

RSCAD Neural Network Library

Bara Masalmeh and Aleksandra Lekić

September 20, 2024

Chapter 1

CBuilder Building Process

The CBuilder is a feature found in RSCAD that allows the user to manually design his own component through utilization of low-level C programming. The CBuilder interface is shown in the Fig. 1.1.

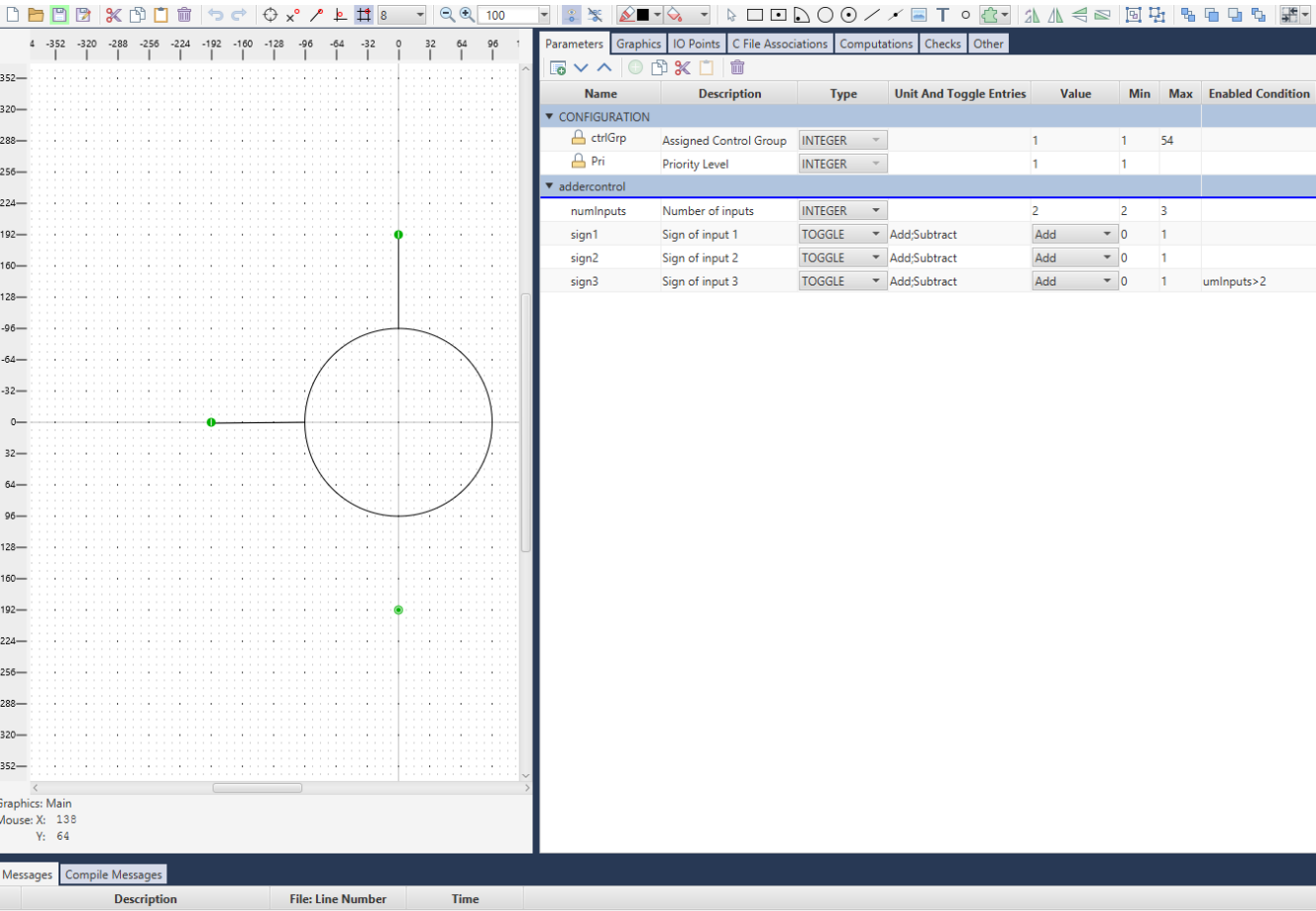


Figure 1.1: CBuilder Interface

1.1 Initial Set-up

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 1.1. 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).

1.2 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 to initialize the component and generate the relevant header files. Unfortunately, attempting to use the Simulink Conversion tool to create complex components such as Neural Networks usually raises errors 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 1.2.

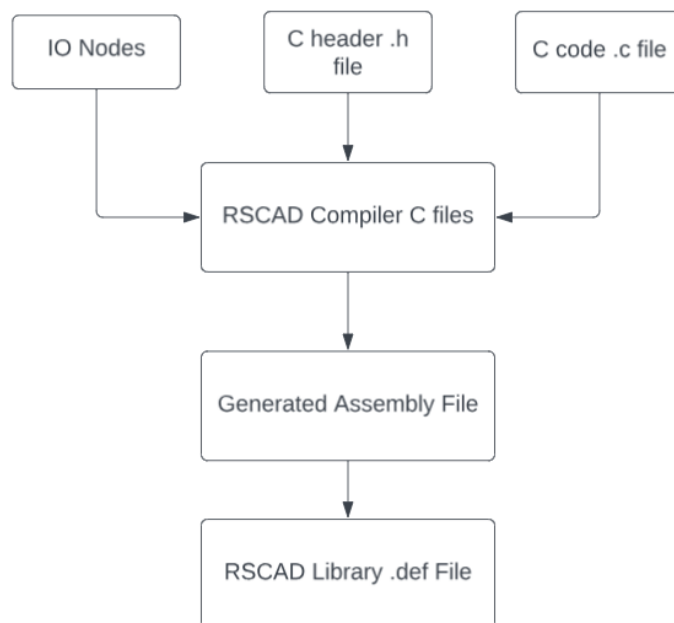


Figure 1.2: CBuilder compilation process (from .c files to .def).

As discussed previously, the main timestep of the simulation is $50 \mu s$ 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 1.3. 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 1.3 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 backpropagation 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.

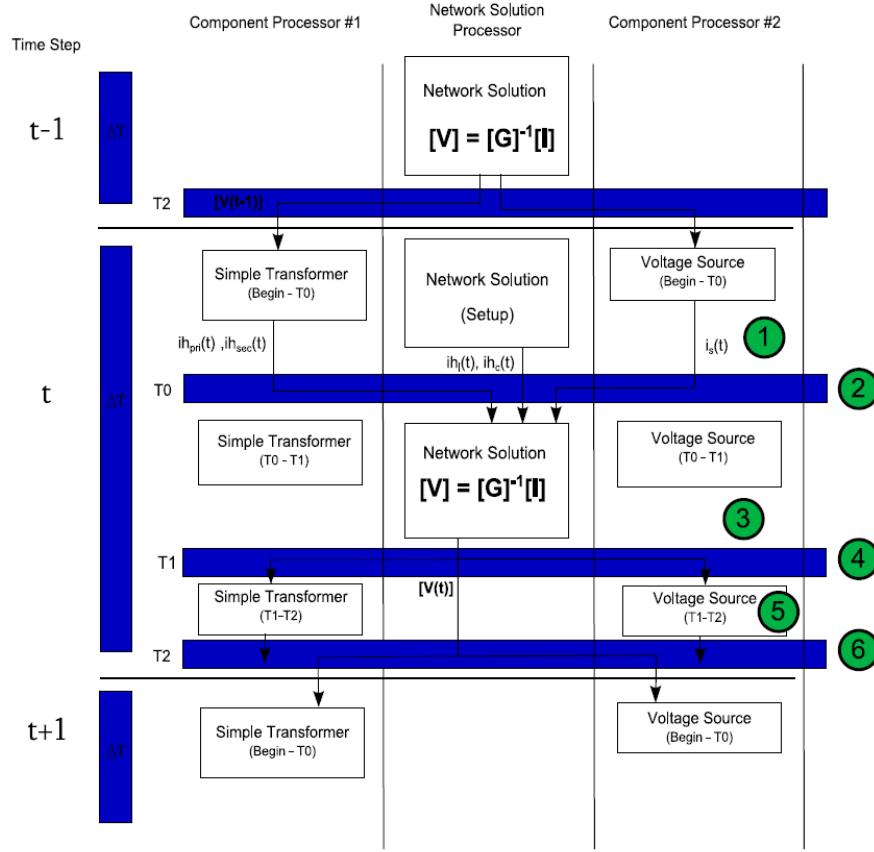


Figure 1.3: A schematic representing the computation and communication intervals within a timestep in RTDS, taken from manual

1.3 CBuilder Neural Network Model

The final CBuilder component graphic is shown in Figure 1.4, following the logic depicted in Figure 1.5.

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:

$$Loss = \lambda_1(y_{1pred} - y_1)^2 + \lambda_2(y_{2pred} - y_2)^2 \quad (1.1)$$

The computing logic of each Neural 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 1.6

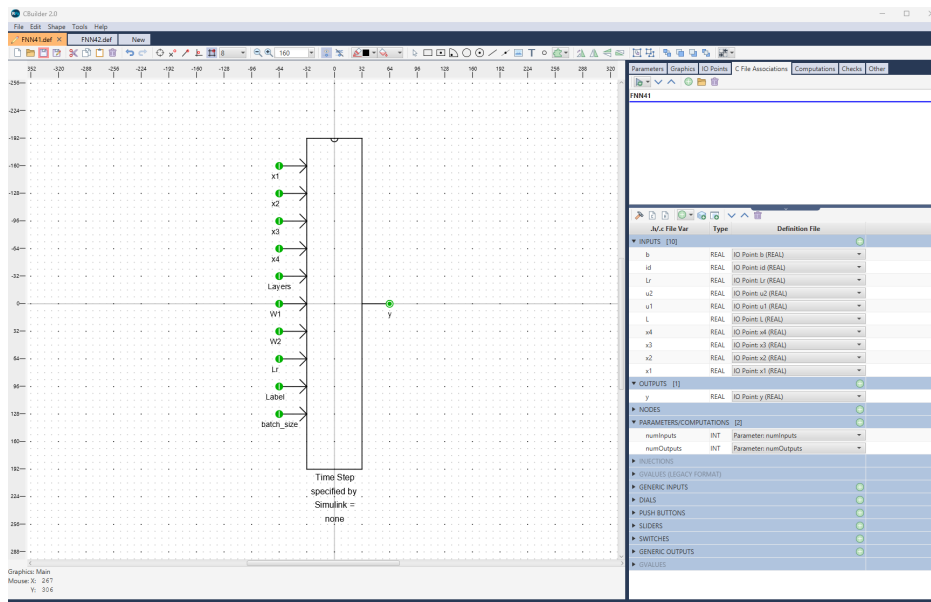


Figure 1.4: CBuilder interface depicting the graphics and I/O points of the component for a 4 feature 2 output 2 hidden layer Neural Network

