EE272 : EXPLORATORY PROJECT PRESENTATION

EVENT DETECTION IN A POWER MICROGRID USING DEEP LEARNING

DEPARTMENT OF ELECTRICAL ENGINEERING
IIT(BHU), VARANASI

PROJECT CO-ORDINATOR

Dr. Soumya Ranjan Mohanty

PROJECT MEMBERS

Rishav Singh	**20085084**
Rohan Saha	**20085086**
Souvik Karmakar	**20085096**

ACKNOWLEDGEMENT

Its our Privilege to express our Sincerest Regards to our Project Coordinator, **Dr. Soumya Ranjan Mohanty** for his valuable Inputs, Able Guidance, Encouragement, Whole-Hearted Cooperation & Constructive Criticism throughout the Duration of our Project. It has been a Great Learning Experience for us, and we have been able to Broaden our Knowledge of both **Deep Learning** and **Event Detection in Power Microgrids**.

We also Dedicate this Space to Express our Heartiest Regards to Each & Every Person who has Supported us to give our Best for this Project and to our **Family & Friends** who have always Motivated us. Finally, we Express Gratitude to our **Team Members** for their Cooperation & Continuous Dedicated Efforts for this Project.

DEPARTMENT OF ELECTRICAL ENGINEERING
IIT(BHU), VARANASI

ABSTRACT / OVERVIEW OF THE PROJECT

AIM - Fault Detection & Location in a **Power Microgrid** using **Artificial Neural Networks [ANN] & Other Deep Learning Features**

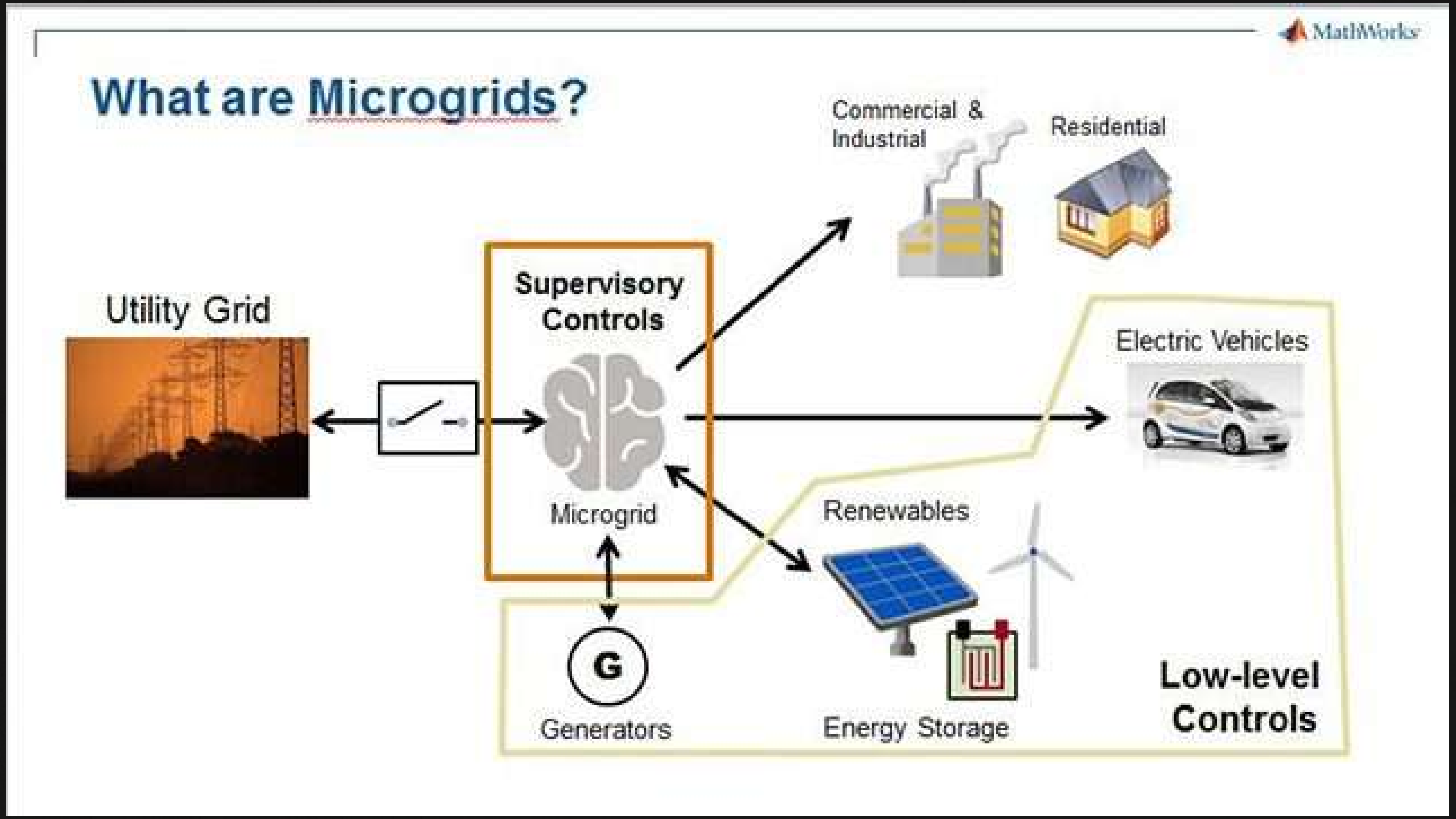
● An Event of **Fault** occurs when Two or more **Conductors come in contact with each other** or the **Ground**.
● Due to the **High Current Discharge** at the time of Fault Occurrence, the whole System might get **De-Energized** which would adversely impact on the entire System.
● **Ground Faults** are considered as one of the Leading Problems in Power Systems and account for more than 80% of all Faults.
● Hence an Effective Method to **Detect, Isolate**, and **Protect** the Power Microgrid System against the effects of Short Circuit Faults is Extremely Important.

OUR PROPOSED SOLUTION

● We have used a Highly Effective Method to Protect the Microgrid System using an **Artificial Neural Network [ANN]** that will **Detect** and **Locate** the **Fault** before it can affect other parts of the System.
● This Protection Network is distributed all along the Power Microgrid System, **Protecting** the **entire Network**, and is connected to the other Protective Devices in the System.
● This Project focuses on **Detecting & Accurately Locating** the Faults on Electric Power Transmission Lines in the Power Microgrid Network.
● We've used **3 Breakers** to Generate Events of Fault in a Power Network.
● On Availability of more data, our Approach can be **Generalised** to any Event that might take place in a Power Network.

INTRODUCTION TO MICROGRID

- With Increasing Environmental Concern, the Demand for introduction of Conventional Energy Systems as an Alternative for Old Energy Systems is of much Prevalence, but Disadvantages & Difficulties in Power Generation & Distribution such as Overvoltage, Fault Protection & Frequency Fluctuations are Obstacles that should be dealt with as well.
- The Electricity Network of the Future will need to accommodate Large Scale Distributed Generation Units [DGs] and facilitate the Connection of Grand Scale Centralized Generation at suitable locations.
- Microgrid has been considered as an Effective Way to manage the DGs and other Distributed Energy Resources [DERs] on the Distribution System Level & the User Level. Microgrids provide Efficient, Low Cost and Clean Energy. Their Infrastructure also Increases Reliability & Resilience and Reduces Grid "Congestion" & Peak Loads. Still, the Protection of Microgrids remains a Problem. When a Fault occurs in the System, it creates a Huge Current that could adversely affect the Entire System. The Possibility of Locating Faults quickly and Isolating that part from the other parts of the System would allow the rest of the Power Microgrid to continue working safely.



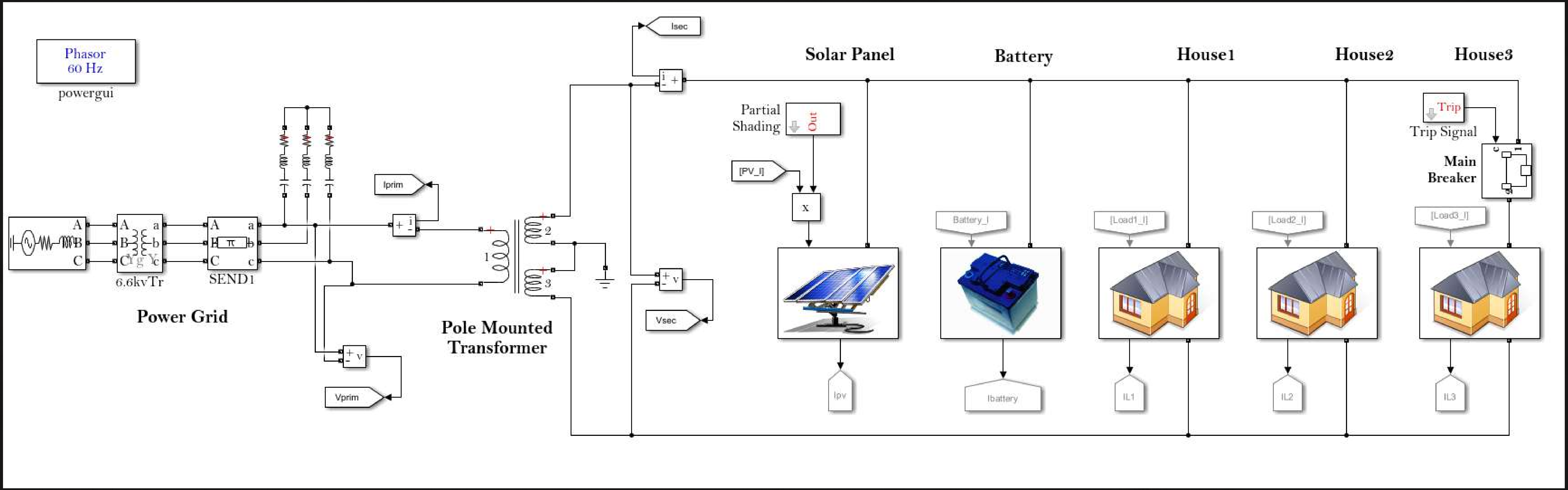
Simplified Schematics of a Microgrid

OUR MICROGRID MODEL

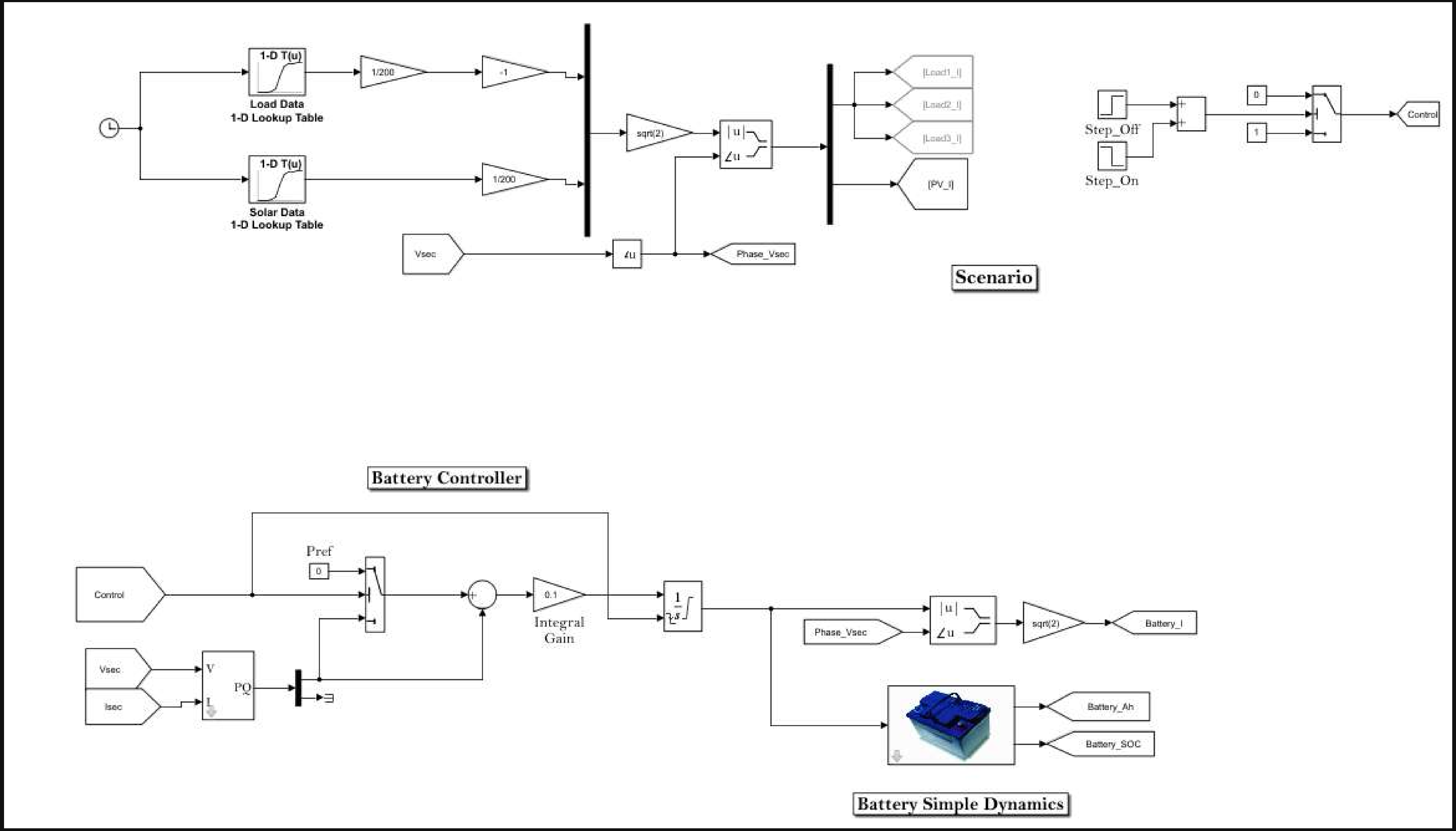
- The **Micro-grid** is a **Single-Phase AC Network**. The **Energy Sources** used in our Model here are an **Electricity Network**, a **Solar Power Generation System** and a **Storage Battery**.
- The **Storage Battery** is controlled by a **Battery Controller**. It absorbs Surplus Delta [Δ] Power when there is excess energy in the Micro-Network, and provides Additional Delta [Δ] Power if there is a Power Shortage in the Micro-Network.
- **Three [3] Ordinary Houses** consume Energy (maximum of 2.5 kW) as Electric Charges.
- The **Microarray** is connected to the Power Network via a **Transformer** mounted on a post which lowers the voltage of 6.6 kV to 200 V.
- The Solar Power Generation and Storage Battery are **DC Power Sources** that are converted to **Single-Phase AC**.
- The **Control Strategy** assumes that the Microarray does not depend entirely on the power supplied by the Power Grid, and the power supplied by the Solar Power Generation and Storage are sufficient at all times.

