Neuroidal Network Software Development Class Project for Detection of Arrhythmia in Patients Experiencing Heart Failure

Version 1.0

Prepared by CSC 9010-010 Summer 2021 Class

Villanova University

July 29, 2021

Table of Contents

Revision History

Name	Date	Reason for Changes	Version
CSC 9010-010 Class	July 29, 2021	Initial release	1.0

Introduction and General Description

The model of a neuroid [1], was proposed as an improvement over the artificial neural network, structurally and functionally mimicking the biological neuron. The three operations a neuroid performs in processing the incoming information are: comparison of the inputs, pulse modulation, and pulse demodulation. The incoming set of inputs is first fed into an initial execution block which does some cumulative computation on them and checks if the sum signal generated is greater than the permitted minimum threshold value of that neuroid for it to be valid. If it is lesser, the signal is discarded, and no output is produced. If not, another execution block performs the pulse modulation and demodulation on the generated signal and its output is transmitted as the output of the neuroid.

The advantage of using this approach of a neuroid is that the weights of the incoming signals, its values, and other default parameters can be altered for convenience, which is otherwise impossible in regular artificial neural networks. In this project, we attempt to develop a neuroidal network system that is composed of multiple neuroids in combination, which can take up a set of inputs, perform necessary computations and provide the observed output.

Once the neuroidal network system is complete, the software can be translated into various Artificial Intelligence applications such as arrhythmia detection. An arrhythmia is a heart condition where an individual experiences an irregular or abnormal heartbeat, and detection of a patient suffering from an arrhythmia is normally too late before heart failure occurs. Applying the neuroidal network that is to be developed to arrhythmia detection would benefit the user in determining when an abnormality occurs that could help them and their healthcare physician make better decisions with their healthcare plans.

The project was carried out in four continuous phases.

1. The User Flow Design Phase
2. The Front-End Design Phase
3. The Back-End Design Phase
4. The Business Requirements Phase

