2AMM10 2021-2022 Assignment 2 & 3: Group 47

Ahmet Ayrancioglu (1678620) Ricardo Andrade (1725882) Ruben Wolters (1342355)

June 20, 2022

1 Introduction

Simulation models have been evolving extensively to aid several engineering fields and the developments of other sciences. Simulations are preferred over conducting experiments in the real world as it is usually less costly and easier to conduct. These simulation models allow a company or a researcher to test how things will behave in the real world and if the desired outcomes can be reached. However, even though these simulations are more efficient compared to their real-life counterparts, the efficiency can still be improved. To improve this necessary process, machine learning may be used. In this report, neural networks specifically are investigated to find out whether they can be a suitable replacement for these costly simulations, while still achieving the necessary level of precision that these experiments require.

In these assignments, the main focus lies on the simulation of charged particles and their evolution as a system of complex interactions in a plane over time. The following notation is used:

- p_i is the i^{th} particle,
- $\mathbf{x}_i^t \in \mathbb{R}^2$ is the position of particle i at time t,
- $\mathbf{v}_i^t \in \mathbb{R}^2$ is the velocity of particle i at time t,
- $\mathbf{a}_i^t \in \mathbb{R}^2$ is the acceleration of particle i at time t,
- $\mathbf{c}_i \in \mathbb{R}$ is the charge of particle i,
- $\mathbf{v}_i^t = \mathbf{v}_i^{t-\Delta t} + \mathbf{a}_i^t \cdot \Delta t$ is the equation to update the velocity of particle i at time t,
- $\mathbf{x}_i^t = \mathbf{x}_i^{t-\Delta t} + \mathbf{v}_i^t \cdot \Delta t$ is the equation to update the position of particle i at time t,

2 Task 2

2.1 Problem formulation

In this task, a model is developed to predict the dynamics of interacting particles. More concretely, a prediction of the positions of n particles at a time t is made from the initial positions, velocities, and charges ($t = 0$) of these particles, with $c_i \in \{-1, 1\}$.

The dataset that is used is generated by running simulations. A simulation consists of the discretization of time into points with interval $\Delta t > 0$. Smaller intervals generate more precise simulations, while also taking more time to compute. In contrast, smaller intervals yield faster simulations but with less precision. For this task, an interval of $\Delta = 0.001$ is used. A system of $n = 5$ particles initialized with arbitrary positions, velocities, and charges is studied. A preliminary simulation is executed for three seconds. $t = 0$ corresponds to the time directly after the preliminary phase. Finally, the system is simulated for an additional 1.5 seconds, concluding the simulation process.

The data is divided into three parts:

- training set of 10000 simulations,
- validation set of 2000 simulations and
- testing set of 2000 simulations.

2.2 Model formulation

Following the previous section and the task at hand, a simple Feedforward Neural Network (FNN) is used to achieve the desired results. At a first glance, this problem might seem to involve time as a feature, and using a type of Recurrent Neural Network (RNN) might seem appropriate. However, as the only access to the initial positions and the velocities of the particles are available, this problem can be solved with traditional deep learning approaches. A multi-layer perceptron with non-cyclical connections is developed, where each layer is only connected to the next subsequent layer. To train the model, Mean Squared Error (MSE) Loss is used, defined as follows:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2,$$

where N is the batch size, y is the true value, and $\hat{y}$ is the predicted value by the model.

2.3 Implementation and training

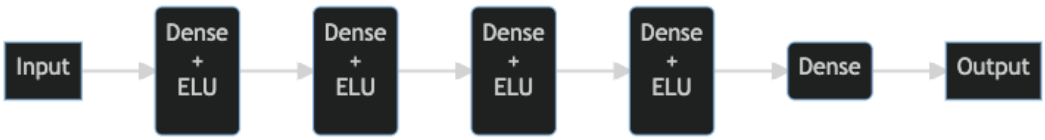


Figure 1: High-level representation of the model design.

The model consists of 5 dense layers. A high-level representation of the model can be seen in Figure 1. All linear layers have the same parameters except the first and last layers which have different input sizes and output sizes respectively. The exact description of these layers can be seen in Table 1. ELU is used which is a variation on the common ReLU activation function. ELU is very similar to ReLU with values over 0 scaling exponentially instead of linearly. There was minimal difference between these two activation functions for the use cases, but ELU was chosen as the activation function.

