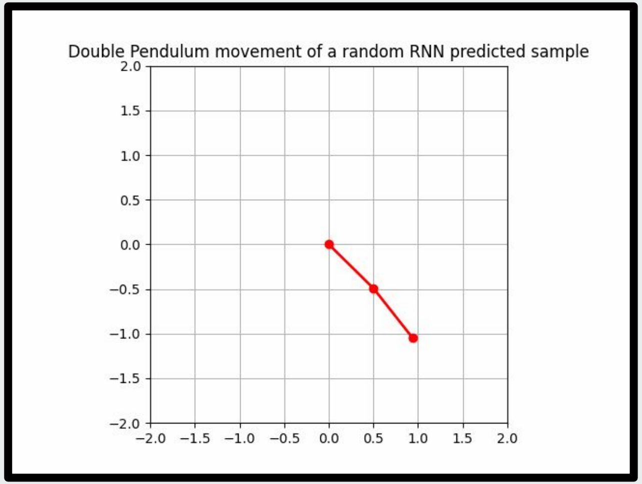
Comparing and Studying Neural Network Models

using a Double Pendulum System

Simen Løken



Abstract

- RNNs and CNNs are able to predict oscillating motion similar to that of a single pendulum system
- Hard to declare a real winner between RNNs and CNNs
  - RNN just barely wins
- NNs are not suited for modeling chaotic systems where the initial variables have such an impact on the system.
  - NNs are best suited for generalization. Given that a double pendulum system innately is a very “un-generic” system, NNs are poorly suited for this task.
  - As stated earlier, able to predict motion along the path a single pendulum system would travel, outside of that there is too much noise or no predictions at all

Introduction and scope of the project

What Neural Network is best suited to model chaotic systems?

1. Use a Runge-Kutta 4 algorithm to generate a data set with random initial variables
2. Train different neural networks with the generated data set
3. Compare the results
4. Try to draw conclusions as to which model can best model a chaotic double pendulum system, if any

Motivation - Why?

Why neural networks over traditional numerical methods?

- Given sufficient training - more numerically stable
- Learns outside of the bounds set by the training data
- Can model relationships outside of known physical models
- Can model non-linear data/systems
  - Systems that can not traditionally be modeled by numerical methods
- Can be faster

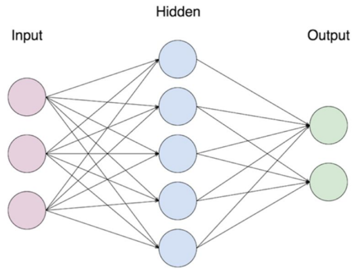
Motivation - Why?

Personal motivation

- Better understanding of neural networks
    - Complex and chaotic system
- Tangentially related to my thesis
    - Studying Quantum Many-body problems using Deep Learning
- Interesting
- Very visual results

Theory - Neural Networks

- Takes inspiration from the brain
    - Specifically neurons
- An input layer, N number of hidden layers and an output layer
- The behavior and function of the hidden layer is decided by the type of Neural Network
- Updating weights and biases as the network trains
    - Weight and bias calculations share some similarities between models



Img src:
https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

Theory - Feed-Forward Neural Networks

- Most simplistic form of Neural Network
- Information only flows in one direction
  - Input layer -> N Hidden layers -> Output layer
- Weighted connections between between nodes and the nodes in the subsequent layers

