RSCAD Neural Network Library

September 19, 2024

THIS IS STILL MISSING A LOT including:

- python file overview
- Cbuilder component building process
- a table with every single file and what it is
- an overview of the C code
- Future optimizations for anyone willing to fix and improve

The Github repository is made has two main folders, NNLib and RSCAD Files. NNLib contains the Cbuilder components to be added in RSCAD that form the Neural Network toolbox, the RSCAD files represent the MMC model this toolbox was initially designed for. In the main page you will also find instructions on how to upload all these files and in what folder on your personal computer. This file will mainly discuss the content of the folder "NNLib".

Chapter 1

Full NNs

The contents of NNLib are summarized as follows:

- Folder Full_ANN: includes files associated with a component representing a basic full ANN structure (MLP)
- Folder LSTM_Final: includes files associated with a component representing a an LSTM Neural Network
- SingleLayerComponents: includes files associated with a component representing a singular layer in an MLP (ANN). This includes the input, hidden, and output layers.

1.1 Full_ANN

This folder includes one .def file, one .c file and one .h file. Once these files are uploaded in the right repositories on your PC as per the document "instructions to add C-Builder Component" you can add the definition file .def to your RSCAD draft file as follows:

the option ANN.def should appear, once it is loaded you expect the following:

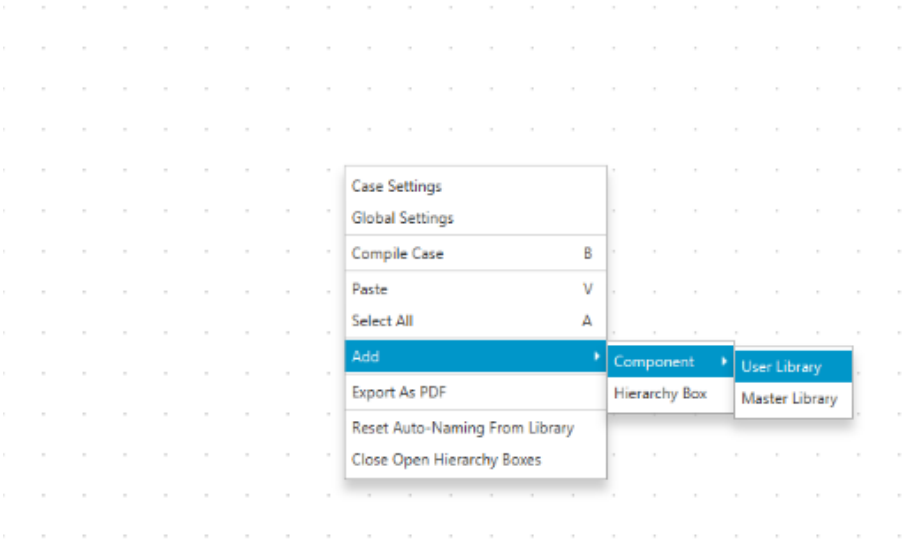


Figure 1.1

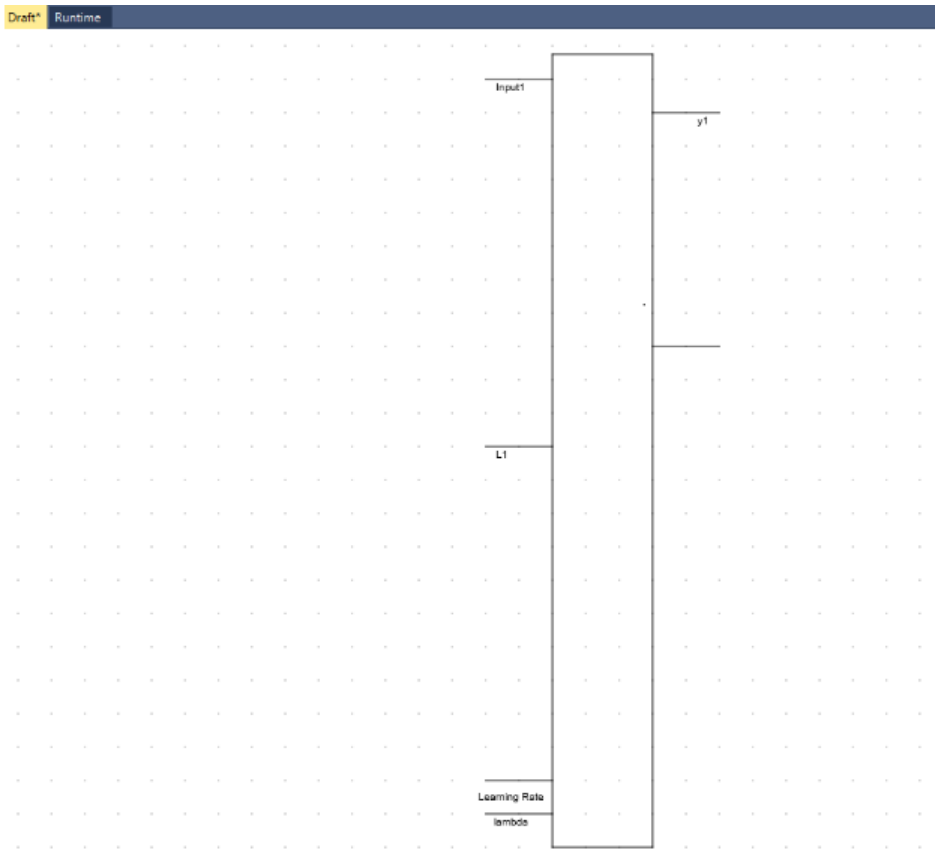


Figure 1.2

by double clicking this component you should get the following menue:

here we define the initial parameters of the Neural Network which are:

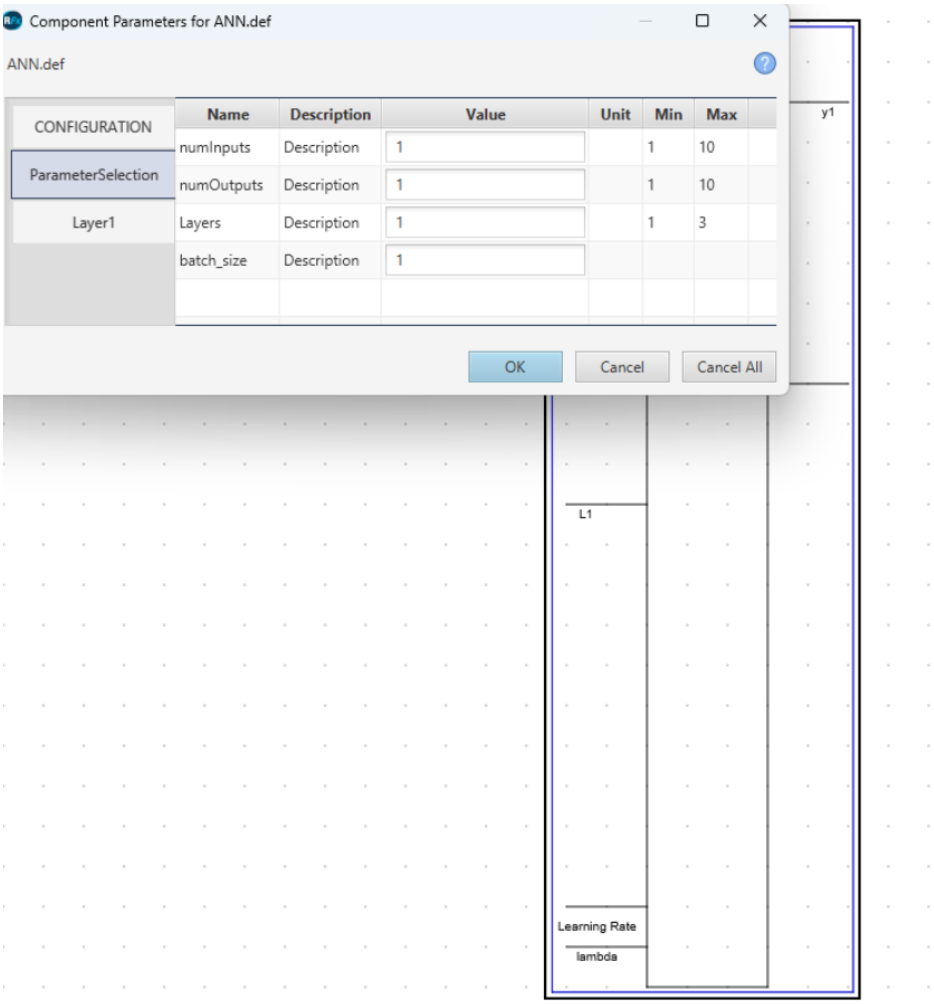


Figure 1.3

- numInputs: How many inputs go into the Neural Network
- numOutputs: How many outputs are expected from the NN
- Layers: How many hidden layers between input and output layers
- Batch_size: How many loss intervals before the weights are updated

under the ParameterSelection tab you can see there is a tab for Layer1, by increasing the number of layers more tabs will show up:

here we can see that tabs for layer1 2 and 3 showed up, which is the maximum number implemented in this component, for each layer there are 2 parameters that can be varied:

- Activation: The activation function of the layer
- Nodes: how many nodes the layer can have (20 maximum)

Once all the parameters are finalized the NN component should look like:

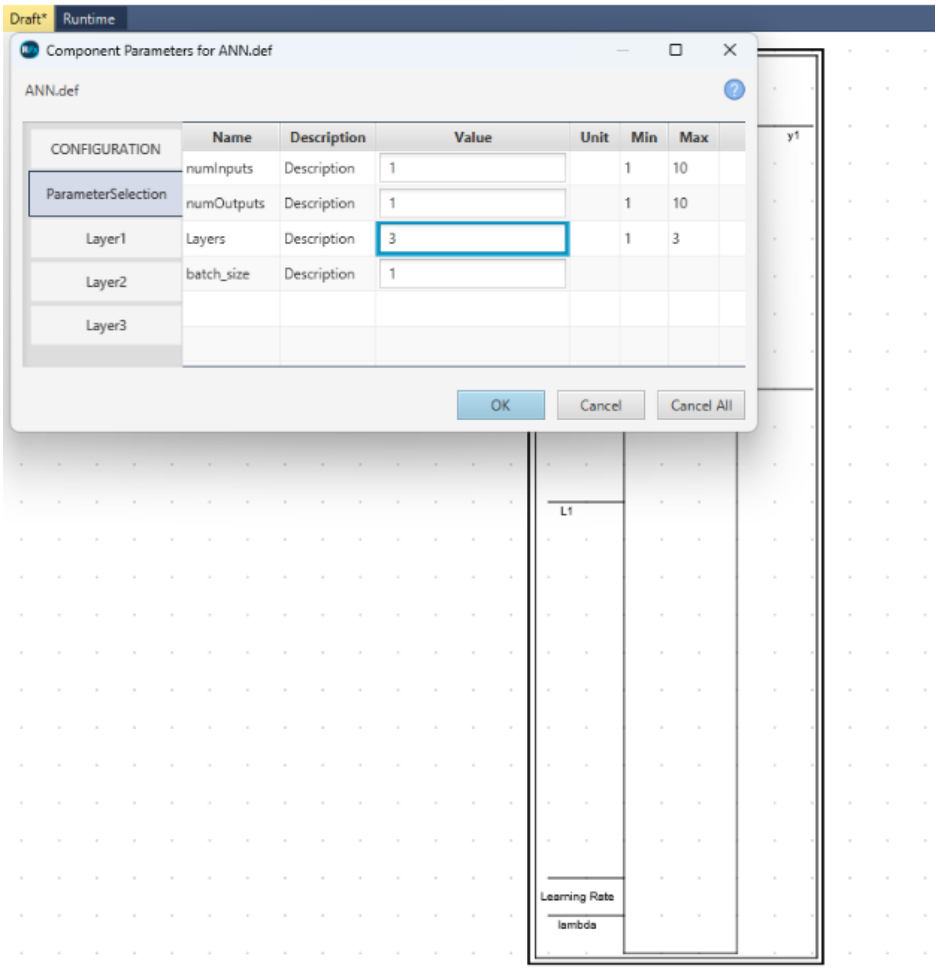


Figure 1.4

In this archticute I chose 4 inputs and 2 outputs, the number of layers is irrelevant in its appearance. On the left side you have the inputs into the component and on the right the outputs which are described as follows:

- Input1,2,3,4: The input variables of the problem you are trying to solve with the Neural network.
- y1,2: The outputs which are the results given by the Neural Network
- L1,2: These are the loss functions for each of the output (explain more maybe add equation) Ill add it here later but for now you can refer to the layered component chapter where how to define a loss in RSCAD is explained in detail.
- Learning Rate: The rate at which the Neural Network learns every iteration
- lambda: Regularization constant

the learning rate and lambda can be connected to a slider to dynamically change their values throughout the simulation. It is important to mention that the regularization is not optimized and is best to keep at zero for now. The activation rate needs to be dynamic because it needs to be set to zero once the training process is over.

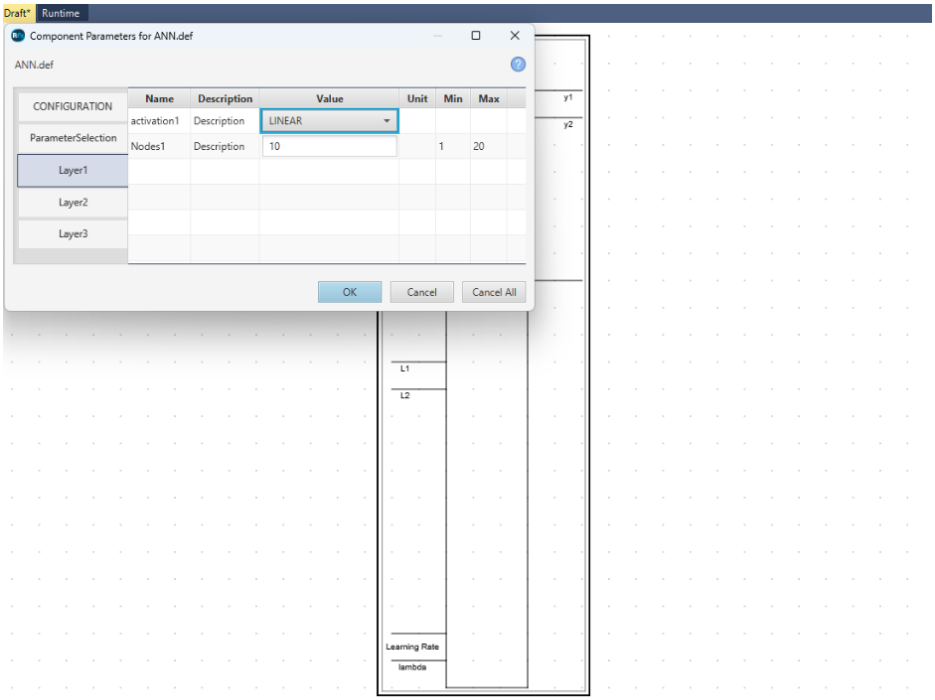


Figure 1.5

1.2 LSTM

the LSTM repository is very similar to the previously dicussed ANN repository. Once the component is loaded:

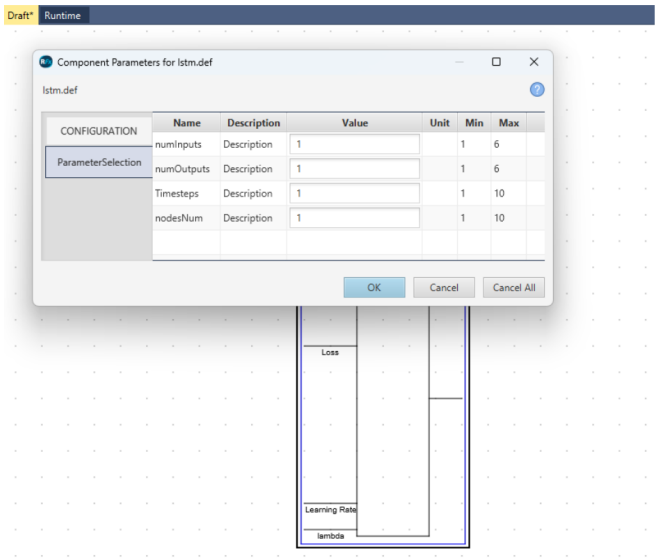


Figure 1.7

Here the parameters are as follows:

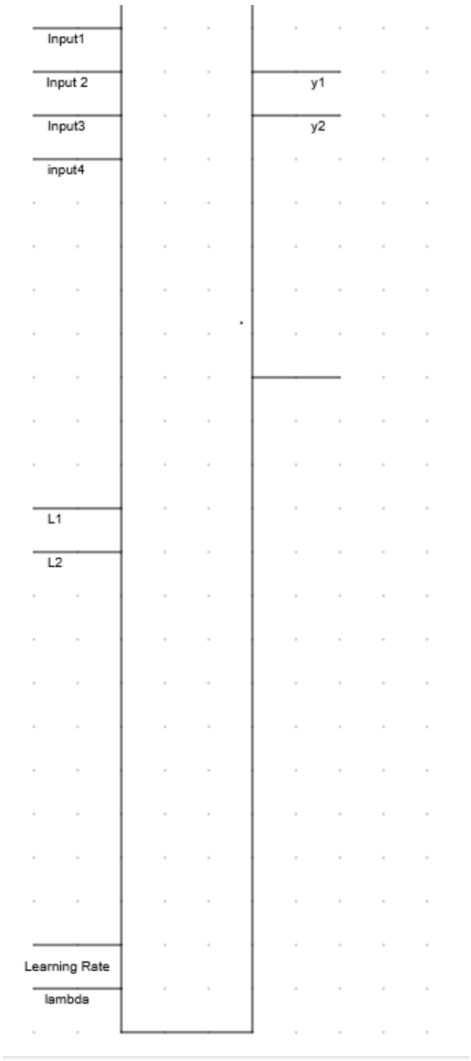


Figure 1.6

- numInputs:
- numOutputs:
- Timesteps: How many timesteps go into the component for one prediction, so if the maximum 10 time steps are taken, the LSTM will take the previous 10 timesteps of inputs 1,2,3... and make the calculations found in equations.....
- nodesNum: LSTM node number (explain more)

Chapter 2

Layered Components

The components in the library can be divided into two, Full Neural Networks and Single Layers. Full Neural Networks are components that include the computation of all layers involved while single layer blocks are used to design Networks one layer at a time.

2.1 Layered Components

There are three different layer components:

- The input Layer (InputLayer.def).
- Hidden Layer (MidLayer.def).
- The Output Layer (OutputLayer.def).

For the ANN layers, each component has 3 parameters to be selected before compiling, these are numInputs, numOutputs, and activation.

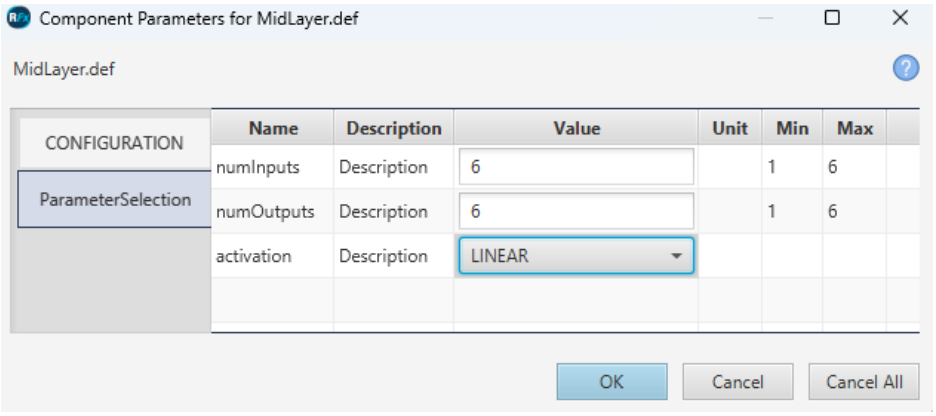


Figure 2.1

Output Layer

The output layer takes in the following inputs:

- Input1 - Input6: The inputs into the layer, used as the outputs of the previous layers
- loss y1/y2: The derivative of the loss in terms of prediction y1 and y2. This was the user can design the loss function outside of the component and feed it as input into the output layer.
- Learning_rate: The rate at which the Neural Network Learns
- Lambda: Regularization Term to drop out weights

the outputs of the layer are:

- y1/y2: The predictions of the network, this output is what should be used for the loss function design.
- Li1-Li6: These are the losses propagated backwards to the previous layer, as in, these outputs are fed as inputs into the previous layer and represent the loss at the node connecting both layers.

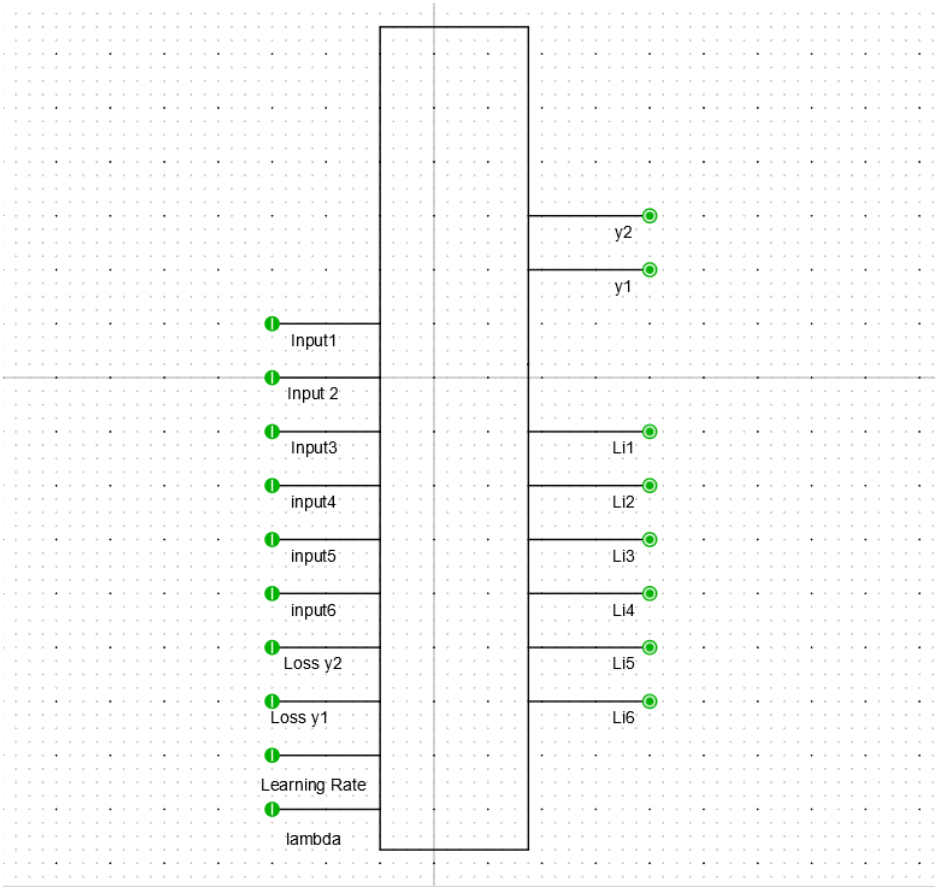


Figure 2.2

Hidden Layer

The hidden layer takes in the following inputs:

- Input1 - Input6: The inputs into the layer, used as the outputs of the previous layers
- L1-L6: The previous loss of the output of this layer, fed back into it from the next layer.
- Learning_rate: The rate at which the Neural Network Learns
- Lambda: Regularization Term to drop out weights

the outputs of the layer are:

- y1-y6: The output nodes of the hidden layer
- Li1-Li6: Losses propagated backwards to the previous layer, these outputs are fed as inputs into the previous layer and represent the loss at the node connecting both layers.

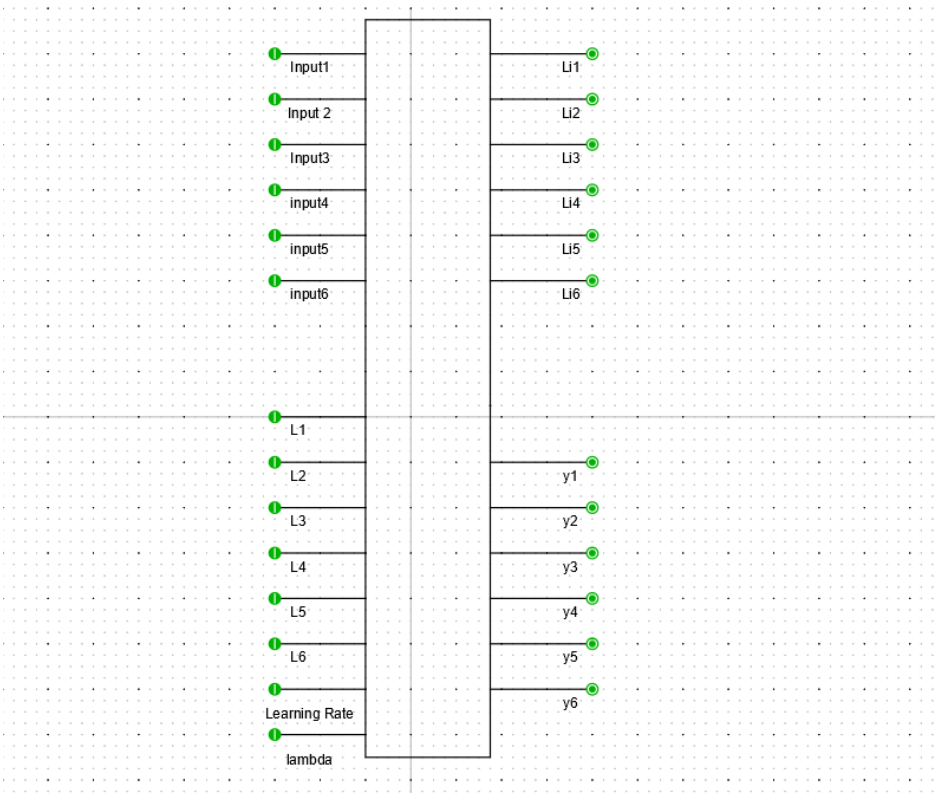


Figure 2.3

Input Layer

The hidden layer takes in the following inputs:

- Input1 - Input4: The input variables into the Neural Network.

