Writeup draft

Tags	
CAP Issues	
Carterochka/capstone PRs	
Last update date	@October 29, 2022

▼ Introduction and rationale

Things that need to be highlighted in this section:

- Purpose - research algorithms for predicting trajectories in classical mechanics scenarios to understand the capability of neural networks to learn the underlying physics.
- Use cases:
  - Robotics - analyze the impact of physical interventions
    - Phyre's underlying goal is to benchmark something similar
  - Military - predict the trajectories of parts of missiles after impact
  - Neuroscience research - model how our brain perceives the dynamics of objects

▼ Literature review

Section to be constructed after the list of articles is summarized.

Literature review notes are here.

For midterm deliverable, my page with literature notes also comes in pdf as a separate appendix.

▼ Simulator overview

This section will rely on example notebook and paper that come with Phyre benchmark.

- Notebook link:
  `https://github.com/facebookresearch/phyre/blob/main/examples/01_phyre_intro.ipynb`

- Paper link: `https://arxiv.org/abs/1908.05656`

Main points to be highlighted:

- So far I am not using the “benchmarking” capabilities of Phyre software, only the simulator that is implemented there.
- Phyre also comes with a set of example models that solve the target problem of the benchmark. Those models can be used as inspiration for the trajectory prediction, and if they will, it will be highlighted in the corresponding part of the capstone body.

▼ Data preparation process

- I use Phyre simulator as a source of data.
- For data preparation, I first created an extension of PyTorch `Dataset` class. This extension is an abstract class called `ClassicalMechanicsDataset`.
  - Implementation of this class can be found on github here. It is only accessible from `epic/free-fall` branch.
  - This class defines an abstract method `generate_data()` that must be implemented in every child class. In child classes, this method will invoke 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.
  - Every other dataset that I will use in my research will extend this class. Child classes have to implement methods `generate_data()` (to generate the dataset for every specific research case) and `__getitem__(self, idx)` (that returns input and ground truth output for the queried data instance).

▼ Free-fall analysis

▼ Free-fall of a single ball

The simplest case for trajectory prediction is a single free-falling object. Just free-fall does not have any non-linearity involved and, thus, doesn’t need

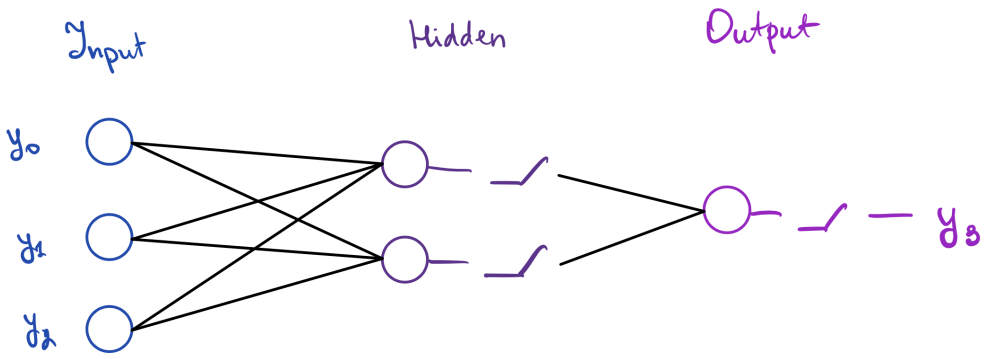
machine learning to predict. So, instead I am trying to train some models that will predict the movement of the free-falling ball that falls on the floor. To simplify in the beginning, I am starting with one-dimensional case (as free-fall is a 1-D movement).

The implementations of the model can be found here.

The notebook with testing the models can be found here. Polishing the contents of the notebook is still work in progress.

- ▼ Input - three consecutive coordinates
  - ▼ Dense network with two neurons in hidden layer and ReLU activations

    Schematic representation of the model is given below:



