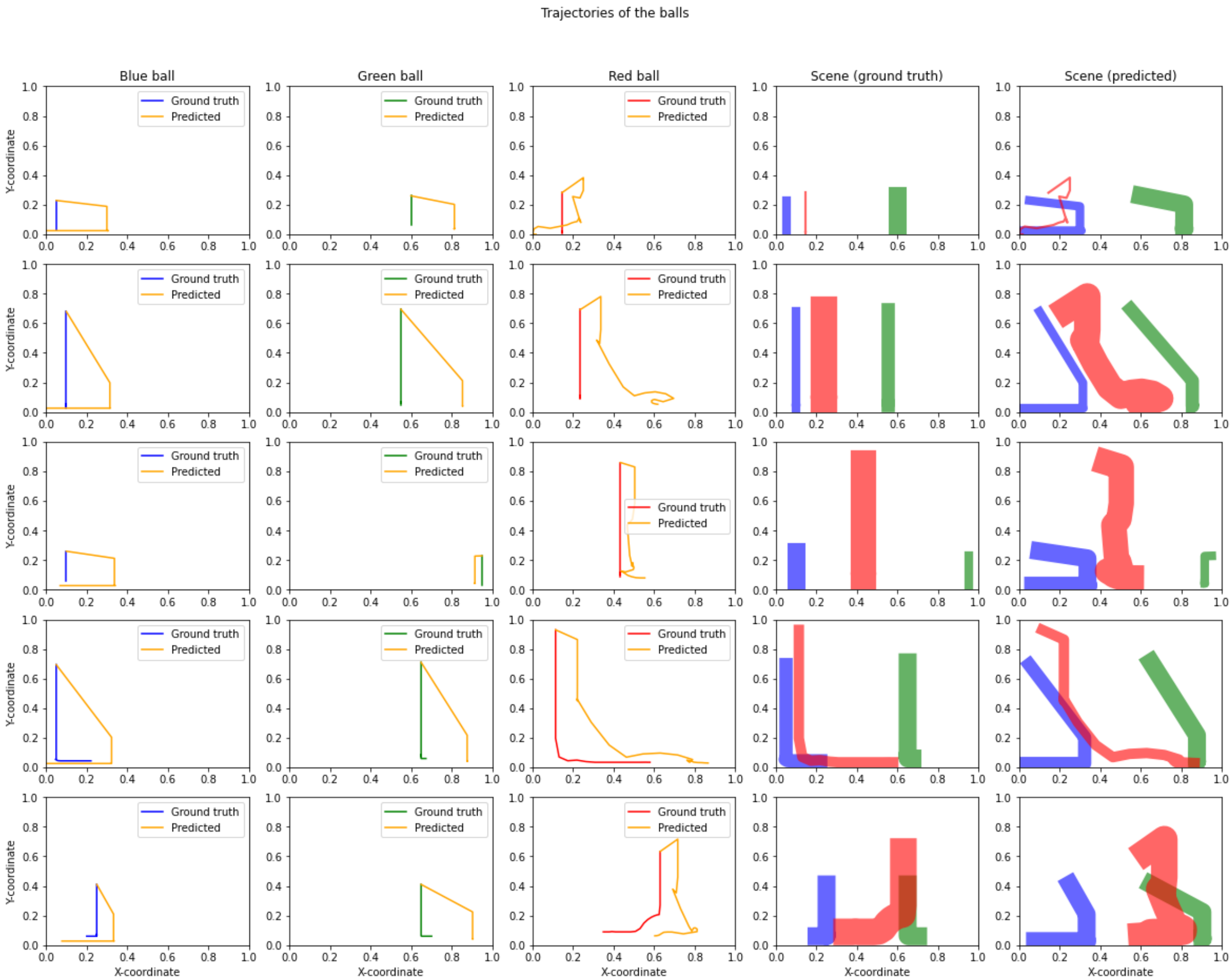


Figure 27. The first 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by Parallel ESN on the test set.