Table 1: Table showing the layers used in the model

Layer	**Type**
1	Linear: input=25, output=256
2	ELU
3	Linear: input=256, output=256
4	ELU
5	Linear: input=256, output=256
6	ELU
7	Linear: input=256, output=256
8	ELU
9	Linear: input=256, output=10

The model was trained using Mean Squared Error Loss (MSE Loss), which is the most common loss function which gives good and reliable results for most regression tasks unless the number of outliers in the dataset is of great quantity. As the dataset is coming from

simulations and does not contain any significant outliers, it works adequately for the use case.

The model uses the AdamW optimizer for training. Adam optimizer is one of the most popular and efficient optimizers. It gives good results and converges to a local minimum relatively quickly. AdamW is a variation of that optimizer that incorporates weight decay. Weight decay helps models to be more generalizable as it is a regularization technique. The parameters for the AdamW optimizer are as follows: betas=(0.9, 0.999), eps=1e-08, lr=0.0001, weight decay=0.01

A batch size of 64 is used for all of the datasets, as it gives a balance between the training time and the quality of gradients that are used by the optimizer. With the reduced time of 1 epoch models can be trained for longer periods. The training is 80 epochs, and an evaluation step is done on the validation dataset every 5 epochs. All training and evaluation were done on Google Colab using a GPU to speed up processing. The best-performing model on the validation set is saved to be later used during the testing stage.

2.4 Experiments and results

To assess the quality of the model, it is evaluated on several combinations of test sizes and prediction horizons. Also, the performance of the model is compared with a simple linear baseline where the final position of particle p_i at time t is

$$\mathbf{x}_i^t = \mathbf{x}_i^0 + \mathbf{v}_i^0 \cdot t.$$

To evaluate the performance of the models and compare them with the linear baseline, 15 experiments were conducted with training sizes of $[100, 500, 1000, 5000, 10000]$ and a prediction horizon of $t = [0.5, 1, 1.5]$. In addition to training sizes and prediction horizons, the number of epochs was also examined to see the effects it has on performance. The above mentioned 15 experiments were conducted for $[20, 40, 60, 80]$ epochs. The results are presented for each prediction horizon in a separate table and can be seen in tables 2, 3, and 4.

Table 2: Comparison between the model's performance and the linear baseline equation for prediction horizon $t = 0.5$

Epochs	**Train Size**	**Model MSE**	**Model R^2**	**Baseline MSE**	**Baseline R^2**
20	100	5.399	0.294		
	500	0.668	0.912		
	1,000	0.187	0.975	0.058	0.992
	5,000	**0.043**	**0.994**		
	10,000	**0.038**	**0.994**		
40	100	2.969	0.612		
	500	0.213	0.972		
	1,000	0.084	0.988	0.058	0.992
	5,000	**0.039**	**0.994**		
	10,000	**0.036**	**0.995**		
60	100	1.642	0.785		
	500	0.140	0.981		
	1,000	0.061	0.991	0.058	0.992
	5,000	**0.038**	**0.994**		
	10,000	**0.036**	**0.995**		
80	100	0.828	0.892		
	500	0.104	0.986		
	1,000	**0.051**	**0.993**	0.058	0.992
	5,000	**0.038**	**0.994**		
	10,000	**0.035**	**0.995**		

Table 3: Comparison between the model's performance and the linear baseline equation for prediction horizon $t = 1.0$

Epochs	**Train Size**	**Model MSE**	**Model R^2**	**Baseline MSE**	**Baseline R^2**
20	100	7.185	0.265		
	500	0.985	0.899		
	1,000	0.347	0.964	0.259	0.974
	5,000	**0.133**	**0.986**		
	10,000	**0.125**	**0.986**		
40	100	4.537	0.536		
	500	0.386	0.960		
	1,000	**0.214**	**0.978**	0.259	0.974
	5,000	**0.126**	**0.986**		
	10,000	**0.122**	**0.987**		
60	100	2.396	0.755		
	500	0.271	0.972		
	1,000	**0.160**	**0.983**	0.259	0.974
	5,000	**0.126**	**0.986**		
	10,000	**0.122**	**0.987**		
80	100	1.233	0.874		
	500	**0.217**	**0.977**		
	1,000	**0.155**	**0.983**	0.259	0.974
	5,000	**0.131**	**0.986**		
	10,000	**0.122**	**0.987**		

