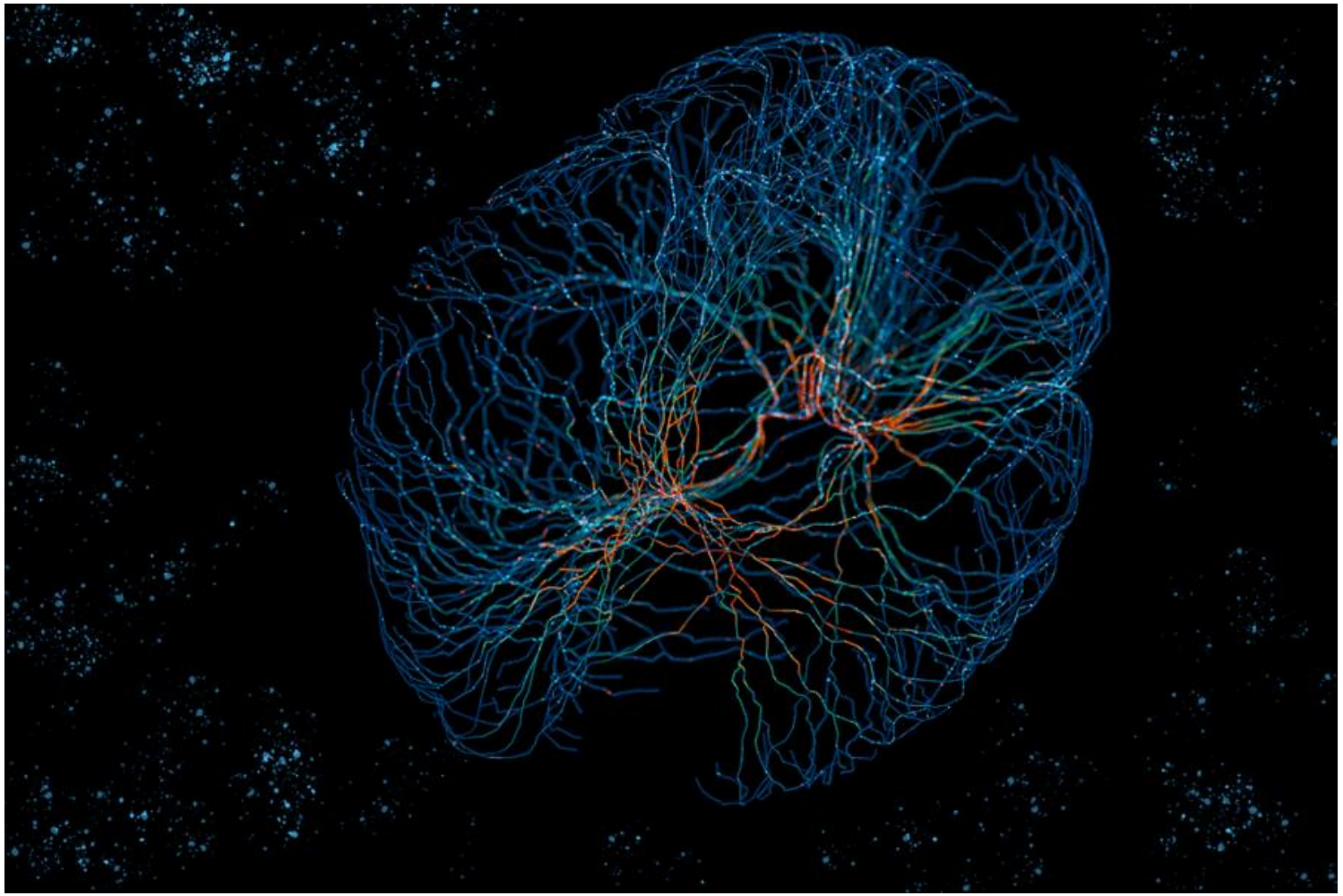


# A Physics-informed Neural Network modelling of a DC Motor

Boboye Oladosu Okeya (BOO)