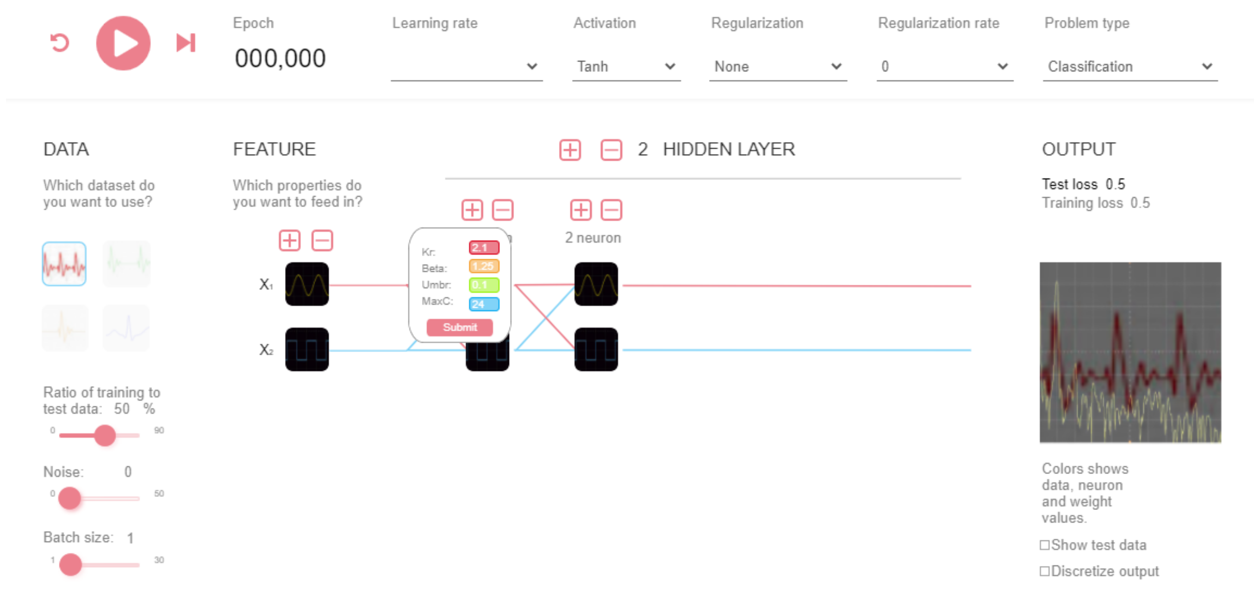
User Flow

The User Flow Design Phase

The aim of the user flow design phase was to develop a prototype outlining the basic structure of the neuroid, its digital representation, its various parameters of usage, and mapping the layout of the environment in which it would be implemented visually. The software that was used for this part of the project was Axure RP. Axure RP is a software for creating prototypes and specifications for websites and applications using drag and drop placement features, resizing, and formatting of widgets. This software was specifically chosen for the flexibility it offers in being able to modify and alter our design as needed and the high level of interactivity that almost is close to a HTML coded web page. It is cross-platform allowing designs to be imported and exported between different operating systems and is also compatible for collaboration with other tools and services like Slack and Microsoft Teams.

Prototype

- Prototype was created by Axure RP



Epoch 000,000

Learning rate

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50 %

Noise: 0

Batch size: 1

FEATURE

Which properties do you want to feed in?