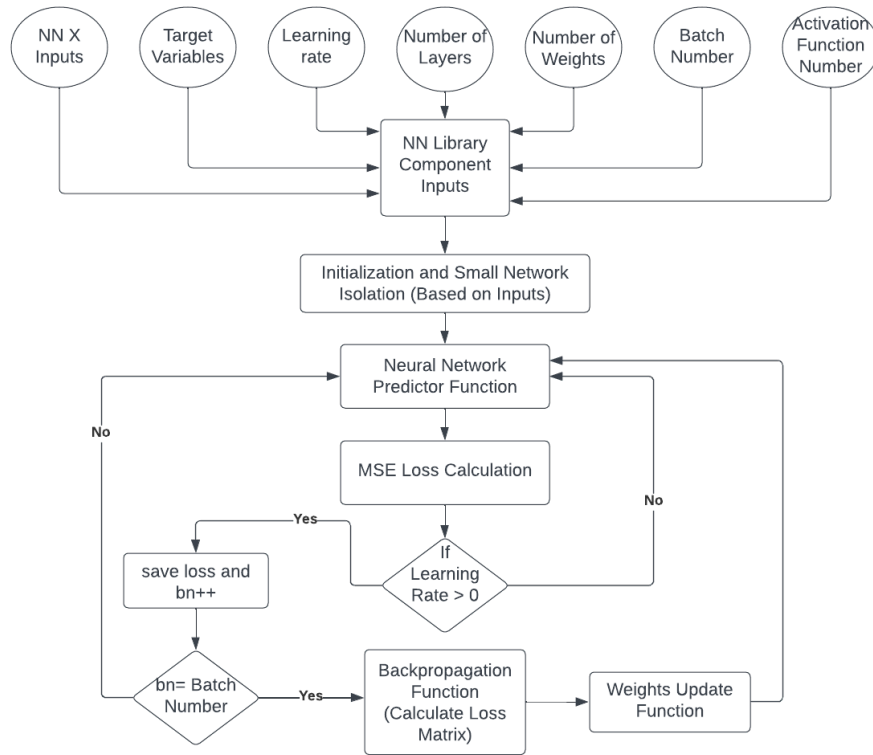


Figure 1.5: Flow Logic of Neural Network Component in RSCAD

The parameters in Figure 1.6 are:

- **numInputs:** Number of features the NN tries to map (1-4).

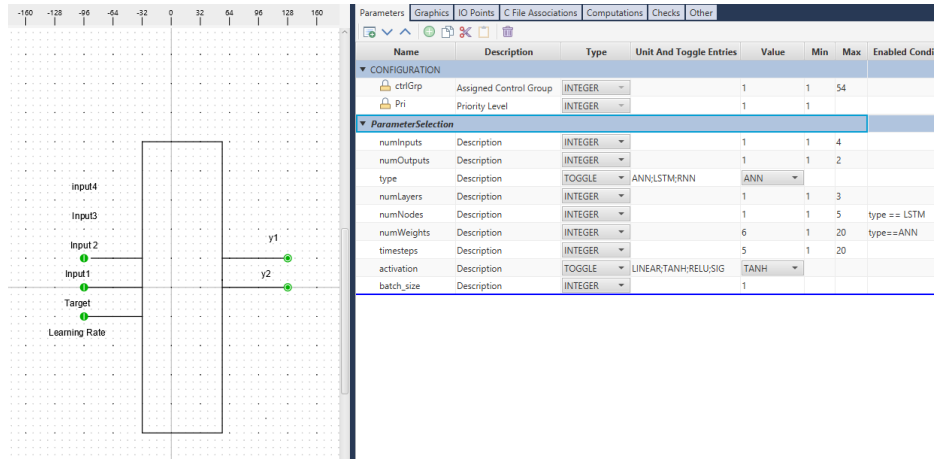


Figure 1.6: CBuilder interface depicting the parameters required for initializing the component in RSCAD. Missing nodes appear when numInputs is adjusted

- **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).
- **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 backpropagation 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 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

1.4 CBuilder Setup of Existing Component

To be able to run the existing component, it is important to place component files at the proper location. Namely, you should follow the steps:

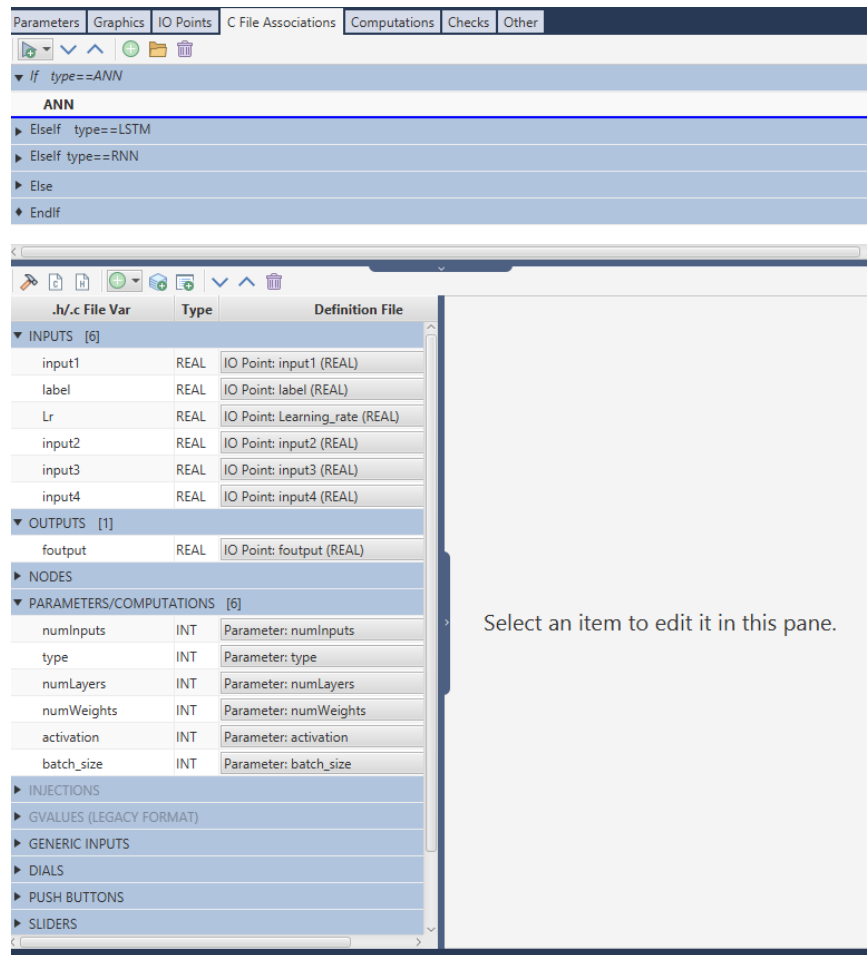


Figure 1.7: C association tab

1. Put all the c-files [.c, .h,] except the def files in `.\RSCAD\RTDS_USER_FX\BIN\CMODEL_SOURCE` (.h file simply contains definitions and outputs, e .c file should include the functionality of the component). [e.g., `C:\Users\alekic\Documents\RSCAD\RTDS_USER_FX\BIN\CMODEL_SOURCE`]
2. Put the def files in `.\RSCAD\RTDS_USER_FX\ULIB\COMPONENTS` folder [e.g., `C:\Users\alekic\Documents\RSCAD\RTDS_USER_FX\ULIB\COMPONENTS`]
3. In the toolbar: Go to **Launch>Component Builder** as seen from Fig. 1.8 and then in the CBuilder open window select **Open From Disk** as seen from Fig. 1.9 and select .def file from `.\RSCAD\RTDS_USER_FX\ULIB\COMPONENTS` folder.
4. The opened component looks like in Fig. 1.10.

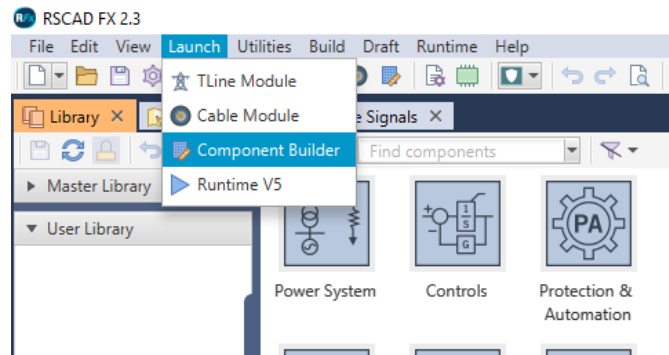


Figure 1.8: Component Builder launch.

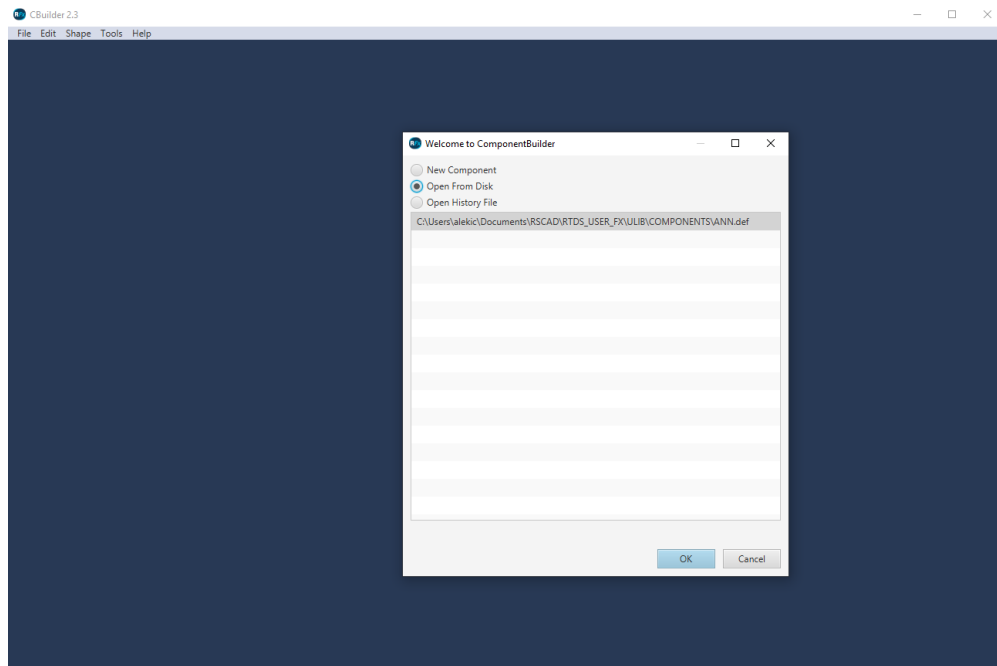


Figure 1.9: Open component.