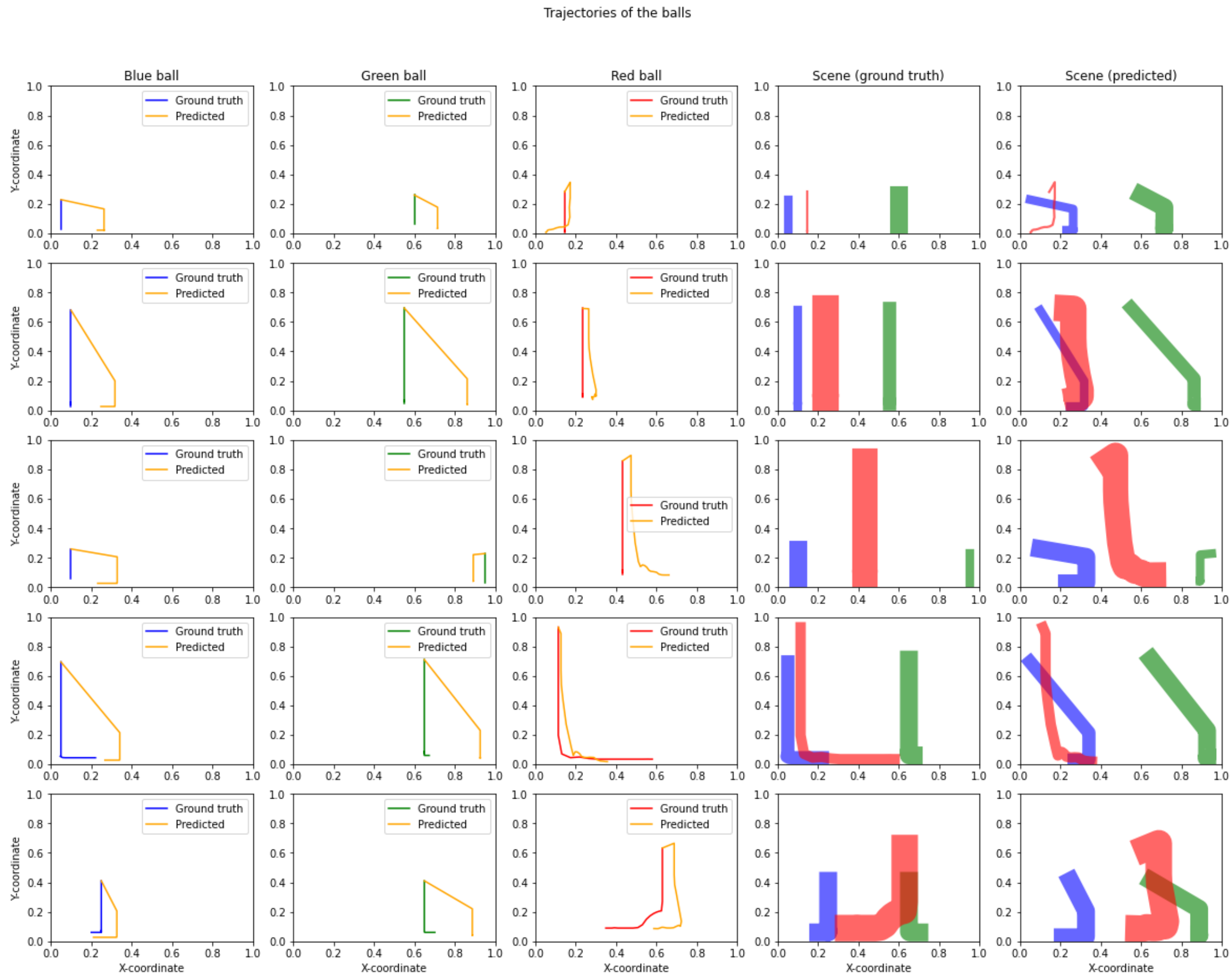


Figure 28. The first 5 predicted and ground-truth trajectories of each ball, ground-truth, and predicted scene evolution done by Grouped ESN on the test set.

In summary, traditional RNN models performed better in terms of RMSE on predicting the entire scene evolution, but RC models demonstrated the ability to reasonably predict the movement of balls for incorrectly predicted collisions. Both model types still struggled to accurately predict the trajectories of the balls and further improvements are needed.

Section 6. Conclusion and Future Work

In this Capstone Project, I investigated the performance of traditional Recurrent Neural Networks (RNNs) and Reservoir Computing (RC) models in predicting the trajectory of a ball in a simulated 2D environment. My primary focus was on comparing the performance of Vanilla RNN, Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), Echo State Network (ESN), and its deep variations (Sequential ESN, Parallel ESN, and Grouped ESN) in three different tasks: predicting free-fall trajectories, predicting single-ball trajectories with collisions, and predicting the evolution of the entire scene with three balls.

For the task of predicting free-fall trajectories, all models showed satisfactory results, with LSTM and GRU models achieving the best performance. In the more complex task of predicting single-ball trajectories with collisions, the performance of all models decreased, but LSTM demonstrated the least noisy predictions. For Reservoir Computing models, the amount of noise in predictions was drastically less, but they were less accurate than traditional RNN models.

In the extension task of predicting the evolution of the entire scene, traditional RNN models showed surprisingly better results compared to single-ball trajectory prediction with collisions. Despite the increase in the number of predicted values, the error per predicted value was three times smaller. On the other hand, while RC models' error rates were higher than traditional RNNs, they demonstrated an ability to reasonably predict the movement of balls for incorrectly predicted collisions.

My study highlights the strengths and weaknesses of traditional RNNs and RC models in predicting complex ball trajectories in a 2D environment. Traditional RNNs, particularly LSTM and GRU, showed better overall performance, but RC models demonstrated some advantages in specific situations, such as predicting the movement of balls in incorrectly predicted collisions. While both model types struggled to accurately predict ball trajectories in complex scenarios, this research provides valuable insights into the capabilities of RNNs and RC models for trajectory prediction tasks and potential improvements that can be made for future studies. **Obtained results allow me to speculate the conclusion that Reservoir Computing models are doing a better job learning the underlying physics compared to traditional RNNs but struggle to make accurate predictions.**

Future work can focus on expanding the complexity of the simulated environment by including more objects on the scene, introducing different types of objects (e.g., different shapes and static objects), transitioning to a 3D environment, and ultimately using real-world data to validate the models' performance. These directions would allow for a more comprehensive understanding of the models' capabilities in predicting trajectories in more realistic and diverse scenarios.