Theory - Recurrent Neural Networks

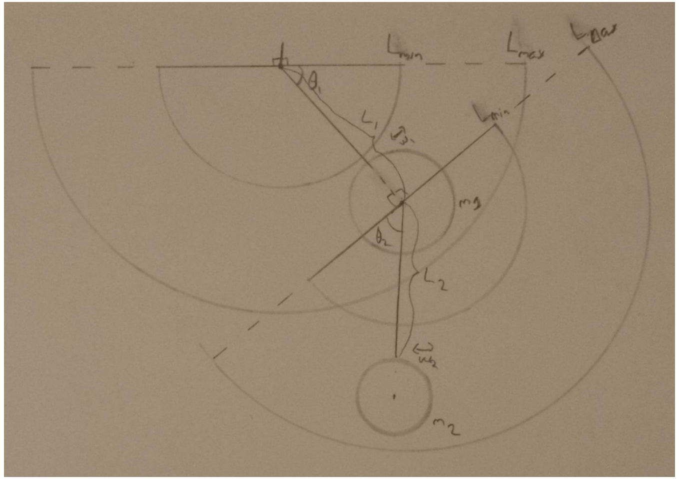
- Made for time-series or sequential data
- Similar to FFNN
  - Input layer -> N Hidden layers -> Output layer
- Has recurrent connections which serve as a “memory” for a given node
- Weights and biases the same as in FFNN
  - A separate set of recurrent weights that temporal dynamics
- Specifically LSTM (Long Short Term Memory)
  - Addresses problems with vanishing/exploding gradients
  - More effectively retain or discard information

Theory - Convolutional Neural Networks

- Made for data of grid-like structure
  - Images
- Different “layer types” to that of FFNN, RNN
  - Convolutional layers
    - Learnable filters extract local features
  - Pooling layers
    - Convoluted feature maps reduce spatial dimensions
  - “Normal” layers
    - Similar to those of traditional neural networks (ie hidden layers)
- Transform our data to look “grid-like”

Method

- Create a sizeable dataset of RK4 Double Pendulum data
- Train the network on a 80/20 split
- Use various methods to gauge the accuracy of our trained network
  - Plots
  - Compare to RK4

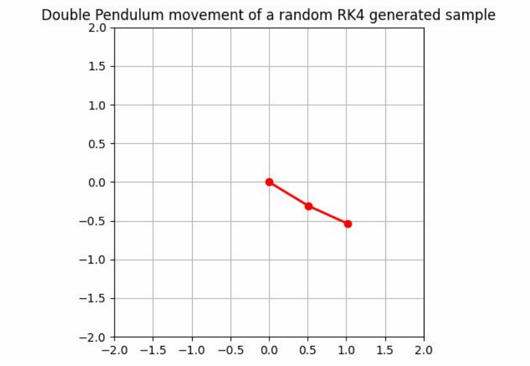


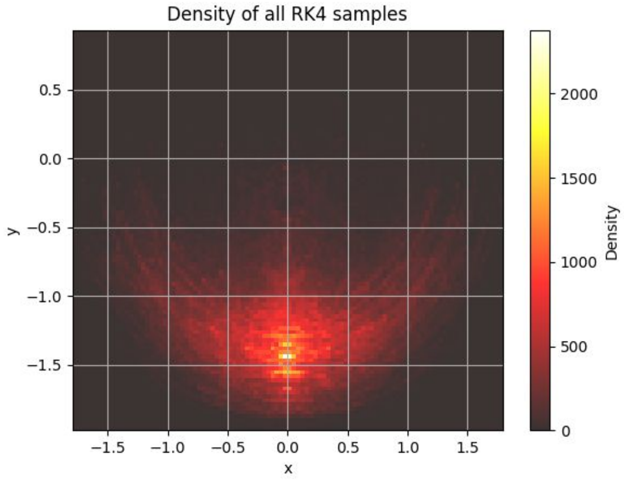
	L [m]	θ [rad]	ω [m/s]	M [kg]
min	0.5	-π/2	-0.05	0.25
max	1	π/2	0.05	0.5

$$L = \frac{1}{2}(M_1 + M_2)L_1^2\dot{\theta}_1^{\,2} + \frac{1}{2}M_2 L_2^2\dot{\theta}_2^{\,2} + M_2 L_1 L_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2) + (M_1 + M_2) g L_1 \cos(\theta_1) + M_2 g L_2 \cos(\theta_2)$$

Results - Runge-Kutta 4

- A benchmark
- Regarded as the “true solution”
    - The closer a model mimics the behavior and plots of RK4, the better the solution is
- Heatmap produces a “characteristic” ω shape