For the evaluation metric, MSE and R^2 score are used. As previously stated, MSE is used in the training step of the model. The R^2 score, also known as the coefficient of determination, measures the proportion of the variance in the predictions that are explainable from the training data set. R^2 score generally takes a value from $[0, 1]$ with 1 meaning there is a perfect correlation between the predictions and the training dataset, and 0 meaning that the model is not better than a naive model that is just using the mean values. Note that the R^2 score can be arbitrary negative meaning that the model is performing worse than the naive mean value model.

One of the most immediate conclusions that can be drawn from Table 4 is that the linear baseline gets worse "faster" than the new model when the prediction horizon is larger. Additionally, when looking at all tables in unison, it can be observed that although the R^2 scores decrease gradually, the MSE metric increases sharply with each 0.5-second increase on the prediction horizon. This suggests that the moving particles get more unpredictable as time goes on, a characteristic that can't be captured well by the simple linear baseline. On the other hand, the model minimizes this effect well especially when it's presented with enough training data.

For $t = 0.5$, the model is only marginally superior to the linear baseline even when trained with 10,000 samples. As the linear baseline gets worse, the model gains superiority for $t = 1.0$ and $t = 1.5$ even with only 500 training samples. However, as already mentioned, the performance of the model isn't very superior to the linear baseline when taking into account R^2 scores. In contrast, it is less error-prone than the linear baseline by a significant amount.

Finally, as shown in tables 2, 3, and 4 training for more epochs makes the models more accurate and perform better. However, it is easy to observe that performance and number of training epochs do not scale linearly, as the model is trained for longer, the performance gain from the training drops drastically. This is in line with the literature on how deep neural networks are trained in general.

Table 4: Comparison between the model's performance and the linear baseline equation for prediction horizon $t = 1.5$

Epochs	**Train Size**	**Model MSE**	**Model R^2**	**Baseline MSE**	**Baseline R^2**
20	100	9.494	0.224		
	500	1.403	0.885		
	1,000	**0.534**	**0.956**	0.607	0.950
	5,000	**0.267**	**0.977**		
	10,000	**0.256**	**0.978**		
40	100	6.053	0.505		
	500	**0.572**	**0.953**		
	1,000	**0.357**	**0.970**	0.607	0.950
	5,000	**0.261**	**0.978**		
	10,000	**0.254**	**0.978**		
60	100	3.614	0.704		
	500	**0.478**	**0.960**		
	1,000	**0.307**	**0.974**	0.607	0.950
	5,000	**0.259**	**0.978**		
	10,000	**0.253**	**0.978**		
80	100	2.148	0.824		
	500	**0.385**	**0.968**		
	1,000	**0.299**	**0.975**	0.607	0.950
	5,000	**0.260**	**0.978**		
	10,000	**0.251**	**0.979**		

2.5 Conclusion

A feed-forward neural network and a baseline linear model were trained with the task of predicting the positions of multiple particles at a given time in the future given the starting positions, velocities, and charges of the particles. To investigate the performance of the deep learning model on this task, different experiments were conducted with varying training epochs, training dataset sizes, and prediction horizons.

From the high scores achieved in the evaluation of the model, machine learning has the potential to be a suitable replacement for the costly computer simulations that are necessary for some engineering and science tasks. However, due to the simplicity of the data, these results may not be very relevant since the linear baseline is also able to get very good predictions for the prediction horizons taken into consideration.

Further research should attempt to evaluate the model with more complicated settings and with larger time horizons. The ideal setting would be one where the linear baseline is incapable of providing reasonably good predictions and a machine learning model still achieves high scores for the task.

3 Task 3.1

3.1 Problem formulation

In this task, a model is developed to deduce information about a particle system with one charged particle moving in a field with fixed-point charged particles. The first goal is to predict the exact charges of the static particles given the path of the moving particle. More formally, the exact charges of c_2, c_3, and c_4 are predicted using the positions of the charged particle p_1. Contrasting with Assignment 2, here a system composed of $n = 4$ particles is considered, where p_2, p_3 and p_4 are fixed. Additionally, the charge $c_1 = 1$ and charges c_2, c_3, c_4 are sampled uniformly from $[-1, 0]$.