Throughout the course of this project, I gained valuable experience and skills in several aspects of machine learning and deep learning research. I learned how to use PyTorch to architect and train deep learning models effectively. I also developed the ability to manipulate data and prepare it for usage with PyTorch, ensuring that the models receive the appropriate input for training and evaluation. I discovered the importance of heuristics in gaining insights into the data and how they can help guide the choice of an appropriate model for a given problem. Additionally, I developed a deeper understanding of hyperparameters and how to optimize them through techniques like Grid Search, which is essential for enhancing the performance of machine learning models.

Furthermore, I delved into the world of Echo State Networks, learning about their unique architecture and how to implement them using the Reservoirpy framework. This exploration allowed me to appreciate the potential benefits and limitations of Reservoir Computing models in comparison to traditional RNNs. Lastly, I applied a bottom-up approach to research, which involved starting with simple scenarios and gradually increasing complexity, to better understand the strengths and weaknesses of the models under study.

This Capstone Project has provided me with a solid foundation in machine learning and deep learning techniques, and the insights gained from this research will undoubtedly be beneficial in my future endeavors in the field of Artificial Intelligence.

References

Bakhtin, A., van der Maaten, L., Johnson, J., Gustafson, L., & Girshick, R. (2019, August 15). *PHYRE: A new benchmark for physical reasoning*. arXiv.org. Retrieved from https://arxiv.org/abs/1908.05656

Bengio, Y., Simard, P., & Frasconi, P. (1994, March). *Learning long-term dependencies with gradient descent is ... - IEEE xplore*. IEEEXplore. Retrieved from https://ieeexplore.ieee.org/document/279181

Malik, Z. K., Hussain, A., & Wu, Q. M. J. (2016, June). *Multilayered echo state machine: A novel architecture and algorithm*. Retrieved from https://www.researchgate.net/publication/304192098_Multilayered_Echo_State_Machine_A_Novel_Architecture_and_Algorithm

Mottaghi, R., Bagherinezhad, H., Rastegari, M., & Farhadi, A. (2015, November 12). *Newtonian image understanding: Unfolding the dynamics of objects in Static Images*. arXiv.org. Retrieved from https://arxiv.org/abs/1511.04048

Mottaghi, R., Rastegari, M., Gupta, A., & Farhadi, A. (2016, March 17). *"What happens if..." Learning to predict the effect of forces in images*. arXiv.org. Retrieved from https://arxiv.org/abs/1603.05600

Lukosevicius, M. (2012). *A practical guide to applying Echo State Networks*. Retrieved from https://www.ai.rug.nl/minds/uploads/PracticalESN.pdf

Pascanu, R., Mikolov, T., & Bengio, Y. (2013, May 26). On the difficulty of training recurrent neural networks. PMLR. Retrieved from https://proceedings.mlr.press/v28/pascanu13.html

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc.

Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high performance-deep-learning-library.pdf

Song, Z., Wu, K., & Shao, J. (2020, April 12). *Destination prediction using Deep Echo State Network*. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0925231220305506

Trouvain, N., Pedrelli, L., Dinh, T. T., & Hinaut, X. (2020). Reservoirpy: An efficient and user-friendly library to design Echo State Networks. *Artificial Neural Networks and Machine Learning – ICANN 2020*, 494–505. https://doi.org/10.1007/978-3-030-61616-8_40

Verzelli, P., Alippi, C., & Livi, L. (2019, September 25). *Echo State Networks with self-normalizing activations on the hyper-sphere*. Nature News. Retrieved from https://www.nature.com/articles/s41598-019-50158-4

Wolchover, N. (2018, April 18). *Machine Learning's 'amazing' ability to predict chaos*. Quanta Magazine. Retrieved from https://www.quantamagazine.org/machine-learnings-amazing-ability-to-predict-chaos-20180418/

Wu, H., Chen, Z., Sun, W., Zheng, B., & Wang, W. (2017, August). *Modeling trajectories with recurrent neural networks*. Retrieved from https://www.ijcai.org/Proceedings/2017/0430.pdf

Ying, X. (2019). *An overview of overfitting and its solutions*. IOPScience. Retrieved from https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022

Zaremba, W., Sutskever, I., & Vinyals, O. (2015, February 19). *Recurrent neural network regularization*. arXiv.org. Retrieved from https://arxiv.org/abs/1409.2329

HCs

#rightproblem: While defining the problem that I want to solve with my capstone, I was following a problem analysis process. Stemming from my desire to do research that combines Computer Science and Physics, I started by researching different aspects of how these fields overlap. By doing an initial literature review, I identified the field of Intuitive Physics as a currently developed research direction. I couldn't find any research paper that implements a bottom-up approach - starting from simplistic scenarios, and slowly complicating tasks to reach real-world complexity. Rather than that, all the research papers that I encountered were trying to take the real data and fit the models that learn specific aspects applicable to such data. This guided an approach that I wanted to implement in my capstone - start from an oversimplified intuitive physics task, and slowly increase the complexity. The rest of the work lay in scoping - from finding efficient interventions to the system to predicting trajectories using Recurrent Neural Networks, to predicting trajectories using RNNs in scenarios with no more than 3 balls. As a result, I defined a problem of predicting complex ball trajectories in a 2D environment, considering collisions and gravitational effects.