SUBSEQUENT PAGES DISCUSS THE MODEL & SCHEMATICS OF-

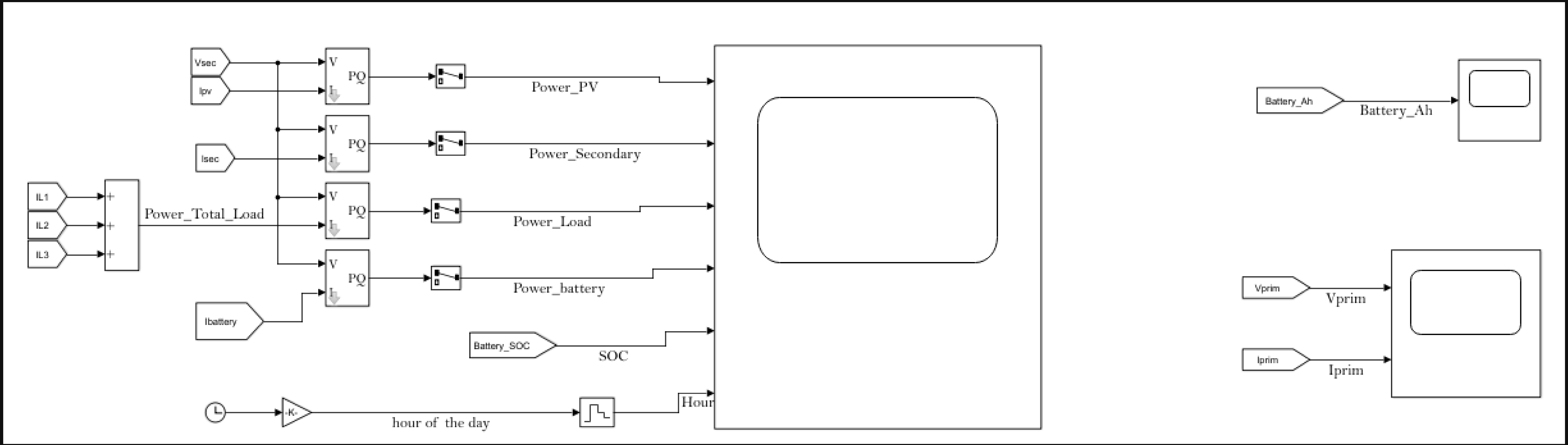
- **Simplified Model of a Microgrid**
- **Design of Scenario & Battery Controller**
- **Design of the Scopes**



Simplified Model of a Microgrid



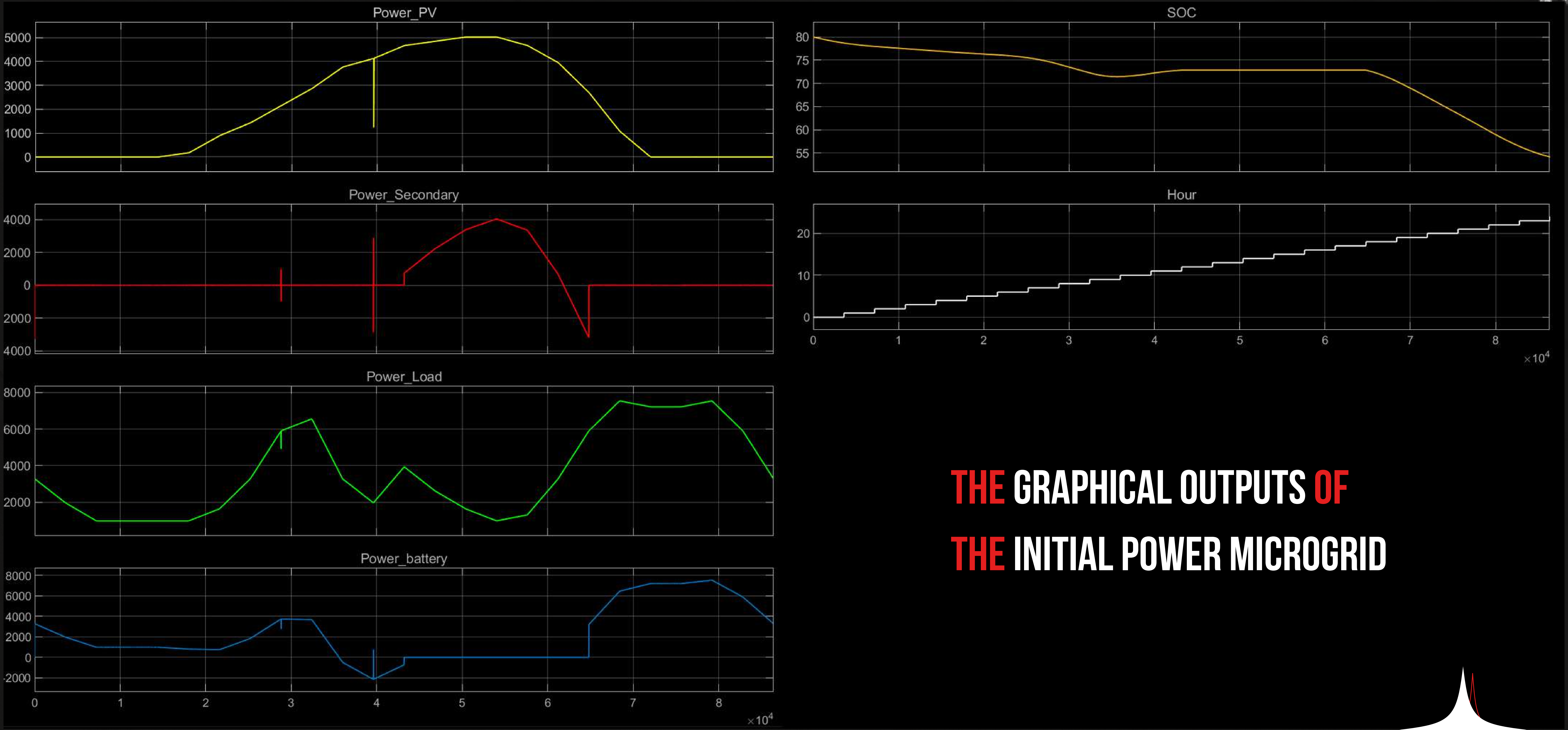
Design of Scenario & Battery Controller



Design of the Scopes

SIMULATION ON **MATLAB**

- The **Solar Power Generation** between 20:00 - 4:00 is zero [0] watts.
- Power Generation is **Maximum** between 14:00 - 15:00.
- As the **Typical Load changes** in ordinary houses, the amount of Electric Power load reaches **Peak Consumption** at 9:00 (6,500 W), 19:00, and 22:00 (7,500 W). From 0:00 to 12:00 and from 18:00 to 24:00, **Battery Control** is performed by the **Battery Controller.**
- The Battery Control performs **Tracking Control** of the Current so that the Active Power which flows into System Power from the Secondary Side of the Pole Transformer is set to 0. Then, the Active Power of the Secondary Side of the Pole mounted Transformer is always around zero [0].
- The **Storage Battery** supplies the **Insufficient Delta [△] Current** when the Power of the Microgrid is insufficient and **absorbs Surplus Delta [△] Current** from the Microgrid when its Power surpasses the Electric Load.
- From 12:00 to 18:00, Battery Control is not performed. **SOC [State Of Charge]** of the Storage Battery is fixed to a constant and does not change since charge or discharge of the Storage Battery are not performed by the Battery Controller.
- When there is a **Power Shortage** in the Micro-grid, the System Power supplies Insufficient Power. When there is a Surplus Power in the Micro-grid, the Excess Delta [△] Power is returned to the System Power. At 8:00, **Electricity Load No. 3** of an ordinary house is set to **OFF** for 10 sec by the Breaker. A **Spike** is observed in the Active Power on the Secondary side of the Pole Transformer and the Electric Power of the Storage Battery.



THE GRAPHICAL OUTPUTS OF THE INITIAL POWER MICROGRID

TYPES OF FAULTS IN A MICROGRID & THEIR DETECTION

● There are broadly **Two [2] types** of **Faults** which occur in a Microgrid-

i. **Line-to-Line Fault**
ii. **Line-to-Ground Fault.**

● The **Majority** of these Faults are Line-to-Ground Fault due to Component or Segment Failure or Lightning. When a **Short Circuit Fault** happens in the Microgrid, the Fault Resistance tends to zero [0] and the Current tends to become infinite.

$$I_f \rightarrow \infty \; , \; R_f \rightarrow 0$$

● Here, I_f is the **Fault Current**
and R_f is the **Fault Resistance**.

KEY IDEA FOR DETECTION OF FAULTS

● In this case, **Three [3] Circuit Breakers** are introduced in the circuits before each house as shown in the next slide. They behave as the Faults.

● The Properties of Breakers are -

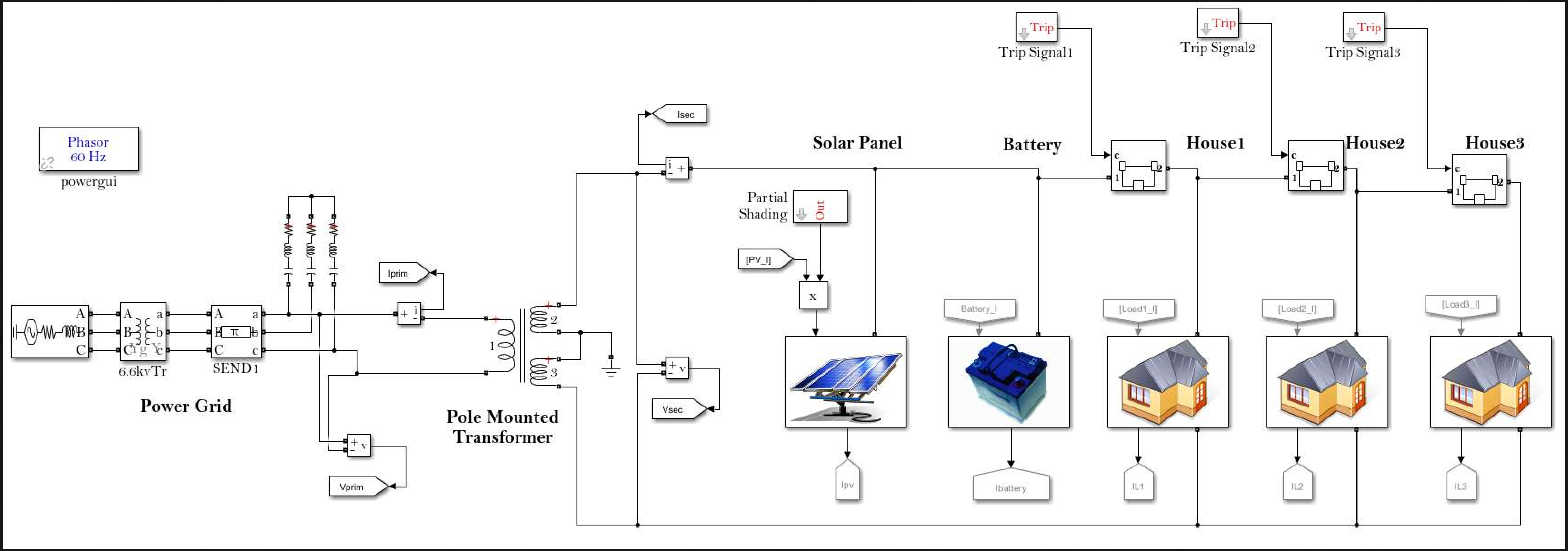
Breaker Resistance = 0.001 ohm,
Snubber Resistance = 1e6 ohm,
Snubber Capacitance → ∞.