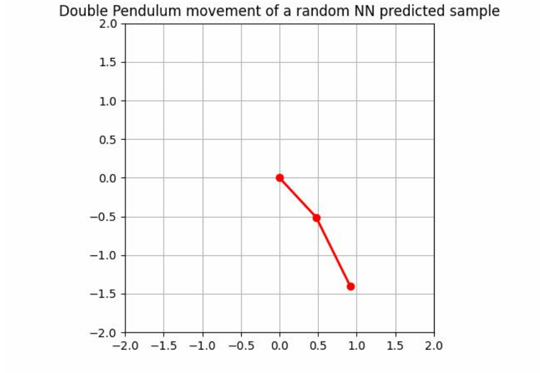


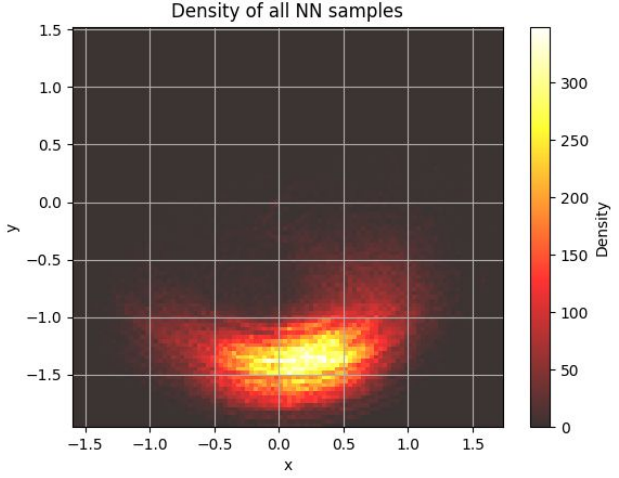


Results - Feed-Forward Neural Networks

- Can somewhat mimic the the movement of the double pendulum
  - Very noisy predictions
- Heatmap shows that the oscillation is not centered on x=0 as it should (it drifts slightly to the right)
- Movement not along the “fully stretched” oscillating path is underrepresented

```
self.typeStr = 'NN'
flatten_input = tf.keras.layers.Flatten()(input2)
flatten_input = tf.keras.layers.Dropout(0.3)(flatten_input)
concat_output = tf.keras.layers.concatenate([input1, flatten_input])
optimizer = 'adam'
```

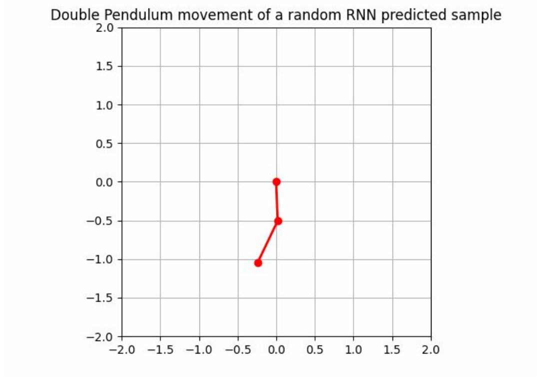


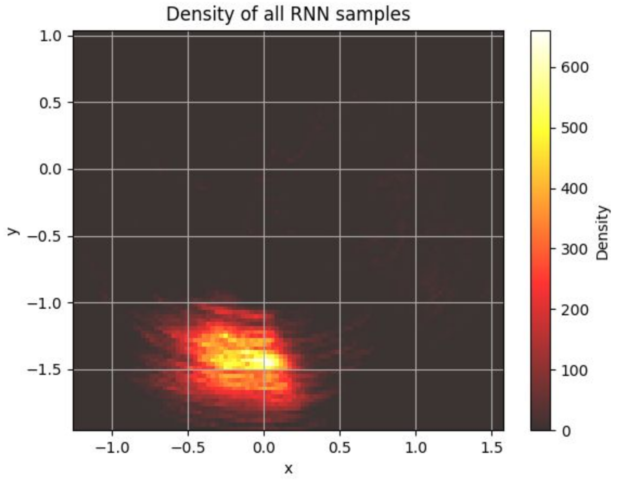


Results - Recurrent Neural Networks

- Less noisy than NN, but suffers from overfitting
  - Doesn’t oscillate much
- Almost no predictions exists for x<-1
  - Suggests all left-side data has been overfitted towards the middle
- Again, very little movement

```
self.typeStr = 'RNN'
rnn_output = tf.keras.layers.LSTM(units=64, return_sequences=True)(input2)
rnn_output = tf.keras.layers.Flatten()(rnn_output)
rnn_output = tf.keras.layers.Dropout(0.3)(rnn_output)
concat_output = tf.keras.layers.concatenate([input1, rnn_output])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
```