5. Open C File Associations tab and select to open the folder, where you should search for your .c and .h files placed in .\RSCAD\RTDS_USER_FX\BIN\CMODEL_SOURCE folder. Then, compile the codes as depicted in Fig. 1.11.
6. Close the component window and then open Draft. On the Draft press mouse right-click and select Add>Component>User Library and add that .def as a component in the draft (see Fig. 1.12). That should open it there.

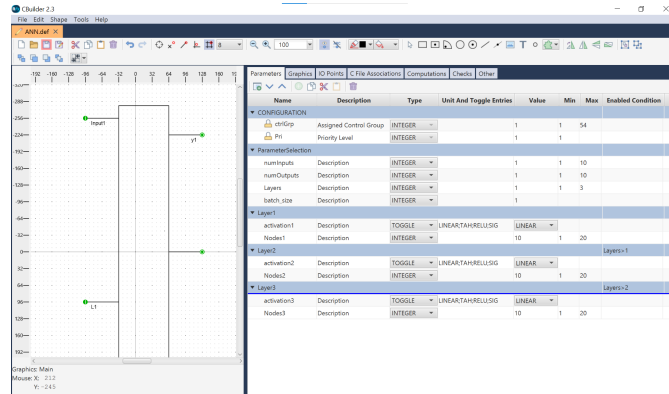


Figure 1.10: Component opened from .def file.

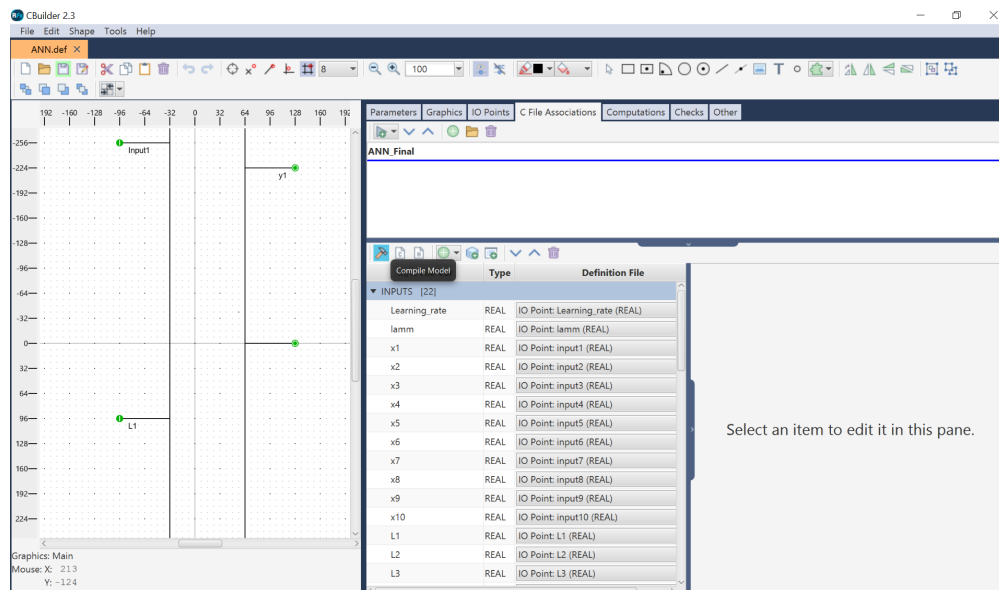


Figure 1.11: Compile model in RSCAD.

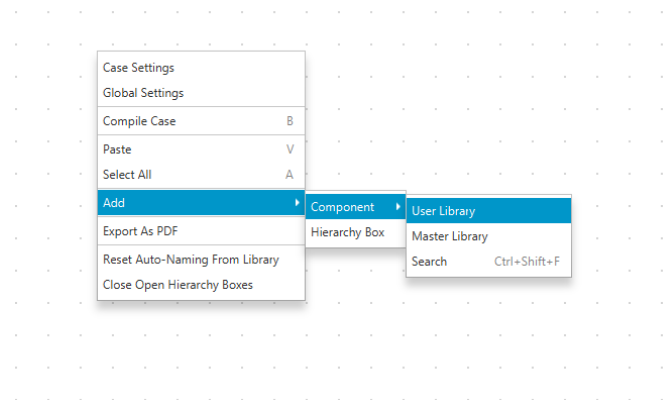


Figure 1.12: Add compiled component.

Chapter 2

NN Component Library

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".

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.

2.1 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 (once compiled as in the previous chapter) as follows in Fig. 2.1.

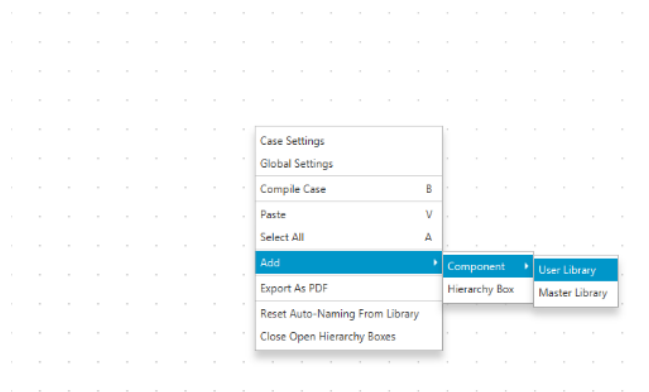


Figure 2.1: Add .def file to RSCAD draft file.

Then, the option ANN.def should appear, once it is loaded you expect the following, see Fig. 2.2.

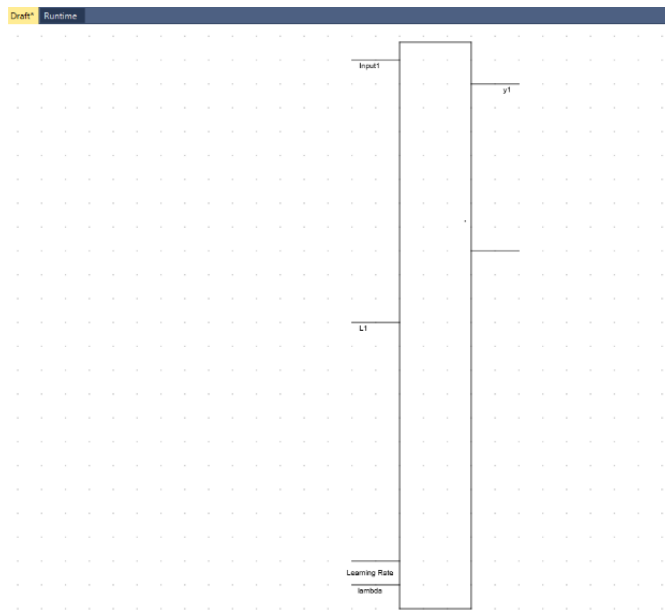


Figure 2.2: ANN in draft.

By double-clicking this component you should get the menu from Fig. 2.3.

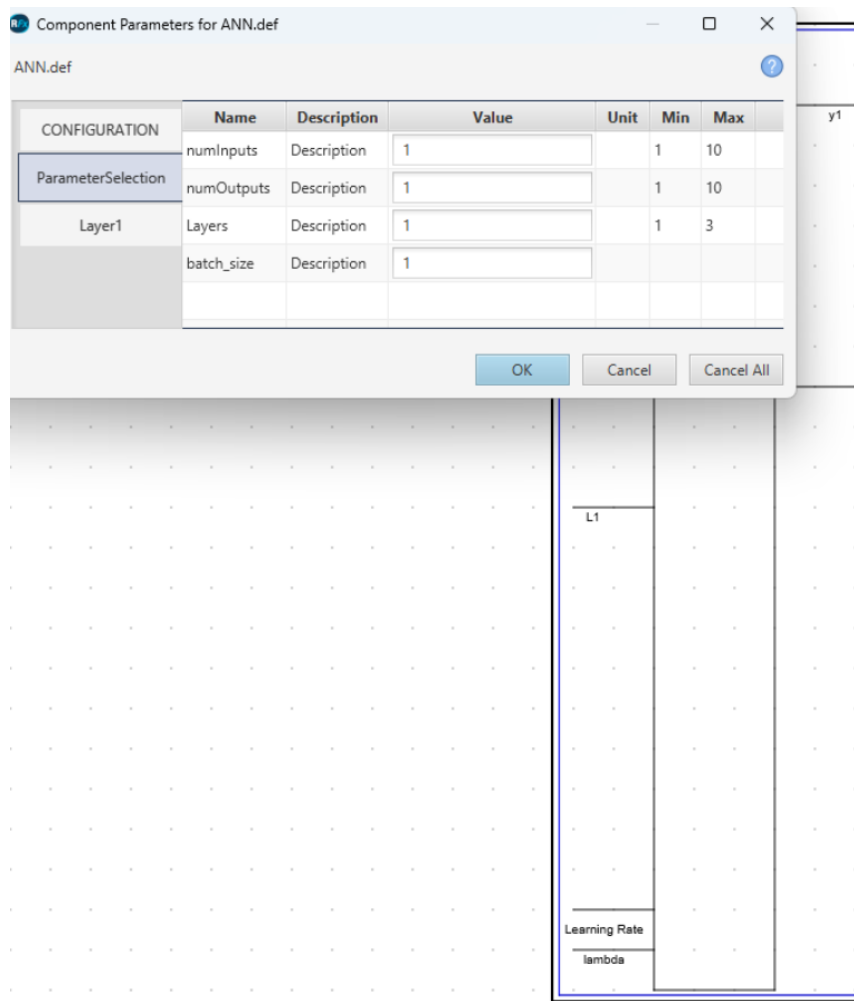


Figure 2.3: ANN parameters.