● The Breakers are initially at **State-1** i.e., Not Activated.

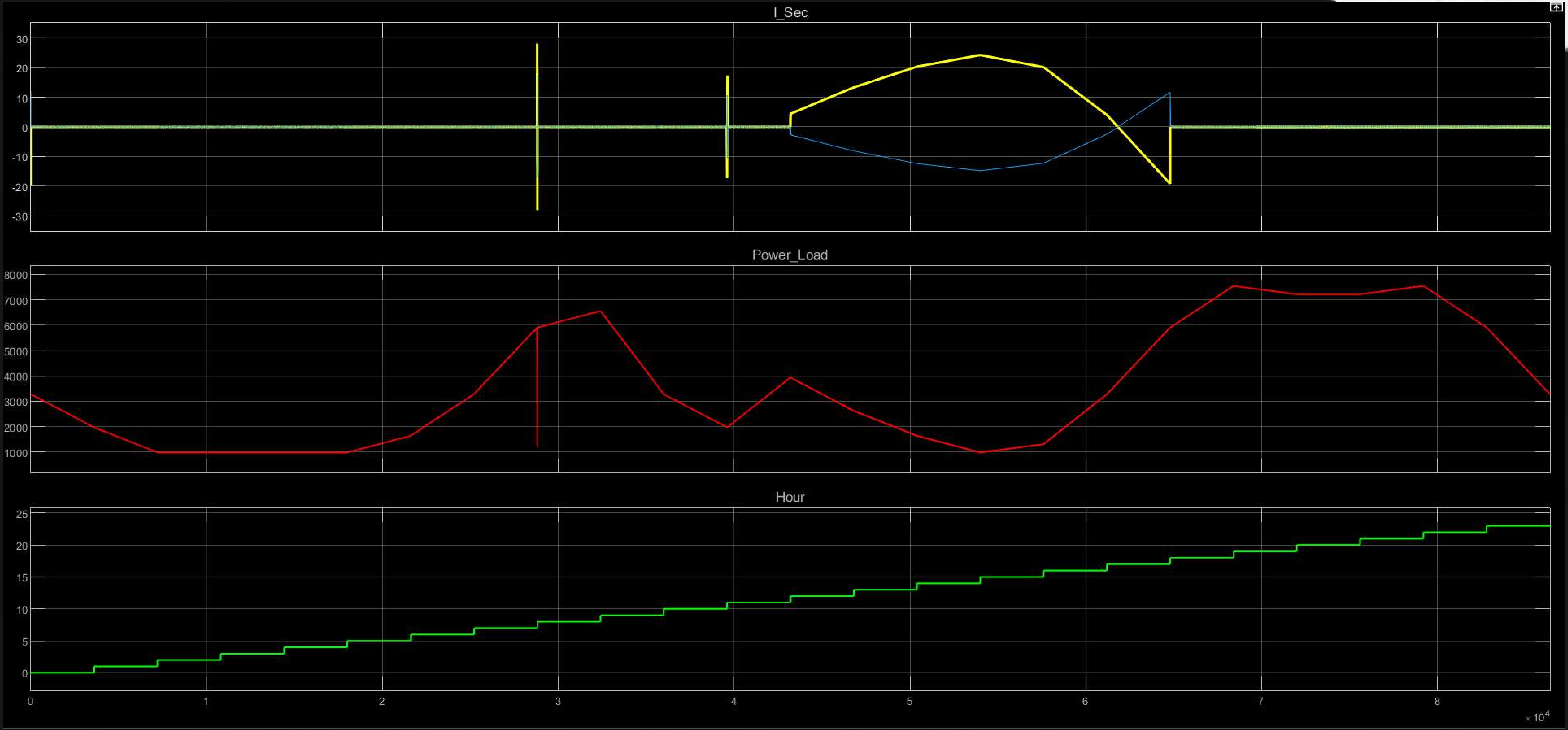
● Now they're Activated one-by-one for 1000s each time to get the Dataset. Each Breaker is activated at every hour of the day. So we get a total of **96 Data Points**.

● Whenever a Fault occurs in any part of the system, the values of **Secondary Current [Isec]** and **Load Power** changes drastically. These changed values play a Central Role in our Model. At every point Secondary Current [Isec] and Load Power value is taken into consideration along with their Time of Activation.

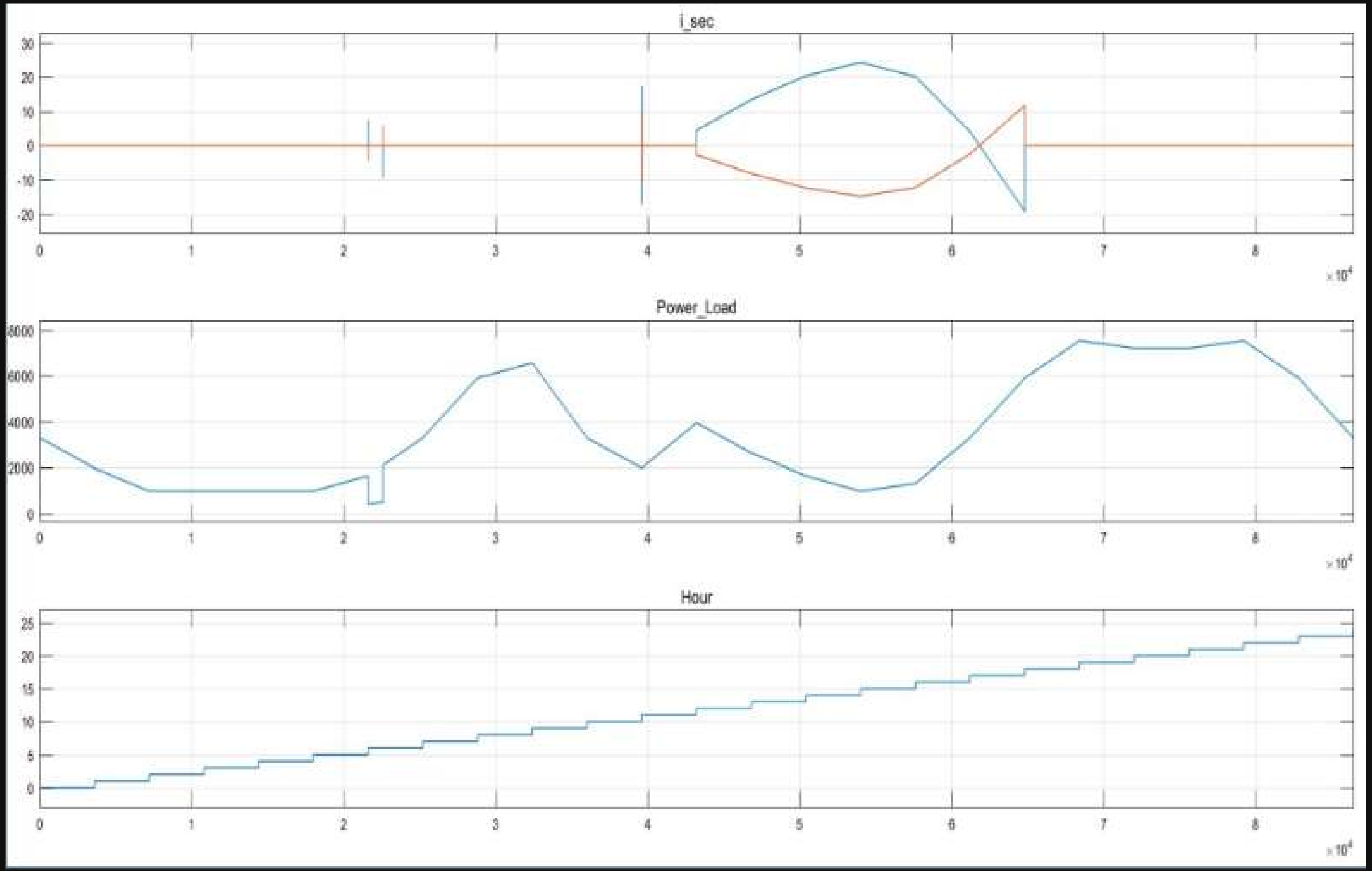
● The **Time of Fault/Activation** plays a Major Role because of the Nature of the Circuit.



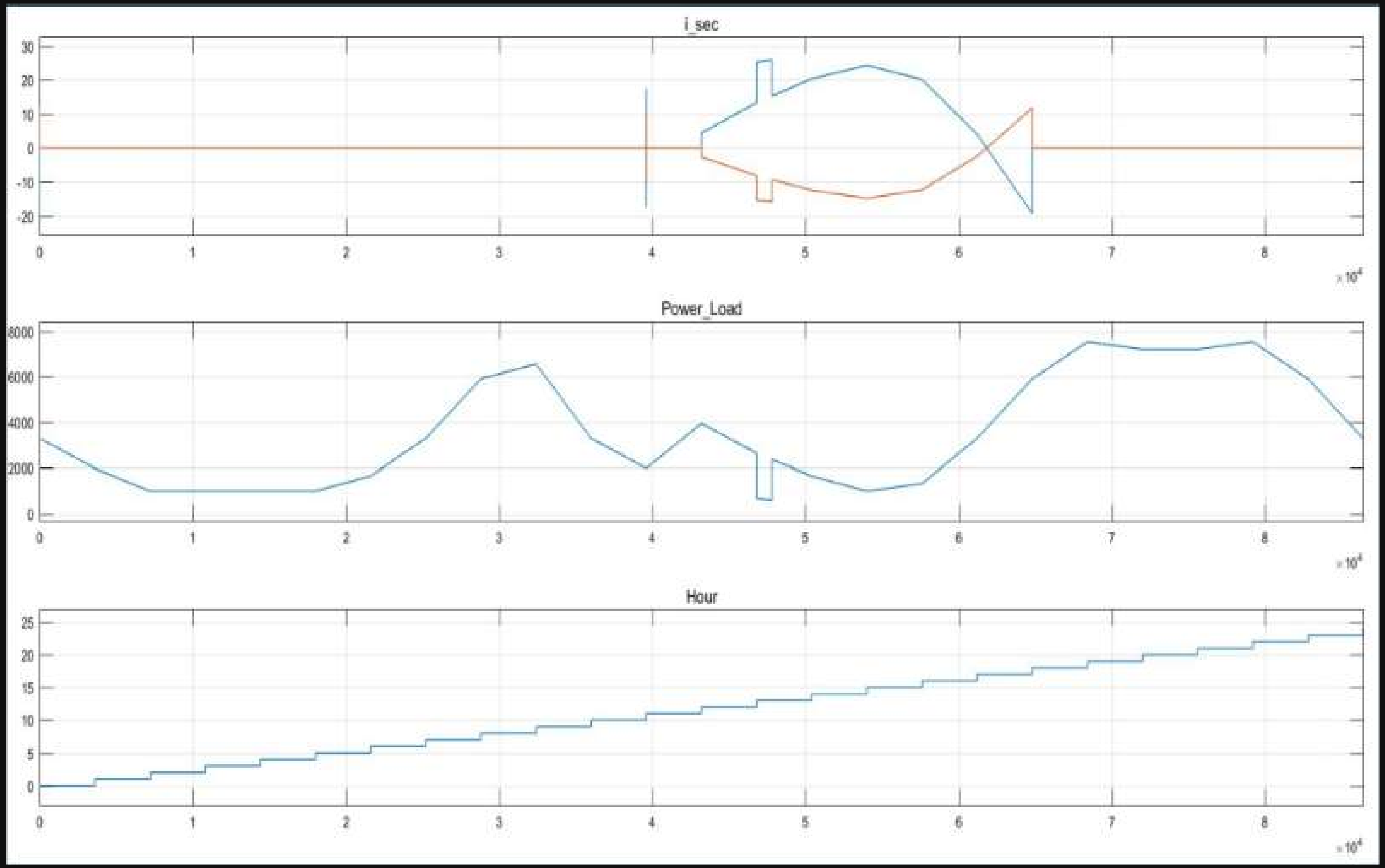
Three [3] Circuit Breakers introduced in the Initial Microgrid