The dataset generation follows the same procedure as the previous task with some key differences. Simulations are sampled with $\Delta t = 0.1$ and the dataset consists only of the positions $\mathbf{x}_1^t \in \mathbb{R}^2$ of particle p_1. Simulations with a longer period are also examined since the dataset provides simulations starting at $t = 0$ and ending at $t = 10 \pm 1$.

Furthermore, the data is more limited. For this task, 800 simulations are used for training, 100 for validation, and 100 for testing.

3.2 Model formulation

To achieve the above-described goal, a Recurrent Neural Network is used. More specifically, a Gated Recurrent Unit (GRU) is used.[2] While in Task 2 only the initial positions of the particles were given, hence time was not a feature to consider; in this case, the entire simulated trajectory of particles p_1 is given. Since the location and velocity of the particle change over the course of the simulation, time is now a relevant feature, therefore using a Recurrent Neural Network is more appropriate. For the loss function, Mean Squared Error is used, much like in Task 2.

3.3 Implementation and training

Figure 2: High-level representation of the model design.

The model consists of a GRU, followed by a dense layer. A high-level representation can be seen in Figure 2. An exact description of the parameters used in each layer can be seen in Table 5.

Layer	Type
1	GRU: input=2, layers=1, output=32
2	Linear: input=32, output=3

Table 5: Table showing the parameters of each of the layers used in the model for Task 3.1

The model was trained using Mean Squared Error Loss (MSE Loss), which is one of the most common loss functions that give good and reliable results for most regression tasks unless the number of outliers in the dataset is large. Since the values of the charges are quite limited, as they are sampled uniformly from $[-1, 0]$, extreme outliers are not possible, hence MSE is appropriate to use as the loss function in this case.

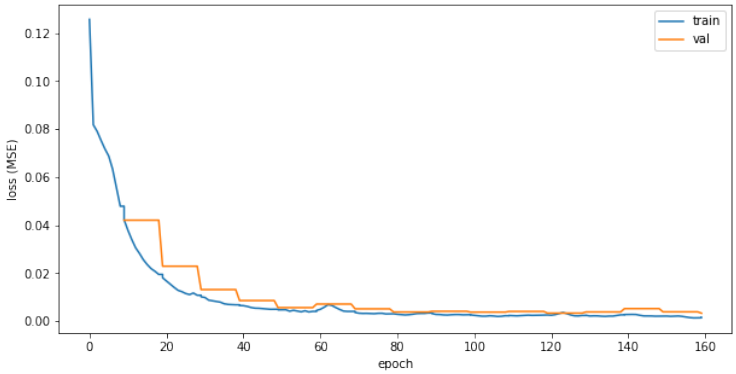
For the optimizer, AdamW is used. The Adam optimizer is a popular and efficient optimizer that gives good results and converges to a local minimum relatively quickly. AdamW is a variation of that optimizer that incorporates weight decay. Weight decay is a regularization technique that helps models be more generalizable. The parameters used for the AdamW optimizer are: betas=$(0.9, 0.999)$, eps=1e-08, lr=0.01 and weight decay=0.01[1].

The simulations have differing lengths. Thus, to batch the simulations for faster predictions, the simulations need to be padded to the length of the longest simulation in the dataset. For the padding, 0s are used and the original lengths of the simulations are recorded to evaluate the results correctly.

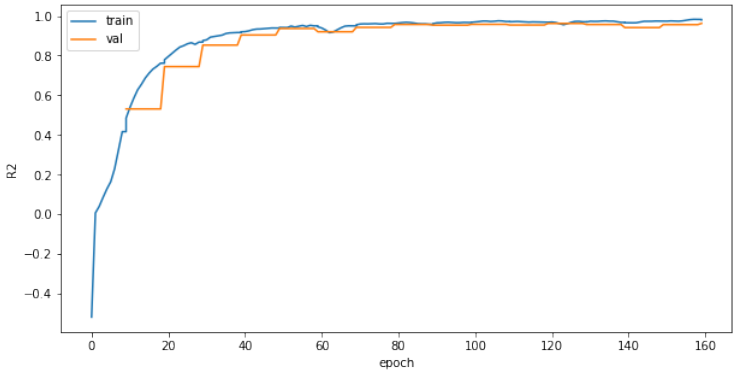
A batch size of 128 is used for the dataset as this gives a balance between training time and the quality of gradients that are used by the optimizer. With the reduced time of 1 epoch, models can be trained for longer periods. The training takes at most 160 epochs, and an evaluation step is done every 10 epochs. All training and evaluation were done on Google Colab using a GPU to speed up processing. The best-performing model on the validation set is saved to be later used during the testing stage.