2 HIDDEN LAYER

2 neuron

Kr: 2.1

Beta: 1.25

Umbr: 0.1

MaxC: 24

Submit

OUTPUT

Test loss 0.5

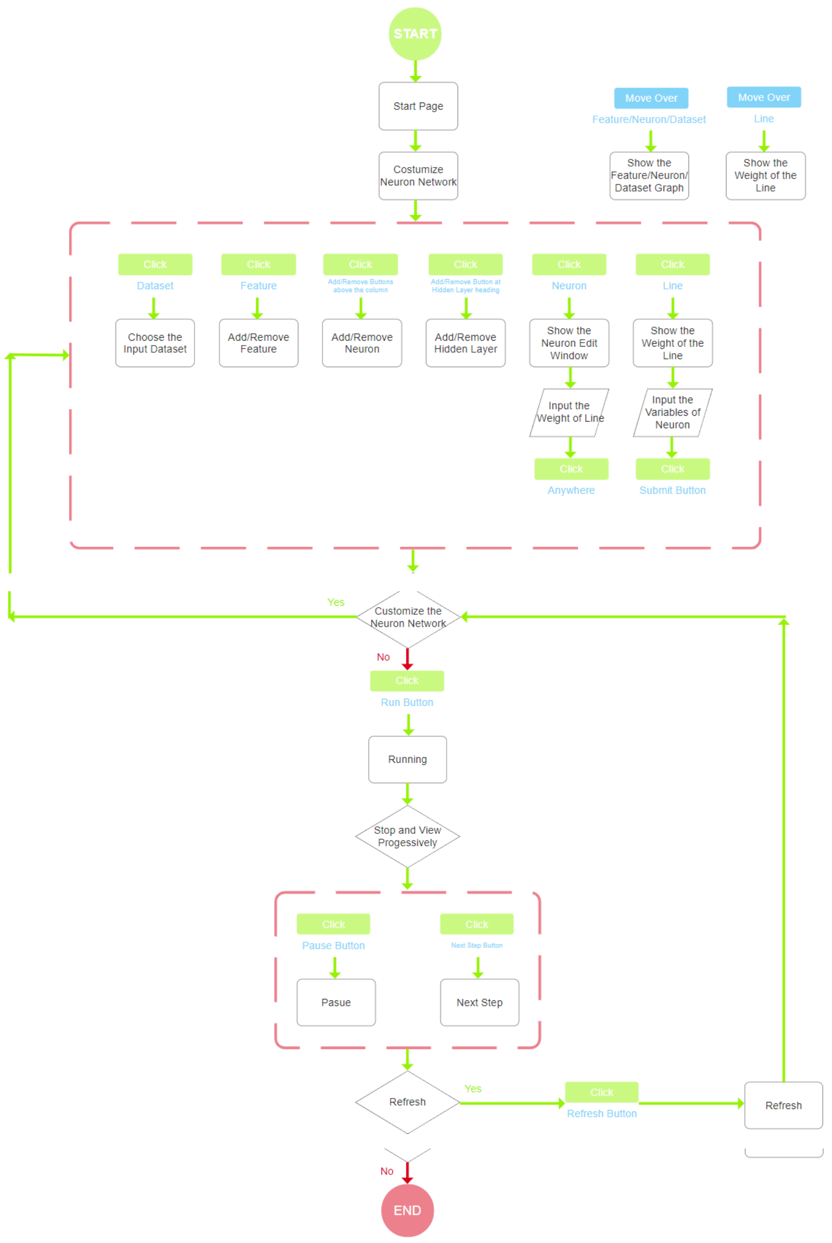
Training loss 0.5

Colors shows data, neuron and weight values.

☐Show test data

☐Discretize output

User Flow Diagram



Front-End

Goals

- Assess business requirements as set by the business requirements team to plan out what functionalities and interactions were needed for our MVP
- Coordinate the UI design for the neuroidal simulator with the user flow team
- Coordinate the implementation of the simulator with the back-end team to achieve maximum functionality within the allotted time
- Choose a tech stack that could easily support the needs of the user flow and back-end teams while giving us the tools to implement these features
- Create menuing tools for users to quickly and easily change the inputs of the simulator
- Map out the neural network on the page in a clean and organized fashion
- Allow the user to have dynamic control over the simulation, for example, the number of layers and the number of neurons within each layer.
- Display the output of the simulation alongside the network that updates as the simulation runs

Tools

The following tools were used to develop the front-end.

- HTML/CSS
- Bootstrap
- Javascript
- GitHub
- Favicon

HTML/CSS and Bootstrap were used to create the structure and appearance of the index.html file of the project where the neuroidal network simulation is visualized. Favicon was used to create the rounded square icons to represent the neuroids. Javascript was leveraged to add various functionalities to the page, where some Bootstrap interactions are dependent. Additionally, custom Javascript was generated to support the visualization of the hidden layer neuroids and the weighted connections. Finally, chart.js was an imported library to model the output.

Code and Detail

Simulation

The title and explanation of the website along with many tools for the user to control the simulation are oriented in the header. On the left is a control panel that allows the user to replay, play, and fast forward steps in the simulation. To the right of these controls are a series of dropdown menus that offer the user a selection of choices to change the learning rate, activation, regularization, regularization rate, and problem type. Through Bootstrap, additional design considerations were added such as up and down arrows to signify the field as a dropdown menu and a blue border around the currently active field allowing the website to feel more dynamic and pleasing to use.

There are input sections for the user to submit the number of neuroids for the system, the number of hidden layers available, and the weights of the connections in these layers as they should be used in the simulation. The Features section has sliders so that the user can set the ratio of training to test data, the noise, and the batch size of the simulation. These customizations alter the type of learning that the neuroidal network can perform. A header that displays the number of hidden layers in use is found above an input menu to add additional neuroids to those layers as needed.

The depiction of the simulation includes neuroids that have customization for the values of kr, beta, umbr, and maxcount (Back-End -- Components and Parameters -- Configurable Parameters). The hidden layer neuroids appear and disappear as the user interacts with the input controls; as the variables change, there is a reaction within the environment. The connections between the main neuroids and the hidden layer neuroids are depicted with colored lines. These lines also have customizable weights via the weight input menu in the visualization. The colors of the line signify which neuroids they connect to and serve as a reminder on the output of the interaction between these neuroids and layers.

The output plots generated the response of the neuroidal network over the time period. The line graph reflects the inherent benefit of neuroids in their ability to react to changes in frequency over time rather than a simple input/output of a traditional neuron.