WHEN ALL THE BREAKERS ARE IN THE OFF STATE I.E., NO FAULT STATE



FAULT WITH BATTERY CONTROLLER



FAULT WITHOUT BATTERY CONTROLLER

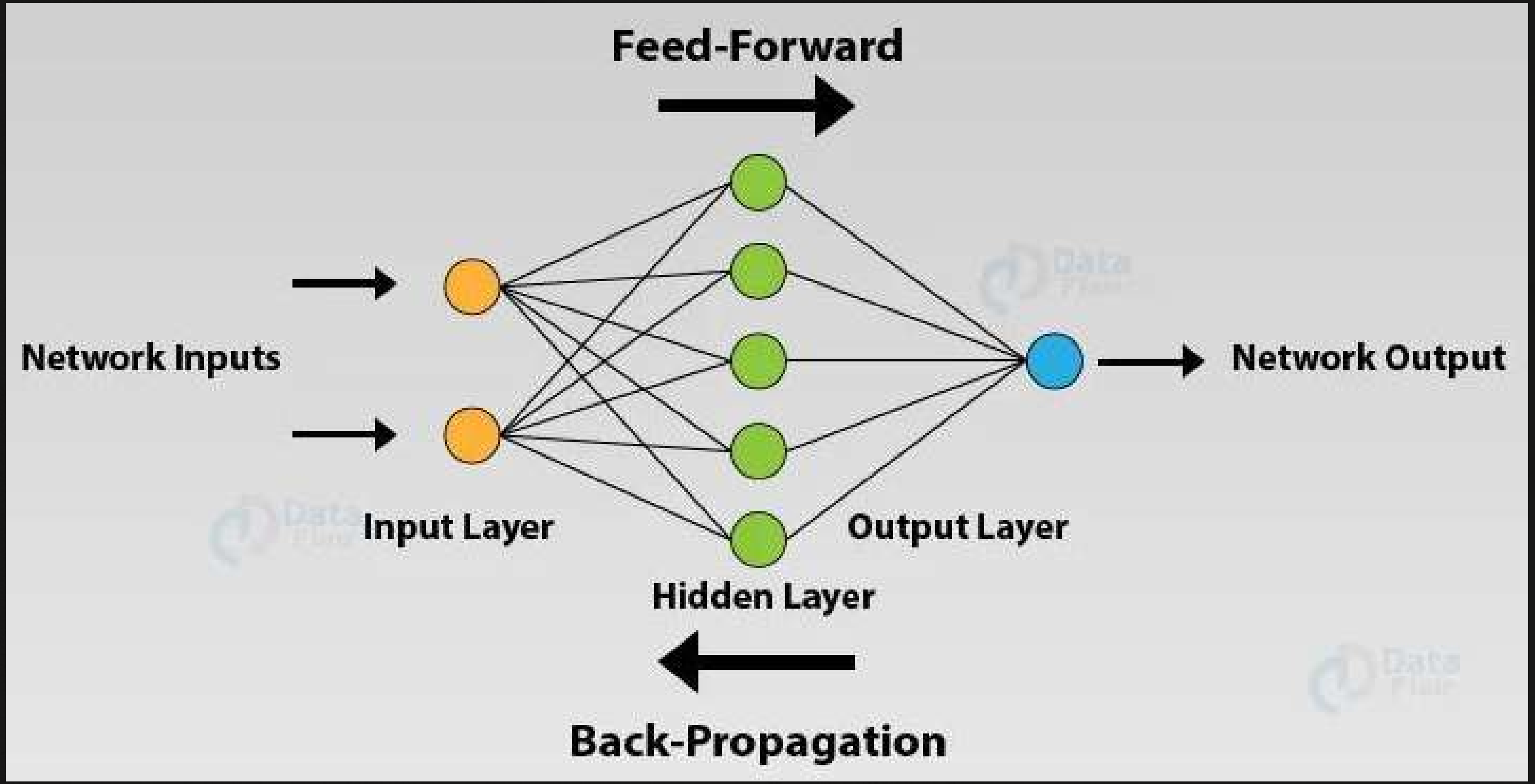
When the Fault is introduced in the circuit, a **Sudden Surge** of **Current** is seen and a **Dip** in the **Load Power**.
When the **Battery Controller** is i. working [**ON**], the **Base** of the **Surge** is **zero [0]**
ii. Switched **OFF**, the **Base** of the **Surge Power** is **different**.

GENERATION OF INPUT DATA FOR ANN MODEL

We measured the values of **Secondary Current [Isec]**, **Load Power [PL]** consumed by the Loads & the **Time** at which this Data is measured.
The Data was measured for a period of 24 hours, with Values being noted at Intervals of 60 minutes. In this way, we got **23 Sets of Values** of all these Parameters. We then **Trained** this **Dataset** [Link for the **Codebase** in References] in our **Artificial Neural Network [ANN] Model**. Now, using this Trained Dataset, **Location** of the Fault could be found out as the Fault before the First house or the Second house Generated the Input Data for ANN Model.
Here, the Model consists of only **Three [3] Houses**, so the Dataset obtained is Small, but a Large Dataset can be obtained by including more Loads i.e., Houses.

WHAT IS AN ARTIFICIAL NEURAL NETWORK [ANN] ?

An **Artificial Neural Network [ANN]** is based on a **Collection** of **Connected Units** or **Nodes** called **Artificial Neurons**. An Artificial Neuron receives an Input, processes it and can Relay Information to other Neurons connected to it. The Output of each Neuron is computed by some **Non-Linear Function** of the Sum of its Inputs. The Non-Linear Function has attributes known as **Weights**. Typically, Neurons are Aggregated into **Layers**. Different Layers perform different **Transformations** on their Inputs. Information travels from the **First layer [Input Layer]**, to the **Last Layer [Output Layer]**, possibly after traversing the layers multiple times. The Information at the end layer helps in **Back-Propagating** the changes to the weights and bias, so that the Prediction and Output of the Network are closer.



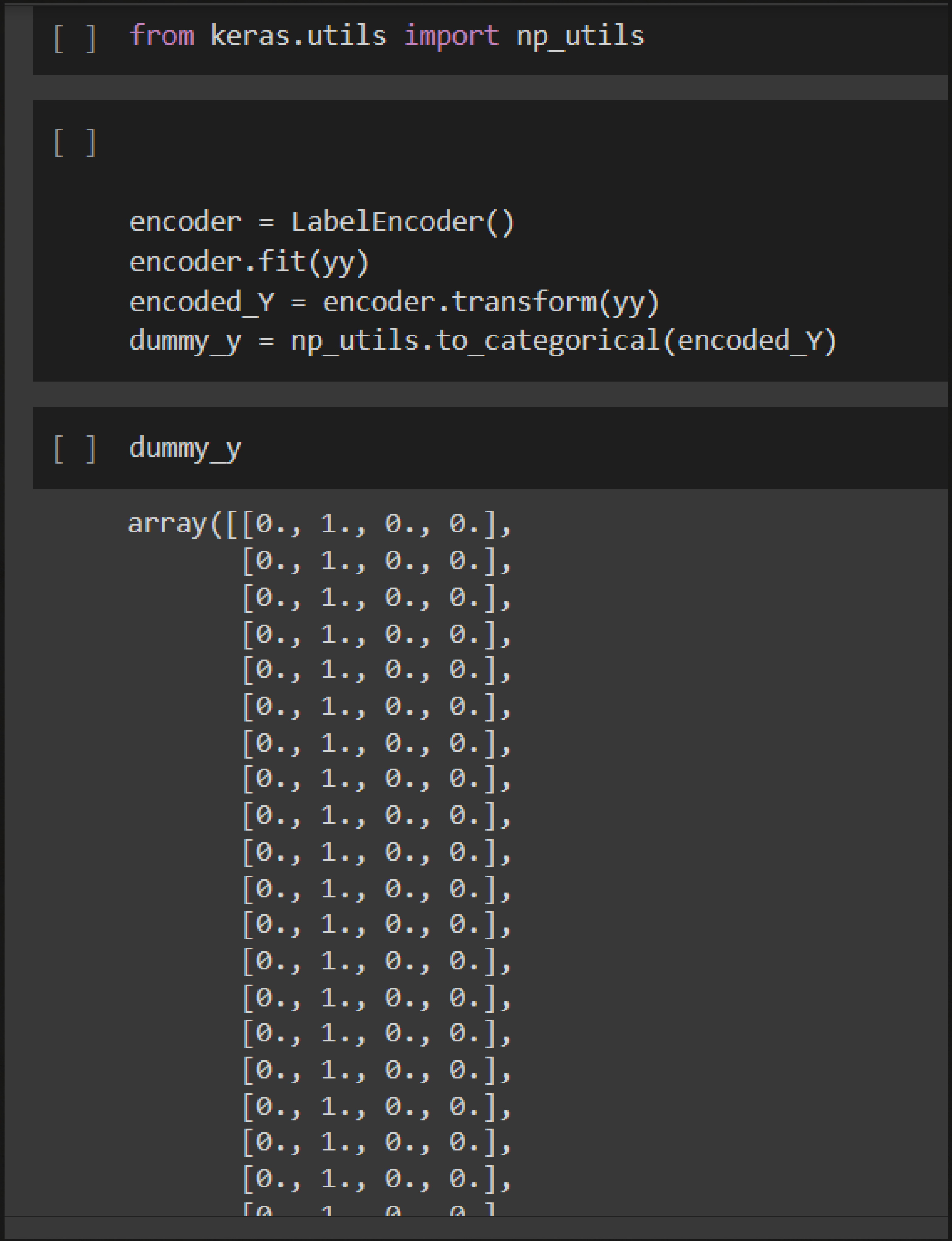
Simplified Schematics of an ANN

SOLUTION THROUGH OUR DEEP LEARNING MODEL

- We gave this **Network** an **Input Layer** of **Dimension 92x27**.
- The Neural Network consists of **6 hidden layers**, **one output layer** with **4 Neurons[outputs][Softmax Layer]**.

- The Features on which the Predictions of the Model depends on are:
  1. The **Load Power**
  2. **Current** in the Circuit
  3. The **Result**
  4. The **Time** at which Breaker was Switched ON.

Since Time as a feature has 24 values, one hot encoding of the features increases the overall input values to 27. Since the input values are spread randomly across values,we scale the data using **Standard Scalar**, process it and encode it for training.



```
[ ]  from keras.utils import np_utils
```

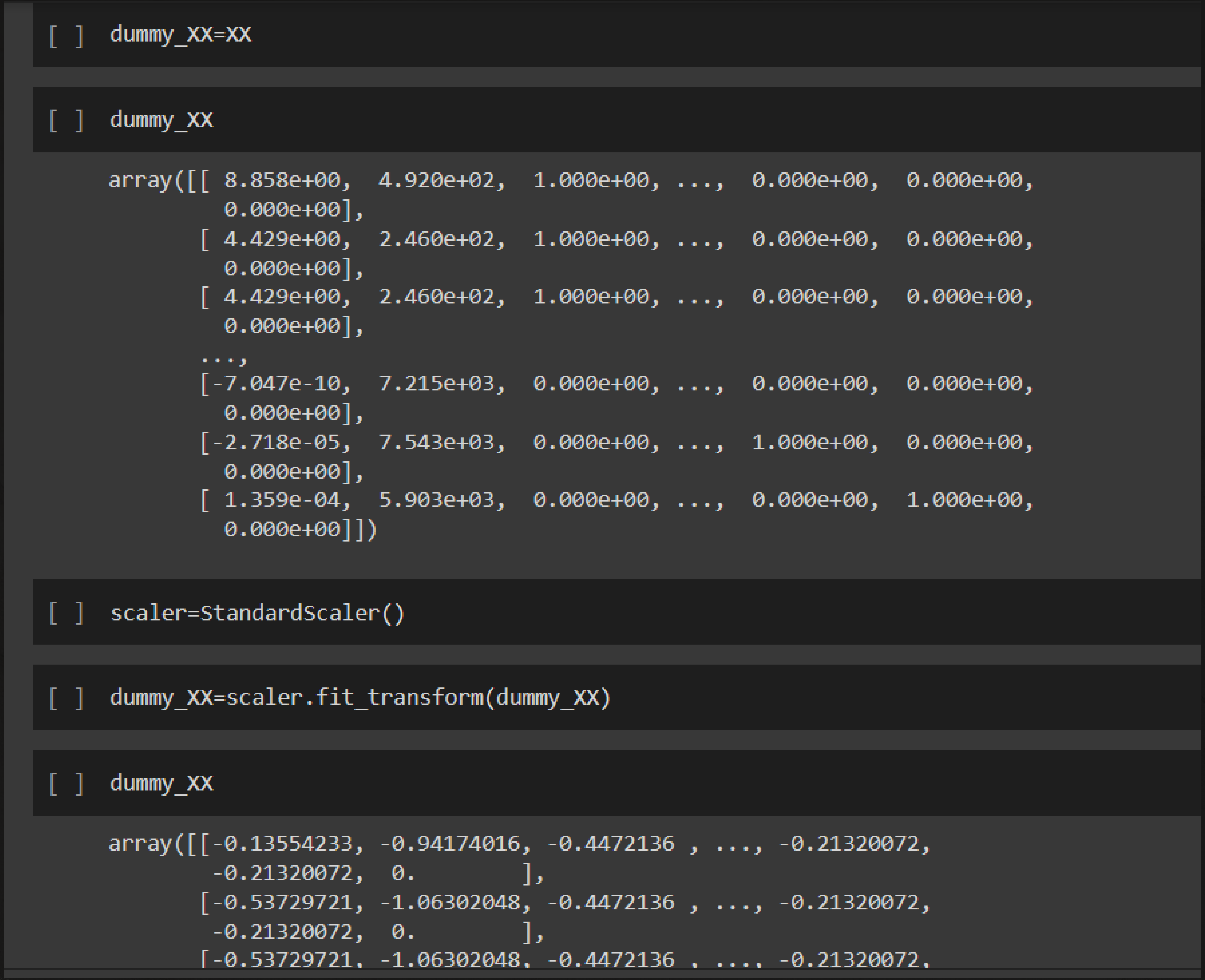
```
[ ]

     encoder = LabelEncoder()
     encoder.fit(yy)
     encoded_Y = encoder.transform(yy)
     dummy_y = np_utils.to_categorical(encoded_Y)
```

```
[ ]  dummy_y
```

```
     array([[0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
            [0., 1., 0., 0.],
```