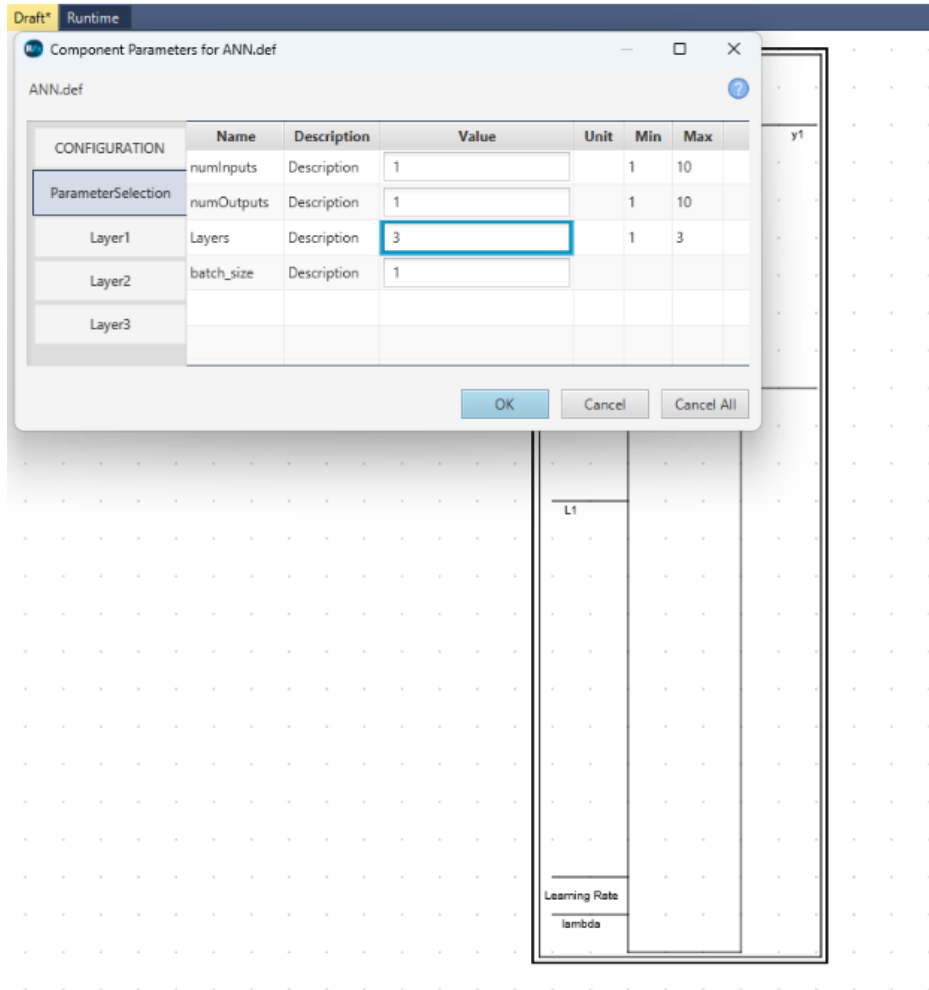
Here, in this menu, we define the initial parameters of the Neural Network which are:

- **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. However, by increasing the number of layers more tabs will show up as in Fig. 2.4.

In Fig. 2.4 we can see that tabs for layers 1, 2, and 3, which is the maximum number implemented in this component. For each layer, 2 parameters can be varied as seen in Fig. 2.5:

- **Activation:** The activation function of the layer which can be RELU, TAH - tangent hyperbolic, LINEAR, and SIG - sigmoid;
- **Nodes:** how many nodes the layer can have (20 maximum).



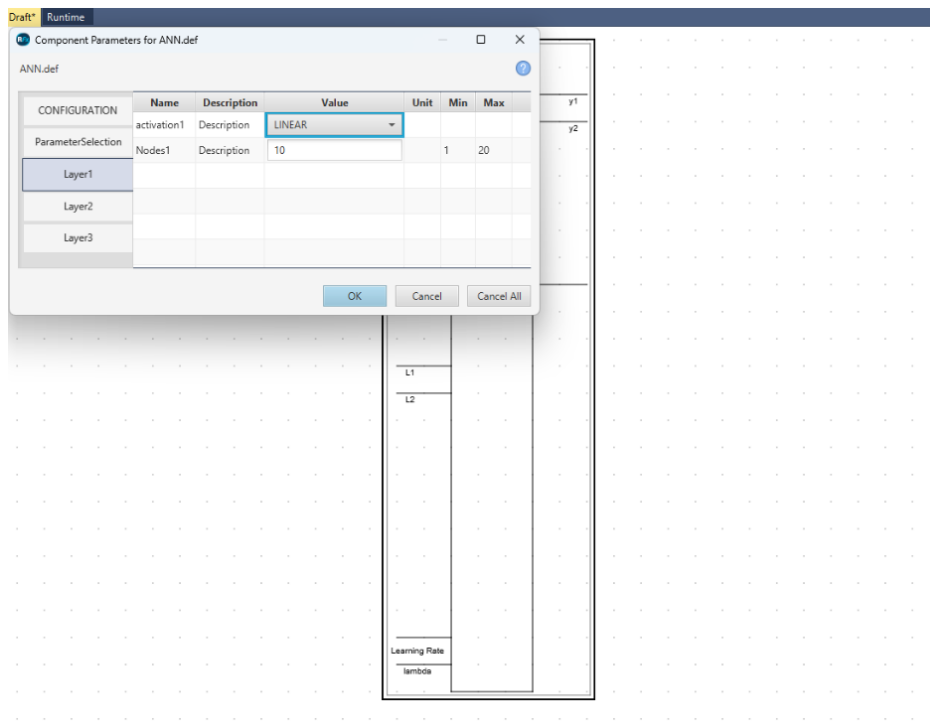


Figure 2.5: ANN - parameters that can be selected for each layer.

Once all the parameters are finalized the NN component should look like in Fig. 2.6.

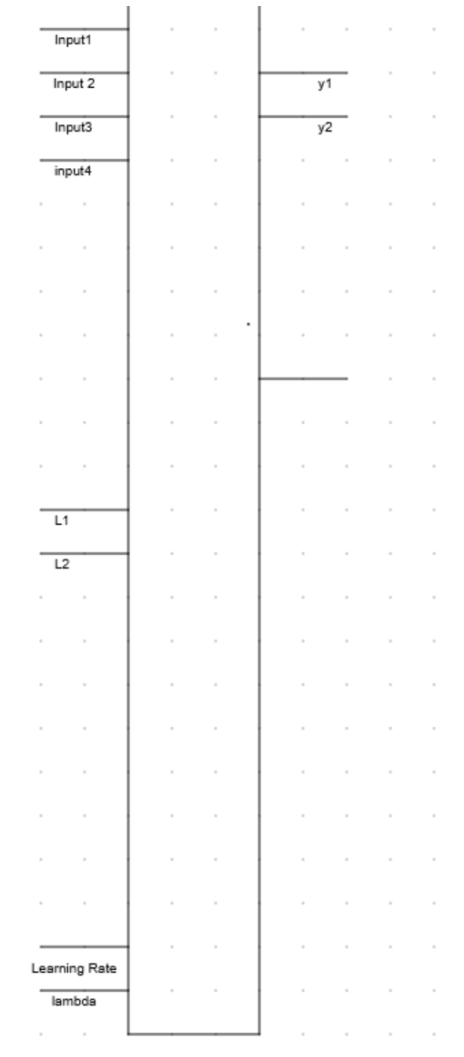


Figure 2.6: Complete ANN.

In this architecture, we chose 4 inputs and 2 outputs. However, the number of layers is irrelevant in its appearance in the draft. 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 outputs.
- **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.

2.2 LSTM

The LSTM repository is very similar to the previously discussed ANN repository. Once the component is loaded it looks like in Fig. 2.7.

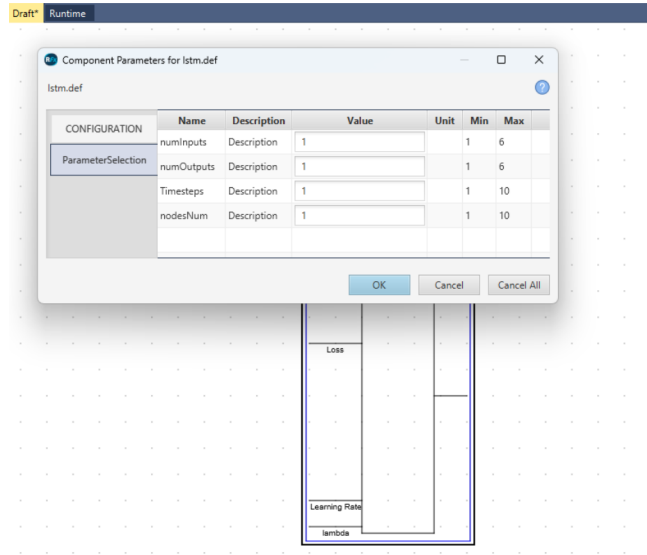


Figure 2.7: LSTM component.

Here the parameters are as follows:

- **numInputs:** How many inputs go into the Neural Network;
- **numOutputs:** How many outputs are expected from the NN;
- **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.

Chapter 3

Layered Components

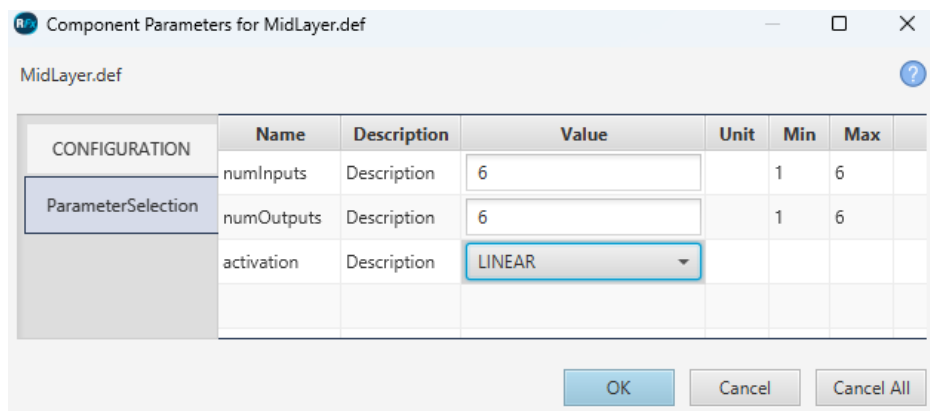
The components in the library can be divided into two, Full Neural Networks and Single Layers. Full Neural Networks are components that include the computation of all layers involved, while single-layer blocks are used to design Networks one layer at a time.