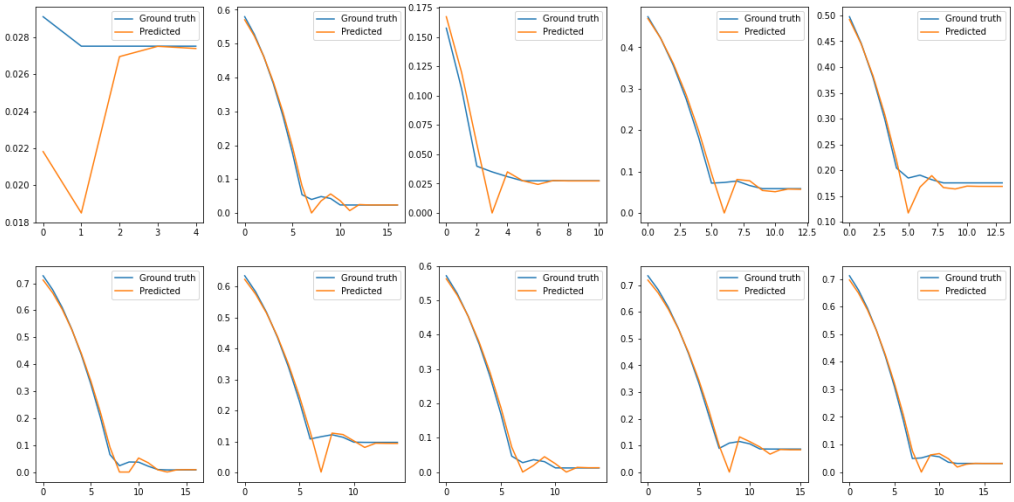
    The idea behind such an architecture is the following.

    - Three consecutive coordinates are enough mathematically to infer the velocity and acceleration of the object.
    - Two neurons in the hidden layer are supposed to represent velocity and acceleration.
    - ReLU activation for hidden layer is the most basic attempt that I could start with.
    - ReLU activation for output is a prevention from negative predictions (as measures of length in Phyre are bound between 0 and 1).

    The output is a single coordinate. For each scenario, predicting each following coordinate is an independent task. However, plotting the predictions for the same scenario together can give us something

similar to a “trajectory” (however, in this case it won’t be a trajectory but rather a “map” of predictions).

I trained this model on a train dataset that consists of 3752 scenarios. After training, I used a test set to see how it performs on the unseen data. The first 10 “trajectory maps” from the test set are shown on the figure below.

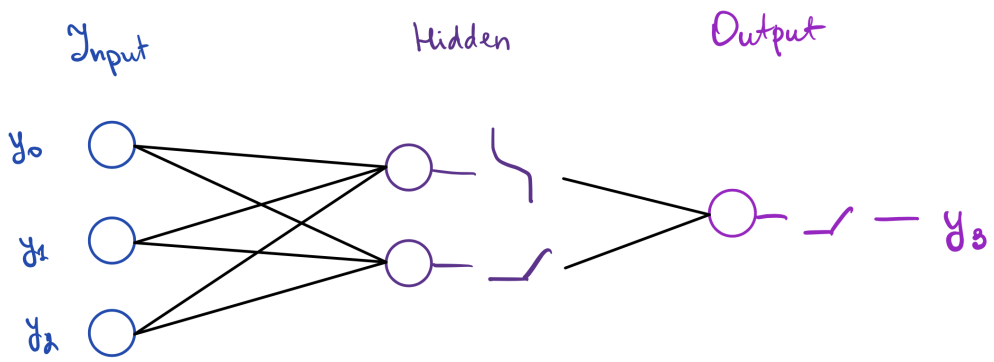


We can see that in majority of cases the model predicts the following coordinate quite well.

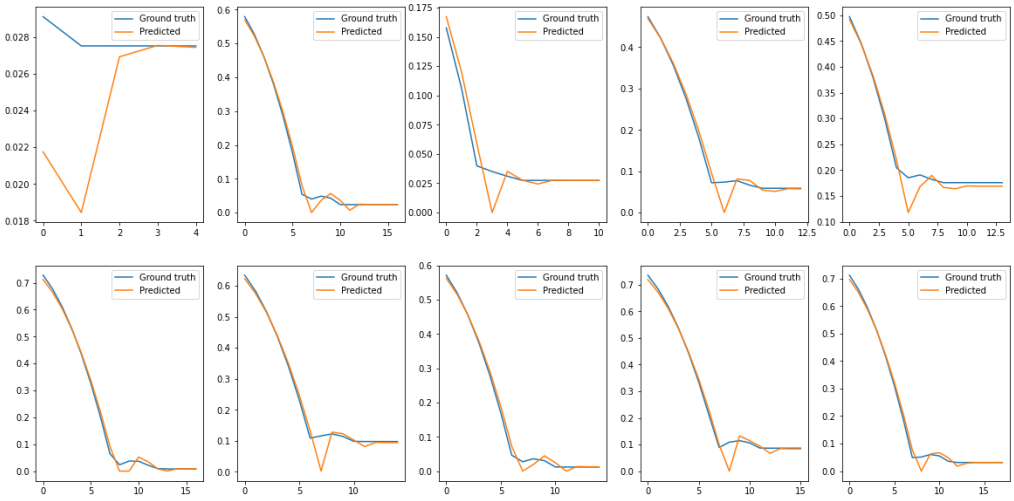
▼ Dense network with two neurons in hidden layer and ReLU/cotan activations

The second model that I tried is similar to the previous, with the exception that one of the activations in the hidden layer is changed to cotangent. The rationale that the velocity can be negative, which is not allowed by ReLU but is allowed for cotangent.