CODE SNIPPETS

for the Encoder
ie. [LabelEncoder()]



```
[ ]  dummy_XX=XX
```

```
[ ]  dummy_XX
```

```
array([[ 8.858e+00,  4.920e+02,  1.000e+00, ...,  0.000e+00,  0.000e+00,
         0.000e+00],
       [ 4.429e+00,  2.460e+02,  1.000e+00, ...,  0.000e+00,  0.000e+00,
         0.000e+00],
       [ 4.429e+00,  2.460e+02,  1.000e+00, ...,  0.000e+00,  0.000e+00,
         0.000e+00],
       ...,
       [-7.047e-10,  7.215e+03,  0.000e+00, ...,  0.000e+00,  0.000e+00,
         0.000e+00],
       [-2.718e-05,  7.543e+03,  0.000e+00, ...,  1.000e+00,  0.000e+00,
         0.000e+00],
       [ 1.359e-04,  5.903e+03,  0.000e+00, ...,  0.000e+00,  1.000e+00,
         0.000e+00]])
```

```
[ ]  scaler=StandardScaler()
```

```
[ ]  dummy_XX=scaler.fit_transform(dummy_XX)
```

```
[ ]  dummy_XX
```

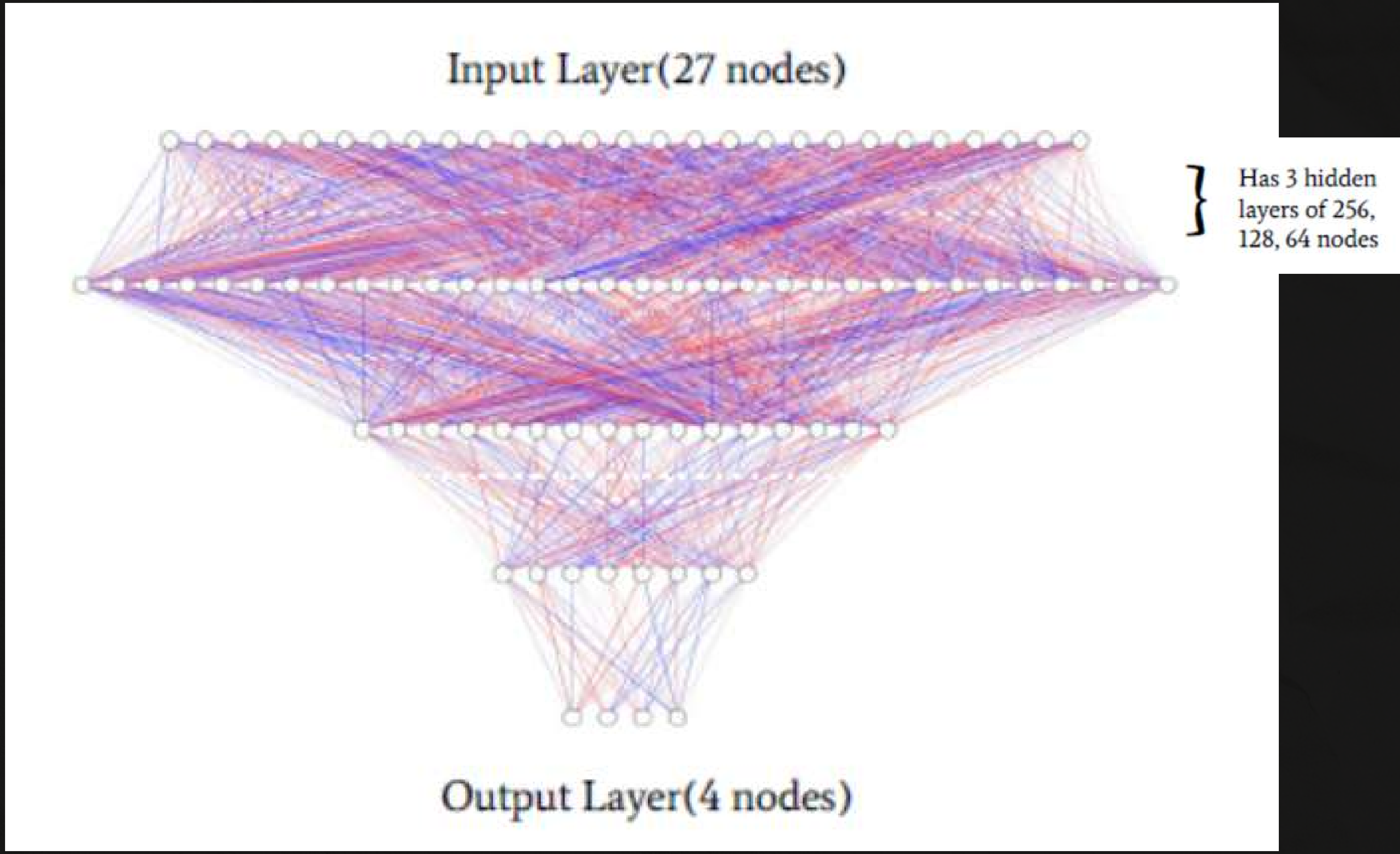
```
array([[-0.13554233, -0.94174016, -0.4472136 , ..., -0.21320072,
        -0.21320072,  0.        ],
       [-0.53729721, -1.06302048, -0.4472136 , ..., -0.21320072,
        -0.21320072,  0.        ],
       [-0.53729721, -1.06302048, -0.4472136 , ..., -0.21320072,
```

CODE SNIPPETS

for the Scaler
ie. [StandardScaler()]

NEURAL NETWORK USED

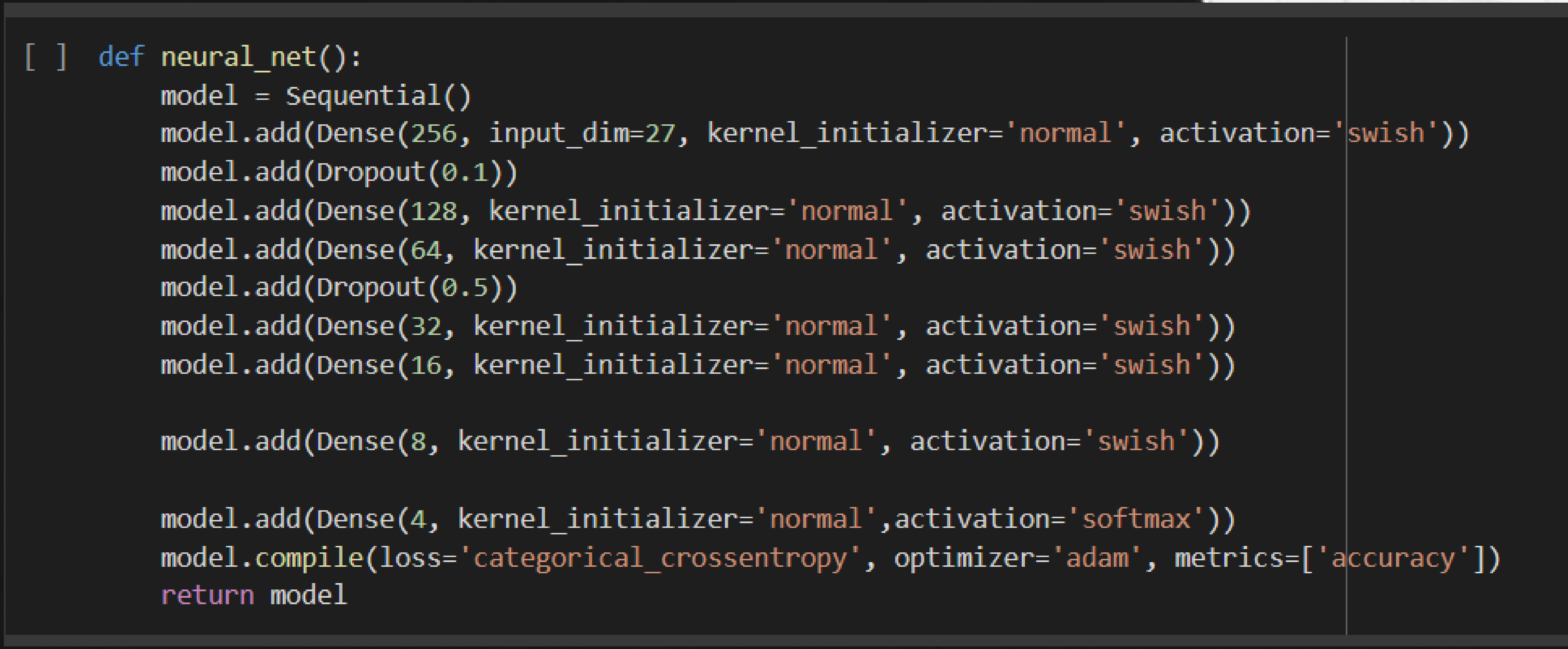
1. The **First Hidden Layer** consists of **256 neurons** and has '**swish**' activation function.
2. The **Second Hidden Layer** consists of **128 neurons** and has '**swish**' activation function.
3. The **Third Hidden Layer** consists of **64 neurons** and has '**swish**' activation function.
4. The **Fourth Hidden Layer** consists of **32 neurons** and has '**swish**' activation function.
5. The **Fifth Hidden Layer** consists of **16 neurons** and has '**swish**' activation function.
6. The **Sixth Hidden Layer** consists of **8 neurons** and has '**swish**' activation function.
7. The **Output Layer** consists of **4 neurons**.



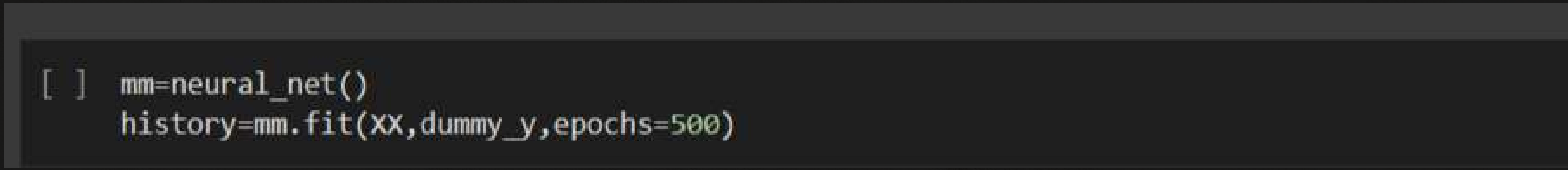
Network Architecture for the **ANN Model** used here

CODE SNIPPETS

for the Artificial Neural Network [ANN] ie. [neural_net()]



```
[ ] def neural_net():
        model = Sequential()
        model.add(Dense(256, input_dim=27, kernel_initializer='normal', activation='swish'))
        model.add(Dropout(0.1))
        model.add(Dense(128, kernel_initializer='normal', activation='swish'))
        model.add(Dense(64, kernel_initializer='normal', activation='swish'))
        model.add(Dropout(0.5))
        model.add(Dense(32, kernel_initializer='normal', activation='swish'))
        model.add(Dense(16, kernel_initializer='normal', activation='swish'))

        model.add(Dense(8, kernel_initializer='normal', activation='swish'))

        model.add(Dense(4, kernel_initializer='normal',activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        return model
```



```
[ ] mm=neural_net()
    history=mm.fit(XX,dummy_y,epochs=500)
```

This Model is compiled using the **'Adam' Optimiser** and the **Number** of **epochs** used are **500** to calculate the **Accuracy** of the Model. We then added **Two [2] Dropout Layers** between the Layers to prevent **Overfitting**.