3.1 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 (see Fig. 3.1), each component has 3 parameters to be selected before compiling: `numInputs`, `numOutputs`, and `activation`.



Component Parameters for MidLayer.def

MidLayer.def

CONFIGURATION	Name	Description	Value	Unit	Min	Max
ParameterSelection	numInputs	Description	6		1	6
	numOutputs	Description	6		1	6
	activation	Description	LINEAR			

OK Cancel Cancel All

Figure 3.1: ANN layered component.

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

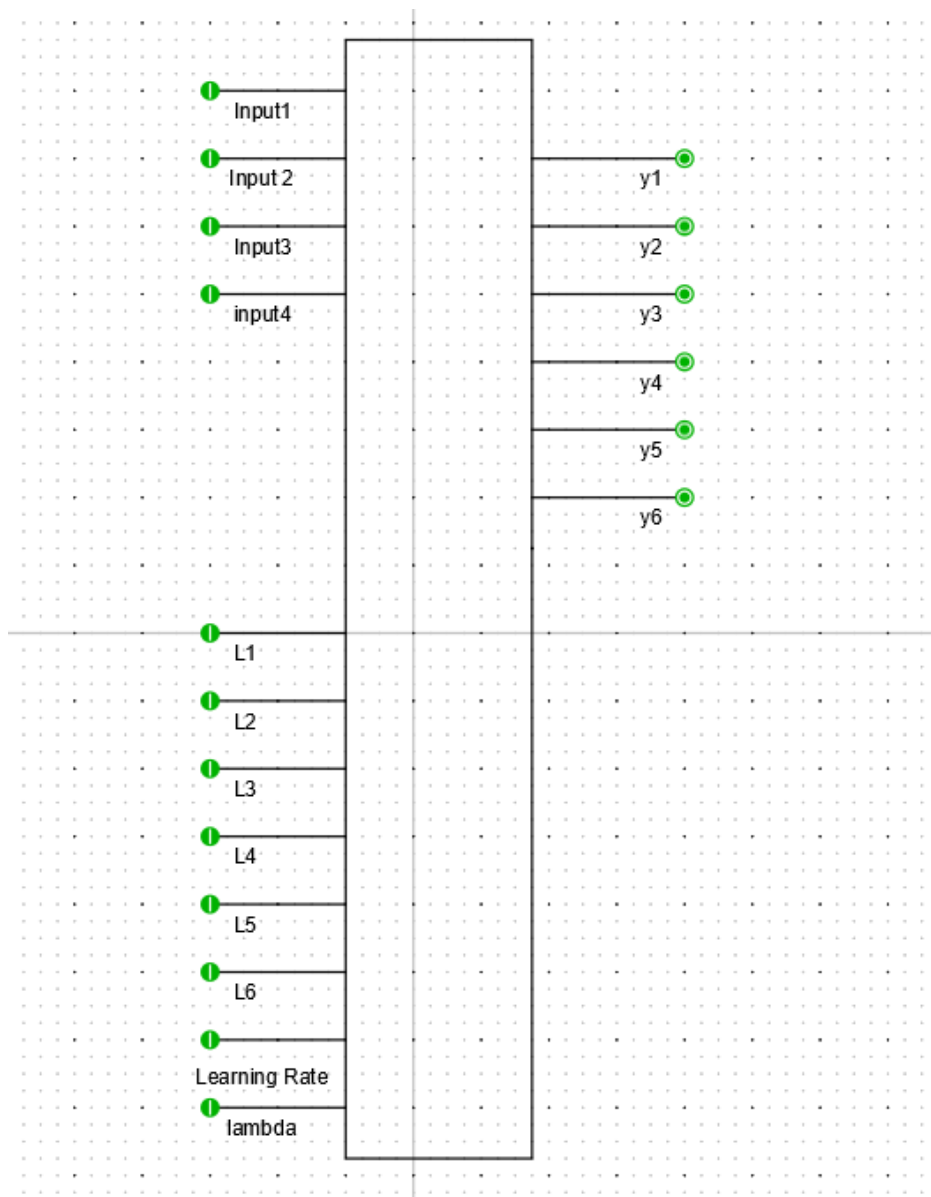


Figure 3.2

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.

Hidden Layer

The hidden layer, see Fig. 3.3, takes in the following inputs:

- **Input1 - Input6:** The inputs into the layer are 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.

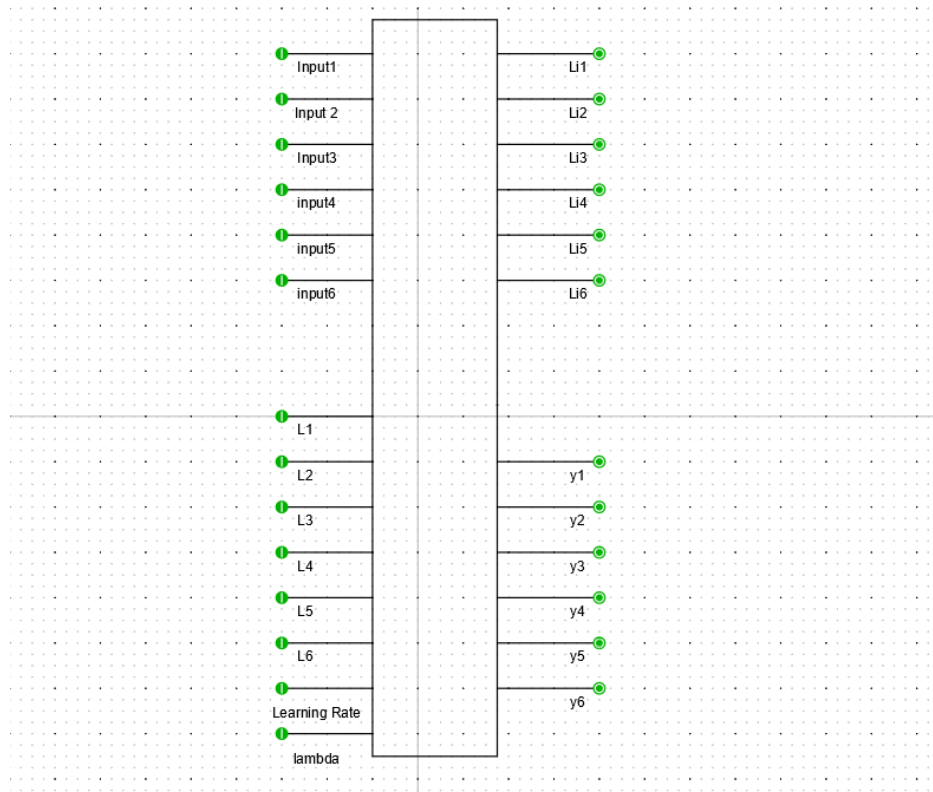


Figure 3.3: Hidden layer.

Output Layer

The output layer (see Fig. 3.4) 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 backward 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.

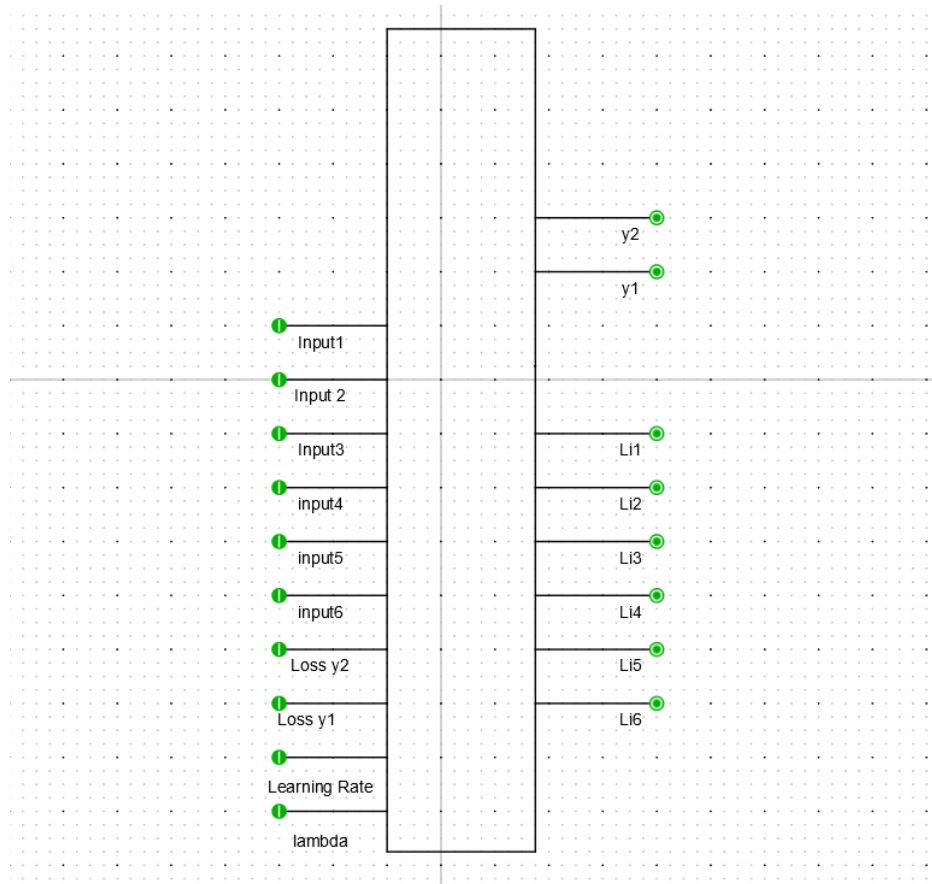


Figure 3.4: Output layer.