# *What is a Physics-informed Neural Network (PINN)?*

*From my research studies, PINN is just your regular backpropagated neural network….*

*With the exception that the network's loss is not just a function of the error between the actual output and the predicted output….*

*The loss includes the residual losses which are derived your systems dynamic equations [1].*

*The inclusion of these residuals helps to ensure that any deviation of your network's output from the underlying physical law is penalized [2].*

# *So why use PINN?*

# Physics-Informed Neural Network (PINN)

- Performs state (speed and current) and parameter estimation (resistance, inertia, etc) [1]
- Faster that conventional numerical solvers [2]
- Has higher accuracy and lesser computational costs than conventional numerical solvers [2]
- Exploits the underlying physical laws [1,2]

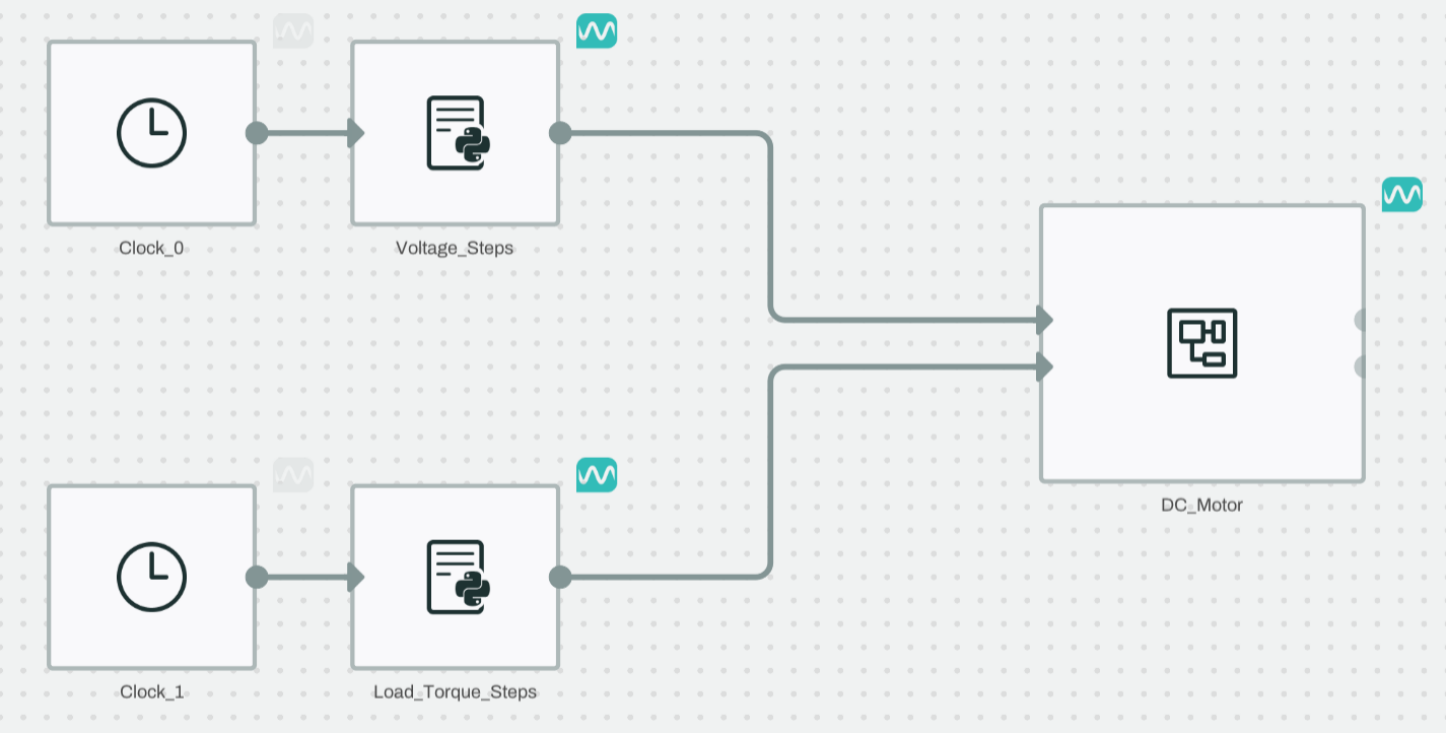
*So let's have some fun and try it out to see how and if it truly works*