CATEGORICAL CROSS-ENTROPY

Cross-Entropy is widely used as a **Loss Function** when we're Optimizing Classification Models. Cross-Entropy includes the **Logistic Regression Algorithm [a Linear Classification Algorithm]**, and Artificial Neural Networks [ANN] that can be used for **Classification** tasks. Using this Cross-Entropy Error Function instead of the Sum-of-Squares for a Classification Problem leads to **Faster Training** as well as **Improved Generalization**. Classification problems involve one or more input variables and the prediction of a class label. Classification tasks that have more than two labels as output variables are referred to as **Categorical** or **Multi-Class Classification Problems**.

$$CCE = -\sum_{i}^{n} y_i * \log(\tilde{y}_i)$$

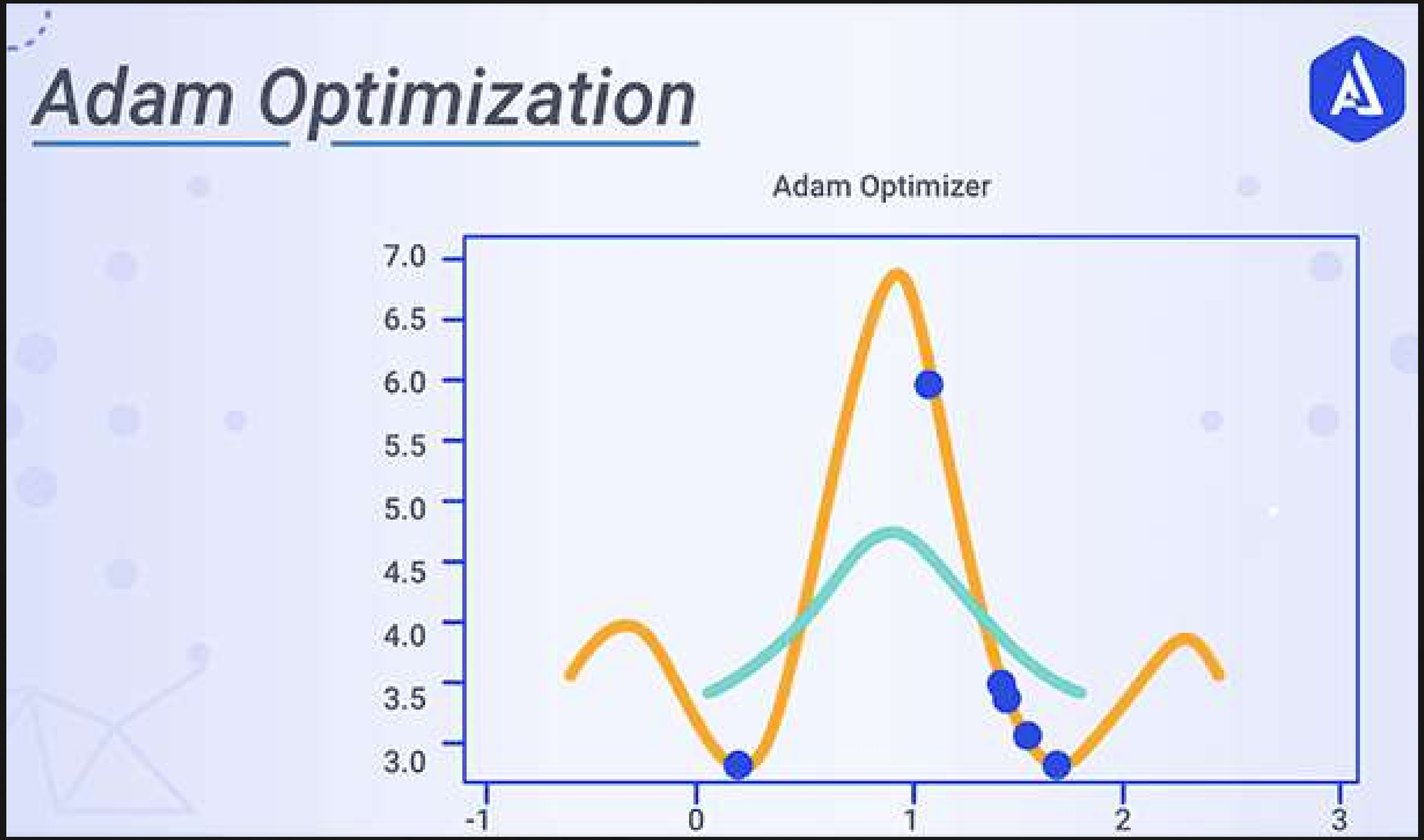
General Formula for the Categorical Cross-Entropy

ADAM OPTIMIZER

Adam Optimizer is used to perform Optimization and is one of the best optimizers at present. Adam Optimizer is good with **Sparse Data**, the **Adaptive Learning Rate** is perfect for this given type of Datasets.

Features of Adam Optimizer are:

- **Updates of Parameters** are **Invariant** to **Re-scaling of Gradient** — It means that if we have some objective function f(x) and we change it to k*f(x) (where k is some constant). There will be No effect on performance.
- The Step-size is approximately bounded by the **Step-size Hyper-Parameter**.
- It doesn't require **Stationary Objective**. That means the f(x) we talked about might change with time and still the algorithm will converge.
- Naturally performs step size annealing. Well remember the classical **Stochastic Gradient Descent [SGD] [tf.keras.optimizers.SGD]**, we used to decrease step size after some epochs, nothing as such is needed here.



General Re-scaling by **Adam Optimizer**

'SWISH' ACTIVATION FUNCTION

There is one glaring issue to the prevalent **Rectified Linear Unit [ReLU]** function. The Rectified Linear Activation Function [ReLU] is a piecewise linear function that will **output** the **input** directly if it is **positive**, **otherwise**, will **output zero [0]**. In Deep Learning, we learn from our **Errors** at the end of our **Forward Path**, then during the Backward Pass, the **Weights** and **Bias** of our Network on each layer get **Updated** to make Better Predictions. What happens during this Backward Pass between Two [2] neurons, one of which returned a **negative number really close to 0** and another one that had a **large negative number**? During this Backward Pass they would be treated as the same ie. 0. There would be no way to know one was closer to 0 than the other one because we removed this information during the forward pass. Once they hit 0 it is rare for the Weight to recover and will remain 0 going forward. This is called the **'Dying ReLU Problem'**. To solve this problem, we use the **'swish' activation function** that **doesn't completely neglect the negative values**. The below graph makes it clear:

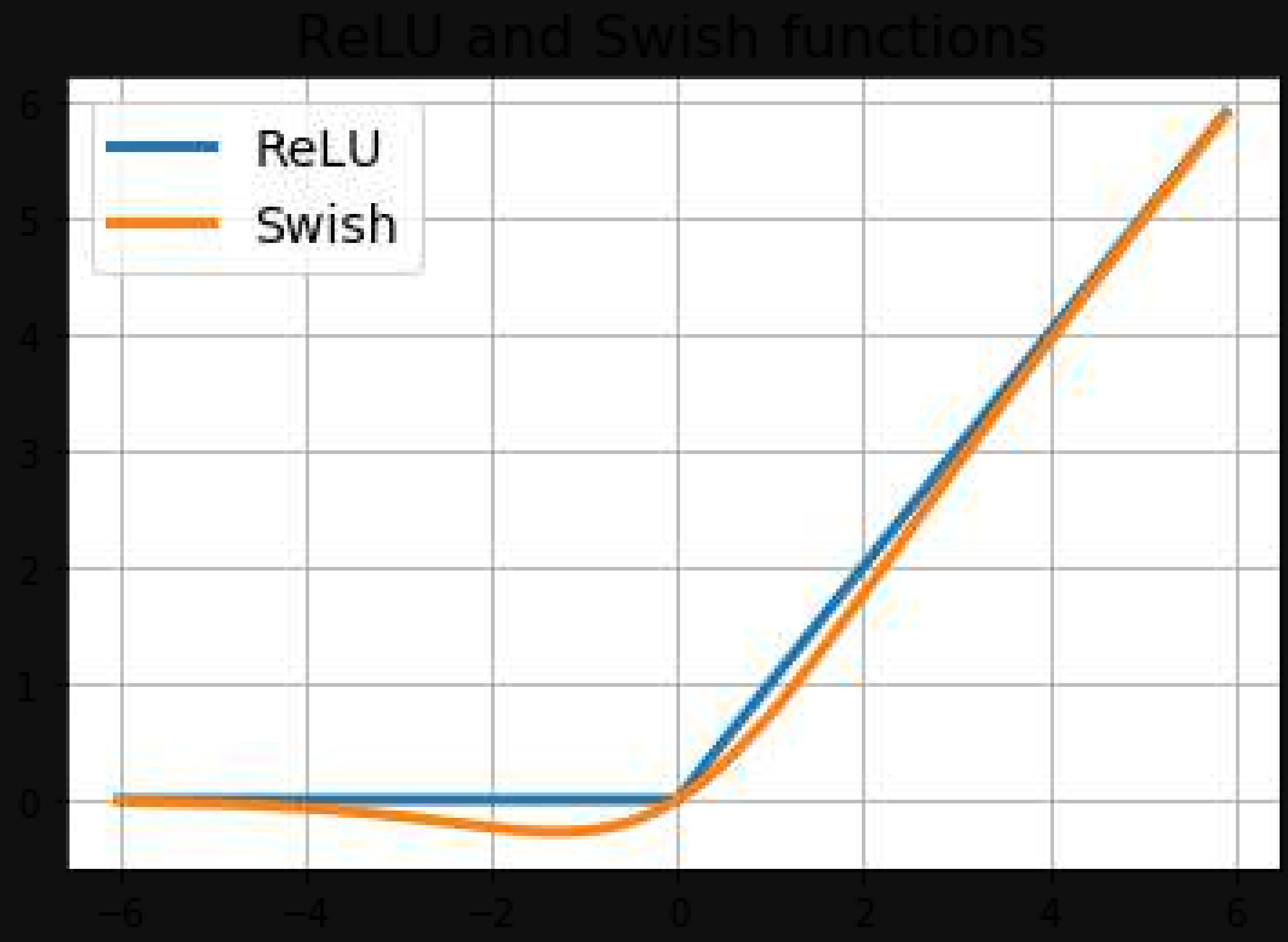
Swish

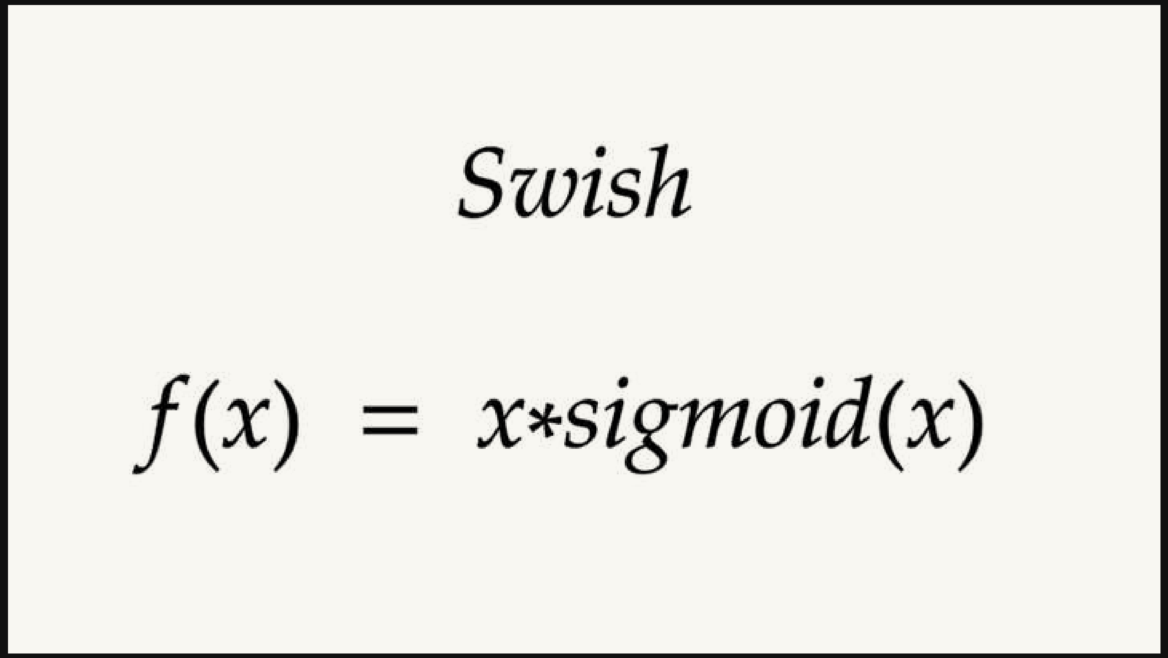
$$f(x) = x * sigmoid(x)$$

Sigmoid / Logistic

$$f(x) = \frac{1}{1 + e^{-x}}$$

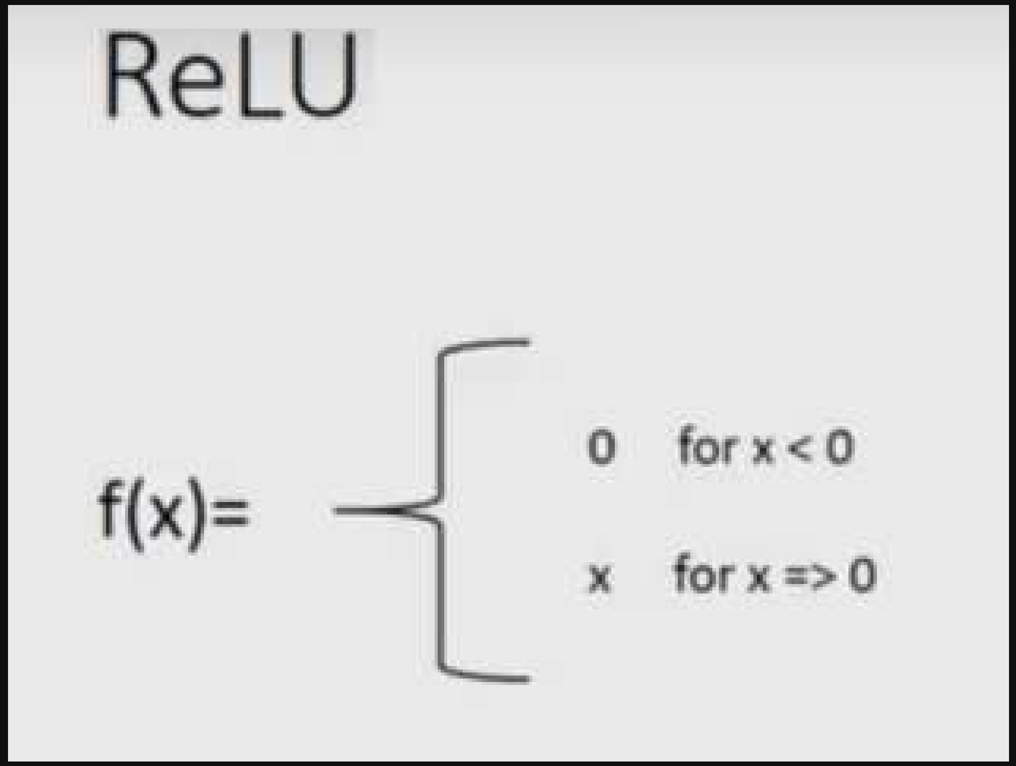
Swish v/s ReLU Activation Functions





Swish

$$f(x) = x * sigmoid(x)$$

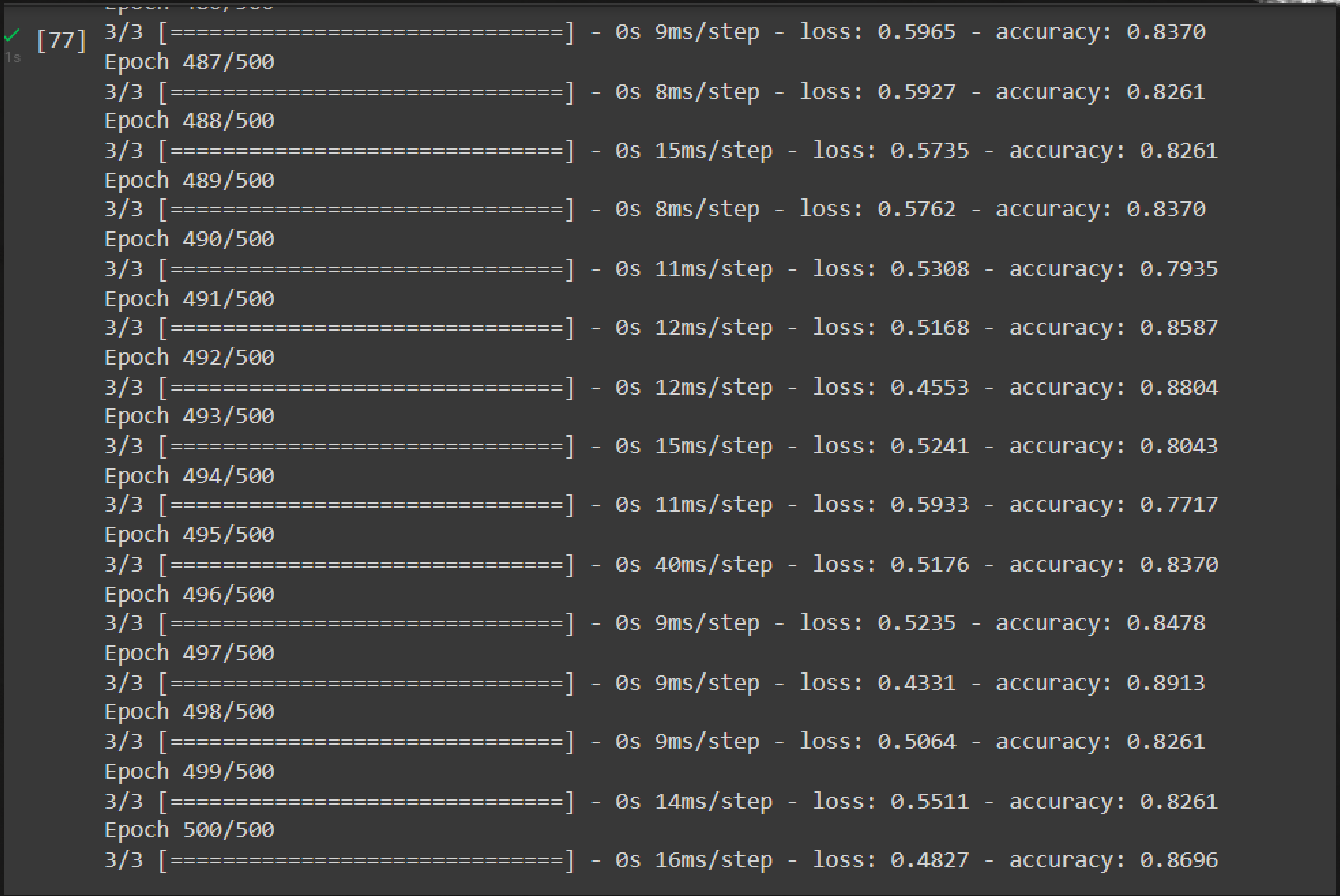


ReLU

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x => 0 \end{cases}$$

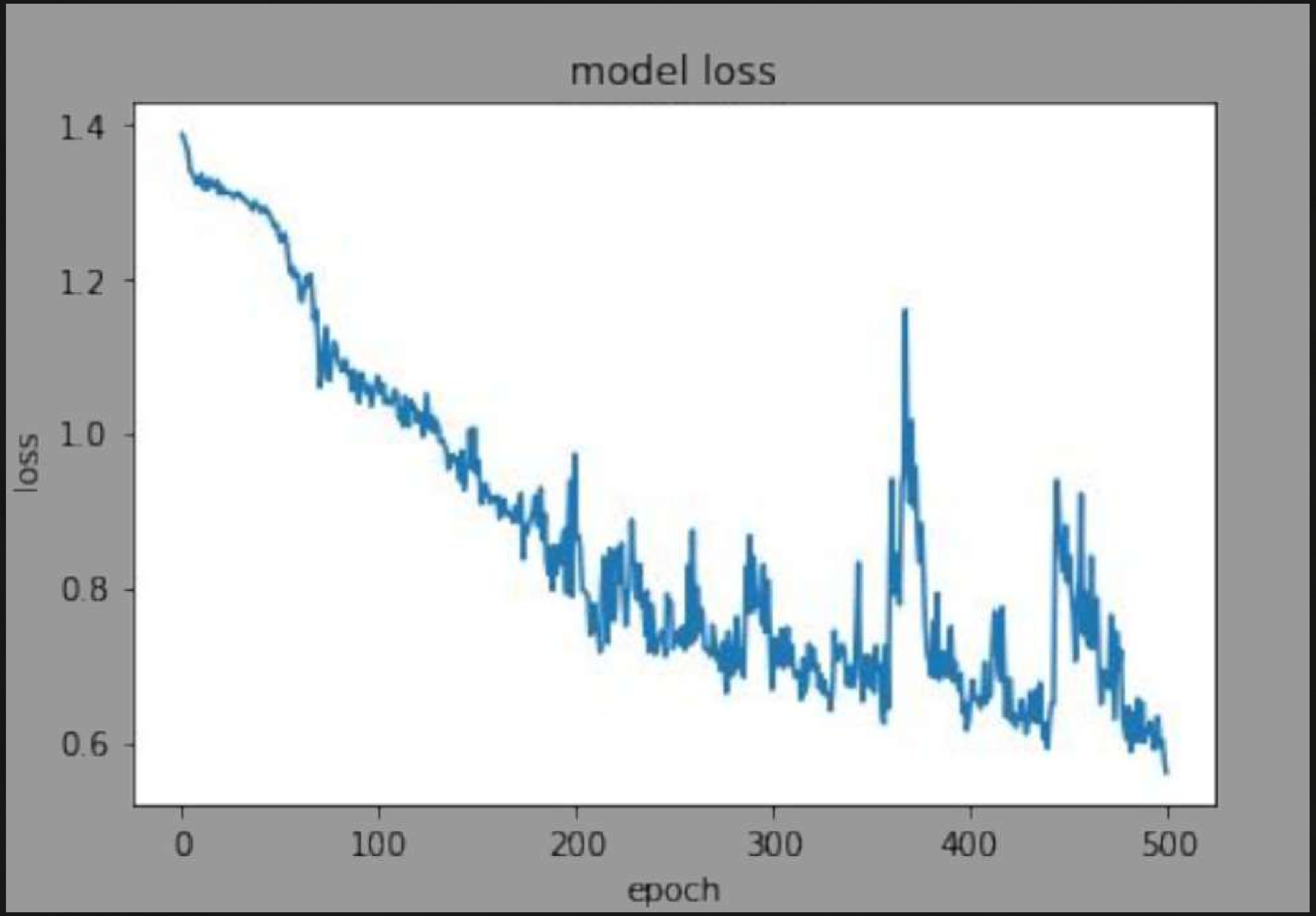
THE TRAINING EPOCHS

86.96% ACCURACY IN PREDICTION AND LOCATION OF FAULT

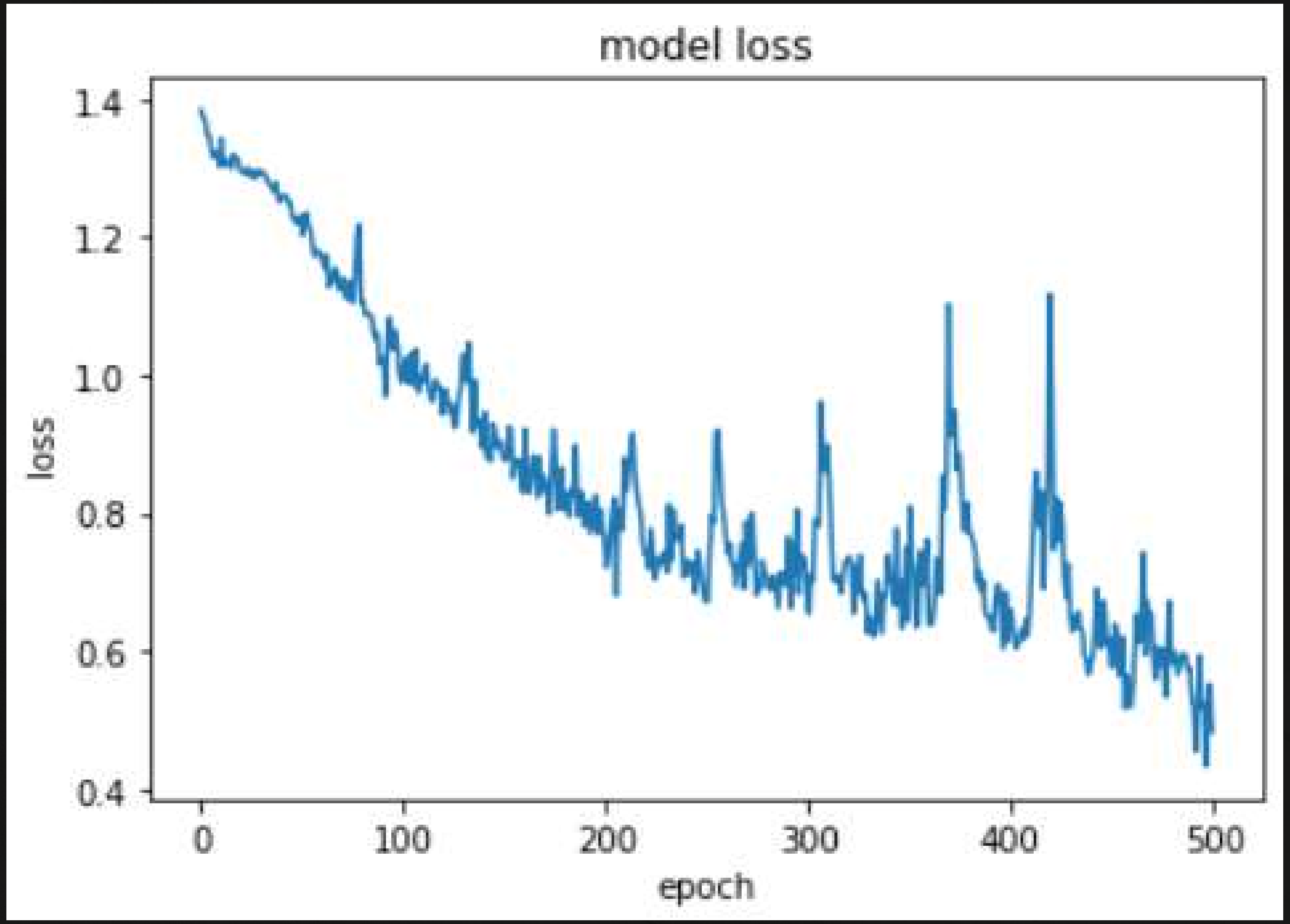


```
[77] 3/3 [==============================] - 0s 9ms/step - loss: 0.5965 - accuracy: 0.8370
     Epoch 487/500
     3/3 [==============================] - 0s 8ms/step - loss: 0.5927 - accuracy: 0.8261
     Epoch 488/500
     3/3 [==============================] - 0s 15ms/step - loss: 0.5735 - accuracy: 0.8261
     Epoch 489/500
     3/3 [==============================] - 0s 8ms/step - loss: 0.5762 - accuracy: 0.8370
     Epoch 490/500
     3/3 [==============================] - 0s 11ms/step - loss: 0.5308 - accuracy: 0.7935
     Epoch 491/500
     3/3 [==============================] - 0s 12ms/step - loss: 0.5168 - accuracy: 0.8587
     Epoch 492/500
     3/3 [==============================] - 0s 12ms/step - loss: 0.4553 - accuracy: 0.8804
     Epoch 493/500
     3/3 [==============================] - 0s 15ms/step - loss: 0.5241 - accuracy: 0.8043
     Epoch 494/500
     3/3 [==============================] - 0s 11ms/step - loss: 0.5933 - accuracy: 0.7717
     Epoch 495/500
     3/3 [==============================] - 0s 40ms/step - loss: 0.5176 - accuracy: 0.8370
     Epoch 496/500
     3/3 [==============================] - 0s 9ms/step - loss: 0.5235 - accuracy: 0.8478
     Epoch 497/500
     3/3 [==============================] - 0s 9ms/step - loss: 0.4331 - accuracy: 0.8913
     Epoch 498/500
     3/3 [==============================] - 0s 9ms/step - loss: 0.5064 - accuracy: 0.8261
     Epoch 499/500
     3/3 [==============================] - 0s 14ms/step - loss: 0.5511 - accuracy: 0.8261
     Epoch 500/500
     3/3 [==============================] - 0s 16ms/step - loss: 0.4827 - accuracy: 0.8696
```