3.4 Experiments and results

To assess the quality of the model, MSE Loss and R^2 score is used. MSE is used in the training step of the model. R^2 score measures the proportion of the variance in the predictions that are explainable from the training data set. Its value usually lies in $[0, 1]$, with 1 meaning the model has a perfect correlation between prediction and training data, while 0 means that it is not better than a naive model using mean values.



(a) Plot of the MSE of the model for both training and validation set.

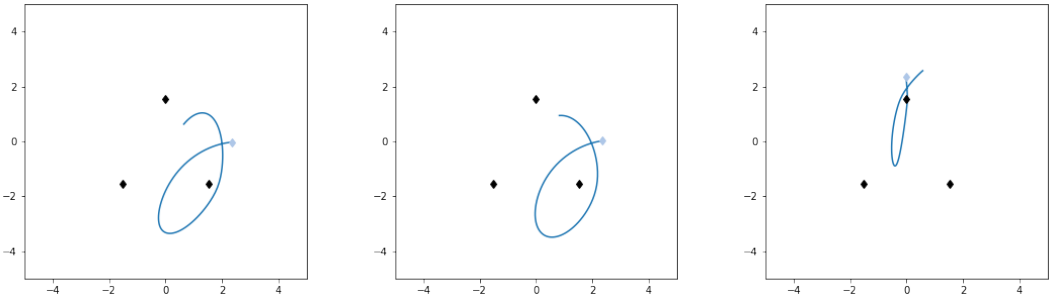


(b) Plot of the R^2 of the model for both training and validation set.

Figure 3: Plots for the evaluation metrics of the model.

In Figure 3a the progress of the MSE is plotted against the epochs for the training set and the validation set. In Figure 3b the progress of the R^2 score is plotted against the epochs for the training set and the validation set. As can be seen in both plots, the score improves rapidly until around epoch 10. From there the improvement slows down and finally settles around epoch 40. Finally, the best MSE is 0.0033, and the best R^2 score is 0.96.

In Figure 4, the three simulations the model performed worst on are shown. Table 6 shows the true and predicted values of each of the charges of the three worst-scored simulations



(a) Simulation, with MSE loss 0.0253

(b) Simulation, with MSE loss 0.016

(c) Simulation, with MSE loss 0.011

Figure 4: 3 worst performing simulations of the model.

of the model. Something to note about each of these cases is that there is a relatively large difference between the most negative charge and the other two fixed charges. This could potentially explain the reason why these particular cases are the worst-performing ones.

Simulation	**Charge**	**Predicted**	**True**
(a)	c_1	-0.2458	-0.1285
	c_2	-1.0815	-0.9564
	c_3	-0.3467	-0.1310
(b)	c_1	-0.1715	-0.0629
	c_2	-0.9927	-0.9295
	c_3	-0.2836	-0.1001
(c)	c_1	-0.1800	-0.1111
	c_2	-0.1159	-0.0509
	c_3	-0.7852	-0.6288

Table 6: Table displaying the predicted values and the true values of each of the charges in each simulation.

3.5 Conclusion

A GRU neural network, which is a type of RNN, was trained with the task of predicting the exact charges of fixed particles in a system given the path of a moving charged particle. The model performed quite well in this task with an R^2 score of 0.96. We can conclude that accurate predictions about the values of charges of particles with fixed locations can be made given a simulated trajectory of an oppositely charged particle. However, this approach is not perfect and there are some cases where the model performs below expectation. This seems to happen when there are large differences between fixed charges. Further research should attempt to evaluate the model with more complicated settings and with a larger set of particles with a large range of charges.

4 Task 3.2

4.1 Problem formulation

This task is a continuation of the previous task, 3.1. Now the task is to predict the continuation of the path of particle p_1 until $t = 14 \pm 3$, from the given simulations of particle p_1 until $t = 10 \pm 1$. As it can be understood, the prediction horizon for each simulation can be different, and not all simulations are of the same length.