*But first, lets generate the data that we will use to build the network…*

*I built a computer model of a DC motor using…Our Very Own--The Collimator*

# Data Generation from Collimator

- The Collimator Model is shown in the figure
- Collimator allows for easy download of your simulations data
- It comes by default in the .csv format



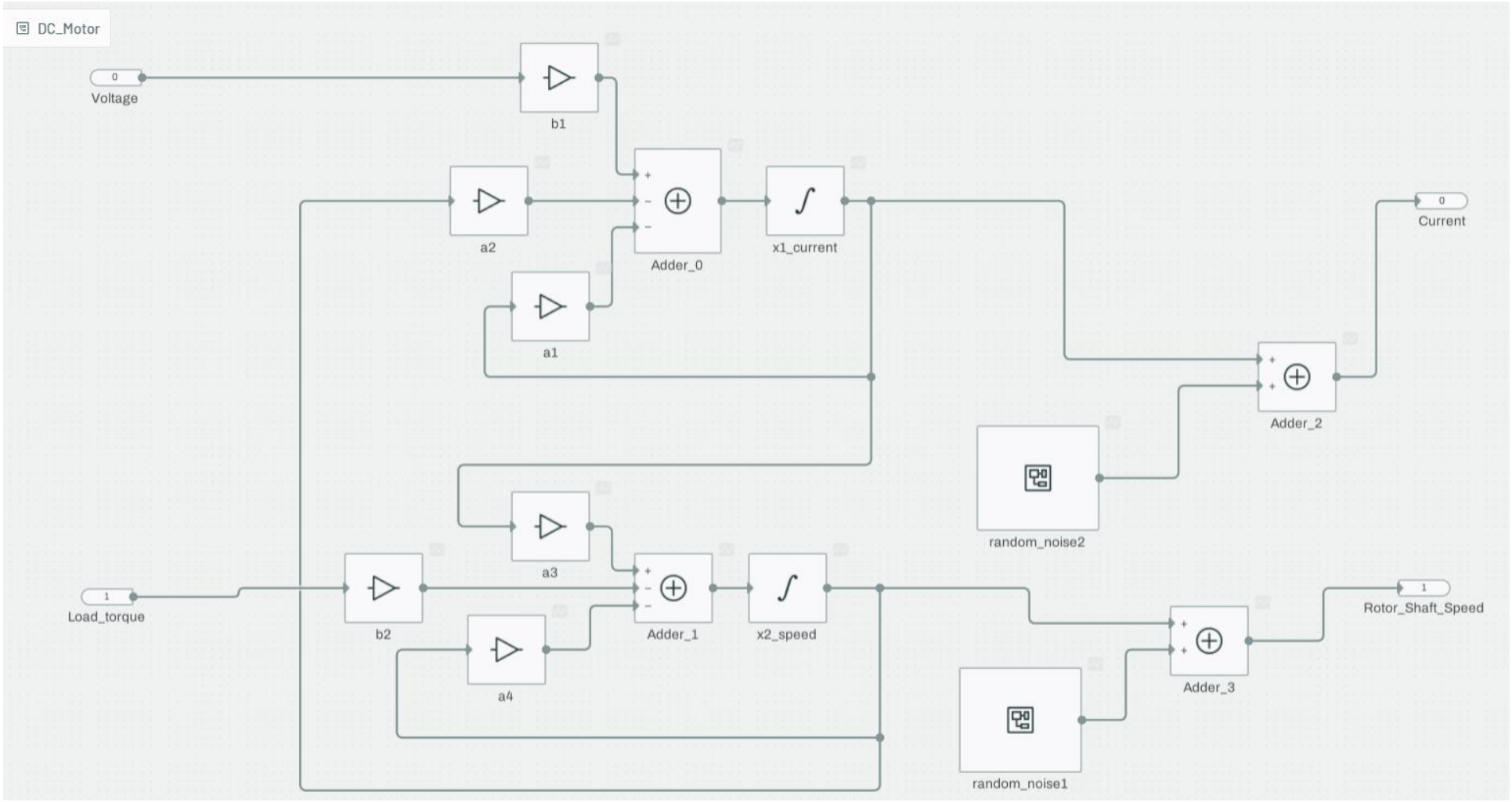
# Simulation Parameters

- Simulation Time: 60 seconds
- Solver: fixed step , 0.01
- Motor Parameters in fig

```
#DC Motor's Parameters
R = 0.6 #ohms armature resistance
L = 0.035 #Henry armature inductance
Kb = 0.0191 #BEMF constant
Kt = 0.0187 #Torque constant
J = 0.000125 #Rotor's momement of inertia
B = 0.0000095 #Damping coefficient
```

# DC Motor Block

- I added some noise to the outputs—speed and current



# Voltage Steps

- Training Profile

Physics_Model / Voltage_Steps

Step

```
#Training Data profile
Ts = 60
if in_0 >=0 and in_0 < 10:
    out_0 = 4
elif in_0 >=10 and in_0 < 20:
    out_0 = 8
