SECQAI PROJECT PROGRESS REPORT

Team Members
Swarup Yeole
Vinayak Shrivastava

Mentor
Dr. Avantika Singh

Problem Statement

Implementation of SNN to mimic the behavior of PID controllers for temperature regulation.

Plan Of Action

1. Understanding about SNN and its Architecture
2. Understanding the PID logic and its potential use in our system.
3. Designing an architecture In order to implement the logic for PID controllers.
4. Exploring publicly available dataset that contains values for PID controller parameters and current temperature, adjustment in temperature.
5. Implementing the PID logic with current available algorithms
6. Exploring available libraries to implement SNN and implementing it
7. Implementing SNN for some basic usage using the libraries.
8. Implementation of SNN for the PID logic
9. Making the model more generalized to cover a wide variety of potential applications.

A. SNN and its Architecture

A spiking neural network (SNN) is a type of artificial neural network that models the behavior of biological neurons in the brain. In an SNN, information is represented by the timing and pattern of spikes, or action potentials, that are generated by individual neurons. These spikes are transmitted from one neuron to another across synapses, forming a network of interconnected neurons that process and transmit information.

The basic building block of an SNN is the spiking neuron, which is modeled as a nonlinear function that generates spikes when the membrane potential of the neuron reaches a certain threshold. The membrane potential is determined by the input from

other neurons and the synaptic weights that connect them. When a neuron generates a spike, it sends a signal to the downstream neurons through its axon and across the synapse, where the signal is received and integrated into the membrane potential of the next neuron.

The behavior of an SNN is determined by the connectivity and properties of its neurons and synapses. The synaptic weights are adjusted during training to learn the desired behavior of the network. There are various methods for training SNNs, including supervised learning, unsupervised learning, and reinforcement learning. During inference, an SNN processes input signals in real time by integrating the incoming spikes over time and generating output spikes in response. The output spikes can be used for various tasks, such as classification, regression, control, and communication.

SNNs are particularly well-suited for processing and transmitting information in a time-dependent and asynchronous manner. They are used in various applications, including image and speech recognition, robotics, and neuroscience research. Recent advances in hardware and software have enabled the development of large-scale and efficient SNNs, which hold promise for achieving intelligent and adaptive behavior in artificial systems. The architecture of a spiking neural network (SNN) is typically represented as a graph, where nodes represent neurons and edges represent synapses. The graph is organized into layers, with each layer consisting of a set of neurons that process and transmit information to the next layer.

Each neuron in the SNN is modeled as a nonlinear function that generates spikes when the membrane potential of the neuron reaches a certain threshold. The membrane potential is determined by the input from other neurons and the synaptic weights that connect them. When a neuron generates a spike, it sends a signal to the downstream neurons through its axon and across the synapse, where the signal is received and integrated into the membrane potential of the next neuron. The synaptic weights are adjusted during training to learn the desired behavior of the network. There are various methods for training SNNs, including supervised learning, unsupervised learning, and reinforcement learning. During inference, an SNN processes input signals in real time by integrating the incoming spikes over time and generating output spikes in response. The output spikes can be used for various tasks, such as classification, regression, control, and communication. SNNs can have various architectures, including feedforward, recurrent, and convolutional. The specific architecture will depend on the particular application and requirements

B. PID Controller

A PID controller is a type of feedback control system commonly used in engineering and industrial applications to regulate the behavior of a process or system. The letters "PID" stand for Proportional-Integral-Derivative, which are the three basic control actions used in a PID controller.

In a PID controller, the proportional control action is proportional to the error between the desired setpoint and the measured process variable. The integral control action is proportional to the integral of the error over time, which helps to eliminate steady-state errors. The derivative control action is proportional to the rate of change of the error, which helps to reduce overshoot and improve system response. The PID controller calculates the output control signal by combining the proportional, integral, and derivative control actions based on their respective gains or tuning parameters. These gains are usually adjusted through a trial-and-error process to achieve the desired system performance. PID controllers are widely used in various control applications, such as temperature control, speed control, pressure control, and level control. They are also used in robotics, automation, and manufacturing processes to regulate the behavior of machines and systems.