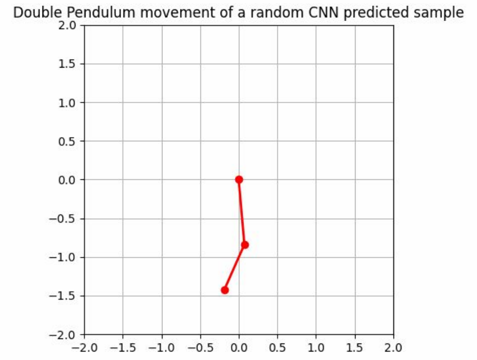


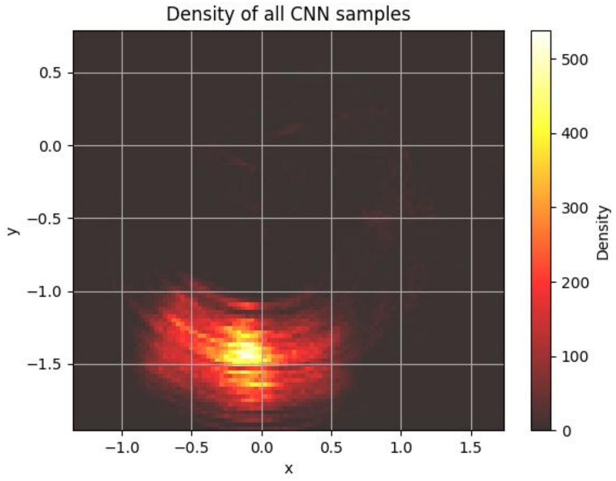


Results - Convolutional Neural Networks

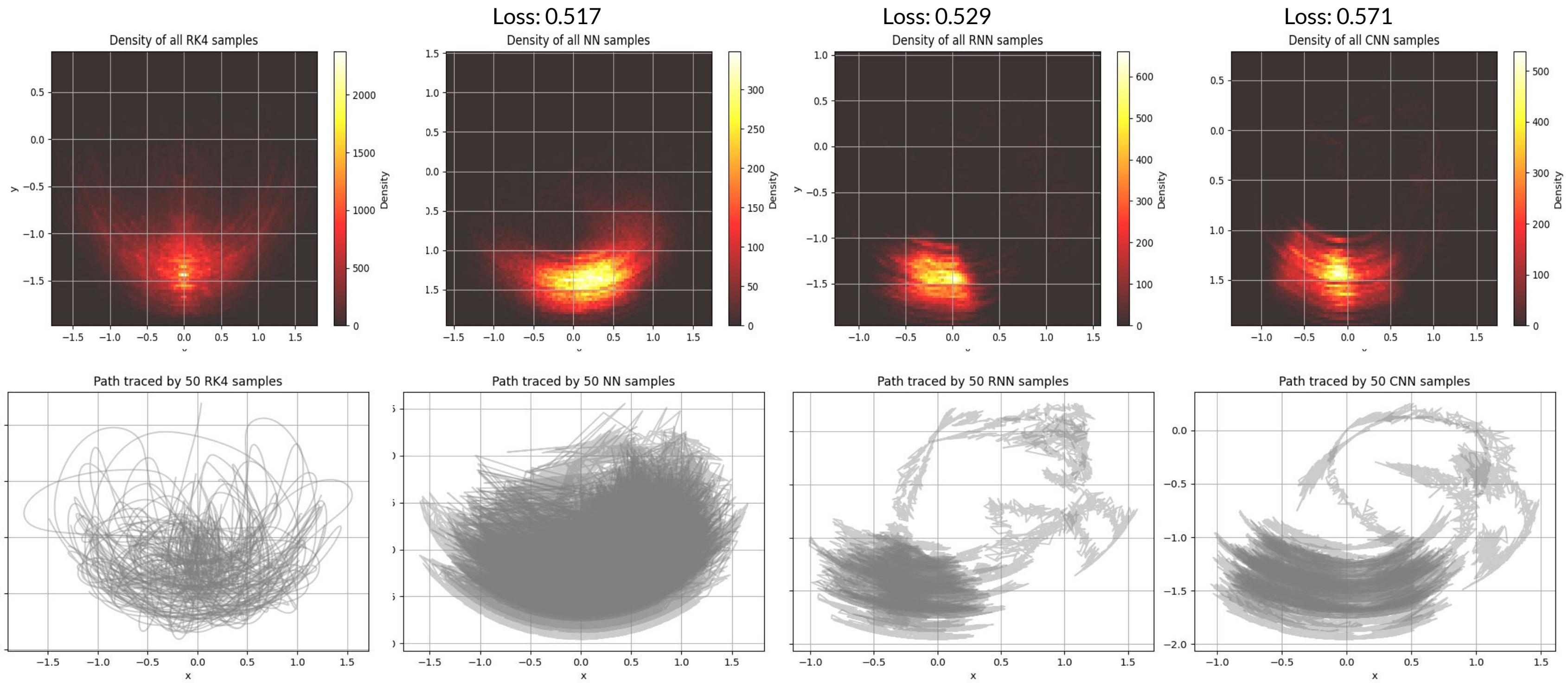
- Similar to that of RNN, but suffers less from overfitting
  - “Wider” range/ more movement than RNN
- Also has almost no predictions for x<-1

```python
self.typeStr = 'CNN'
reshape_input2 = tf.keras.layers.Reshape((1000, 2, 1))(input2)
conv_output = tf.keras.layers.Conv2D(filters=64, kernel_size=(2, 1), activation='relu')(reshape_input2)
flatten_output = tf.keras.layers.Flatten()(conv_output)
flatten_output = tf.keras.layers.Dropout(0.3)(flatten_output)
concat_output = tf.keras.layers.concatenate([input1, flatten_output])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
```