The strength of the HC application lies in covering all the key aspects of the #rightproblem: understanding the current state (i.e. current state of research), analyzing the obstacles (i.e. my lack of knowledge and how I can effectively gain it in the process), and navigating the scope. As a result, I had a clearly scoped capstone that can be explored in depth over the course of two semesters. This in-depth understanding of the problem's depth and scope allowed for a well-informed exploration of various models and techniques, ultimately leading to a deeper comprehension of the problem's complexity, potential solutions, and future work suggestions.

#breakitdown: While I imagined a roadmap of my work with the incremental complexity approach, I still needed to break down each step of that roadmap into specific tasks, the successful completion of which ensures the steady progress of my research. I have broken down the entire process into the following tasks:

- Planning the infrastructure (which included setting up Jira, Github, Cloud folders, and development environment);
- Conducting literature review;
- Planning the code architecture;
- Implementing the algorithm within the currently researched step (which included a dataset generation, creating and debugging machine learning models, and tuning the hyperparameters);
- Grooming new tasks and further research directions;
- Creating and iterating on the capstone write-up.

When it comes to the problem that I researched, I effectively broke down the complex problem of predicting ball trajectories in a 2D environment into tractable components. I started by predicting the free-fall trajectories of a single ball, then proceeded to predicting single-ball trajectories with collisions, and finally extended to predicting the evolution of the entire scene with multiple balls. This decomposition allowed for a step-by-step exploration of the problem, considering various deep learning and reservoir computing models.

My application of this HC is strong because it demonstrates a methodological approach to understanding the problem by dividing it into manageable parts. My proposed decomposition enabled the exploration of each component in detail and allowed for a more comprehensive understanding of the challenges involved. By tackling each component separately and then combining the findings, I was able to reach a clear conclusion on the capabilities of RNNs and RCs to predict trajectories in classical mechanics.

#modeling: In this paper, I proposed various models to predict ball trajectories in a 2D environment. The set of proposed models included simple Feedforward Networks (during the naive initial stage of my exploration), followed by traditional Recurrent Neural Networks (Vanilla RNN, LSTM, and GRU) and Reservoir Computing models (Simple and Deep ESN architectures). I justified the choice of these models by their ability to capture temporal patterns and effectively handle sequential data. Through this research, I compared these models in each of the explored settings (free-fall, single-ball scenarios with collisions, and entire scene evolution), which allowed me to analyze their strengths and weaknesses in the context of trajectory prediction.

My application of #modeling is strong because it demonstrates the selection, tuning, implementation, and usage of the appropriate models to describe the system that I was exploring. By comparing different models, I effectively evaluate the performance of investigated models in predicting ball trajectories, which allowed me to provide insights into the effectiveness of each model. Additionally, in this paper, I acknowledge the simplifying assumptions made in the modeling process, such as the absence of external factors and constrained 2D Physics implementation.

#algorithms: This HC can arguably be considered a central HC for creating quality content for this project. Its applications manifested in multiple different stages.

To start with, I leveraged this HC to generate the datasets. This process involved invoking the Phyre simulator, processing the simulation data (by thresholding invalid simulations and implementing custom filters for each type of scenario that I was investigating during my research), and transforming this data into a custom data structure that is compatible with PyTorch framework.

Following the dataset generation, I applied algorithmic thinking to attempt to solve the problem of trajectory prediction in a 2D environment with colliding balls. To achieve this, I implemented multiple Deep Learning and Reservoir Computing models using such frameworks as PyTorch and Reservoirpy. Once the implementation was done, I compared the performance of traditional RNNs and RC models. To be able to do it, I had to understand not only how to build these models, but also how to train and evaluate them, which I expressed in my paper.

My application of this HC is strong because it identifies appropriate algorithmic strategies for trajectory prediction and justifies their selection based on their potential to handle sequential data. My paper goes beyond merely implementing the algorithms, as I provide a systematic evaluation and comparison of their performance. Part of the process was understanding how different hyperparameters affect the models' performance. By effectively implementing, testing, and debugging the algorithms, I showed a solid grasp of algorithmic thinking and the ability to apply it to solve complex, close-to-read-world problems.

#dataviz: In this paper, I apply #dataviz in multiple instances to create, interpret, and analyze data visualizations that provide insights into the performance of the different models in predicting ball trajectories. For different types of scenarios that I explored, I had to think about how to effectively structurize the visualizations to capture all the information that is essential to demonstrate. As such, one-dimensional free-fall visualizations only captured the ground truth and predicted time series of the Y-coordinate, while two-dimensional cases included an analogous visualization of the X-coordinate, as well as the X- vs Y-coordinate plots that showed the trajectory trace of the ball. For the scene evolution, I had to discard visualizations of individual coordinates, as it would make the visualizations unreasonably loaded. Rather than that, I focused on visualizing the movements of each ball individually and relative to each other. Admitting that visualizations do not constitute comprehensive metrics for assessing the machine learning models' performance, I also provided well-organized tables reporting the RMSE values of the models on the test sets.