Schematically this model can be pictured as below.



Here are the first 10 trajectory maps predicted by the model on the test set.

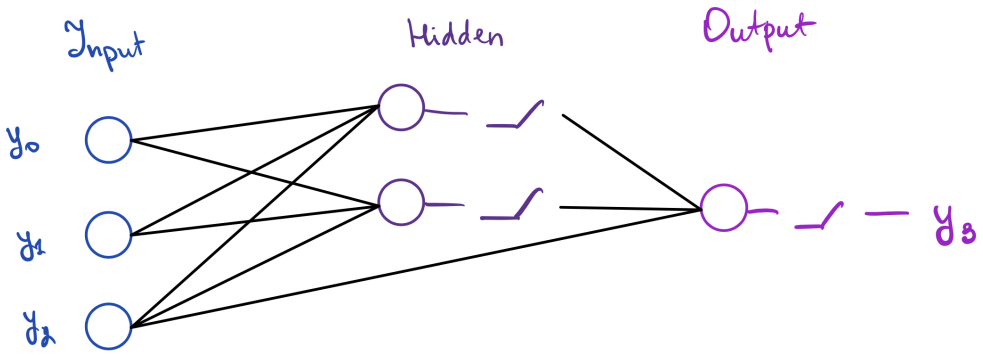


▼ Dense network with two neurons in hidden layer, ReLU activations, and direct connection of one of the input neurons to the output

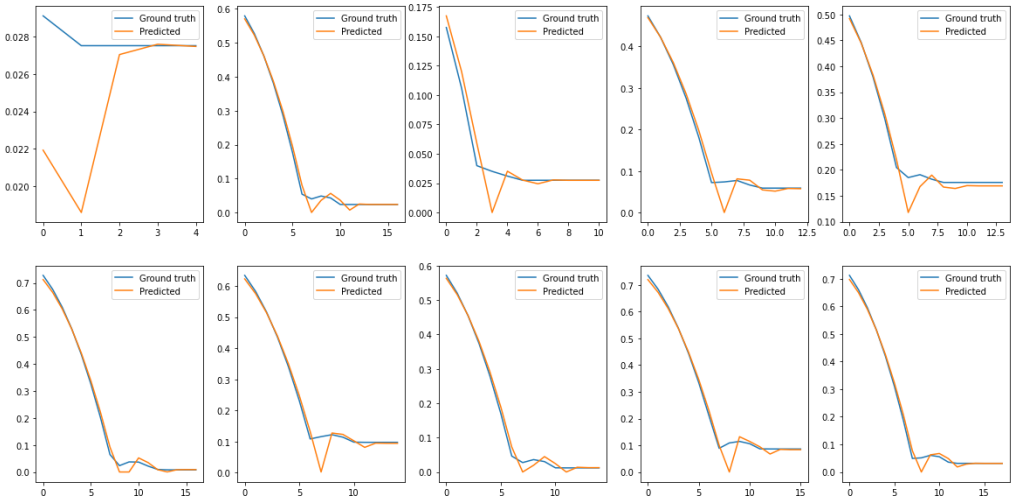
According to kinematics, the coordinate of the object in the free-fall can be calculated as

$y = y_0 - v_0 t - \frac{1}{2} g t^2 = y_0 - vt + \frac{1}{2} g t^2$, where v_0 is initial speed and v is current speed.

To better mock this formula, I decided to add a direct connection from one of the input neurons to the output neuron. Schematically it can be shown as the following:



The first 10 predicted trajectory maps on the test set look the following:



▼ Conclusion

We can see that the results of the models above do not differ much between each other. If the choice was to choose among these three modes, I would opt for the dense network with ReLU activations, as it is the simplest one among the three.

As a next step, I want to train the models that would predict the entire trajectory rather than a single coordinate at a time.

▼ Input - a single initial coordinate coordinate

The next step in incrementing complexity is to train models that are able to predict the entire trajectory of the free-fall and bounce. As an input I decided to use just an initial coordinate.

The notebook with the linear networks tests can be found here.