Results - Overall



New Results

- Dataset size increased
  - 1000 -> 10000
- Deeper networks
  - More aggressive L2-regularization to combat overfitting
  - RNN and CNN are no longer 1.5Gb and 3Gb respectively
    - Faster training time - more efficient iteration
- MAE as opposed to MSE for loss
- Absolute difference in heatmaps
- Noise reduction for animated plots

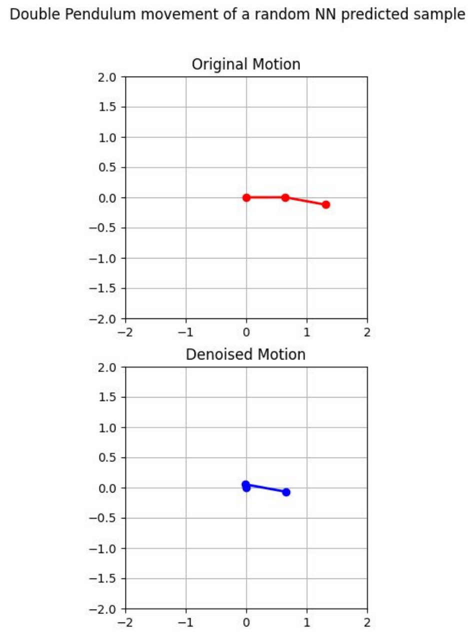
New Results - Feed-Forward Neural Networks

- Motion now more closely resembles that of RK4
  - Can predict normal oscillation
- Still some noise

Code:

```python
self.typeStr = 'NN'
flatten_input = tf.keras.layers.Flatten()(input2)
flatten_input = tf.keras.layers.Dropout(0.3)(flatten_input)
concat_output = tf.keras.layers.concatenate([input1, flatten_input])
optimizer = 'adam'
```

```python
self.typeStr = 'NN'
regu = 0.001
loss = 'mae'
acti=None
flatten_input = tf.keras.layers.Flatten()(input2)
flatten_input = tf.keras.layers.Dense(units=128, activation='relu')(flatten_input)
flatten_input = tf.keras.layers.Dense(units=64, activation='linear')(flatten_input)
flatten_input = tf.keras.layers.Dropout(0.3)(flatten_input)
concat_output = tf.keras.layers.concatenate([input1, flatten_input])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.00075)
```