- L1-L6: The previous loss of the output of this layer, fed back into it from the next layer.
- Learning_rate: The rate at which the Neural Network Learns
- Lambda: Regularization Term to drop out weights

the outputs of the layer are:

- y1-y6: The output nodes of the hidden layer

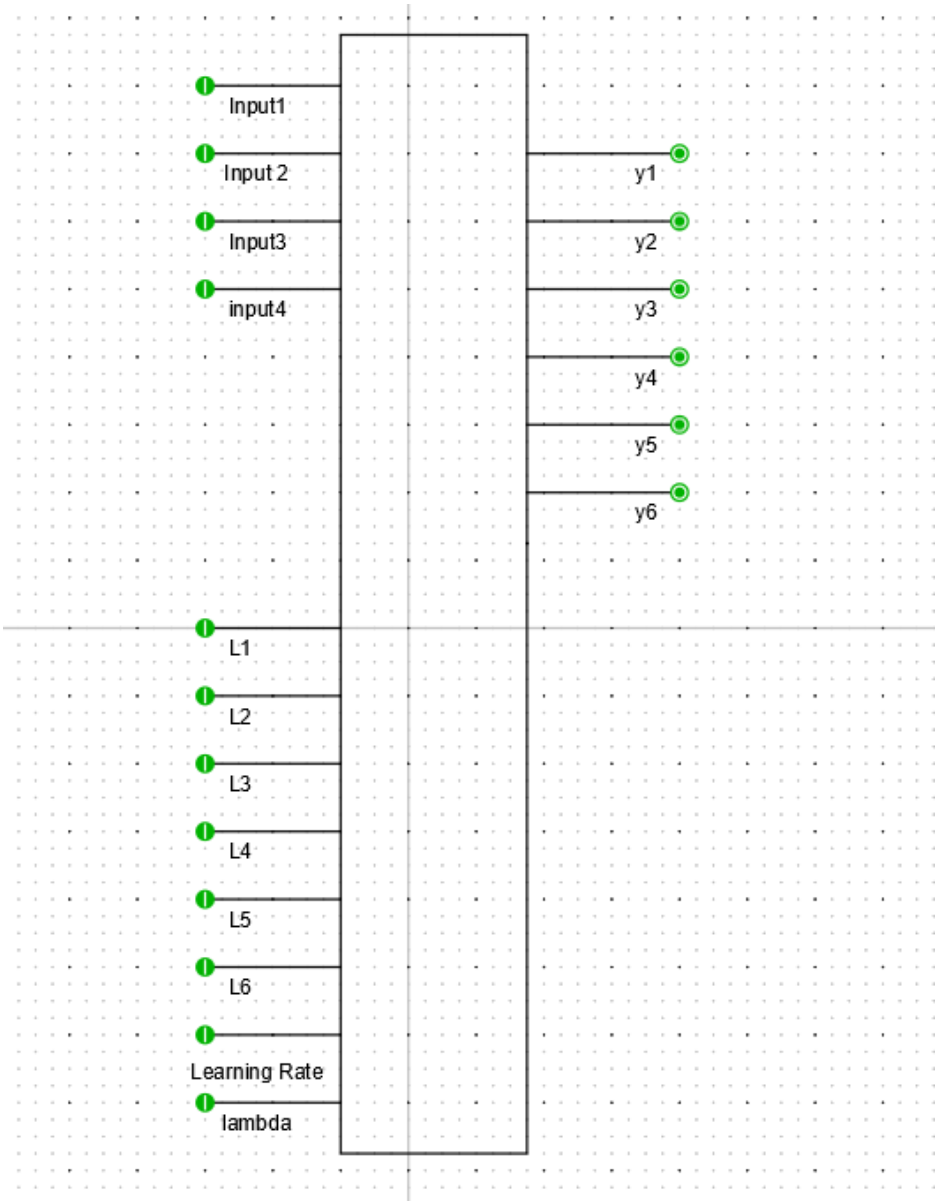


Figure 2.4

Using these components the user can construct their own Neural Network, to give an example, I will construct a 2 and 1 hidden layer Neural Network with 2 inputs and 1 output.

3 Layer Network

First we define the required variables:

- Loss
- Learning Rate
- Regularization Term

We can take a simple example of MSE loss function, the derivative of the loss function with respect to the prediction is simply the difference between the prediction and the target (neglecting the 2 multiplication term). The learning rate and regularization lambda will be sourced from sliders as shown below:

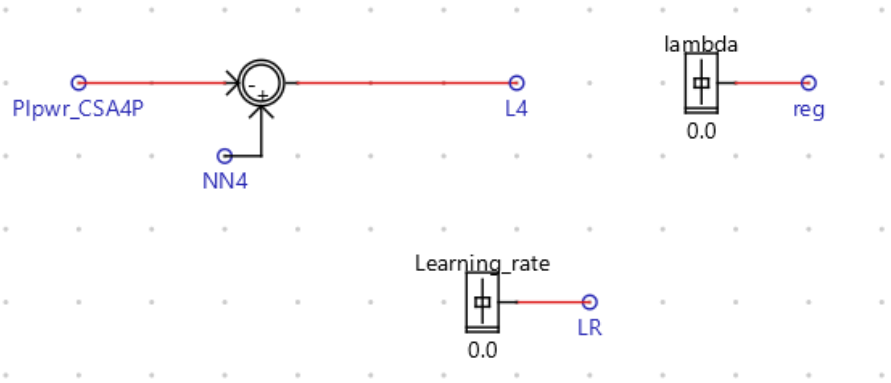


Figure 2.5

To construct a Neural Network with one hidden layer, we can connect the input layer to the output layer as shown below:

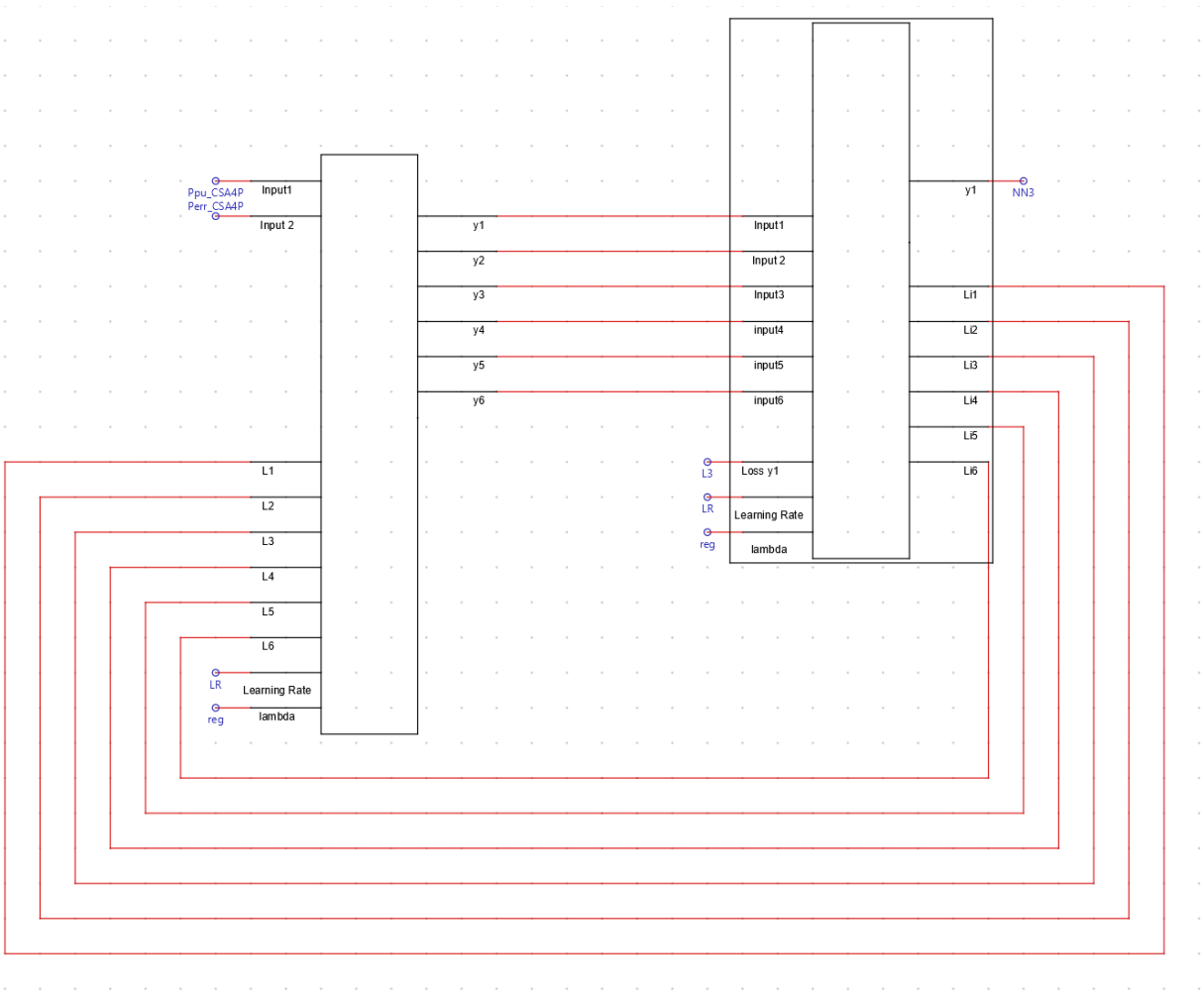


Figure 2.6

4 Layer Network

Using the same logic and parameters, we can construct a 2 hidden layer network by adding a MidLayer.def component as shown below.

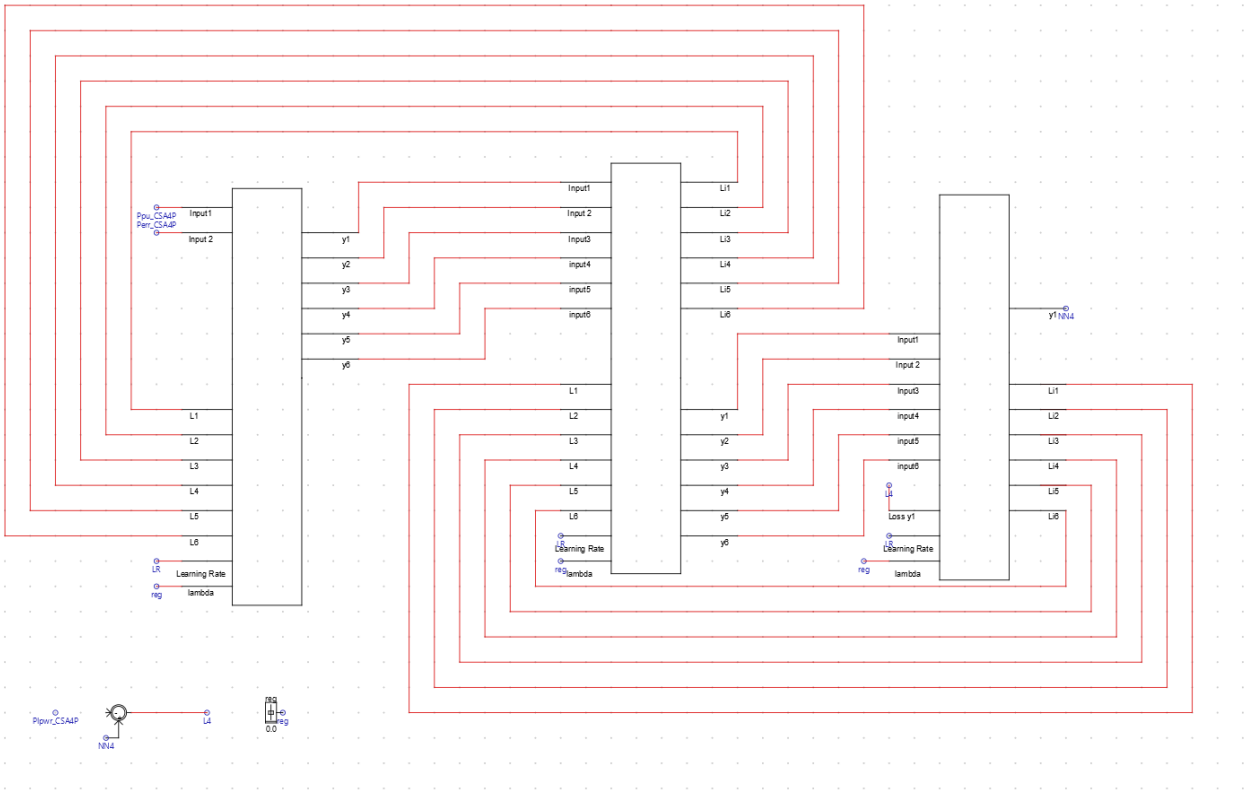


Figure 2.7

2.1.1 Training and Regularization

Since there is no learning rate optimizer, the rate should be adjusted according to the behaviour visible to the user. For example, during training, if the direction of prediction and the direction of the target are not similar during a sudden change in input parameters, then the weights do not have the correct signs. In this case the learning rate will be set high (0.01-0.0001) until the Neural Network output and the Target variable move in the same direction during changes in input parameters. Once that is achieved, the learning rate should be then reduced to somewhere 1e-4 - 1e-9, the user should adjust based on apparent results. Make sure not to keep the same input parameter range for extended duration's of time. If, during training, the shape of the prediction during transient is very odd, you can stop the training during transient and resume after, training is stopped by setting the learning rate to zero.

For the Regularization term, a value of 0.0 can be taken to avoid complications. However, if it is required, then start with a low number (1e-8) and adjust accordingly. Setting lambda larger ($\approx$ 1e-4) can lead to most of the weights dropping out, an appropriate value of lambda most likely is found in the range 1e-8 - 1e-4.

Chapter 3

CBuilder Building Process

3.1 Initial Set-up

In order to design a component, the first step would be to set up the Graphics and IO (Input/Output) points in the tabs shown in Figure **??**. For each IO point, the data type of the point needs to be specified (Real, Integer, or Complex). The user can then proceed to the 'C File Associations' tab, here the user can specify the I/O points that would be associated with the script (from the already defined points).

3.2 Scripts

The next step would be to set up the header and script files (.h and .c files). The .h file simply contains definitions of inputs and outputs. However, when using complex operations multiple header files might be needed to define the data structures and functions necessary for implementing these algorithms. These might include definitions for managing buffers or queues to hold past values of inputs (Such as delay implementation), as well as functions for updating these structures and computing the desired outputs. Additionally, it may require utility functions for initializing these data structures, handling memory allocation, and ensuring data integrity over successive iterations. This layered approach allows for modular, readable, and maintainable code.

In collaboration with tools like Simulink, the complexity of setting up such functionalities is reduced. Simulink, a MATLAB-based graphical programming environment, allows for the modeling and simulation of systems through a block diagram approach. When specific operations, like the integration of a delay, are required as part of a larger system model, Simulink provides pre-built blocks that simplify its implementation. These blocks are converted to relevant .c and .h files that are specifically tuned for the RTDS hardware. The Simulink conversion tool was utilized for initializing the component and generating the relevant header files. Unfortunately, attempting to use the Simulink Conversion tool for creating complex components such as Neural Networks usually leads to errors raised as the CBuilder compiles the model to an Assembly file. Therefore this feature is only used for initializing the inputs and outputs (specifically for RNN implementation).