A spiking neural network (SNN) can be trained to mimic the behavior of a PID controller. In fact, SNNs have been shown to be effective in implementing control systems and can be used to replace traditional control algorithms, such as PID controllers. The basic principles of a PID controller, such as feedback control and adjustment of control parameters based on error, can be mapped onto an SNN architecture. By adjusting the synaptic weights and other parameters of the SNN, it is possible to achieve the desired control behavior. In some cases, an SNN-based control system can outperform a traditional PID controller, especially in complex and nonlinear systems. However, the design and training of an SNN-based control system can be more challenging than a PID controller, and requires specialized knowledge and expertise in neural networks and control theory.

$$OP = OP_{bias} + \underbrace{K_c e(t)}_{\text{Proportional}} + \underbrace{\frac{K_c}{\tau_I}\int e(t)dt}_{\text{Integral}} + \underbrace{K_c \tau_D \frac{de(t)}{dt}}_{\text{Derivative}}$$

Fig 1. Fundamental equation of PID controller

C. Designing Architecture To Implement PID Logic

i) Generating PID controller data

The initial data was generated using several arrays to store data and running the controller for 90 minutes to generate initial data**.** We have recently increased the number of samples in our dataset by applying data augmentation techniques to increase the dataset . This technique allowed us to generate additional data points with varying PID logic parameters for a given temperature range. The dataset includes columns for temperature values to be achieved and the original temperature, as well as the different parameters used in the PID logic. These additional data points will be beneficial in further training and testing our machine learning models, which will ultimately improve the accuracy of our temperature control system.

```
# PID Parameters
Kc   = 6.0
tauI = 75.0 # sec
tauD = 0.0  # sec
# PID coefficients in terms of tuning parameters
KP = Kc
KI = Kc / tauI
KD = Kc * tauD

# OPbias for controller (initial heater)
op0 = 0
# upper and lower bounds on heater level
ophi = 100
oplo = 0
```

Fig 2. Tuning parameters used in the data generation

List of Parameters in the data :

a) tm : Time stamp
b) Tsp1 : Temperature set point
c) T1 : Measured Temperature
d) Q1 : Heater Values
e) P : P values of PID controller
g) D : D values for PID controller
h) iae : Integral absolute error
i) ierr : Integral error

```
Temperature 1: 23.73 °C
Temperature 2: 23.15 °C
   1.0  23.67  23.64   0.18   0.18   0.00  -0.00   0.03
   2.0  23.67  23.67   0.00   0.00   0.00  -0.00   0.03
   3.0  23.67  23.80   0.00  -0.78   0.00  -0.00   0.16
   4.0  23.67  23.80   0.00  -0.78   0.00  -0.00   0.29
   5.0  23.67  23.80   0.00  -0.78   0.00  -0.00   0.42
   6.0  23.67  23.80   0.00  -0.78   0.00  -0.00   0.55
   7.0  23.67  23.77   0.00  -0.60   0.00   0.00   0.65
   8.0  23.67  23.80   0.00  -0.78   0.00  -0.00   0.78
   9.0  23.67  23.67   0.00   0.00   0.00   0.00   0.78
  10.0  23.67  23.77   0.00  -0.60   0.00  -0.00   0.88
  11.0  23.67  23.77   0.00  -0.60   0.00  -0.00   0.98
  12.0  23.67  23.77   0.00  -0.60   0.00  -0.00   1.08
  13.0  23.67  23.77   0.00  -0.60   0.00  -0.00   1.18
  14.0  23.67  23.80   0.00  -0.78   0.00  -0.00   1.31
```

Fig 3. Sample Data Image

ii) Training the PID with LSTM

The data generated has been used to train an LSTM network. This model uses the previous 15 seconds of sensor data from the PID controller run and uses it to emulate the heater output that the PID controller would use. The model is saved for use in other notebooks. The notable feature of this model are as follows:

1) The model was trained using a deep learning approach, specifically using the Keras library with a TensorFlow backend. The architecture used was a recurrent neural network (RNN) with long short-term memory (LSTM) cells. This type of architecture is well-suited for sequential data, such as time-series data, which is what the temperature control system in this project involves.

2) The hyperparameters used to train the model were chosen based on heuristics and prior experience, but they are not necessarily optimal for this particular problem. For example, the number of layers, dropout rate, batch size, number of units in each layer, and window size (i.e., how many time steps to look back when making predictions) could all be further optimized to potentially improve model performance.

3) Despite this, the model performs well in emulating the behavior of a PID controller, which is a common control algorithm used in temperature control systems. This is demonstrated by the fact that the model is able to accurately

predict future temperature values based on current and past temperature readings, as well as the setpoint and error values.

4) Feature selection was performed using the SelectKBest method from the scikit-learn library. This method evaluates the statistical significance of each feature and selects the k best ones based on a scoring function. In this case, the results indicate that only the setpoint and error features are necessary for good model development, while the actual temperature value does not directly contribute to a better model. However, additional features could be derived and tested to potentially improve model performance further.