New Results -
Recurrent Neural Networks

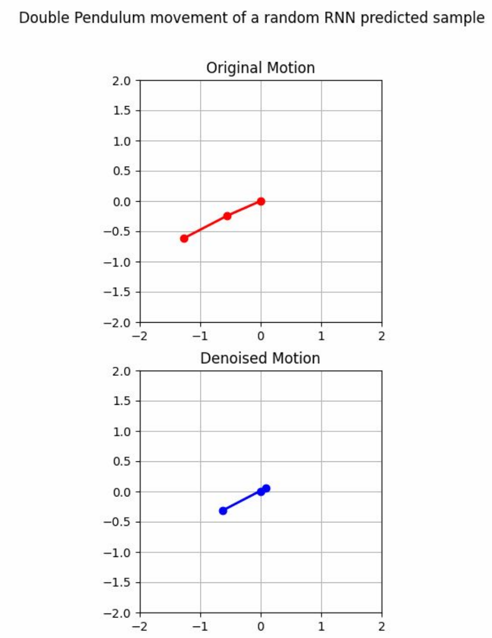
- Again much closer resembles that of RK4
- Still some noise

Code:

```
self.typeStr = 'RNN'
rnn_output = tf.keras.layers.LSTM(units=64, return_sequences=True)(input2)
rnn_output = tf.keras.layers.Flatten()(rnn_output)
rnn_output = tf.keras.layers.Dropout(0.3)(rnn_output)
concat_output = tf.keras.layers.concatenate([input1, rnn_output])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
```

↓

```
self.typeStr = 'RNN'
regu = 0.0002
loss = 'mae'
acti=None
rnn_output = tf.keras.layers.LSTM(units=32, return_sequences=True, activation='relu')(input2)
rnn_output = tf.keras.layers.LSTM(units=16, return_sequences=True, activation='linear')(rnn_output)
rnn_output = tf.keras.layers.Flatten()(rnn_output)
rnn_output = tf.keras.layers.Dropout(0.15)(rnn_output)
concat_output = tf.keras.layers.concatenate([input1, rnn_output])
concat_output = tf.keras.layers.Dense(units=8, activation='relu')(concat_output)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.00025)
```



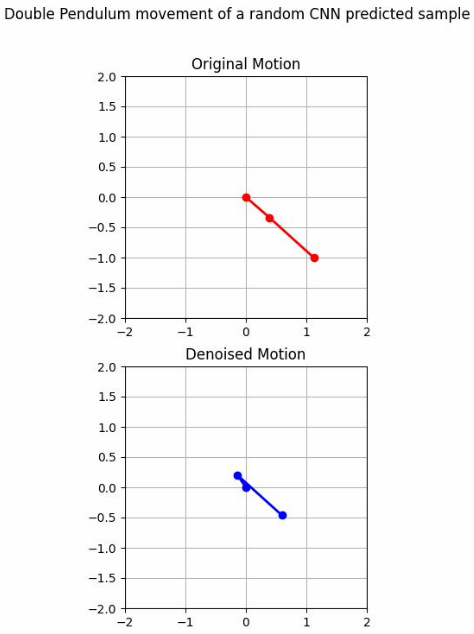
New Results - Convolutional Neural Networks