After the header files are generated. The .c file should include the functionality of the component. This file is divided into the following sections

- Static: Used for declaring variables that retain their value from one timestep to the next. These variables are accessible in the RAM_PASS1, RAM_PASS2, and CODE sections.
- RAM_FUNCTIONS: Contains definitions for user-defined functions to be called within the RAM_PASS1 or RAM_PASS2 sections.
- RAM (RAM_PASS2): Code in this section is executed once when a DRAFT case is compiled. It is not subject to real-time performance constraints, allowing for complex and time-consuming algorithms, such as curve fitting, to be implemented here. However, since it only compiles at the start, only the results of these computations can be referenced in the CODE section.
- RAM_PASS1: Similar to RAM_PASS2 but executed before it during the compilation of a case. Specific function calls and variable assignments that are crucial for the subsequent computation steps are placed here, such as the definition of overlay matrices.
- CODE_FUNCTIONS: This section includes user-defined functions that will be called within the CODE section.
- CODE: Contains all the code executed on the RTDS Simulation hardware for each timestep. Due to real-time constraints, this code must be efficient and cannot take excessively long to execute to avoid triggering a timestep overflow error.

In the context of building a Neural Network component, the Static and RAM sections are not utilized besides for weight initialization. This is due to the fact that there are no constants. However, for a Neural Network without an online training functionality, static values and structures can be used for faster computation of more complex neural networks. For this Thesis, online training functionality was successfully added in the CODE_FUNCTIONS section.

After writing the script, the CBuilder compiles all the relevant .h and .c files and generates C files of its own, most likely for compatibility with hardware. The files are then further decoded into an Assembly file which is finally used to construct the .def (RSCAD Library) file. This is shown in Figure 3.1.

As discussed previously, the main timestep of the simulation in question 50 μs for the model to retain its real-time behaviour. However, this does not mean that the component will have this time to compute its solution. The communication within the Network can be seen in Figure 3.2. Here each timestep is divided into 6 layers as depicted by the figure, where the blue lines indicate a communication layer while the blocks indicate computational layer. During the first computational layer, all computations contributing to the network solution are required to be completed. However, the Neural Network does not directly contribute to the network solution but would still require fast execution when used for Inner-loop control as the modulation indices need to respond to change within the timestep. The significance of the structure shown in Figure 3.2 is that there are three more computational intervals within the timestep that can be used by the components in the network to process calculations that do not directly alter the current network solution. As in, a backpropogation algorithm does not require full execution before the component gives its output, this can be largely leveraged due to the computational complexity involved in BPTT (Back-Propagation Through Time) algorithms.

3.3 CBuilder Neural Network Model

The final CBuilder component graphic is shown in Figure 3.3, following the logic depicted in Figure 3.4. The mathematical logic can be understood by looking at Chapter **??**.

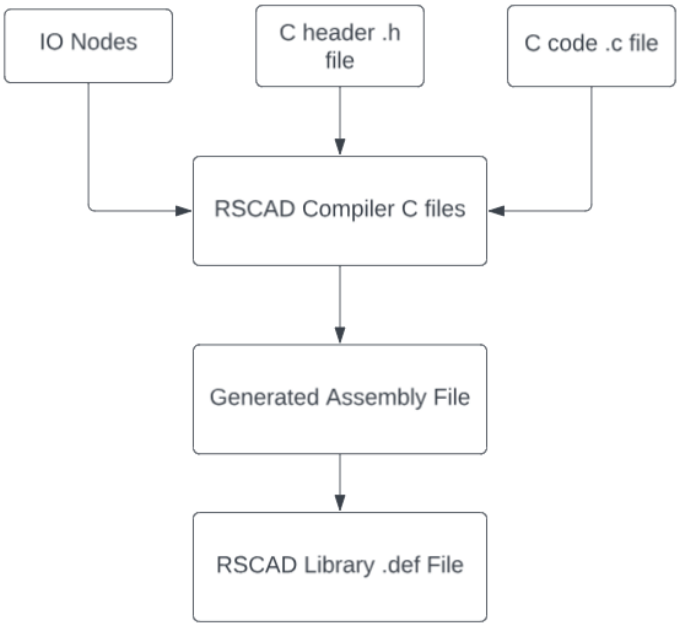


Figure 3.1: CBuilder compilation process (from .c files to .def)

In this component, the inputs are the 4 X features (x_1, x_2, x_3, x_4), the number of layer L, number of weights in hidden layer 1 u_1, number of weights in hidden layer 2 u_2, learning rate Lr, target variables y_1 and y_2, batch_size, and finally, lambda 1 and lambda 2. The values of lambda 1 and 2 can be changed throughout the simulation and are used to prioritize a certain target variable over the other such that the loss function is:

$$Loss = \lambda_1(y_{1pred} - y_1)^2 + \lambda_2(y_{2pred} - y_2)^2 \tag{3.1}$$

The computing logic of each Nueral Network component can be understood by the following flow chart:

After building each component separately, the different scripts can be used to form one component that includes all separate networks, this is done by introducing parameter definition prior to RSCAD compilation, this is done as follows:

Besides the inputs/outputs of the component, define a new section in the parameter window as shown on the right of Figure 3.5

The parameters in Figure 3.5 are:

- numInputs: Number of features the NN tries to map (1-4).
- numOutputs: Number of prediction variables (1-2).
- type: Type of NN model (ANN, LSTM, RNN).
- numLayers: Number of hidden layers between input and output layer (1-2).
- numNodes: Number of LSTM nodes (1-5), activated only when type==LSTM.
- numWeights: Number of Weights in hidden layers (1-20).

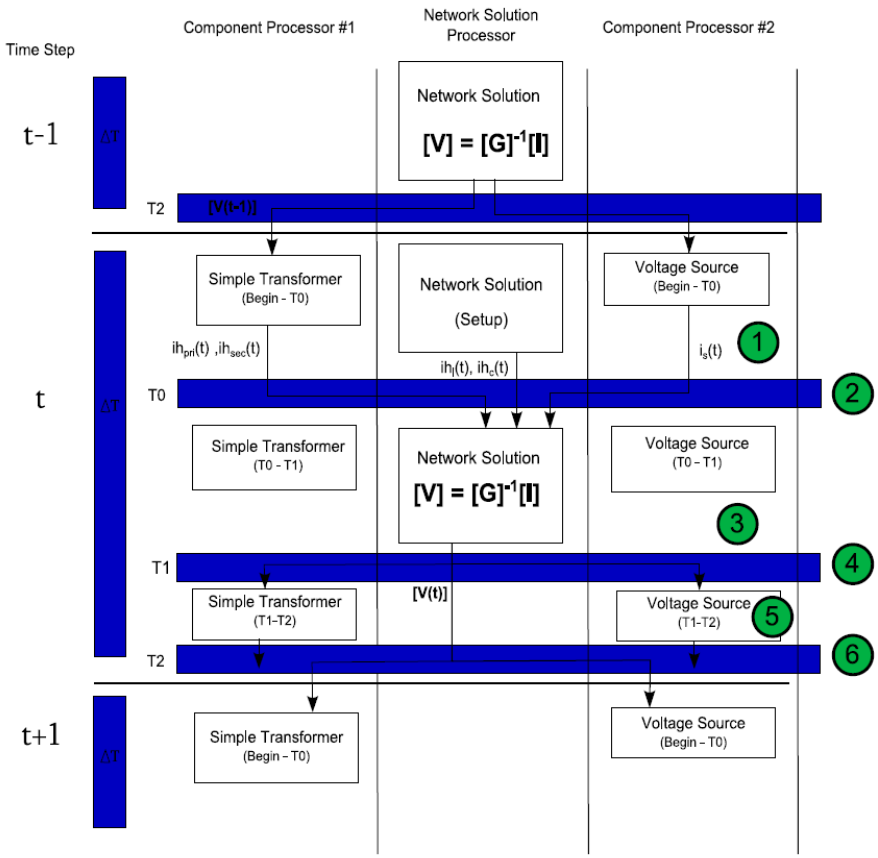


Figure 3.2: A schematic representing the computation and communication intervals within a timestep in RTDS, taken from manual