3.1.1 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:

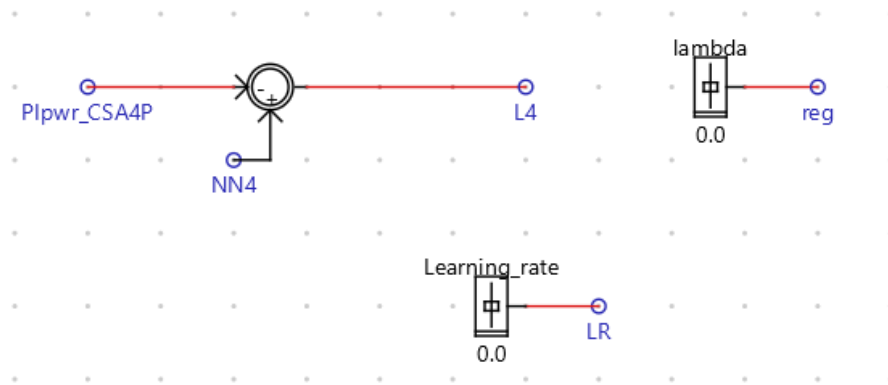


Figure 3.5

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

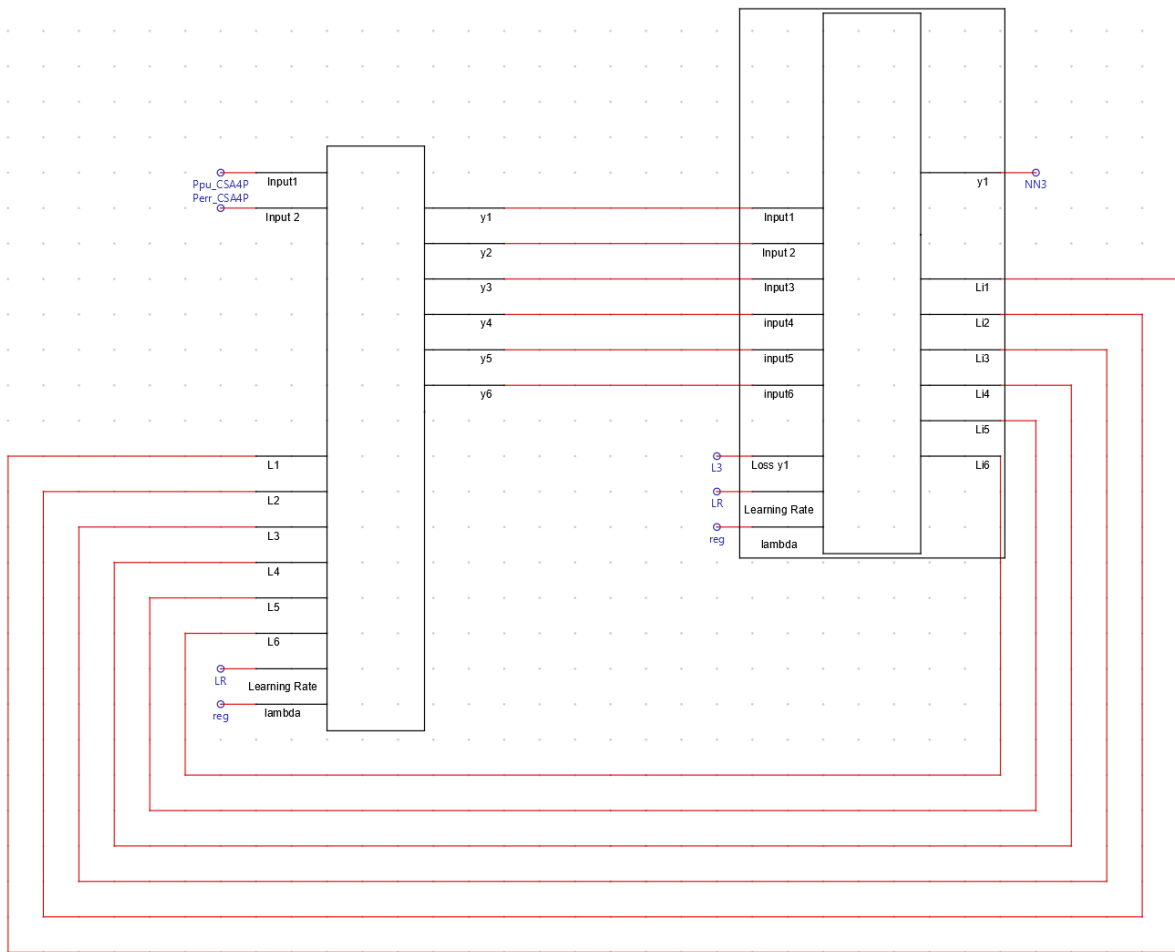


Figure 3.6

3.1.2 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.

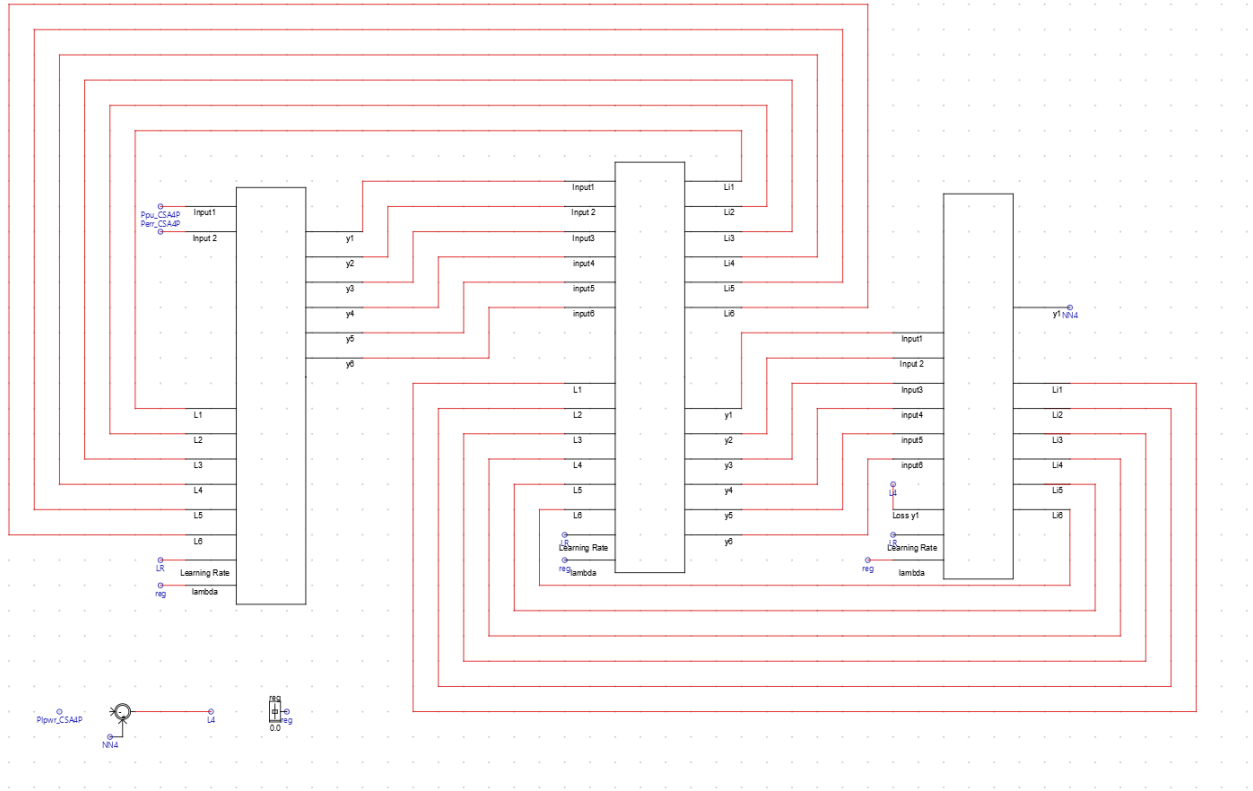


Figure 3.7

3.1.3 Training and Regularization

Since there is no learning rate optimizer, the rate should be adjusted according to the behavior 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 ($0.0001 - 0.01$) 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 $10^{-9} - 10^{-4}$, the user should adjust based on apparent results. Make sure not to keep the same input parameter range for extended duration 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 (e.g., 10^{-8}) and adjust accordingly. Setting lambda larger ($\approx 10^{-4}$) can lead to most of the weights dropping out, an appropriate value of lambda most likely is found in the range $10^{-8} - 10^{-4}$.

3.2 Components in Library

In the folder `\NNLib\SingleLayerComponents`, you will find files mentioned in Table 3.1. Please note that all these layers are ANNs.

Table 3.1: Single layer components

Definition file	Corresponding .c and .h files	Meaning
InputLayer10.def	InputLv2.c/.h	input layer with LINEAR activation function
	InputLv2Tanh.c/.h	input layer with TANH activation function
HiddenLayer10.def	HiddenLv2.c/.h	hidden layer with LINEAR activation function
	HiddenLv2TAN.c/.h	hidden layer with TANH activation function
OutputLv2.def	OutputLv2.c/.h	output layer with LINEAR activation function and dense negative
	OutputLv2LinDense.c/.h	output layer with LINEAR activation function and dense applied
	OutputLv2SigDense.c/.h	output layer with SIGMOID activation function and dense applied
	OutputLv2TanhDense.c/.h	output layer with SIGMOID activation function and dense applied

To use all layers combined, it is necessary to add all input, hidden, and output layer definitions and to compile all of their associated .c/.h files as seen from Fig. 3.8.