- Predicts in line with RK4 movement
- Noisier than NN and RNN

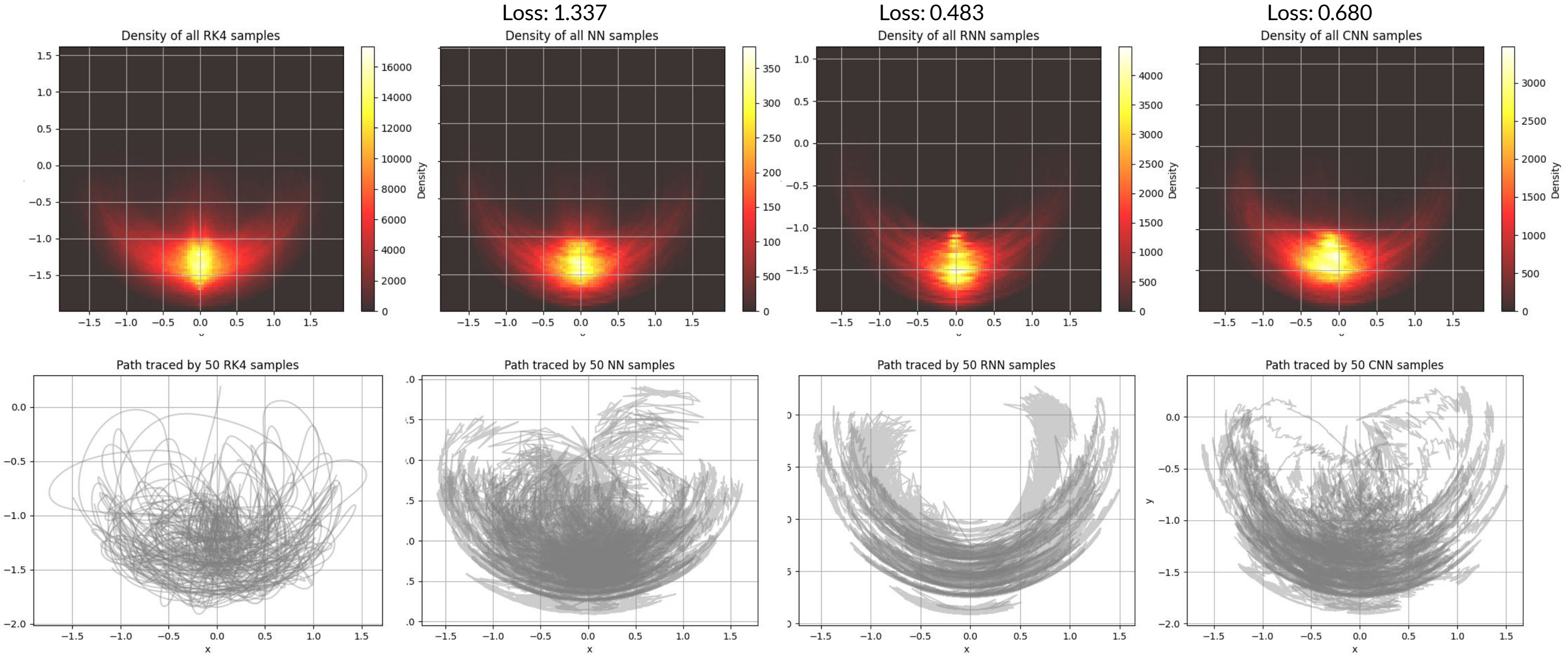
Code:

```
self.typeStr = 'CNN'
reshape_input2 = tf.keras.layers.Reshape((1000, 2, 1))(input2)
conv_output = tf.keras.layers.Conv2D(filters=64, kernel_size=(2, 1), activation='relu')(reshape_input2)
flatten_output = tf.keras.layers.Flatten()(conv_output)
flatten_output = tf.keras.layers.Dropout(0.3)(flatten_output)
concat_output = tf.keras.layers.concatenate([input1, flatten_output])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
```

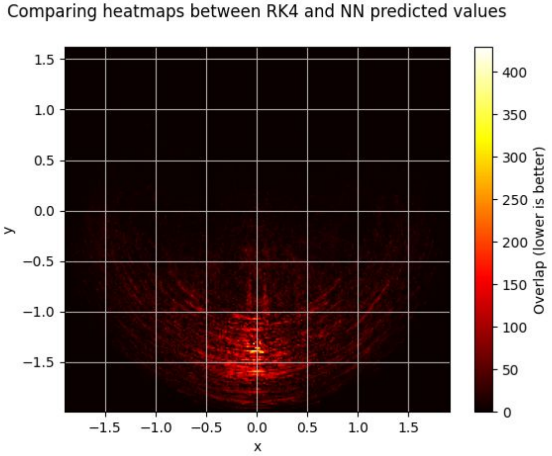
```
self.typeStr = 'CNN'
regu = 0.001
loss = 'mae'
acti='linear'
reshape_input2 = tf.keras.layers.Reshape((1000, 2, 1))(input2)
conv_output = tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 1), activation='relu')(reshape_input2)
conv_output = tf.keras.layers.MaxPooling2D(pool_size=(2, 1))(conv_output)
flatten_output = tf.keras.layers.Flatten()(conv_output)
flatten_output = tf.keras.layers.Dropout(0.5)(flatten_output)
concat_output = tf.keras.layers.concatenate([input1, flatten_output])
concat_output = tf.keras.layers.Dense(units=16, activation='relu')(concat_output)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0003)
```