elif in_0 >=20 and in_0 < 30:
    out_0 = 12
elif in_0 >=30 and in_0 < 40:
    out_0 = 16
elif in_0 >=40 and in_0 < 50:
    out_0 = 20
elif in_0 >=50 and in_0 <=60:
    out_0 = 24

```

- Testing Profile

Physics_Model / Voltage_Steps

Step

```
#Testing Data profile
Ts = 60
if in_0 >=0 and in_0 < 8:
    out_0 = 3
elif in_0 >=8 and in_0 < 15:
    out_0 = 6
elif in_0 >=15 and in_0 < 28:
    out_0 = 8
elif in_0 >=28 and in_0 < 35:
    out_0 = 15
elif in_0 >=35 and in_0 < 48:
    out_0 = 18
elif in_0 >=48 and in_0 <=60:
    out_0 = 24
```

# Load Torque Steps

- Training Profile

Physics_Model / Load_Torque_Steps

Step

```
# # Training Data Profile
Ts = 60
if in_0 >=0 and in_0 < 5:
    out_0 = 0.1
elif in_0 >=5 and in_0 < 15:
    out_0 = 0.15
elif in_0 >=15 and in_0 < 25:
    out_0 = 0.2
elif in_0 >=25 and in_0 < 30:
    out_0 = 0.3
elif in_0 >=30 and in_0 < 35:
    out_0 = 0.4
elif in_0 >=35 and in_0 < 55:
    out_0 = 0.55
elif in_0 >=55 and in_0 <=60:
    out_0 = 0.6
```