▼ Linear network (3 layers)

I decided to start with a simple 3-layer dense network that only takes an initial coordinate as an input and predicts the entire trajectory as an output.

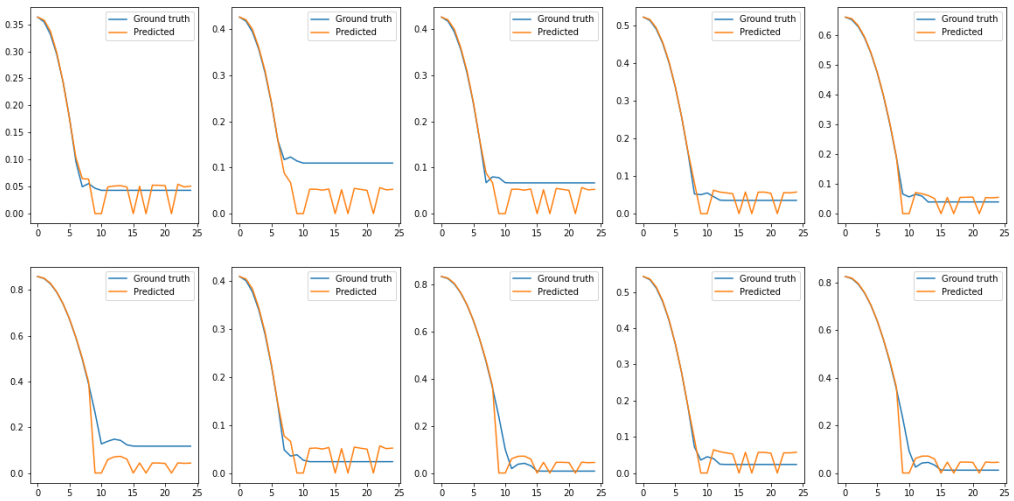
To be able to implement such a model, I needed to assure consistency of the output size. Thus, I generated a dataset in such a way that the length of each simulation is 25 frames.

The model architecture is the following:

- Input size of 1 corresponds to initial coordinate
- There are 2 hidden layers of 64 and 32 neurons respectively
- The output layer has 24 neurons that together with initial coordinate form the full trajectory.

I trained the model on 4398 scenarios, and then tested using the test set of size 1072. The error function that I used to compare the models is RMSE (root mean square error). For this model, it's value on test set equals 0.0468.

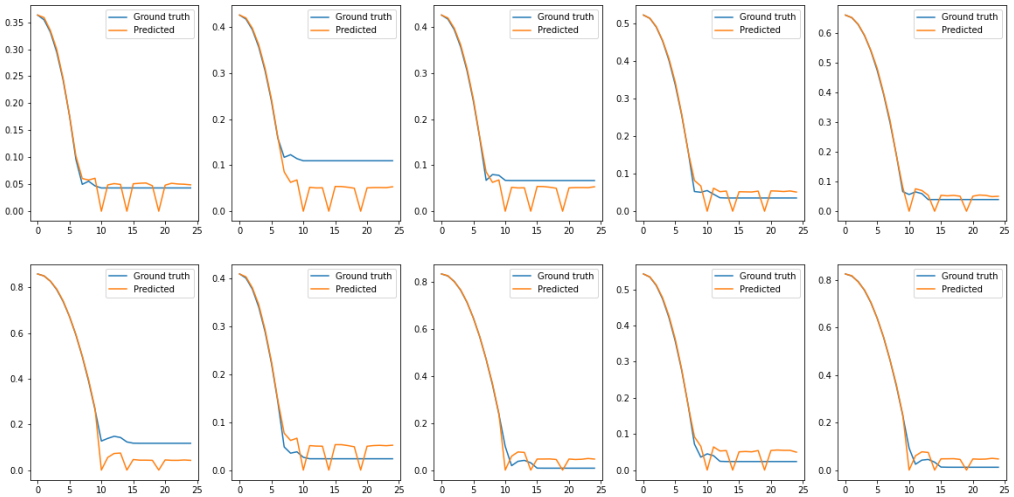
The first 10 predicted trajectories with ground truth from the test set are shown below.



▼ Linear network (5 layers)

Then, I decided to increase the number of layers in the dense network to 5, adding two more hidden layers with 256 and 128 neurons respectively. Test error of this network is 0.0359.

The first 10 predicted trajectories from the test set are shown below.



▼ Conclusion

Note: I did not include regularization or dropout to these two networks as I only used them as reference point and inspiration for other models. Linear networks would not scale for the type of trajectory prediction tasks that I plan to explore further.