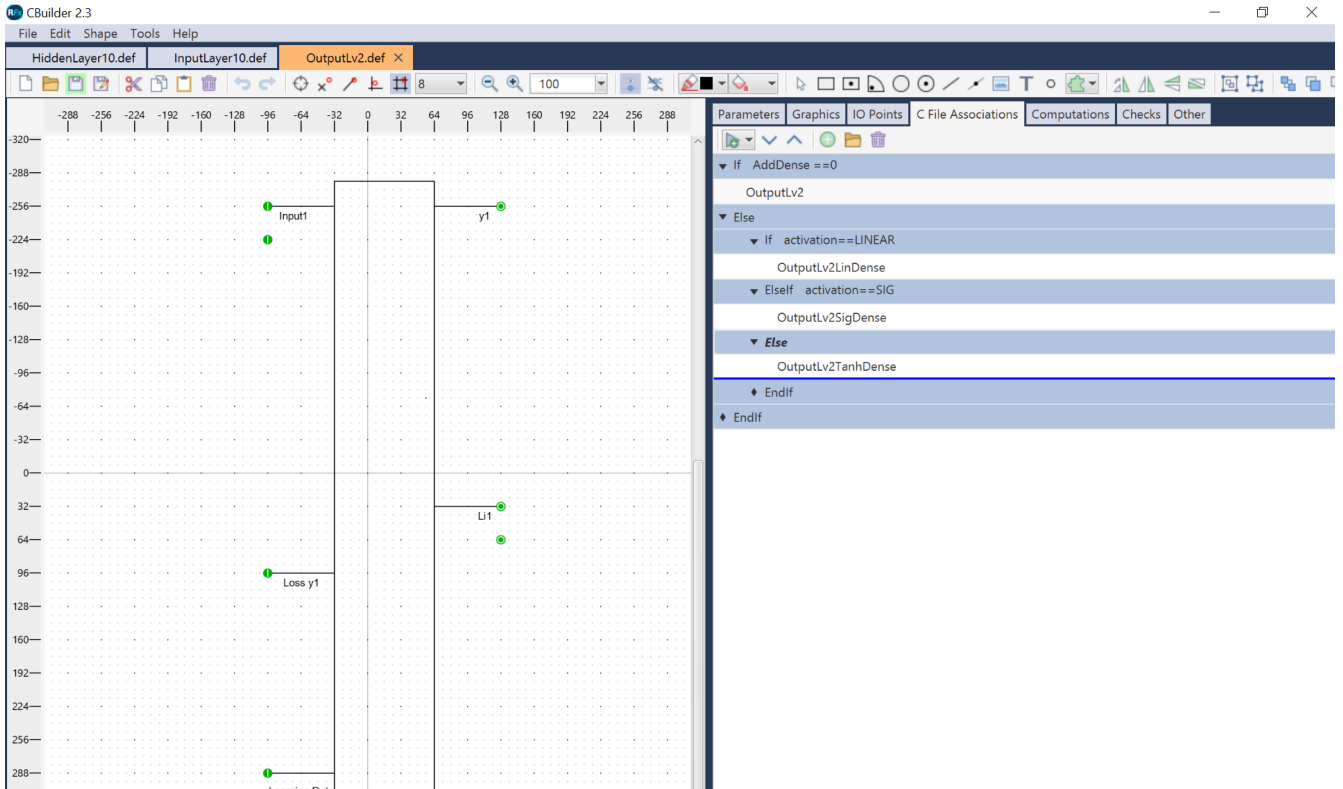


Figure 3.8: Compile all layers.

To use controls as described in the associated paper, it is necessary to compile all of these layers and place them in the desired draft file (e.g., simulated HVDC networks) in the way depicted in Fig. 3.9. Namely, first, drag the ‘Hierarchy Box’ component in the desired draft and then add all three definitions (for input, hidden, and output layer). Like this, you will be able to use these layers for your designs.

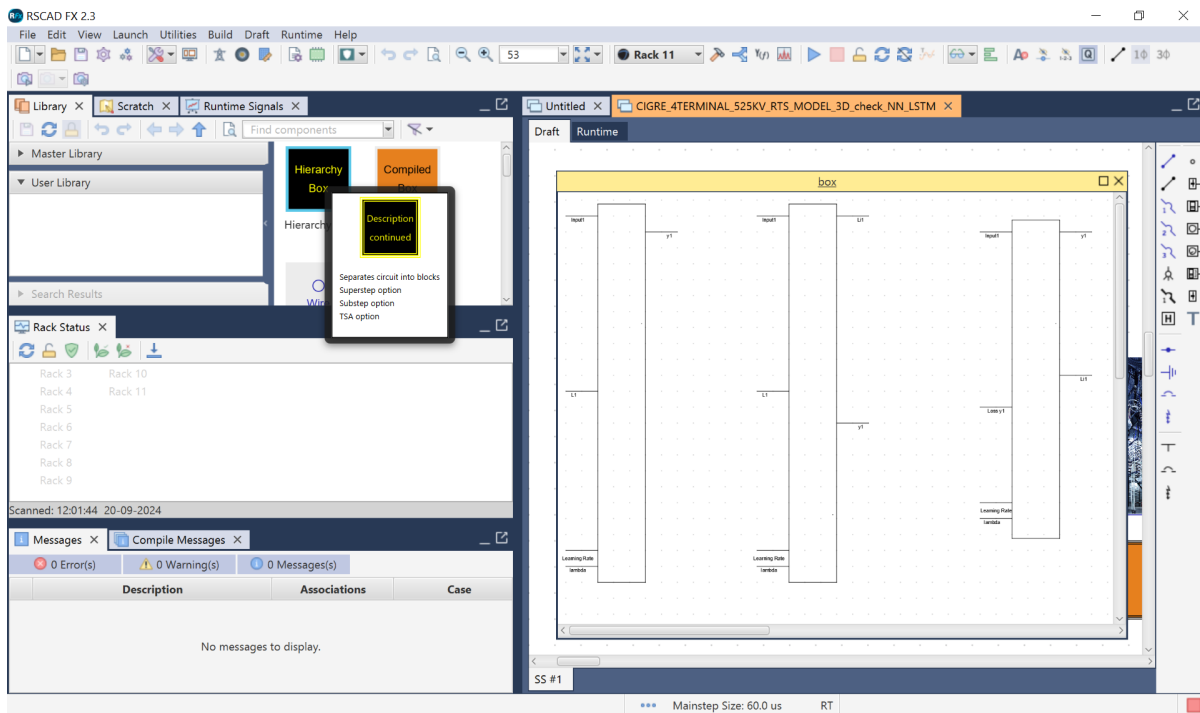


Figure 3.9: Add all layers in hierarchy box.

Make sure to exclude this box from the load flow calculation and from the circuit if you do not attach them directly. You do this by right-clicking on the component and then **Edit>Parameters**, and selecting fields like in Fig. 3.10 and in draft like in Fig. 3.11.

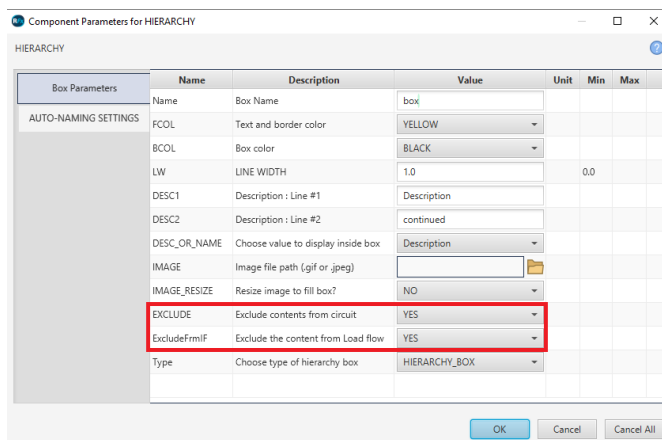


Figure 3.10: Exclude box.



Figure 3.11: Excluded box in draft.

Chapter 4

Running Existing Models

4.1 HVDC-based RSCAD Model Control

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

- Step 1: Download the files from the git directory and copy RSCAD files to the fileman: \RSCAD\RTDS_USER_FX\fileman.
- Step 2: Open file CIGRE_4TERMINAL_525KV_RTS_MODEL_3D_check_NN_LSTM.rtfx. Add NN layers in the hierarchy box as described in section 3.2.
- Step 3: The control is implemented using NNs for the converter CSA4, so by opening its control as seen in Fig. 4.1.

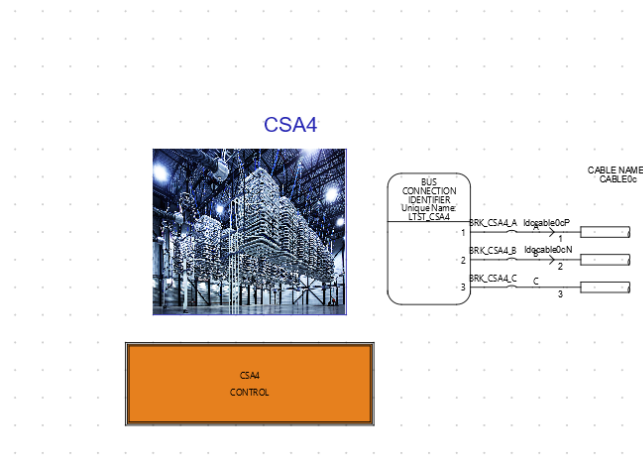


Figure 4.1: Control of CSA4.

Double click on the CSA4 Control block opens a window, which should open the CSA4P Controls block with a double click.

- Step 4: CONCSAP4P window looks like in Fig. 4.2. You can notice 4 black blocks that represent already constructed (example) controllers using NNs.

Controller models that are given with these boxes are:

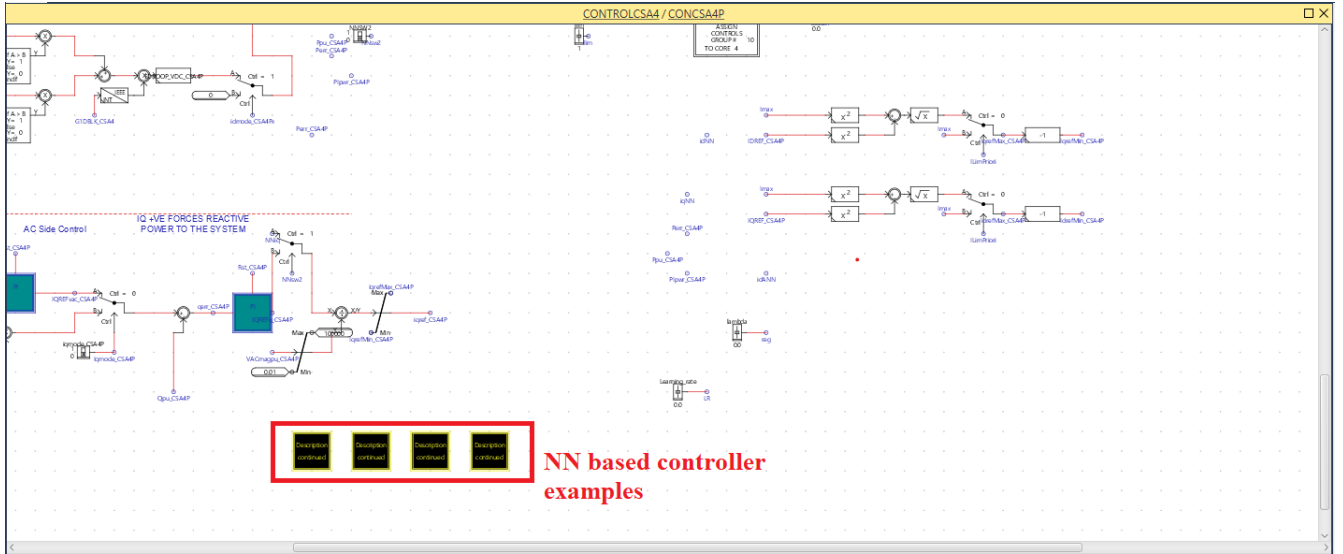


Figure 4.2: Block CSA4P.