- Testing Profile

Physics_Model / Load_Torque_Steps

Step

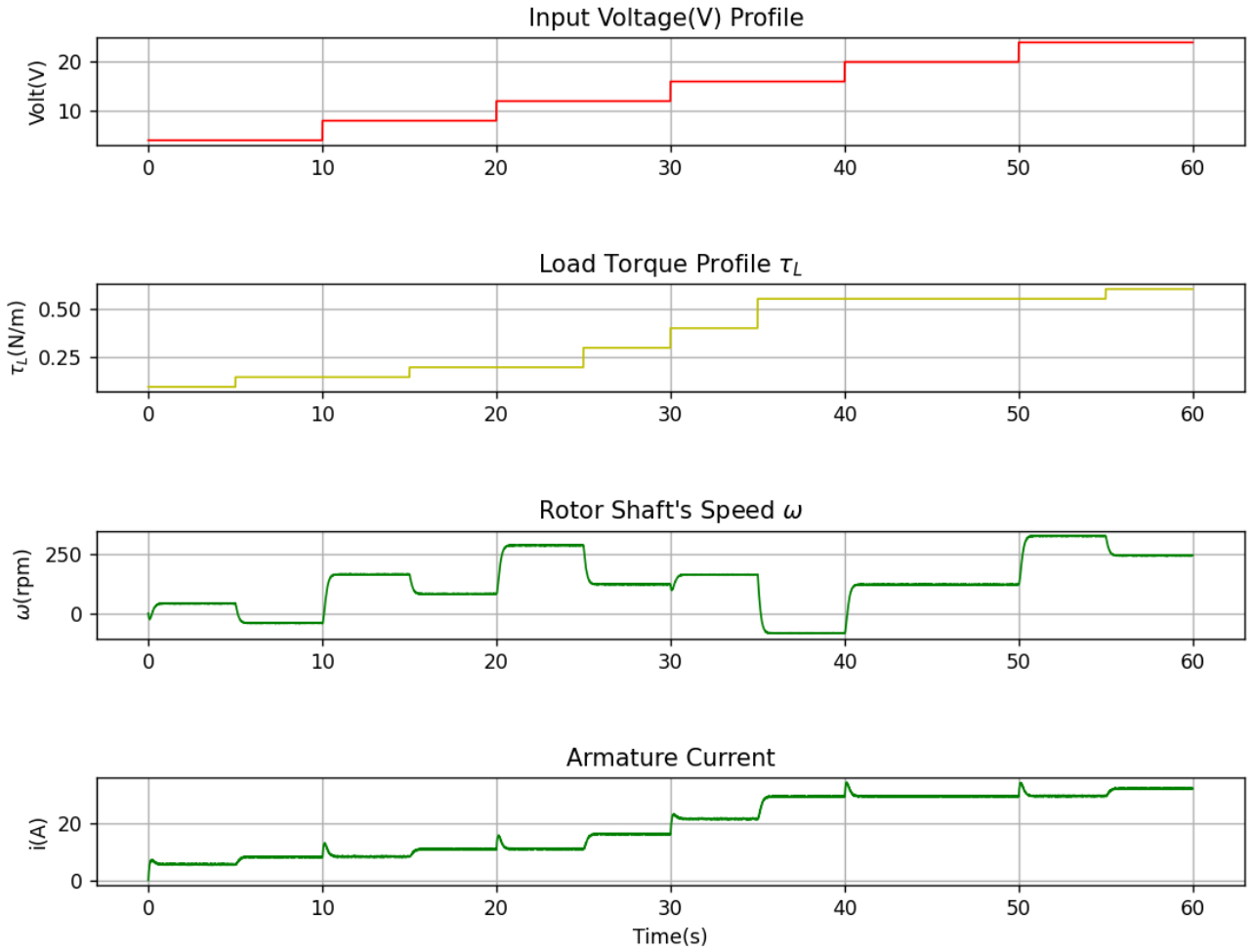
```
# Testing Data Profile
Ts = 60
if in_0 >=0 and in_0 < 10:
    out_0 = 0.05
elif in_0 >=10 and in_0 < 20:
    out_0 = 0.2
elif in_0 >=20 and in_0 < 27:
    out_0 = 0.3
elif in_0 >=27 and in_0 < 30:
    out_0 = 0.1
elif in_0 >=30 and in_0 < 38:
    out_0 = 0.35
elif in_0 >=38 and in_0 < 52:
    out_0 = 0.4
elif in_0 >=52 and in_0 <=60:
    out_0 = 0.6
```

*Super!*

*Let's visualize the training data gotten from collimator*

# Visualization of the Training Data

- The training data included all the DC motor's operation in 2 modes:
  a. Forward motoring
  b. Reverse braking
- Such training allows the PINN to learn about those modes