Front-End to Back-End Transfer

Accessing Function from Front-End

The configurable parameters are passed in through a form: (*neuroid parameters -- form value name*)

- umbr -- umbrValue
- beta -- betaValue
- kr -- KrValue
- maxcount -- maxcountValue

The form's action calls the /input endpoint and returns an array from the Flask that is called "results".

Connecting the front-end and back-end code was established via a shared GitHub repository. Flask was utilized as a Web Server Gateway Interface (WSGI), which allows information to transfer from both the front-end and the back-end. A POST method call on the forms provided the connections needed for Flask to communicate data. Submitting the form provides the action for the backend code to go to the /input endpoint. When this endpoint is loaded, the code in Flask runs and pulls the data from what the user entered into the form. The variables in the neuroid algorithm are set according to the form data and then the algorithm's outputs are returned as an array. To process the output, the array of values generated by the neuroid back-end code is printed to the front-end via a conditional statement that validates the entered data. Then, a Javascript document.getElementById() call reads what was printed and transfers it to the chart.js function, where the output renders as a line graph on the front-end where the output is plotted over the amount of time the simulation was run.

Back-End

Tools

- Python
    - Python is a programming language that allows for the creation of the logic and processing of the neuroid code.
- Flask
    - Flask combines the frontend and backend files. Flask allows the user to input data points to the frontend interface and sends it to the backend where the neuroid code processes the user input. At this point, the neuroid code returns an array of integers that gets displayed on a graph.

Code and Detail

Configurable Parameters

- umbr: Threshold to trigger activation.
- beta: Proportionality constant that controls the slope of the frequency-intensity.
- kr: Proportionality constant that controls amplitude of output.
- t: Length of sliding window that represents the inputs coming in per time unit.
- maxcount: The maximum number of iterations that can pass without nt_out being updated before it gets reset to

Components

- Comparator
    - The comparator checks if the sum is greater than the threshold.

```
def run_comparator(self, inputs, weights):
   input_sum = round(sum(inputs), 3)

   if input_sum > self.umbr:
       if self.count1 > self.beta / (input_sum - self.umbr):
           self.count1 = 0
       else:
           self.count1 += 1
   else:
       self.count1 = 0
```

- Frequency Modulator
    - The frequency modulator returns a value of one the moment the sum of the inputs passes the threshold. Before and after this moment, it returns 0.

```
def run_freq_modulator(self):
   if self.count1 == 1:
       self.y = 1
   else:
       self.y = 0
```

- Frequency Demodulator
  - When the output of the modulator is 1, the frequency demodulator calculates an output. This is the final value and the main output of the neuroid.