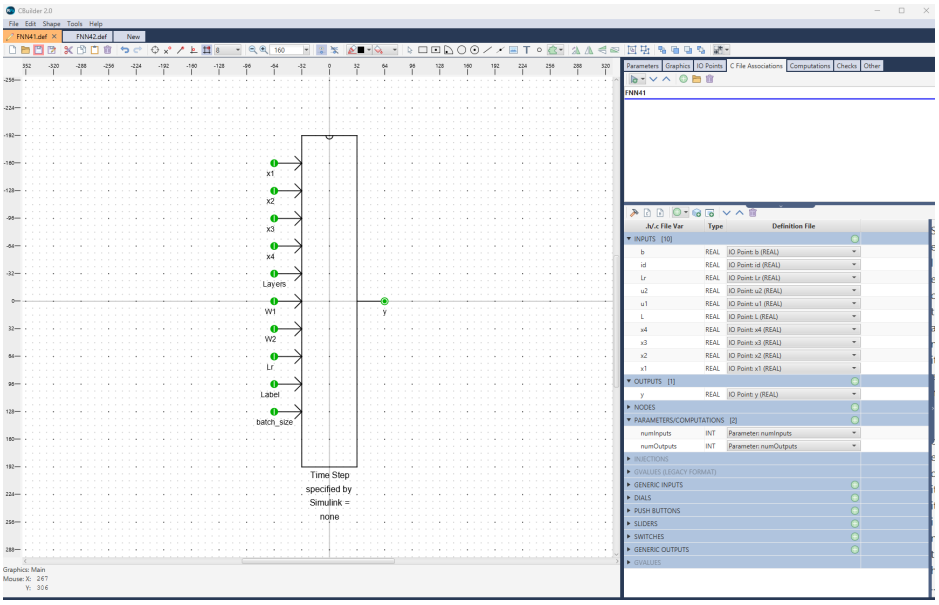


Figure 3.3: CBuilder interface depicting the graphics and I/O points of the component for a 4 feature 2 output 2 hidden layer Neural Network

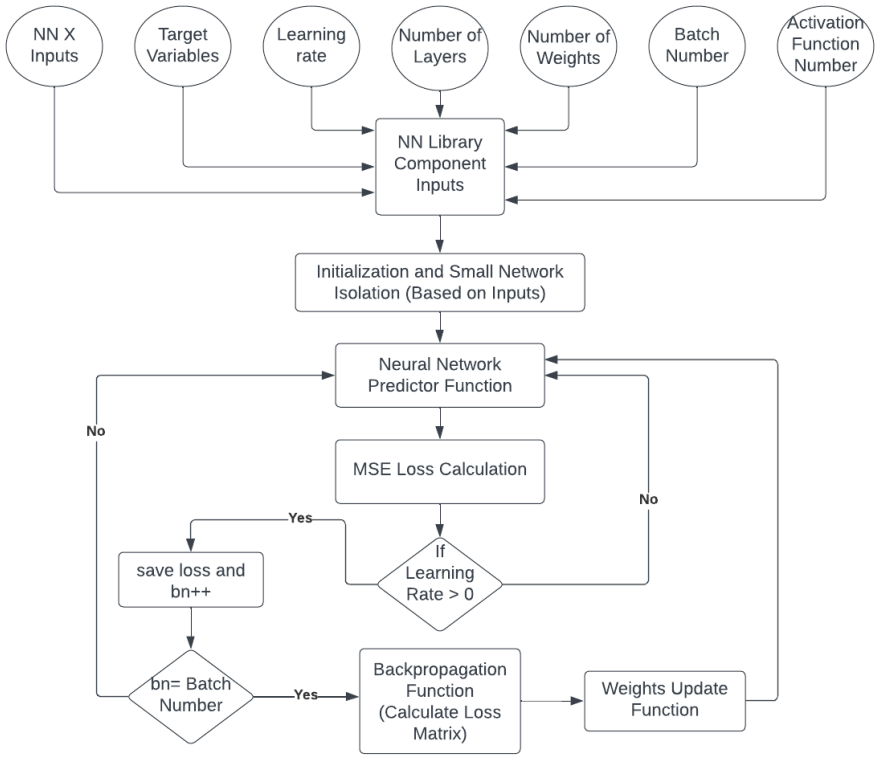


Figure 3.4: Flow Logic of Neural Network Component in RSCAD

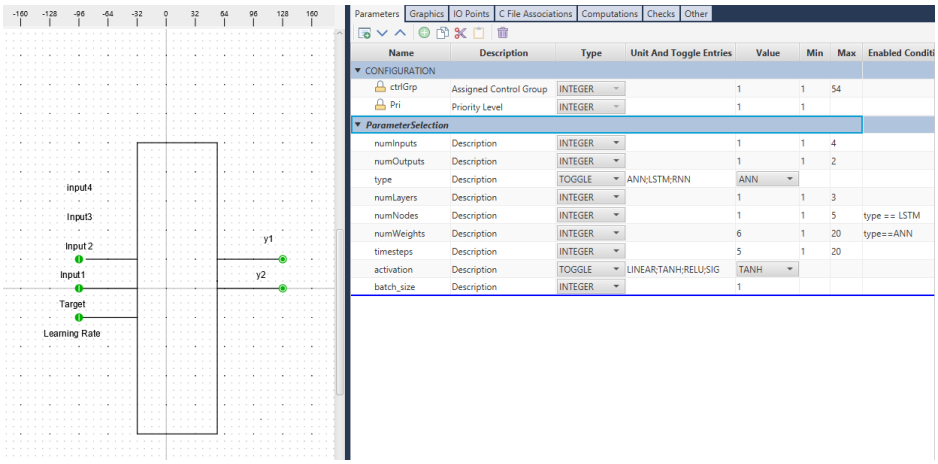


Figure 3.5: CBuilder interface depicting the parameters required for initializing the component in RSCAD. Missing nodes appear when numInputs is adjusted

- timesteps: Number of previous timesteps taken by the LSTM (1-20).
- activation: Activation function in the hidden layers (tanh, linear, ReLu, Sigmoid), does not work for LSTM models as they have predefined activation.
- batch_size: The number of iterations before the backpropogation algorithm is executed.

The inputs/outputs of the Neural Network component are straightforward, besides the values of X and y, the user needs to input the target variable as an input and the learning rate. The learning rate

was taken as a dynamic input to give the user the ability to oscillate through the range and find the most appropriate rate, which is as defined in Chapter **??** determines how much change the weights go through every backpropagation iteration.

After defining the initial parameters, the nodes (inputs/outputs), and the graphics, the C association files need to be initialized. Cbuilder allows the utilization of a different C script based on the parameters defined in the list above, therefore, multiple scripts can be included in one component but only the relevant script will be computed once the parameters are selected. The I/O nodes have to be defined again in the inputs/outputs tabs in order to be used within the C script as shown in the figure below

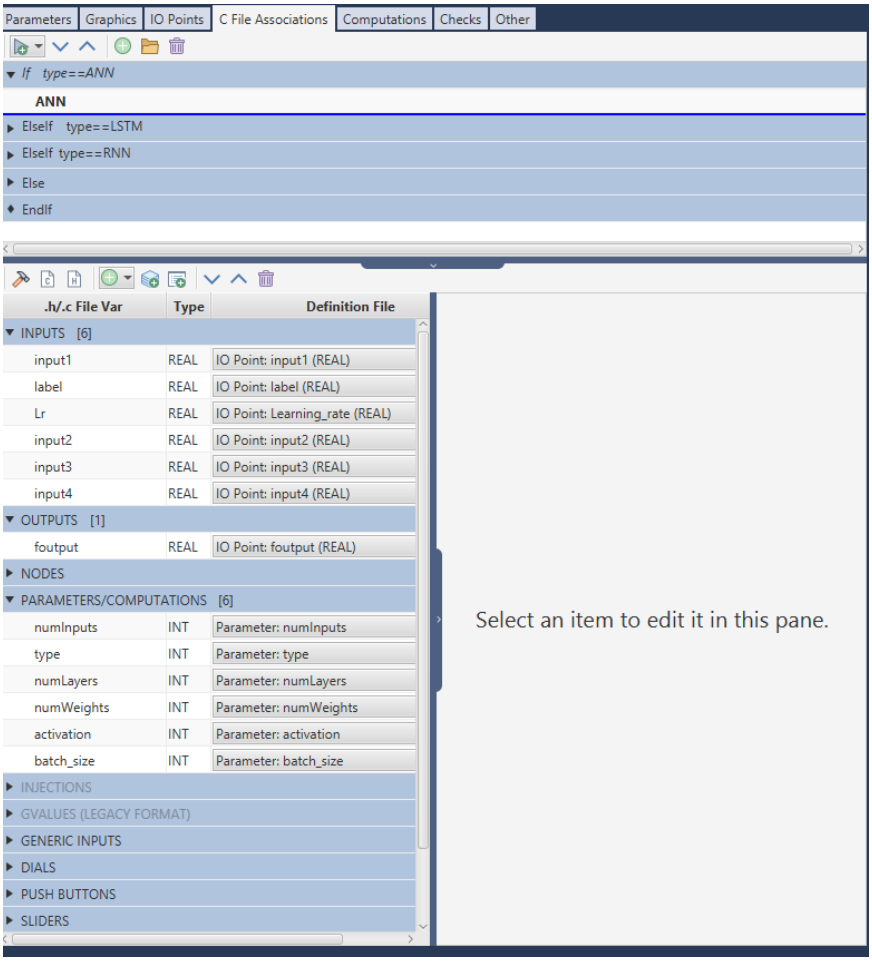


Figure 3.6: C association tab

Chapter 4

Running Existing Models

4.0.1 HVDC-RSCAD Model

The following steps need to be followed to run the files:

Step 1: Download the files from the git directory.