*Gute!!!*

*Next, lets build a PINN*

# PINN's Architecture

- I have used the exact architecture used in [1] as shown in the figure 11
- In [1], 2 PINNs where used to model the electrical and mechanical subsystems separately
- I used the same architecture as [1]

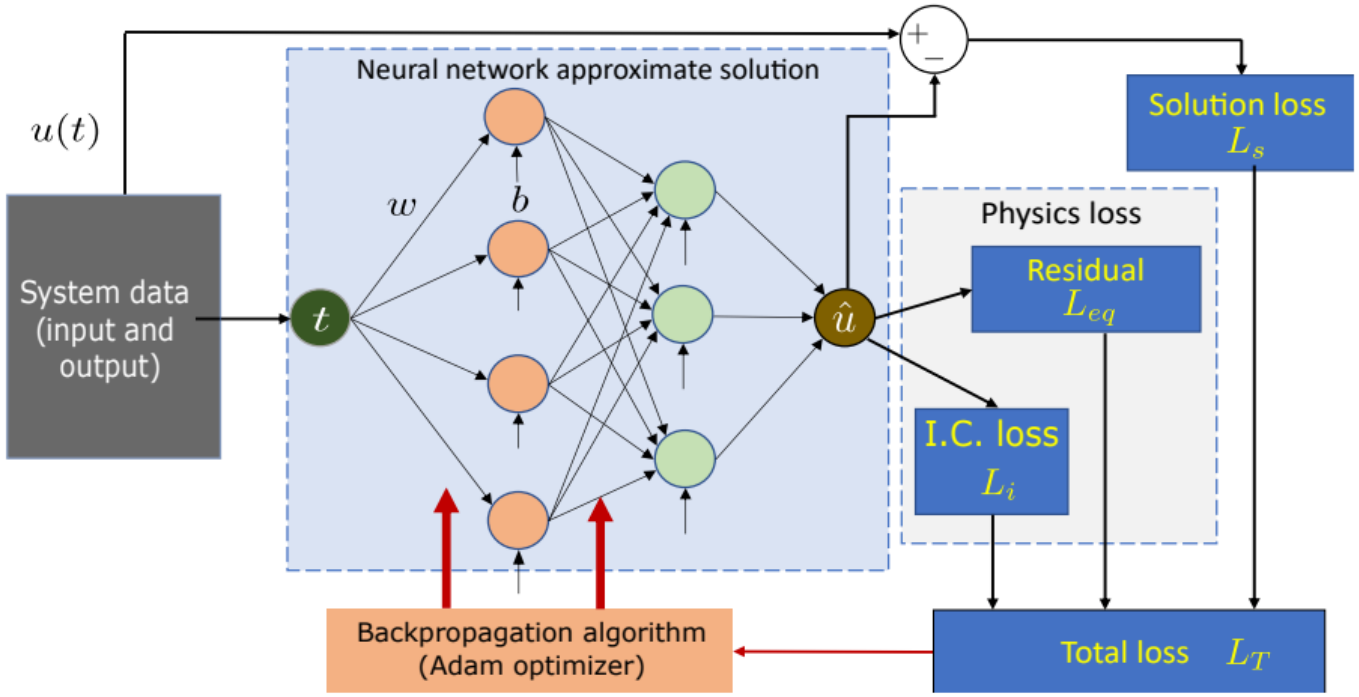



**Figure 11.** A PINN approach based on physics-guided loss formulation.

*After building the PINN, let's test out some hyperparameters to find the most optimal.*

*And finally we will used the optimal set of hyperparameter to train the PINN and test the PINN with our test data*