The results of these networks suggest two important insights.

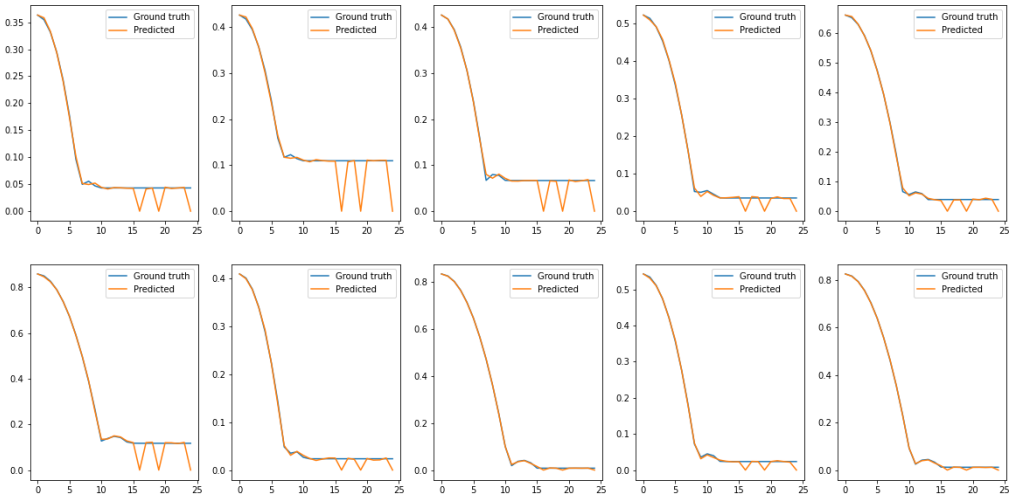
- First, these networks are able to predict free-fall almost perfectly, and are able to learn that the ball will bounce after hitting the floor. However, the models are not able to predict the state of rest, which manifests in the “spikes” on the plots after the ball hits the ground.
- Second, the models predict that the ball will hit the ground at the hight of about 0.06 for all the scenarios. I assume that this is a weighted average for the center of mass of the ball after it hits the ground among all the scenarios in the training set. This insight suggests that I need to include the diameter of the ball into the input, as it is crucial for learning the position of the center of mass after the ball appears on the floor.

▼ Input - initial coordinate and ball diameter

▼ Linear network - 5 layers

I decided to test the same 5-layer dense network as above, but including the ball diameter as one of the inputs.

The testing error that I got for such a network is 0.0226. We can see that with diameter included in the input, the network does a good job predicting where the center of the ball will stay in rest.



▼ Echo State Networks

A number of resources suggest that Echo State Networks can be good for trajectory prediction tasks. Among them:

- This paper claims recurrent neural networks are good for predicting trajectories in scenarios with behavioral components involved.
  - Echo State Network is one of the types of RNNs.
- This article claims that ESNs perform well for predicting the development of chaotic systems.
  - The simulated scenarios that I am using are completely deterministic, and, thus, ESNs should predict them well.

The notebook with ESN testing can be found here.

▼ Simple Echo State Network

I decided to start with the simplest Echo State Network design, which consists of a single reservoir, followed by a linear layer that performs the ridge regression.

- Reservoir is a recurrent neural network with a set of neurons that are randomly interconnected. The connectivity of neurons is about 10%. The weights between input and reservoir are randomly initialized and not trainable. The weights between reservoir and output are trainable and trained in a single epoch using the entire dataset.
- Ridge was chosen because it is designed to have good performance on the correlated data.

To choose the hyperparameters for the ESN, I decided to perform a grid search.

▼ Grid search

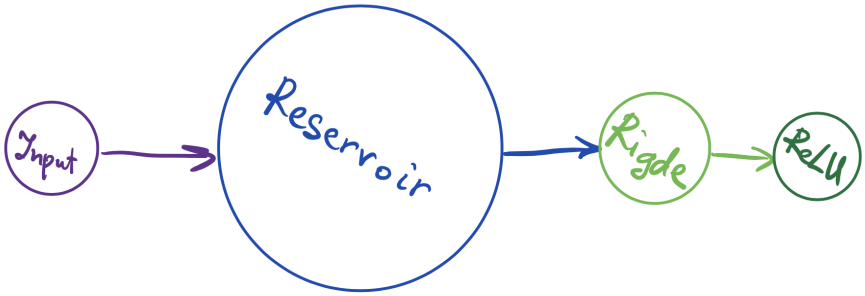
The hyperparameters that I tried for grid search are:

- Reservoir size: 50, 70, 100, 150, 300
- Learning rate: 0.3, 0.5, 0.7
- Spectral radius: 0.95, 0.99, 0.998
- Ridge parameter: 1e-5, 1e-4, 1e-3, 1e-2

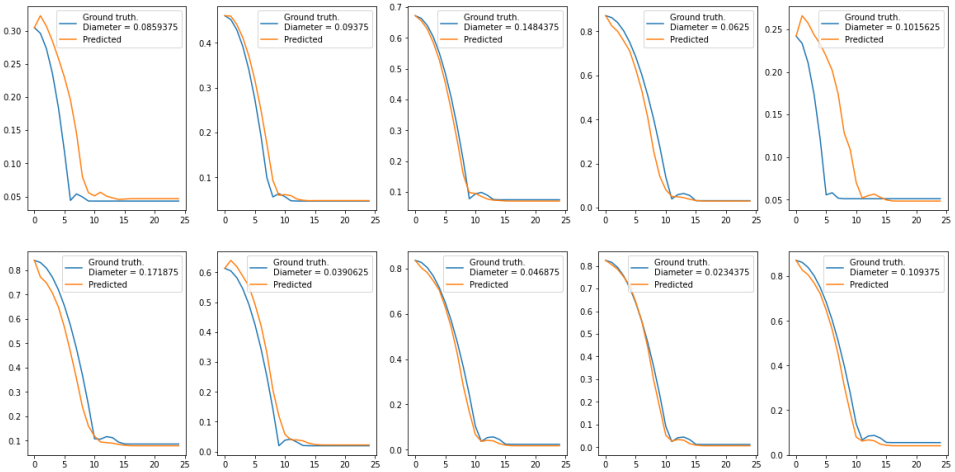
The best set of hyperparameters on the test set were found to be:

- Reservoir size: 70
- Learning rate: 0.7
- Spectral radius: 0.95
- Ridge parameter: 1e-2

Schematically, the network can be shown as the following:

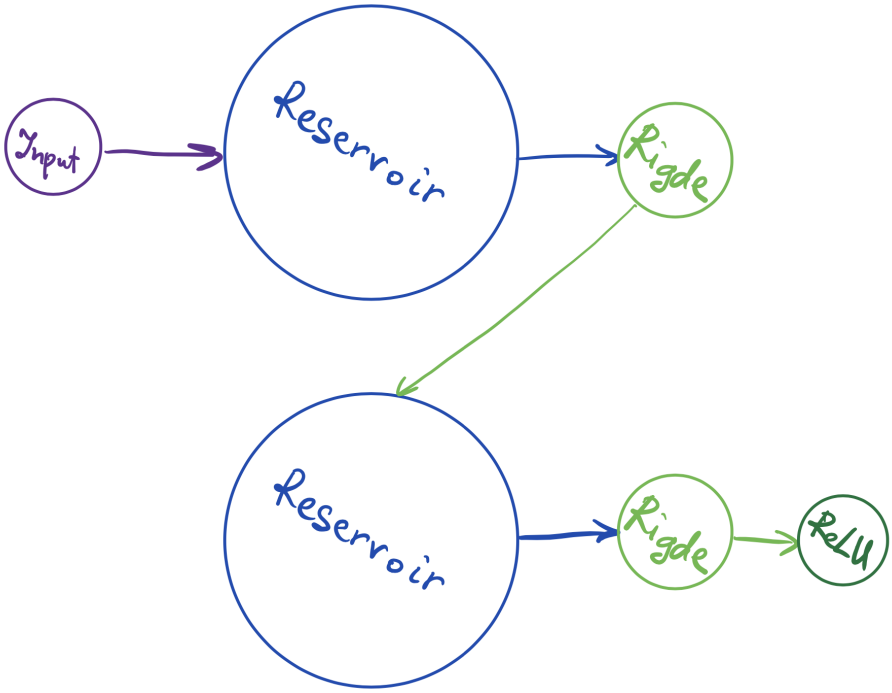


The testing error that I received with this network is 0.0462. It is bigger than using the linear networks; however, if we look at the plots of predicted trajectories, we can see that this network does a good job learning the shape of the trajectories with.