The application of this HC in my paper is strong because I effectively generate detailed data visualizations that are appropriate for the problem context. By using a combination of trajectory plots and tables, these visualizations facilitate the understanding of the models' performance in both qualitative and quantitative ways. Appropriate plot types were shown to convey different information – such as line plots were used to visualize time series, while histogram was used to support the statistical testing. Furthermore, the visualizations are presented in a clear and organized manner, making them easily interpretable and insightful for the reader. Each visualization is appropriately labeled, with the legend presented when needed, according to professional guidelines. This effective use of data visualization techniques demonstrates the ability to convey complex information in an accessible and meaningful way.

#designthinking: In this paper, I apply #designthinking through the process of developing and refining data generation algorithms and deep learning models to predict ball trajectories in a simulated environment. I began the process by understanding what I aim to achieve with my research, namely attempt to achieve the best possible performance of the machine learning model on the trajectory prediction task. Next, I explored several different models from the Reservoir Computing paradigm. By incorporating the feedback obtained from Professors along the way, I also considered implementing traditional Recurrent Neural Networks, which resulted in a better performance on certain types of tasks and interesting findings for others.

A part of my application of this HC was by constantly reviewing the work I've done, both its technical aspects and the paper write-up. As such, I employed a pull request system on GitHub to review and make adjustments to my work with a fresh perspective. I've done it by incrementally pushing my work on GitHub, and rather than merging it to the master branch right away, I was living the pull request open for a few days to be able to review it later, which sometimes resulted in identified issues that I kept track of by creating corresponding bug tickets in Jira.

A strong application of this HC is demonstrated by my use of external instruments like GitHub and Jira to keep track of drafting progress and pull reviews to evaluate my progress. By engaging in continuous evaluation and refinement of both research and writeup, I showed my commitment to an iterative design thinking process, which resulted in a quality deliverable produced for my Capstone Project.

#strategize: My application of this HC is evident through the bottom-up approach that I chose for my research. This approach required me to develop a plan that incrementally increases the complexity of the task I am attempting to solve while maintaining clear objectives. For the problem that I tried to solve, I created a proper breakdown, identified the knowledge gaps (such as a proper understanding of RNNs and RC models), and once the gap was filled I exploited the strengths of the chosen deep learning models to try to solve the problem. I demonstrated a strategic approach in my paper through the use and tuning of different models and the systematic increase in complexity from single timestep prediction of free fall to full trajectory prediction of complex movement.

A part of my use of #strategize was through curating my work to always meet the following three criteria:

- It moves in the right direction, i.e. continuing the incremental increase of complexity should eventually transform my problem into trajectory prediction in real-world scenarios but not some other tangential problem;
- Every complexity increasing step must lead to significant new insights but must be tractable to implement building on the work I already have;
- At every step of my research the task I am currently attempting to solve must be defined clearly and unambiguously.

My application of this HC is strong because I effectively diagnosed the problem, developed a guiding policy in the form of a bottom-up approach, and implemented coherent actions that increase the complexity of the task incrementally.

#organization: My application of this HC in the paper is demonstrated through the clear and coherent structure that guides the reader through the research process. I started the paper with a well-defined introduction, providing context and outlining the problem at hand. I then proceed to discussing the methodology, implementation, and evaluation of various deep learning models for each task that I was trying to solve, organizing the narrative in the order of increasing task complexity. The paper concludes with a summary of the findings and a discussion of future work, which ties together the entire study.

Another significant part of my application of #organization was through organizing the complementary resources, namely Jira and GitHub. In the former, I divided all the tasks into two epics – for free fall scenarios and for scenarios with collisions - successful completion of each of which constitutes reaching an important milestone. I also had a clear hierarchy of tasks: epics consisted of stories – tasks, completion of which resulted in an outcome that has meaning when considered independently of others. A good example of a story from a free-fall epic would be "Exploring traditional RNNs performance on 1D free-fall scenarios," the completion of which results in gaining insights into how this type of model performs for free-fall prediction. Stories were further broken into subtasks, each of which can be implemented separately. An example of the task from the story above would be to explore only Vanilla RNN. Such an organization made it clear to me where exactly I am in the process of my research, as well as helped me to set up deadlines for specific parts of work that needs to be done.

My application of #organization is strong because the paper is arranged in a sophisticated manner that effectively communicates the message to the reader. The logical flow of content, from problem definition to future work, allows the reader to follow the research progression and understand the significance of each step in the process. It makes my paper both accessible and informative for the intended audience. Complimentary to the organization of the paper, my technical resources are organized in a way that makes it easy to move my research forward, which further strengthens my application of this HC.

#variables: In this paper, an application of this HC manifests in careful identification, classification, and examination of the relevant variables and parameters involved in the deep learning models used for time series prediction. To start with, I had to put work into defining a complete set of independent variables to be used as input to the model. Through the method of tries and errors, I identified that the minimum set of variables required to correctly predict the movement of the ball is the x- and y-coordinates of the ball and its size. The dependent variables in my problem are the predicted future coordinates of the ball. The deep learning models that I used to make the prediction have several intrinsic parameters that define their structure and functionality, which were treated as decision variables in the hyperparameter optimization problem but served as constraints during the model training and evaluation.