# Hyperparameter Tuning

- I tested out 36 combinations of 11 types of hyperparameters as shown below

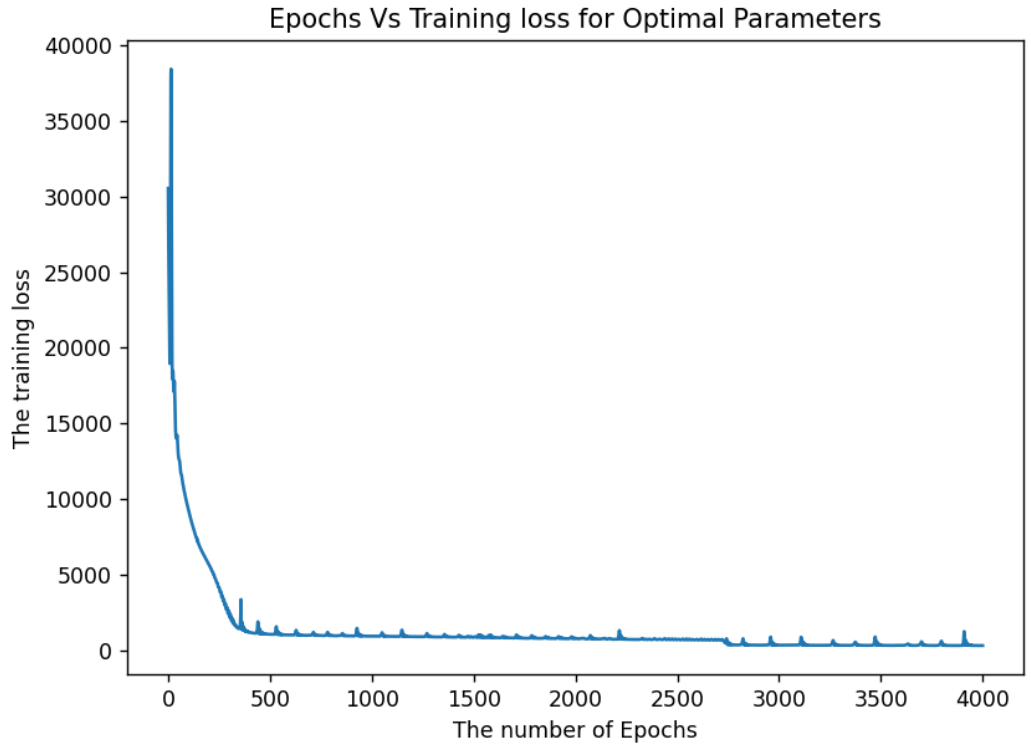
```
    # Define hyperparameter search space
    param_grid = {
        'activation1': ['relu'],
        'activation1_2': ['relu'],
        'epochs': [4000],
        'kernel_initializer': ['glorot_normal'],
        'learning_rate': [0.01],
          'no_of_hidden_layers': [3],
          'no_of_hidden_layers2': [4],
          'no_of_neurons_per_layer': [32,64],
          'no_of_neurons_per_layer2': [32,64],
        'dropout': [0.001, 0.01,0.1],
        'dropout_2':[0.001,0.01,0.1]
    }

    # Generate all possible combinations of hyperparameters
    param_combinations = list(ParameterGrid(param_grid))
```