5) Due to the stochastic nature of model training, the model can change slightly each time it is trained. This means that it is important to train the model multiple times and report the average performance to get a more accurate representation of its capabilities.

6) To prevent overfitting, the model is trained with an early stopping method that stops training after the validation set loss value does not improve for 25 epochs. This helps to ensure that the model is not simply memorizing the training data and is instead generalizing well to new data.

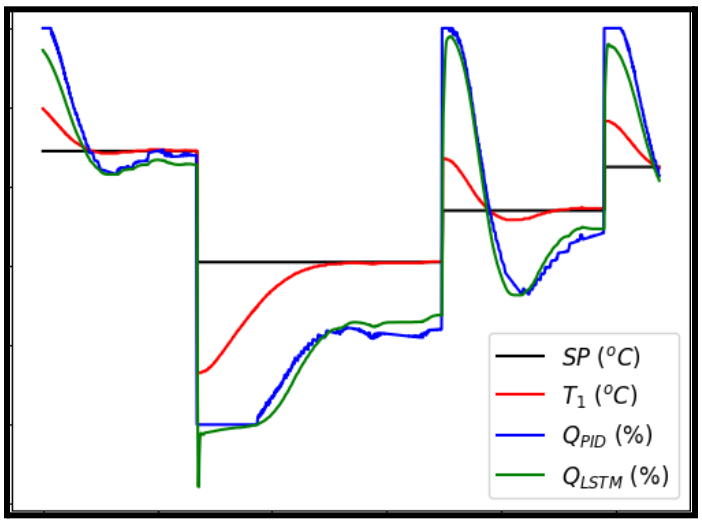

Fig 4. Plot of Setpoint, Current Temperature, PID Response and trained LSTM response.

iii) Emulate PID controller with LSTM

With the trained model saved we run the experiment with the set point data. The LSTM model will emulate the behavior it learned from the controller and predict the heater output to achieve the given setpoint. The experiment was run in three different modes, the result for each is as follows:

i) In the first mode, the controller runs with both the PID and LSTM, with the output of both controllers being compared to determine which one is more effective at controlling the temperature. This mode is likely used to evaluate the performance of the LSTM controller and compare it to the traditional PID controller.

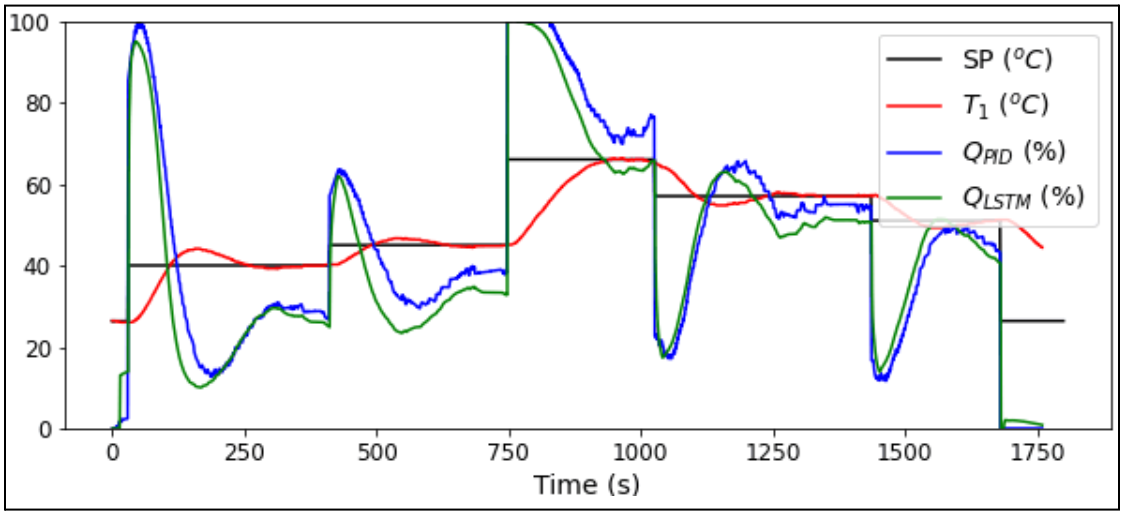

Fig 5. Plot of PID controller and LSTM behaviour

ii) In the second mode, only the LSTM controller is used to control the heater. This mode is likely used to demonstrate the capabilities of the LSTM controller and evaluate its performance in comparison to the PID controller.