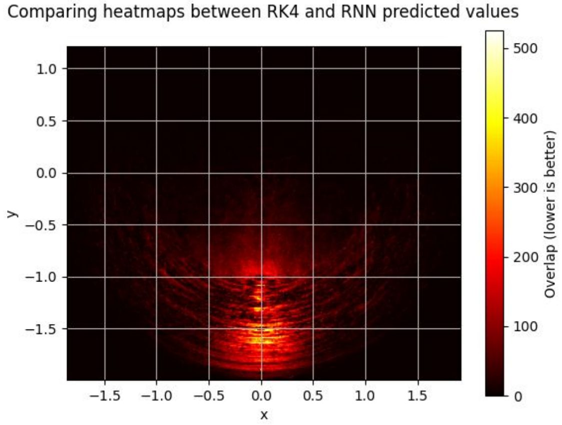


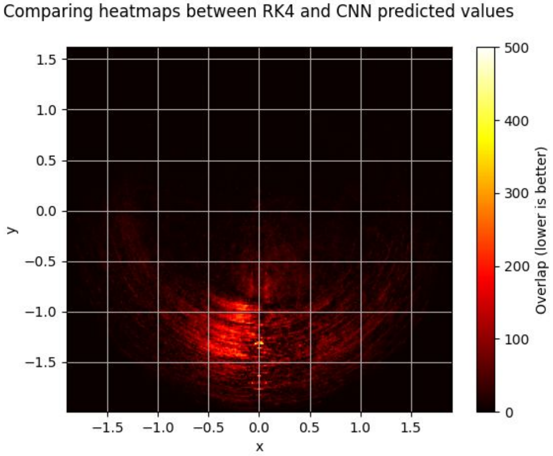
New Results - Overall



New Results - Absolute difference

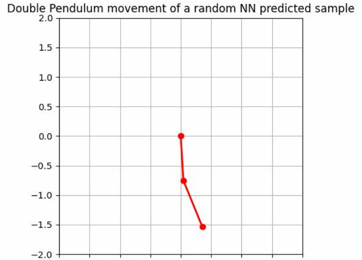


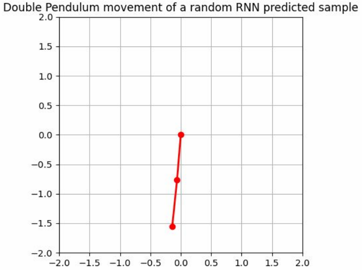


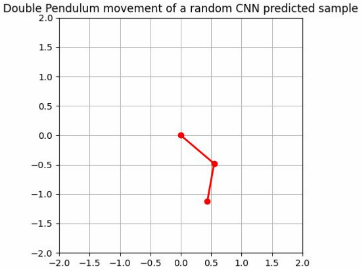


Discussion

- Overfitting
  - Unable to sufficiently capture the complexity of the system
- Reducing the complexity of the system
  - Constants
- Add angular velocity as an “input” to the model
- Unimplemented methods
  - Custom loss function for penalizing “illegal” changes in energy
- Three neural networks triples the time it takes to find a good model
- Utilizing a GPU for faster workflow/model iteration
  - Tensorflow 2.10 last version to support GPUs on Windows
  - Would allow for a bigger dataset/more complex model
- Not all predictions are “good”
- A better question to ask may be:
  Are Neural Networks suitable for predicting chaotic systems?







Conclusion

- RNN and CNN are closest in performance
    - In the original results CNN performed the best
    - In the new results, RNN performs the best
        - Still overfit
- Hard to define a clear winner
    - NNs not suited to chaotic system with high volatility in regards to initial variables
    - Can predict motion along a single pendulum path, outside of that general area there is generally too much noise to draw any conclusions.