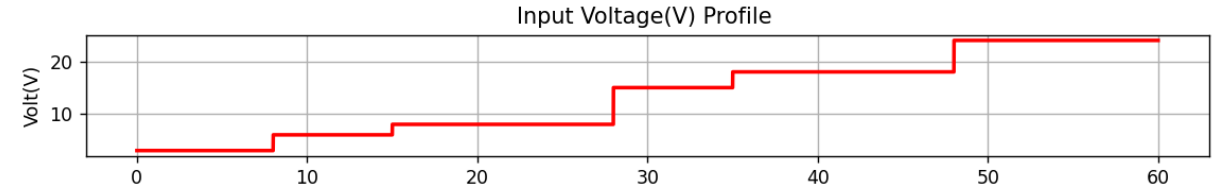
# Optimal Parameter

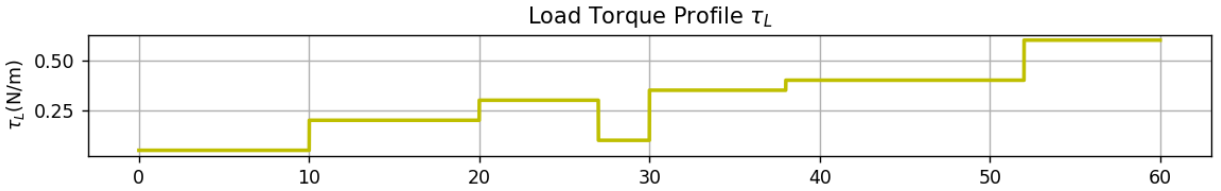
- Total tuning time ~3.75Hours
- The combination of hyperparameters that produced the lowest loss(~199.7)is as shown below
- The plot of the loss per epoch is shown in the figure to the right

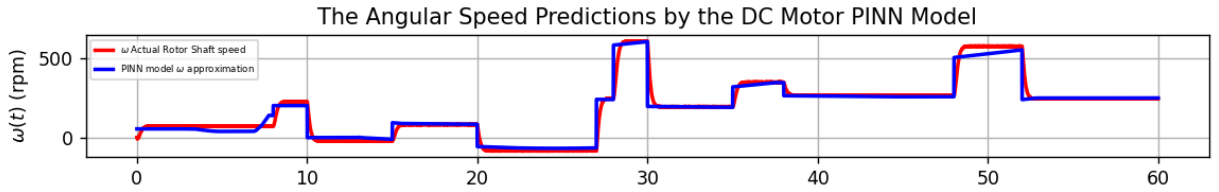
```
{'activation1': 'relu',
 'activation1_2': 'relu',
 'dropout': 0.01,
 'dropout_2': 0.01,
 'epochs': 4000,
 'kernel_initializer': 'glorot_normal',
 'learning_rate': 0.01,
 'no_of_hidden_layers': 3,
 'no_of_hidden_layers2': 4,
 'no_of_neurons_per_layer': 64,
 'no_of_neurons_per_layer2': 64}
```

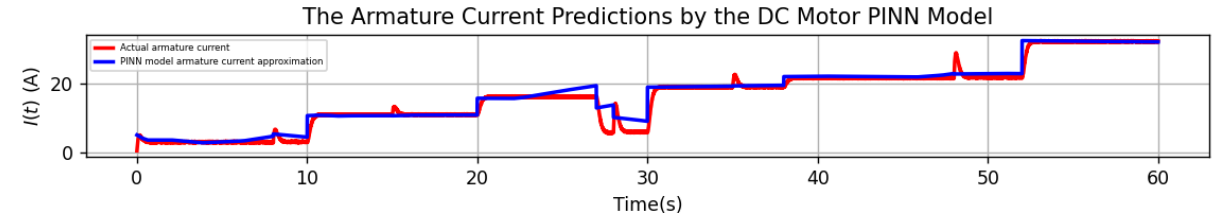


# Testing Results



## *What's next?*

1. *Generate and use data that spans the DC motor's 4 quadrants of operation*
2. *Improve the PINN architecture: Combine both the electrical &mechanical subsystems into a single PINN network with two outputs?*
3. *Deploy the models to a simulator and test its performance in a control application against a regular physics model*

*For more details 'click -> here' for the jupyter notebook and data*

# References

[1] Ayankoso S, Olejnik P. Time-Series Machine Learning Techniques for Modeling and Identification of Mechatronic Systems with Friction: A Review and Real Application. Electronics. 2023; 12(17):3669. https://doi.org/10.3390/electronics12173669

(Link to the research's codes: [here])

[2] G. S. Misyris, A. Venzke and S. Chatzivasileiadis, "Physics-Informed Neural Networks for Power Systems," 2020 IEEE Power & Energy Society General Meeting (PESGM), Montreal, QC, Canada, 2020, pp. 1-5, doi: 10.1109/PESGM41954.2020.9282004.

(Link to the research's code: [here])