The dataset follows the same generation pattern as in Task 3.1. However, even less data is used to evaluate the model, since the dataset consists of 150 simulations for training, 100 for validation, and 100 for testing.

4.2 Model formulation

Just like in Task 3.1, a Recurrent Neural Network is used, namely the Gated Recurrent Unit.[2] This is because, just like in the previous task, a full trajectory is given, meaning that time is a relevant feature. This makes using an RNN a better choice than a simple Feedforward Network.

One thing to note about model formulation is that this model is not an autoregressive model, where the output of the model is fed back into the model as the input in the next timestep. Instead, this model predicts the whole path of the particle in one step. This allows for more stable paths as the model predicts the whole path together. However, as the prediction horizons differ from simulation to simulation, predictions need to be made considering the maximum length of all prediction horizons.

4.3 Implementation and training

Figure 5: High level representation of the model design

The model consists of a GRU, followed by a dense layer. A high level representation can be seen in Figure 5. An exact description of the parameters used in each layer can be seen in Table 7.

Layer	Type
1	GRU: input=2, layers=2, output=128
2	Linear: input=128, output=120

Table 7: Table showing the parameters used in each layer of the model for Task 3.2

The model was trained using Mean Squared Error Loss (MSE Loss), as it gives good and reliable results for most regression tasks, given a consistent enough dataset. For the optimizer, AdamW is used again. The parameters used for this optimizer are: betas=$(0.9, 0.999)$, eps=1e-08, lr=0.001 and weight decay=0.01[1].

As mentioned in the previous sections of the report, the simulation has differing lengths and prediction horizons. Thus, to batch the simulations for faster predictions, the simulations need to be padded to the length of the longest simulation in the dataset. For the padding, 0s are used and the original lengths of the simulations are recorded to evaluate the results correctly.

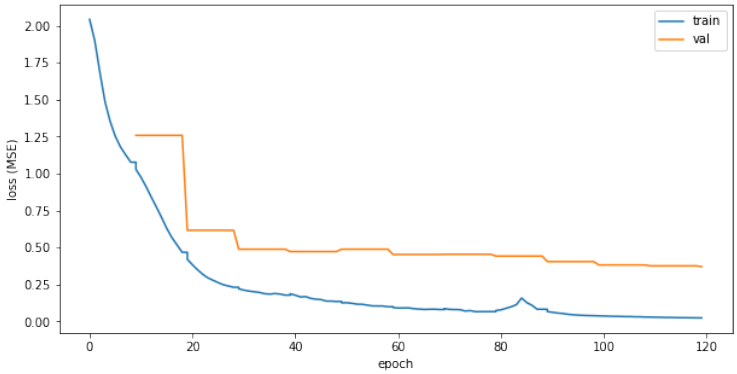
For the training, batches of size 16 are used. Since the amount of training data is very small, using a large batch size gets closer to using full gradient descent. Using a smaller batch size allows the use of mini-batch gradient descent. Furthermore having smaller batch sizes allows for faster convergence, as the simulations can be quite complex. The training

takes 120 epochs, with an evaluation step being performed every 10 epochs. All training was done on Google Colab using a GPU.

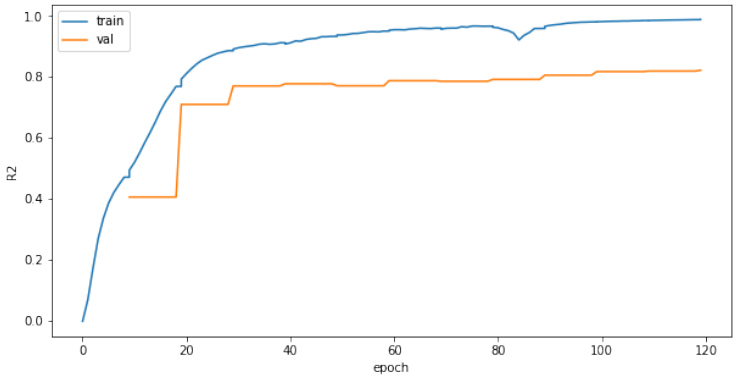
4.4 Experiments and results

To evaluate the model, it is compared to a linear baseline, as well as MSE Loss and R^2 score being used as evaluation metrics. R^2 score is explained in detail in subsection 2.4.