COMPARATIVE ANALYSIS FOR RELU AND SWISH ACTIVATION



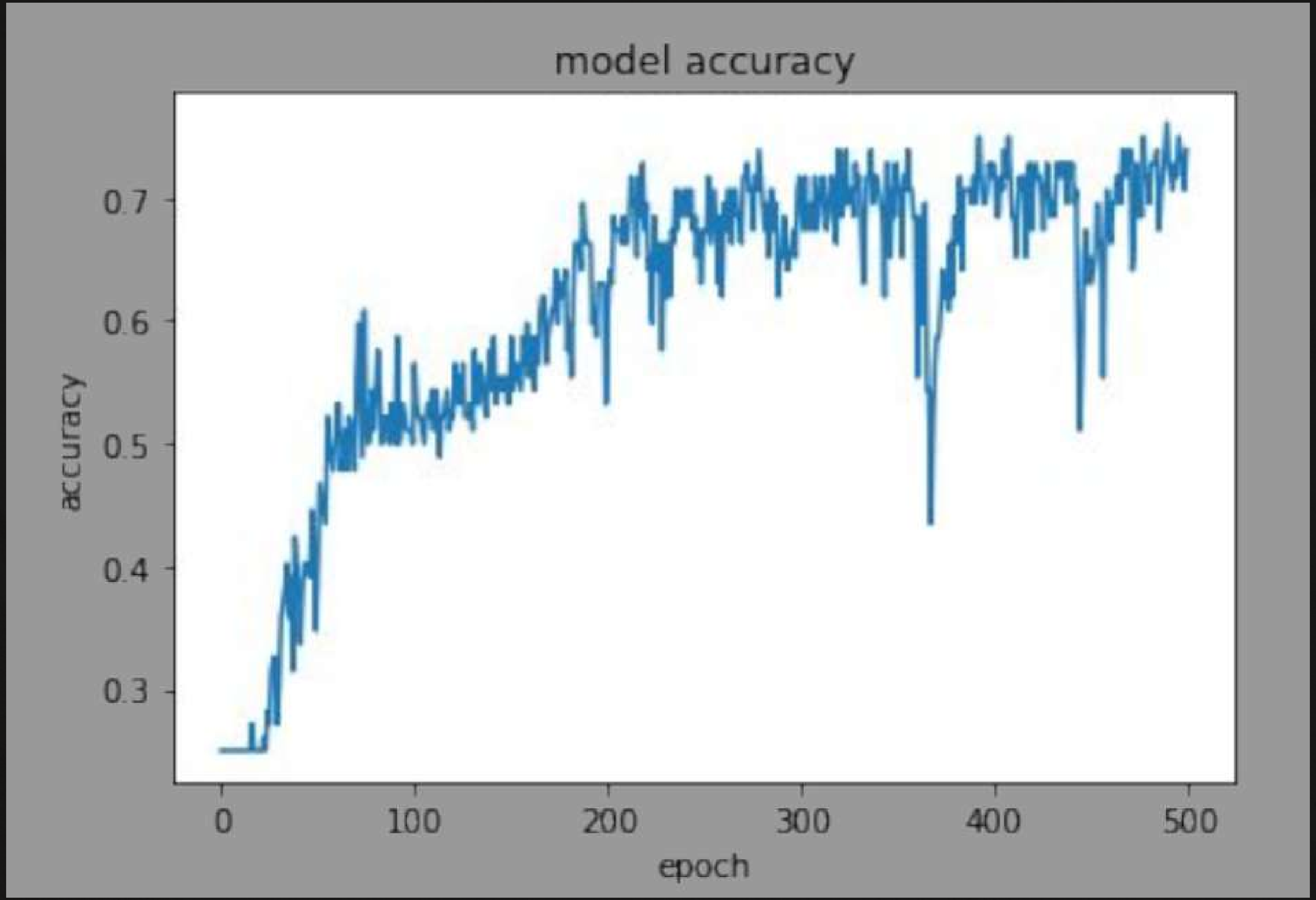
RELU



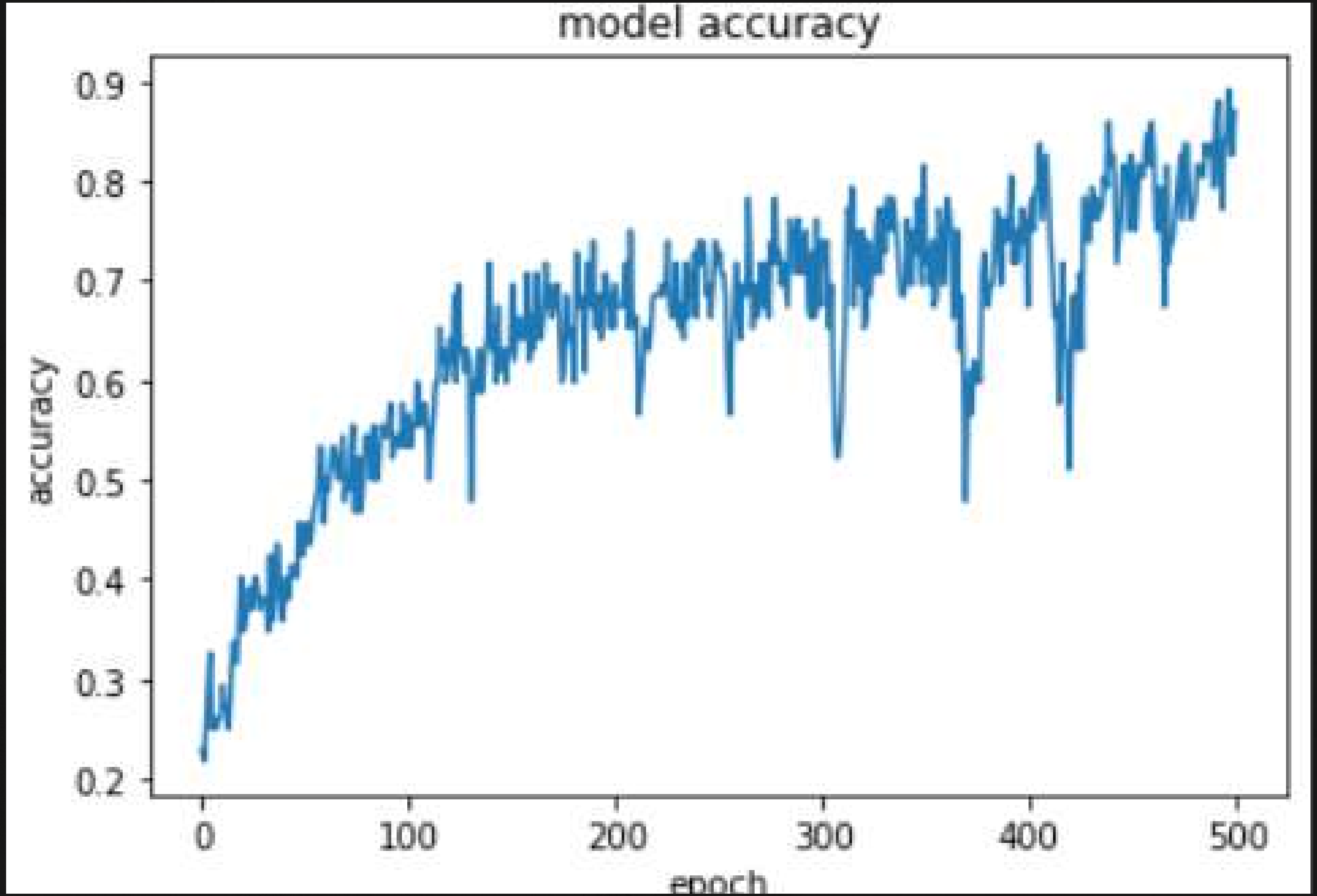
SWISH

i. **Model Loss**

COMPARATIVE ANALYSIS FOR RELU AND SWISH ACTIVATION



RELU



SWISH

ii. Model Accuracy

CONCLUSION

In this Project, we Successfully Studied & Implemented the **Fault Detection & Location of a Power Microgrid** based on **Artificial Neural Network [ANN]**. We Utilised **Small-scale Modeling** of the Power Microgrid including Solar Panels, Battery Energy Storage System, Loads and AC Grid, and simulated the Model in **MATLAB**. A Sudden Change in the value of **Secondary Current [Isec]** and the **Load Power [PL]** is the basis for the Algorithm that Detects the Fault. Here we observe the **Spikes** in the Graphs of Isec, which give us the Faults. These Faults are generated manually by switching on each of the **Circuit Breakers** at various instances in a day. We studied the Faults by utilising a **Battery Controller**. Having Generated the Data at various times of a day and Training this Dataset in ANN fetched a **Remarkable Accuracy** of **86.96%**. This is how we learnt to predict if there is a fault or not, from a Dataset of Load Power [PL], Secondary Current [Isec] and Time of the Day, and if there is, the Location of the Fault can be determined very accurately.

FINAL ACCURACY OF OUR MODEL

No. of epoch's used = 500

Final Accuracy ie. Accuracy @ Epoch 500/500 = **86.96%**

Highest Accuracy Recorded ie. Accuracy @ Epoch 497/500 = **89.13%**

RESOURCES / REFERENCES

[1]. Simplified Model of a Small Scale Micro-Grid on MATLAB
[2]. Dataset Obtained from MATLAB Micro-Grid Model by introducing Sudden Changes in Isec & PL using Circuit Breakers-
(2i). RawDataset.xlsx
(2ii). FeaturesToBeTrained.xlsx
[3]. Artificial Neural Network [ANN] Model Codebase-
Exploratory Project Code SRR.ipynb
[4]. Research Papers relevant to our Project-
(4i). Artificial Neural Network Based Fault Detection and Fault Location in the DC Microgrid
(4ii). Artificial Neural Network Method for Fault Detection on Transmission Line
(4iii). Deep Learning based Techniques to Enhance the Performance of Microgrids: A Review
(4iv). Power flow adjustment for smart microgrid based on edge computing and multi-agent deep reinforcement learning
(4v). Deep reinforcement learning for energy management in a microgrid with flexible demand
[5]. Drive Link with all the Resources Compiled (README.doc, Codebase, Simulink Model, Papers, etc.)
[6]. Github Link [https://github.com/rohansaha13/Event-Detection-in-a-Power-Microgrid-using-Deep-Learning]

PLANS WITH THIS PROJECT MOVING FORWARD

[1]. Further Refining the **Code + Model** for **Increased Efficiency**
[2]. Compile a **Report** for this Project 'Event Detection in a Power Microgrid using Deep Learning'
[3]. Further Refining the Project & Uploading it to **GitHub** as a Fully-Fledged Project

THANK YOU

EE272 : EXPLORATORY PROJECT PRESENTATION

EVENT DETECTION IN A POWER MICROGRID USING DEEP LEARNING

PRESENTED TO YOU BY-

Rishav Singh	20085084
Rohan Saha	20085086
Souvik Karmakar	20085096