My application of this HC is strong because I accurately define and describe the relationships between the independent and dependent variables in the context of time series prediction, as well as clearly define and optimize the parameters of each deep learning model in the provided search space. The clear distinction between these variables and parameters aids in understanding the structure of the models, the nature of the problem, and the potential limitations and biases in the model predictions.

#plausibility: In my capstone paper, the application of #plausibility is demonstrated by thoroughly evaluating the premises and assumptions underlying the hypotheses related to the effectiveness of various deep learning models for time series prediction. This includes considering the foundational principles behind each model, such as the ability of RNNs to capture temporal dependencies, the advantages of LSTM and GRU architectures in mitigating the vanishing gradient problem, and the unique properties of ESNs, which rely on a non-trainable reservoir of neurons. Additional plausibility considerations are coming from the properties of data that are normalized between 0 and 1. As such, whenever the model predicted the values outside these boundaries, it would indicate implausible results, the analysis of which inspired an idea to post-process the predictions of deep learning models by ceiling and flooring them in the boundaries between 0 and 1.

My application of this HC is strong because I effectively identify and explain the premises and assumptions behind each hypothesis that I claim in the paper, ensuring that they are well-defined and internally consistent. This ensures a solid foundation for the research and strengthens the credibility of the drawn conclusions. The careful consideration of #plausibility prevented me from drawing conclusions based on poorly constructed hypotheses, ultimately contributing to more impactful research results.

#professionalism: The application of #professionalism in my paper manifests in the presentation of my work. My writing, while presenting my thought in a personalized style, adheres to the conventions of academic writing by using a formal tone and being clear and concise. I paid close attention to proofreading, excessively using AI tools such as Grammarly to assist me in catching typos and phrasing mistakes to ensure the absence of errors in the text. Additionally, I maintained a proper attribution of quotations throughout the paper with a consistent use of APA7 citation style. The formatting of my paper follows the expected standards of academic research, contributing to the overall appearance of the work.

I think that my application of this HC is strong because I demonstrated a thorough understanding of the nuances of effective communication within the context of Computer Science as a discipline. By maintaining a high level of professionalism in my presentation, I enhanced the credibility of my work, as well as demonstrated a commitment to adhering to the expectations of the field.

As an extra effort to strengthen the application of this HC, I'd like to highlight the effort that I put into designing my presentation for the Capstone Defense. In addition to crafting the slides to support my narrative in a professional manner, I also created an extensive set of slides dedicated solely to answering potential questions. While most of the prepared backup slides were not actually used in the Q&A session, the fact of their preparation constitutes a professional attitude to the defense session.

#composition: My application of this HC is evident in the approach I took to refine the paper drafts. I initially wrote down my thoughts as I had them in my head. Such writing constituted the majority of the content that I delivered as the Full Draft and Revised Full Draft. I then took an effort to improve my writing for the final capstone submission, which I achieved with the help of several AI tools. Firstly, I read the entire paper again, attempting to fix the sections that I thought were not written clearly. Then, the sections that I still was not satisfied with were put into ChatGPT (with GPT-4 base model) with a prompt to paraphrase in a clear and scientific style. However, I did not want my paper to sound dry and all scientific, so the output from ChatGPT was changed by me again to ensure that the writing style is consistent with the rest of the paper but the phrasing remains clear. Finally, I run the entire paper through Grammarly to fix typos, mistakes, and weird grammatical constructions.

This commitment to iterative improvement, which involves using AI resources, demonstrates my dedication to communicating in a clear and precise manner. This approach constitutes a strong application of #composition because not only it highlights my ability to skillfully adapt and refine my writing but also to employ modern computational tools to assist me in the process of creating quality work.

#sourcequality: In this paper, the application of #sourcequality manifested in the utilization of multiple sources to gather accurate information relevant to my research. I paid close attention to the source quality by considering the relevance, authority, and purpose of each source. I prioritized peer-reviewed articles (which almost exclusively form my references section) to ensure that the information is well-vetted and reliable.

By carefully selecting and evaluating the quality of the sources used in my research paper, I demonstrated a strong application of this HC. My approach allowed me to build a solid foundation for my work, ensuring that the information I present is well-supported, reliable, and relevant to my topic, ultimately contributing to enhancing the credibility of my research paper.

#constraints: In my paper, this HC was applied by identifying and imposing the key constraints within the problem of predicting ball trajectories in a simulated 2D environment. Among the constraints that I impose are the usage of this simulated 2D environment with the limitation of 3 balls on the scene. The constraints that I identified and that are crucial for understanding in my research are the implementation of physics in the simulator (which includes implementation of gravity, conservation of momentum, and restitution) and the limitations of the neural network models employed to predict trajectories. By acknowledging these constraints, I focused on developing models that could account for them in attempting to generate accurate predictions.

My application of #constraints is strong because I demonstrate a thorough understanding of the problem's inherent challenges and the limitations posed by constraints. In my paper, I effectively navigate these constraints by selecting and comparing various recurrent neural network (including Echo State Network) models that could potentially address them. By analyzing the performance of these models in the context of identified constraints, I present a comprehensive and well-reasoned approach to solving the problem that strongly showcases constraint satisfaction.