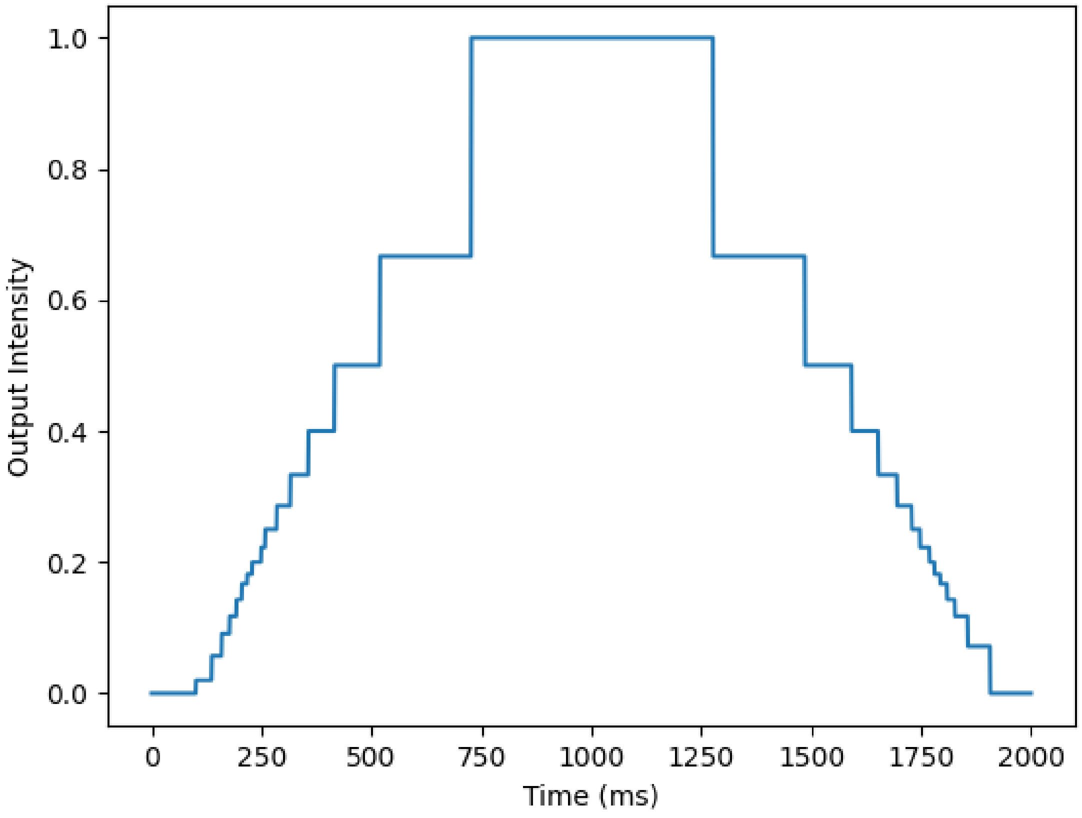
```
def run_freq_demodulator(self):
   if self.y == 1:
       self.nt_out = self.kr / self.count2
       self.count2 = 0
   else:
       self.count2 += 1

   if self.count2 > self.maxcount:
       self.nt_out = 0

   self.nt_out_stream.append(self.nt_out)
```

Output Example

- Input: A signal that starts at 0, increments up to 1, and then decrements back to 0, all by increments/decrements of 0.001.
    - umbr = 0.1
    - beta = 1.25
    - kr = 2
    - maxcount = 50
    - t = 1
- Output:



Requirements

System Feature 1: User Flow

Description

- The user interacts with the software by entering values in editable text fields or selecting from drop down menus.
- The user can view processing time of
- The user can view outputs generated by the software in graphical format.

Requirements

- The neuroidal network must implement hidden neuroid layers.
- Time parameter must be visible and run real time when the execute button is started.
- Question text boxes must be visible and be present to guide user on how to interact with form and features.

System Feature 2: Front-End

Description

- The inputs that are provided by the user associates with values connected to the back-end for processing.
- Form functionality is available allowing the user to interact with the software.

Requirements

- “Execute” button must run back-end code to generate output
- “Refresh” button must reset front-end form and clear stored values from previous trial
- “Forward” or “Next” button must cycle through code for next iteration in signal processing
- Number of neuroids that can be entered must be a positive integer
- Headers must match variable association in back-end code
  - User must be able to select from list of inputs or have field available for user defined input
- “Test Loss” and “Training Loss” must be present to user after trial is executed
- Training properties must be able to be modified before trial is executed using slider object

- Training properties must include “Ratio of Training to Test Data”, “Noise”, and “Batch Size”

System Feature 3: Connectivity

Description

- The front-end user interface button and fields must connect to back-end code, so software may function as intended.

Requirements

- Inputs entered by the user must transfer to back-end code for algorithm processing.
- Outputs generated by back-end code must transfer to the front-end form output section for graphical presentation.
- Graphical presentation must exhibit input signal and output signal.
- Form values must match neuroid parameters.

System Feature 4: Back-End and Neuroid

Description

- The neuroid is the most basic functioning unit of the neuroidal network.
- The neuroid can perform three operations that are carried out to process incoming information such as comparison, frequency pulse modulation, and frequency pulse demodulation.
- Back-end team implements the processing of the neuroid that extracts form inputs and processes them using signal inputs defined by the user.