Step 2: Extract the zip "HVDC-RTDS-Models.zip" in a temporary folder.

Step 3: From the extracted folder, copy the "`CSE_MODEL_FX2.0`" folder, and paste it to your RSCAD user directory (example: "`C:\RSCAD\RTDS_USER_FX\fileman\`").

Step 4: Now open the RSCAD FX2.0, and load the desired models from the `CSE_MODEL_FX2.0` folder from your directory. I am considering the "`CIGRE_4TERMINAL_525KV_RTS_MODEL_3D_check.rtfx`" test case for this explanation.

Step 5: Upon loading the above file, you will see the following Draft tab.

Modify the model to incorporate the neural network library (see section B).

Step 6: Compile the file on a rack with suitable cores, in this case, a minimum of seven cores.

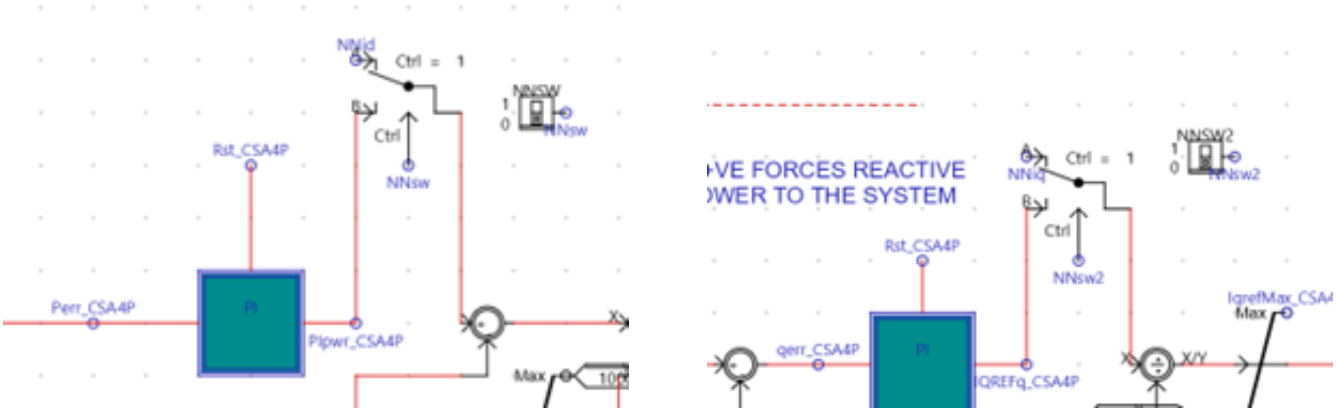
Step 7: Now go to the Runtime tab.

Step 8: Now open and run the script "`Parameter_setting`". This will configure the draft variables in the runtime.

Step 9: Now open and run the script "`starting_squence_5T_file`". This will start the simulation with a starting sequence mentioned in the paper. Once the script is completed now, you can begin your study.

4.0.2 Modification in Model for NN Application

Add a switch to include NN output for both active and reactive power as depicted in Fig. 4.1.



(a) (b)

Figure 4.1: Inclusion of switches for the control of: (a) active power; and (b) reactive power.

4.0.3 CBuilder Setup

1. Put all the c-files [.c, .h,] except the def files in Bin-CMODELSOURCE (.h file simply contains definitions of inputs and outputs, e .c file should include the functionality of the component, see details in Chapter 6 of thesis) [`C:\Users\rashmiprasad\Documents\RSCAD\BIN\CMODEL_SOURCE`]
2. Put the def files on -Component folder `C:\Users\rashmiprasad\Documents\RSCAD\ULIB\COMPONENTS`
3. In the toolbar: Go to Launch- Component Builder- File Open- .def file from `C:\Users\rashmiprasad\Documents\RSCAD\ULIB\COMPONENTS`
4. Open C File Associations and select and compile the codes as depicted in Fig. 4.2.
5. Goto-Draft-add-component-User lib-add that .def as a component in the draft.

4.0.4 Offline Training

1. Saving files: Get the Data from the runtime.- plot the inner current control and outer voltage control PI input and output parameters- [(Perr_CSA4P, PIpwr_CSA4P), (qerr_CSA4P, IQREFq_CSA4P), (iderr_CSA4P, outpi1_t_CSA4P), (iqerr_CSA4P, outpi2_t_CSA4P] Use the script file to run multiple case to save data by varying P and Q.
2. Code
3. Generation of weights and updating

4.0.5 Online Training

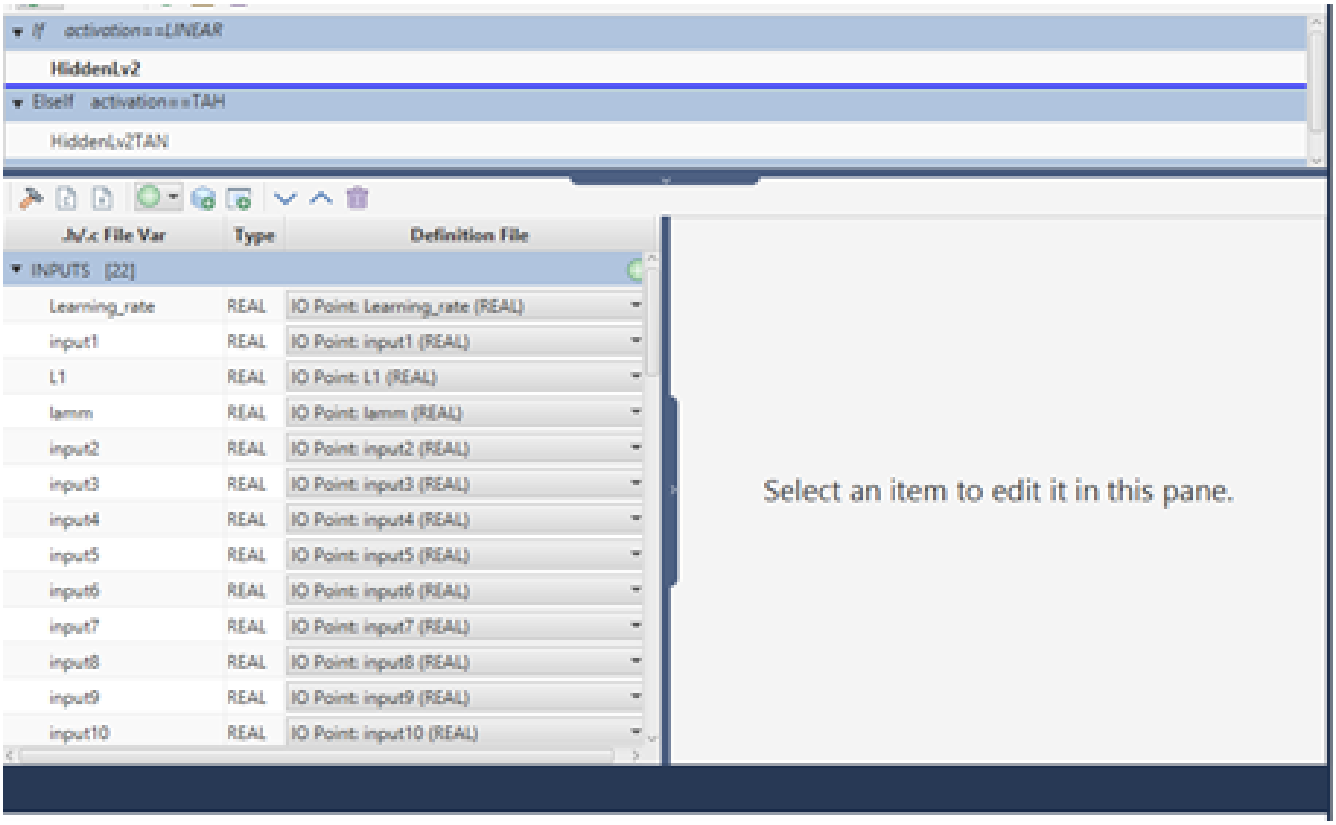


Figure 4.2: Compile codes in RSCAD.