#purpose: In my paper, this HC was applied by clearly defining the research objectives and the underlying motivation behind the study. The purpose of the study was to investigate and compare the performance of traditional Recurrent Neural Networks and Reservoir Computing models in predicting ball trajectories in a simulated 2D environment. My paper focused on specific goals, such as evaluating the models' performance in predicting free-fall trajectories, single-ball trajectories with collisions, and the evolution of the entire scene with multiple balls under the guiding principles of accuracy, robustness, and the potential for future improvements.

My application of this HC is strong because I provide a coherent connection between the fundamental motivation and the research goals. By articulating my personal interests and goals of the study, I establish a clear framework that guides the study and makes it easier for readers to follow the chosen methods and their evaluation. Additionally, in this paper, I demonstrated a commitment to improving the state of the art in trajectory prediction and highlighted the need for further research in this area, also suggesting potential directions for this research. This strong focus on purpose ensures that my work remains relevant and valuable for potential future research.

LOs

#cs156-neuralnetworks:

In my paper, I applied this LO by implementing and analyzing various recurrent neural network architectures, such as Vanilla RNN, GRU, LSTM, and reservoir computing models like Echo State Network (ESN), Parallel ESN, Grouped ESN, and custom designed Sequential ESN. These models were trained and evaluated for their ability to predict ball trajectories in a simulated 2D environment. In my paper, I explore and compare the performance of these models on three different tasks: predicting free-fall trajectories, predicting single-ball trajectories with collisions, and predicting evolution of the entire scene with three balls. I implemented the traditional RNNs using PyTorch and ESNs using Reservoirpy frameworks.

The application of #neuralnetworks LO is strong in my paper because it demonstrates a comprehensive understanding of various RNN and ESN architectures with their strengths and weaknesses in the context of the task I attempted to solve. My paper provides detailed explanations and justifications for selecting these models for the trajectory prediction tasks. Moreover, my paper effectively compares the performance of different models within the same problem using a Root Mean Squared Error as a quantitative metric, and provides visual representations of the actual and predicted trajectories. The careful selection and analysis of various RNN models, combined with a rigorous evaluation of their performance, demonstrate my strong application of this LO in my work.

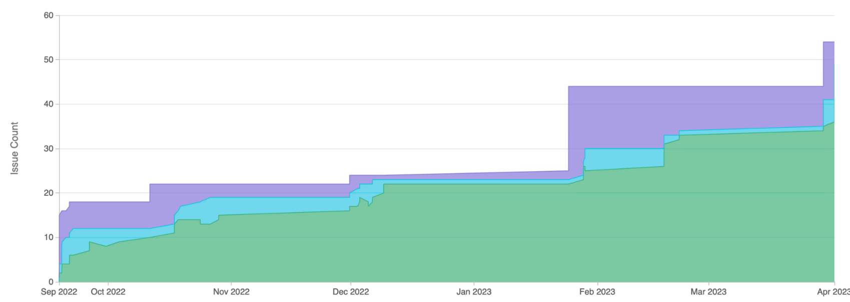
#cs156-

In my paper, I applied his LO by using various recurrent neural network and reservoir computing models as regression algorithms

regressionalorithm:	for the supervised learning task of predicting ball trajectories in a simulated 2D environment. My paper focused on the regression performance of Vanilla RNN, GRU, LSTM, ESN, Sequential ESN, Parallel ESN, and Grouped ESN models. These models were trained on input-output pairs and evaluated on their ability to predict the future trajectories of the ball based on the given initial states. The regression problem was defined as predicting the future time sequence of the ball positions given the initial locations and sizes of the balls present on the scene, according to the physical representations learned in the process of supervised learning. The application of the #regressionalgorithm is strong in my paper because I provide a clear definition of the regression problem and effectively apply the selected models to this task. I also provide a comprehensive performance comparison of different models using the chosen evaluation metric. By analyzing the strengths and weaknesses of various models in the context of regression tasks, I demonstrate a robust application of regression methods for supervised learning tasks.
#cs162- agile:	In my work, I applied this LO by adopting an agile approach in the research process. It allowed me to be flexible and responsive to the changes and findings that emerged throughout my work on the capstone project. By dividing my work into 2-week sprints, I ensured that my progress was continuously assessed and adjusted as needed, and regular task grooming sessions allowed for clarification and adjustment of my future roadmap. It was especially effective to apply this LO with the help of Jira – a tool that allowed me to visualize how I strategize my work, categorize tasks into epics, manage sprints, and assess my productivity using

analytical tools like cumulative flow diagram (shown below).

The application of #agile LO in my work is strong as it shows my adaptivity and responsiveness to changes in research direction while ensuring that my work progresses steadily. The agile approach allowed me to improve continuously, enabling me to adjust my research strategies and objectives as new information and findings became available. My knowledge and effective employment of project management tools (like Jira) only strengthen the application of #agile LO in the context of my capstone project.



#cs110-ComputationalSolutions:

In my work, I applied this LO by creating algorithmic solutions for data generation and machine learning problems within the intuitive physics domain that I chose. I systematically broke down the problem into a clear, ordered set of steps, for which I created programming implementations using Python, PyTorch and Reservoirpy deep learning frameworks, and Phyre library for intuitive physics benchmarking. I implemented different variations of deep learning models attempting to find one with the optimal performance. The process itself required the usage of custom data structures (such as `ClassicalMechanicsDataset` that extends abstract PyTorch `Dataset` to ensure that my data is compatible with the chosen machine learning framework), as well as computational techniques like Grid Search to optimize my solutions.