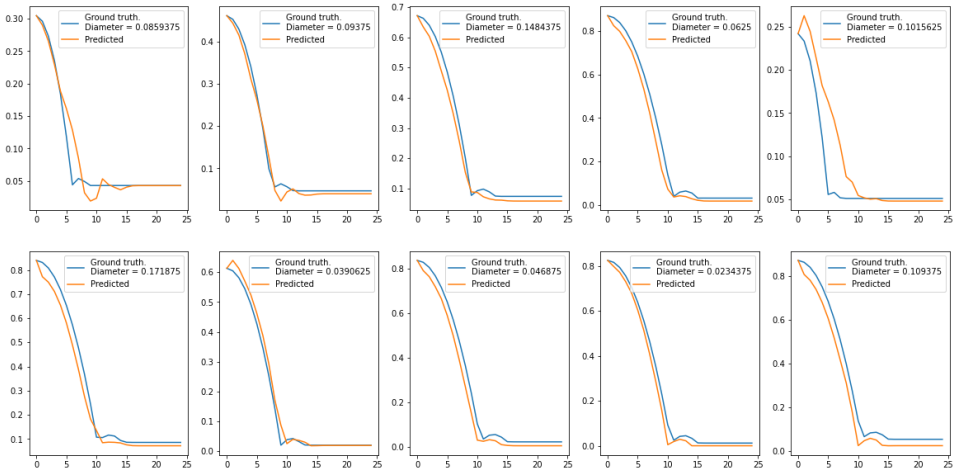
▼ Echo State Network with sequential reservoirs

I then decided to connect multiple reservoirs in sequence, as suggested in this paper, to see if this will improve the results. Schematically it would look the following.



The testing error for this network is 0.0423. Depending on the initialization of the weights (as it is done randomly), it can be better or worse compared to a single reservoir case. I don't have any reasons to believe that such design has any improvements compared to the single reservoir design.

The first 10 predicted trajectories from the test set are shown below.

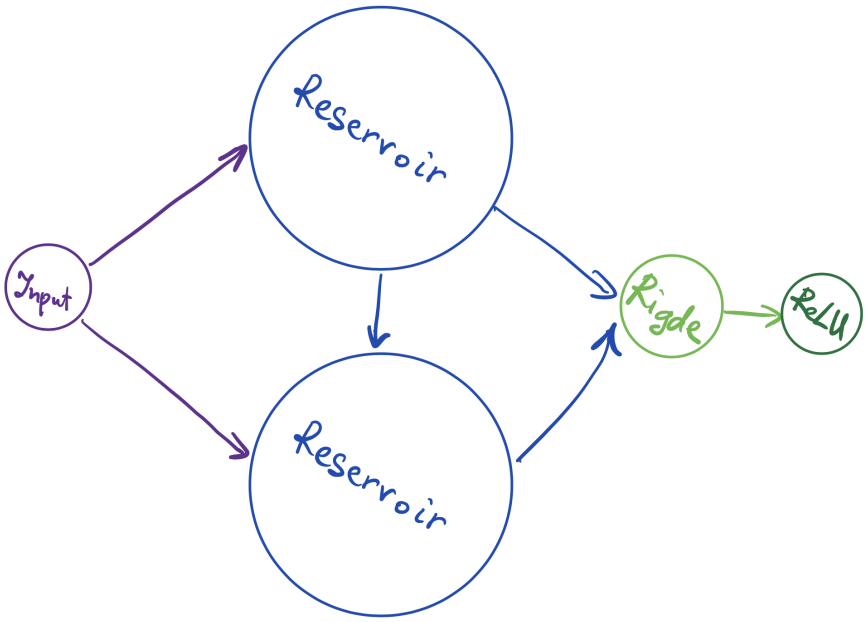


▼ Deep ESN

This paper suggests a DeepESN architecture that consists of several reservoirs connected in parallel rather than sequentially. I decided to test such architecture on my problem.

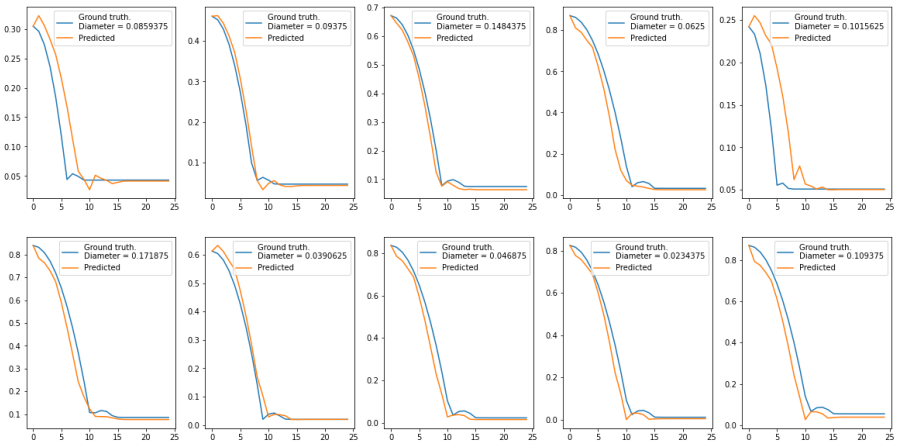
▼ two layers

Schematically, this design can be pictured like below.