Requirements

- The neuroid must have three main functions: a comparator, a frequency modulator, and a frequency demodulator.
    - The comparator must compare the incoming signal against the activation threshold.
    - The incoming signal must be represented by any value between -1 and 1.
    - Frequency modulator must facilitate downstream neuroid.
        - The frequency modulator must generate an impulse train with a frequency that varies proportionally to the input amplitude.
    - Frequency demodulator must inhibit downstream neuroid.

        - The frequency demodulator must be able to demodulate the impulse train.
- The neuroid must be able to represent normally silent and tonically active neurons.
- The neuroid must provide an amplitude-discrete version of the input signal with variable adjustment
- A threshold value must be established for the neuroid
    - If the threshold is surpassed, the frequency modulator or demodulator must be activated.
    - If the threshold is not surpassed, the outcome must be zero.
- Back-end development team implements the processing of the neuroid that extracts inputs.
- "Test Loss" and "Training Loss" must be generated for variable output to user on front-end

Final Thoughts

User Flow

We can say that developing a Neuroid could be an interesting choice when computational resources are limited. Our aim is to develop a prototype outlining the basic structure of the neuroid. We have created a prototype in Axure RP.

Later on a multi-scale system simulation tool will play an important role during the design, development, and testing of drugs. Development of medical devices and prosthetics will also be accelerated if their interactions could be tested on a multi-scale simulation platform before clinical trials. These industry trends are still in their infancy and will probably take a long time to materialize. However, there is certainly a need for platforms to accelerate in silico design and development.

Requirements

The requirements of the neuroidal networks software were initially extracted from the research paper proposed by Erick Javier as referenced. Even though the terminology of biology was unfamiliar for most of the group members, we made subsequent progress in understanding the topic during classes and by interacting with Pr. Ricardo. Overall we were able to provide the requirements of the single unit neuroid that serve to build the back end of the project.

Front-End

Overall, we accomplished a visualization of the neuroidal network with the look and feel of the UI diagram as well as the functionality of the neuroid backend. Using GitHub, we were able to connect many divergent approaches to code and create a unified product. Ultimately, due to the time constraints, some of the feature buttons from the UI diagram are present in the mockup, but have no impact on the visualization. The final product we do have, especially the graph of the output from the neuroidal network, is the culmination of a rigorous term of work, collaboration, and innovation.

Back-End

We made significant progress on finalizing our code in the last few days before our presentation. We were able to replicate an output similar to Erick's code by representing our input data in increments of 0.001 instead of 0.1. Additionally, we discovered that T is the value we were looking for to determine the size of the sliding window, whereas maxcount is the number of stagnant iterations to timeout and return nt_out to 0. Overall, we were successful in implementing the neuroid in our code.

We know, however, that there are still many improvements that can be made. For example, on some inputs, there is a small blip at the beginning where the output goes up and then returns to 0 before beginning the step pyramid pattern. This should not occur and is likely due to a bug we have yet to discover in our logic. There is also the case of maxcount. In an optimal solution, we would not have to manually set a value for when to return to 0, and instead it should automatically return there itself.

In addition, we know that creating a neural network from the neuroids was the end goal of this project, but it ended up being something to tackle beyond the scope of this class.

References

1. Arguello, E., et al. “The Neuroid: A Novel and Simplified Neuron-Model.” *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2012, doi:10.1109/embc.2012.6346160.
2. Carter, Daniel Smilkov and Shan. “Tensorflow - Neural Network Playground.” A Neural Network Playground, playground.tensorflow.org/.
3. Chart.js contributors. “Line Chart.” *Line Chart | Chart.js*, 2021, www.chartjs.org/docs/master/charts/line.html.
4. Pena-Ortega, Fernando. “Pacemaker Neurons and Neuronal Networks in Health and Disease.” *Advances in Clinical Neurophysiology*, 2012, doi:10.5772/50264.

5. Prada, Erick Javier, et al. “Lamina Specific Loss of Inhibition May Lead to Distinct Neuropathic Manifestations: A Computational Modeling Approach.” *Research on Biomedical Engineering*, vol. 31, no. 2, 2015, pp. 133–147., doi:10.1590/2446-4740.0734.