Both of the metrics are used in each of the training steps and work the same as in the other tasks. After training, the following results are obtained on the test set.



(a) Plot of the MSE of the model for both training and validation set.



(b) Plot of the R^2 of the model for both training and validation set.

Figure 6: Plots for the evaluation metrics of the model.

Metric	Value
MSE	0.270
R^2	0.863

Table 8: MSE and R^2 obtained on the test set for 3.2

In Figure 6 the MSE and R^2 are plotted against the epochs for both training and validation set. From the first epoch to the 20th epoch there is a sharp decrease in the loss and a sharp increase in R^2, then the loss gradually decreases until epoch 85, where there is a spike in the loss. However, after training further the loss decreases again and stagnates until epoch 120. From these plots, it can be observed that for both MSE and R^2 scores, the validation set scores were considerably worse than the training set. This indicates that the model may be slightly overfitting to the training set. Due to the size of the training set, this is not unexpected. The size of the training set limits how much the model can learn and generalize. However, in the light of all these limitations, as it can be seen from Table 8, the model manages to achieve impressive metrics on the test set.

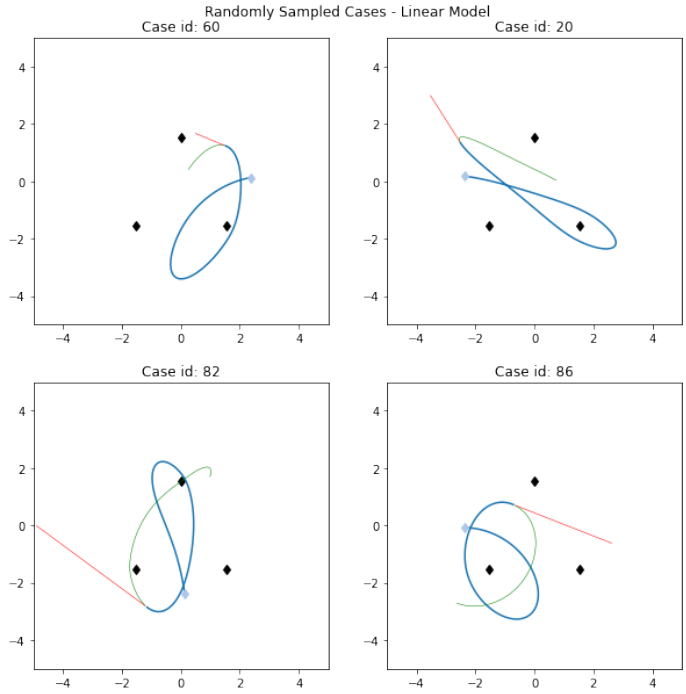


Figure 7: 4 randomly chosen comparisons between the ground truth path and the linear baseline: ground truth in green, baseline in red.

For the linear baseline, the direction between the last two simulation steps is taken and extrapolated to produce a straight line. In Figure 7 4 examples are shown of the comparison between the baseline and the ground truth. The MSE for this baseline is 2.331 and its R^2 score is -1.507. Clearly, this baseline and the ground truth are not very alike, as the R^2 score is very negative and the MSE is quite high as well. The awful performance of the linear model is expected as this task requires the path of the particle to evolve over time according to the charged particles in the system. However, the linear baseline model completely ignores these interactions and acts like the particle exists in isolation. This can be clearly seen in Figure 7.

Furthermore, 4 failure cases and 4 success cases are investigated. These can be seen in Figures 8, and 9. For both of the figures, the prediction is shown in red, while the true trajectory is shown in green. By observing Figure 8, it can be stated that for some cases the model fails tremendously. One interesting observation is that sometimes the path does not continue where it left off for the model's prediction. This is weird but not inexplicable. As there is no constraint of the physical world imposed on the model, the model does not know that it needs to start exactly in the same position where the path left off, the model only tries to minimize the loss function. Additionally, the cases where the model fails seem to involve long looping paths, where the ground truth path has an extensive curvature. This again makes sense as the curvature is created by the other particles which are hard to model. One last remark is that the model-created path is extremely jagged and does not look natural. This is again caused by the model not being aware of the physical constraints.