My work demonstrates a strong application of #ComputationalSolutions LO because it showcases my ability to effectively break down complex problems into manageable steps and implement an algorithmic solution using programming languages and deep learning frameworks. By comparing and contrasting different approaches, I demonstrated a deep grasp of the problem and what could potentially be effective ways to solve it. Combining it with my usage of data structures highlights my ability to create efficient computational solutions to complex problems.

#cs162-separationofconcerns:	In my work, I applied #separationofconcerns LO by designing my code architecture, which was organized into modules that handle specific tasks. The `data` module was responsible for the data generation process and encapsulating data into structures that could be understood by PyTorch. The `models` module contains implementations of all the deep learning models used in my research, making them easily importable as needed. Lastly, the `notebooks` module comprised Python notebooks documenting my research process, including a playground folder for initial experimentation. The application of this LO in my work is strong as it demonstrates my ability to design clean and efficient systems by ensuring that each component handles as a single task and that conceptually similar tasks are grouped together. This approach not only led to a more organized codebase but also makes it more maintainable and extendable in the future. By applying #separationofconcerns

effectively, I demonstrated my proficiency in designing modular systems that reduce complexity and improve the overall development process in my research project.

Capstone LOs

#curation: In my paper, I applied #curation LO by selecting, organizing, and presenting essential content relevant to my research on comparing traditional RNNs and RC models for predicting ball trajectories in the simulated 2D environment. I carefully chose the appropriate deep learning models, data generation methods, and statistical tests to present in my paper while maintaining a coherent structure throughout the write-up. I organized my paper into sections that focus on specific aspects of the study - namely, an introduction, a literature review, a few sections on methodology and discussions, and ultimately leading to a well-argued conclusion - to curate my deliverable in the style of academic research papers.

The application of #curation LO in my paper is strong because it demonstrates my ability to present essential content in a well-structured manner, ensuring that each section serves a specific purpose within the overall narrative. By effectively curating the content, I have made my research easily accessible not only to the audience of profound experts but also to people who have very tangential expertise in the explored field. The thoughtful organization and presentation of content in my paper demonstrate my proficiency in communication and my understanding of how to convey complex ideas in a clear and concise manner.

#navigation: Throughout my work on the capstone project, I applied this LO by working strategically to accomplish the aims of my research project by relevant deadlines. Meeting the deadlines, as well as being flexible and adaptive to the changes in the research process, was greatly supported by my adopting the agile methodology. I used peer and professor feedback to analyze my progress and adjust my research methods as needed by performing these actions during the iterative sprints. I greatly relied on Jira to keep me accountable in the process while also showing a clear roadmap of work that I had to do to produce a quality capstone project.

The application of #navigation LO throughout my work was strong because it demonstrated my ability to work strategically, efficiently, and adaptively to achieve my research goals by specified deadlines. By thoughtfully applying working strategies and embracing agile methodology I was able to meet my commitments and accomplish the delivery of my ambitious project.

#outcomeanalysis: In my deliverable, I applied this LO by identifying and utilizing appropriate measures to develop and evaluate the performance of traditional RNNs and RC models for predicting ball trajectories in a simulated 2D environment. Using the Root Mean Square Error as a quantifiable metric, as well as visual outputs of the models, I honestly concluded that despite my best attempts to achieve good predictive performance for the problem I was trying to solve, none of the models that I tried achieved satisfactory results. That said, based on my research, I was able to conclude that traditional RNNs are doing a somewhat good job minimizing the error while ESNs are better at grasping the underlying physics. This finding allowed me to suggest a future research direction that would explore the combination of these two types of models.

My application of #outcomeanalysis LO is strong because I demonstrated my ability to accurately and effectively use measures and rubrics to assess my findings. By analyzing, explaining, and justifying the application of performance metrics and rubrics, I have successfully demonstrated my expertise in evaluating work products and drawing meaningful conclusions from my research, honestly concluding that my research project cannot be called successful on its own but provides a good fundament for suggested future research.

#qualitydeliverables: My paper is a strong application of #qualitydeliverables LO because it represents a comprehensive summary of my research project according to the scope I defined for it over the past year. In this paper, I thoroughly investigate the performance of traditional RNNs and RC models for predicting ball trajectories in a simulated 2D environment. This paper includes a thorough summary of my experiments with different models, highlighting the strengths and weaknesses of each of them. Together with this, it also contains a description of the underlying data generation process, curated for the non-expert audience. The paper is well-structured, containing a clear introduction, methodology, results, discussion, and conclusion sections, providing a solid basis for understanding the research and its implications.

The application of this LO in my deliverable is strong because it demonstrates my ability to produce high-quality research work that meets the expectations of the research setting. My work exhibits a sophisticated scope, depth, and rigor, addressing the problem comprehensively and providing appropriate justification for my findings. All this scoping, together with an extensive summary of my results process and results, are effectively summarized in the paper, which further highlights my ability to produce quality deliverables.