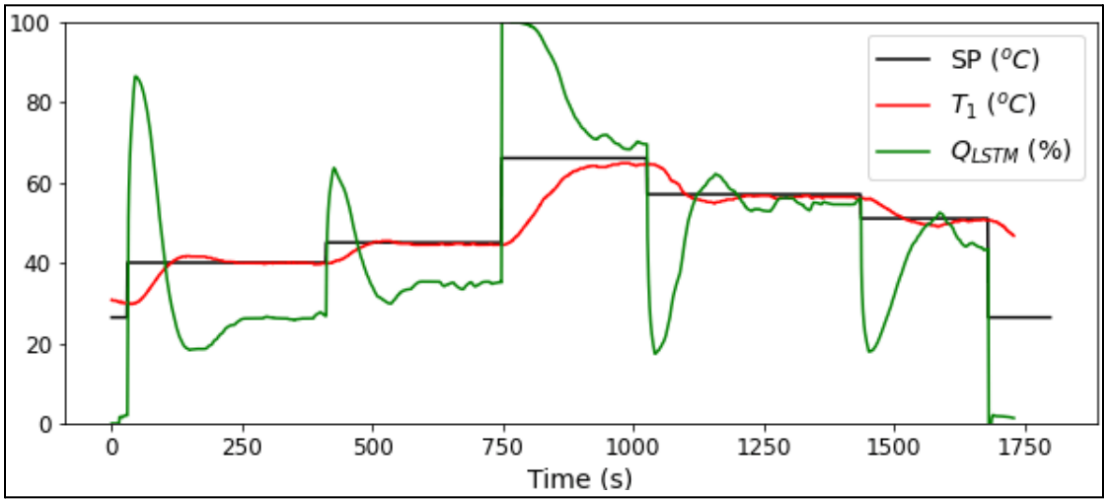

Fig 6. Plot of LSTM behaviour in emulating PID controller behaviour

D. Researching Tools and Libraries To implement PID using SNN

i) SNN Torch :

The package is built on top of PyTorch, a popular machine learning library that provides efficient tensor computation on GPUs. By leveraging PyTorch's capabilities, snnTorch allows users to apply gradient-based learning algorithms to networks of spiking neurons, enabling them to train these networks to perform various tasks.

One of the key advantages of snnTorch is its intuitive integration with PyTorch. Users can treat spiking neurons as just another activation in a sequence of layers, making it easy to incorporate them into existing neural network architectures. This allows users to take advantage of a wide range of pre-existing PyTorch layers, such as fully-connected layers, convolutional layers, and residual connections, without having to modify their code significantly.

In addition, snnTorch provides a number of specialized layers and functions for building spiking neural networks, such as leaky integrate-and-fire (LIF) neurons and spike-time-dependent plasticity (STDP) learning rules. These features make it easier for users to build and train spiking neural networks from scratch, without having to worry about the low-level details of the underlying spiking neuron model.

ii) Tonic :

conversion to different event representations. This can be useful for increasing the size of datasets, which is a common challenge in machine learning. By converting existing datasets to event-based representations, it is possible to create larger and more diverse datasets that can be used to train more robust and accurate models.

Tonic caters to both the event-based world that works directly with events or time surfaces as well as to more conventional frameworks which might convert events into dense representations in one way or another. This means that users can choose the representation that works best for their specific task or application. For example, some applications may benefit from working directly with event-based representations, while others may require conversion to dense representations.

E. Targets Achieved

1. Understanding about SNN and its Architecture
2. Understanding the PID logic and its potential use in our system.
3. Understanding the PID logic and its potential use in our system.
4. Designing an architecture In order to implement the logic for PID controllers.
5. Exploring publicly available dataset that contains values for PID controller parameters and current temperature, adjustment in temperature.
6. Implementing the PID logic with current available algorithms

F. Future Scope

1.Implementing SNN for some basic usage using the libraries.
2. Implementation of SNN for the PID logic
3 . Making the model more generalized to cover a wide variety of potential applications.

G. References

Github Implementation Link : https://github.com/swarupyeole11/SequaiSNN

Online Resources

[1]https://towardsdatascience.com/anomaly-detection-in-process-control-data-with-machine-learning-35056a867f5b

[2]https://www.elprocus.com/the-working-of-a-pid-controller/

[3]https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_1.html#delta-modulation

Research Papers

[5] Jason K. Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. “Training Spiking Neural Networks Using Lessons From Deep Learning”. arXiv preprint arXiv:2109.12894, September 2021. https://arxiv.org/abs/2109.12894