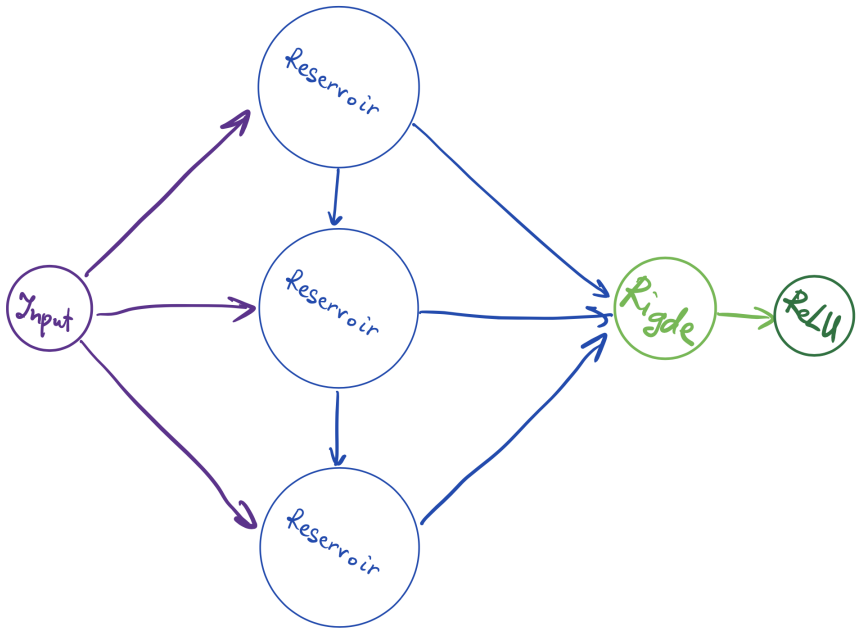
The testing error that I got for this model is 0.04951.

The first 10 predicted trajectories from the test set are shown below.

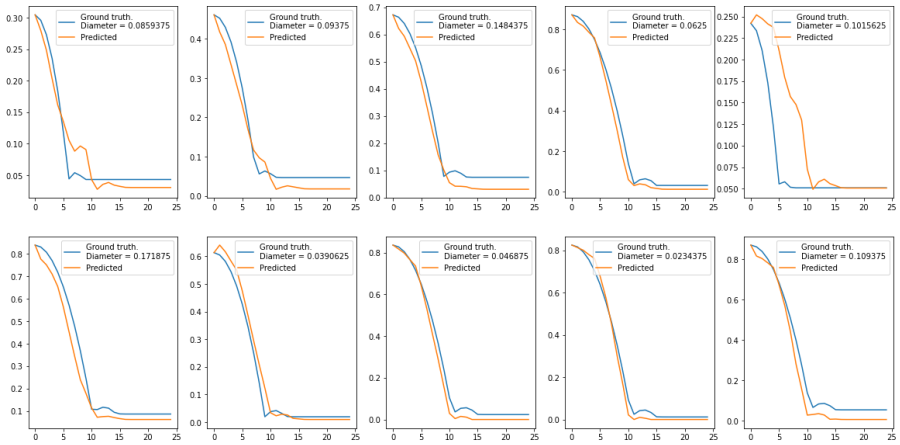


▼ three layers

I also decided to try connecting 3 reservoirs in parallel, as in the figure below.



The testing error that I received is 0.0725, which is higher than for the models with smaller number of reservoirs. Here are the first 10 predicted trajectories from the testing set.



- Conclusion

  It seems that Echo State Networks are capable of learning the trajectories, at least for a simple free-fall with bounce scenarios. I definitely plan to explore them more for more complex scenarios.

  As for the deep ESN architectures, on this task they don't show any improvement compared to simple ones. However, I still plan to explore how they will perform for scenarios with more complexity involved.

- Free-fall with multiple balls on the scene.

- Scenarios with collisions
- Scenarios with static objects
- More complex scenarios
- Conclusion