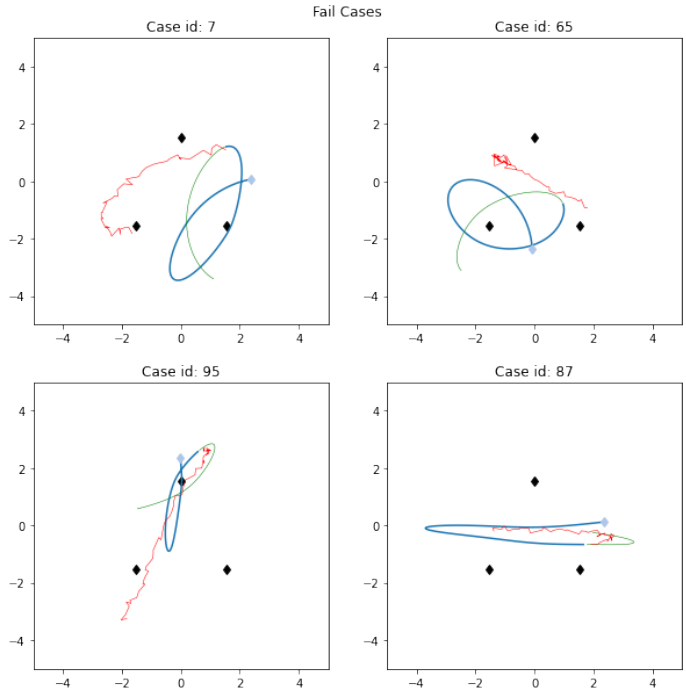


Figure 8: 4 worst predictions, model prediction in red, true trajectory in green.

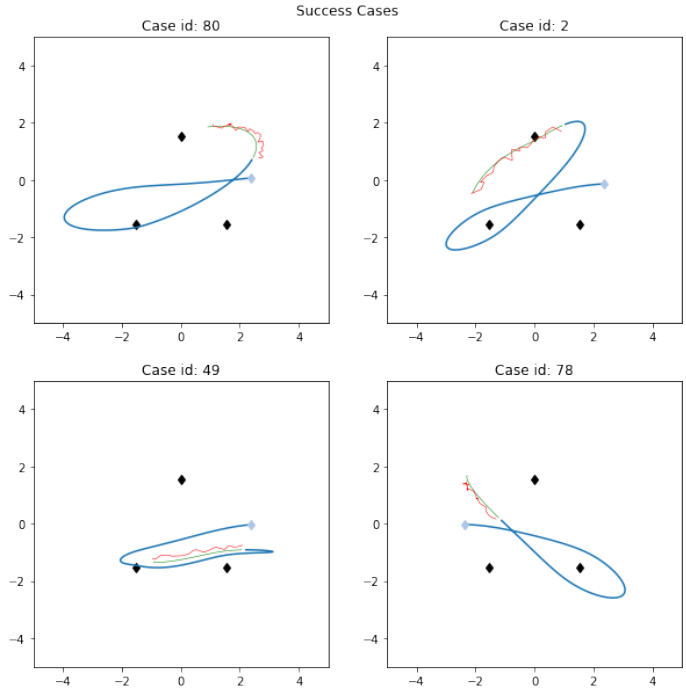


Figure 9: 4 best predictions, model prediction in red, true trajectory in green.

By observing Figure 9, we can make some deductions about where the model succeeds. The model seems to predict paths that are short and have less curvature with decent accuracy. Additionally, less jaggedness can be observed in the predicted paths. This makes sense, as these cases are closer to straight paths where the effects from the other charged particles in the system are minimized. With this reduced complexity, the model can achieve higher performance

4.5 Conclusion

In conclusion, a GRU RNN is created to predict the continuation of a trajectory of a charged particle traversing the plane with 3 fixed point charges. Using the model gives an MSE of 0.270 and an R^2 score of 0.863. While these values are not terrible, there is clearly still room for improvement. One of the main limitations is the size of the training data. Because it is the same size as both the validation set and the test set, there is not enough data for the model to distinguish relevant features that can be applied to the other data sets. Larger data sets could help solve this issue.

References

[1] Torch Contributors. Adamw. `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html`, 2019.

[2] Torch Contributors. Gru. `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`, 2019.