- **Q_control, error + V.D** block controls reactive power, error and inclusion of v_D^G in NN inputs. It gives the reference $i_{q,ref}^\Delta$ as an output. The output is denoted as NNiq.
- **P_control, error** block controls active power and has an option of active power error tracking. It produces a reference for the d-current component as an output ($i_{d,ref}^\Delta$) denoted as NNid.
- **P_control, error + V.Q** block controls active power and has an option of active power error tracking and inclusion of v_Q^G in NN inputs. It produces a reference for the d-current component as an output ($i_{d,ref}^\Delta$) denoted as NNid2.
- **P,Q_control, error + V** block controls active and reactive powers. It produces references for both dq-current components as an output denoted as NNid3 and NNiq3, respectively.

Step 5: To include the desired controller, add a proper input signal to a switch to include NN output for both active and reactive power as depicted in Fig. 4.3. Please note that these controls are also present in the same file where are the controller boxes.

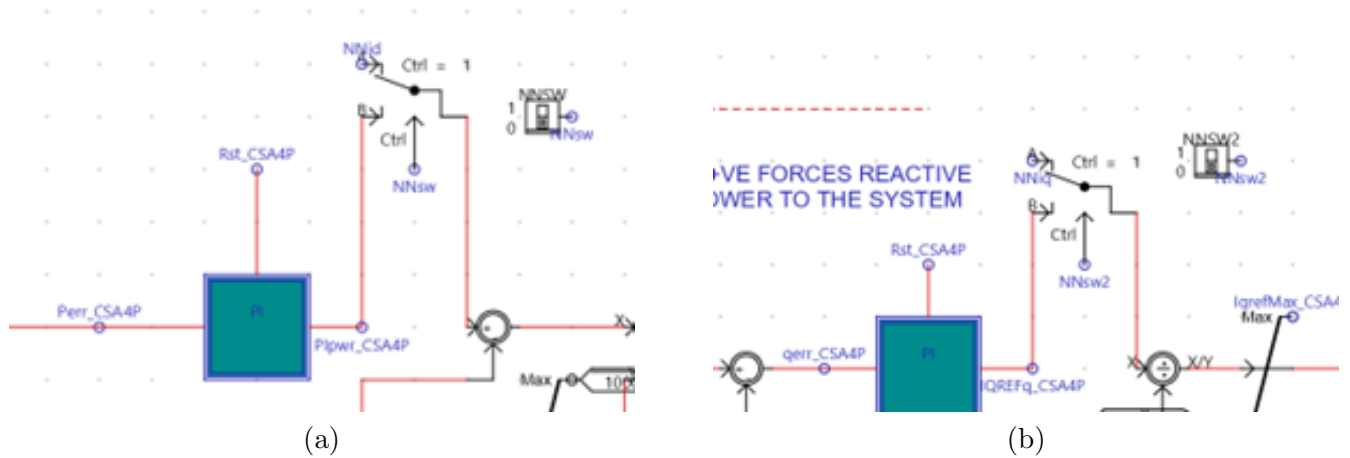


Figure 4.3: Inclusion of switches for the control of: (a) active power; and (b) reactive power.

To include different NN controllers, please make sure that you place the desired label for currents $i_{dq,ref}^\Delta$.

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 “starting_sequence_4T_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.

4.1.1 Training

For offline training, it is necessary to collect data of the RSCAD model running, which are then used for training the NN. Namely, the following steps should be respected:

1. Saving files: Get the Data from the runtime. Since the text converter in our case is CSA4, it is necessary in runtime to plot in a diagram AC current value, and its error, furthermore, active and reactive powers and their errors, and also outer voltage control PI input and output parameters. These parameters are: Ppu_CSA4P, Perr_CSA4P, PIpwr_CSA4P, Qpu_CSA4P, qerr_CSA4P, IQREFq_CSA4P, id_CSA5P, iderr_CSA4P, outpi1.t_CSA4P, iq_CSA4P, iqerr_CSA4P, outpi2.t_CSA4P. Rename the plots e.g. to ‘collection_data’ as seen in Fig. 4.4.

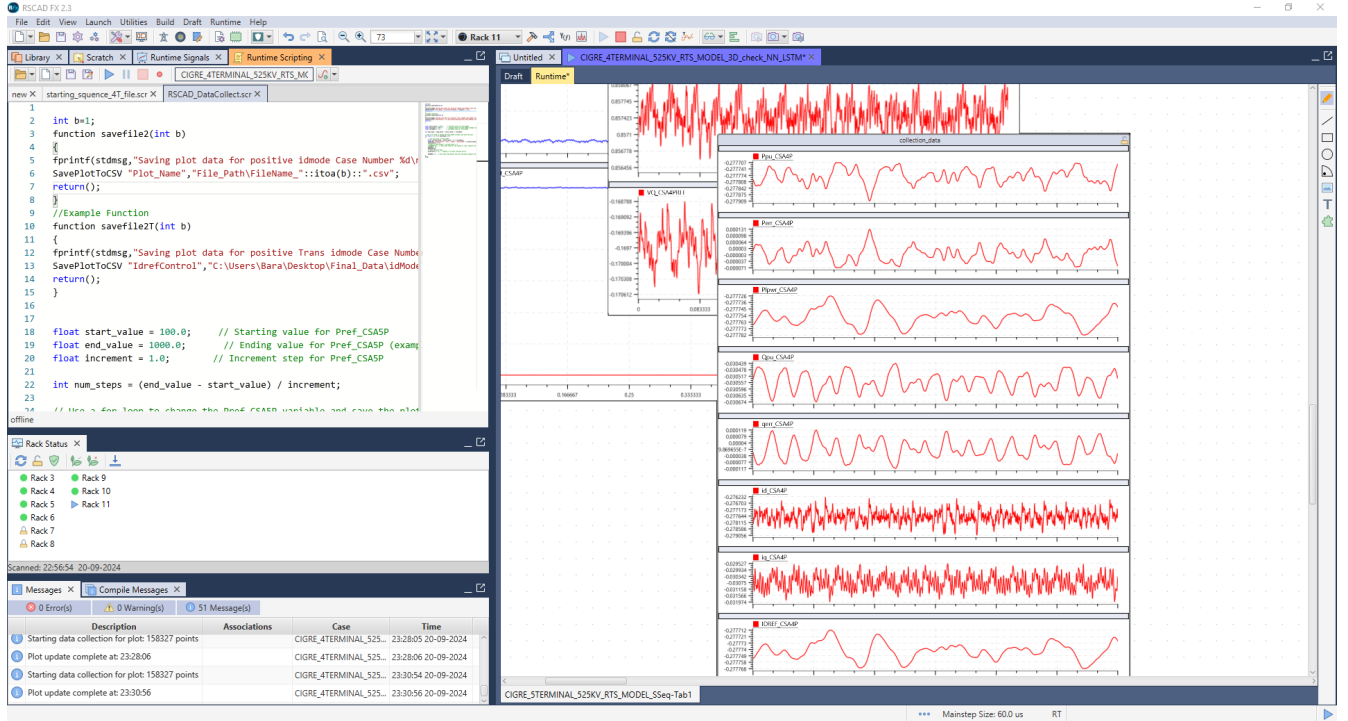


Figure 4.4: Collection of data plot.

Use the script file RSCAD_DataCollect.scr available in the NNLib folder to run multiple cases to save data by varying P and Q. Namely, you can copy the content of the script and change the necessary fields like Plot_name (in our case collection_data), add desired file path and file name.

```
int b=1;
function savefile2(int b)
```

```

{
fprintf(stdmsg,"Saving plot data %d\n",b);
SavePlotToCSV "Plot_Name","File_Path\FileName_":::itoa(b):".csv";
return();
}
//Example Function for transient
function savefile2T(int b)
{
fprintf(stdmsg,"Saving plot data  %d\n",b);
SavePlotToCSV "Plot_Name","File_Path\FileName_":::itoa(b):".csv";
return();
}

float start_value = 100.0; // Starting value for Pref
float end_value = 1000.0; // Ending value for Pref (example value, adjust as needed)
float increment = 1.0; // Increment step for Pref

int num_steps = (end_value - start_value) / increment;

// Use a for loop to change the Pref_CSA5P variable and save the plot
for (int i = 0; i <= num_steps; i++)
{
    // Set the value of Pref_CSA5P
    float current_value = start_value + (i * increment);
    SetSlider "Subsystem #1 : CTLs : Inputs : Pref_CSA5P" = current_value;
    savefile2T(i + 1);
    // Give some time for the system to react (adjust the time as needed)
    SUSPEND 5.0;
    UpdatePlots;
    // Save the plot data
    savefile2(i + 1); // Adding 1 to avoid starting from 0

    // Give some time before the next iteration (adjust the time as needed)
    SUSPEND 0.1;
}
Stop;

```

2. The code to be run can be like in Fig. 4.5. The script generates a number of files in the desired directory. Please note that before calling this data collection script, it is necessary to initialize the system by first calling `starting_sequence_4T_file.scr`. Only then you can start with running the data collection as in Fig. 4.5.
3. Generation of weights and updating is being done based on collected data using the `trainNN.py` file available in the NNLib folder in GitHub.

There are lines of code that need to be changed to fit the simulated test case. Namely, in code line 63 it is necessary to add the location of the .csv data collections. Furthermore, in line 130 the string array `all_c` should contain the key values that were plotted and their names and it should be corrected based on it (it is sufficient to look at the first line with names in the generated .csv files from RSCAD).

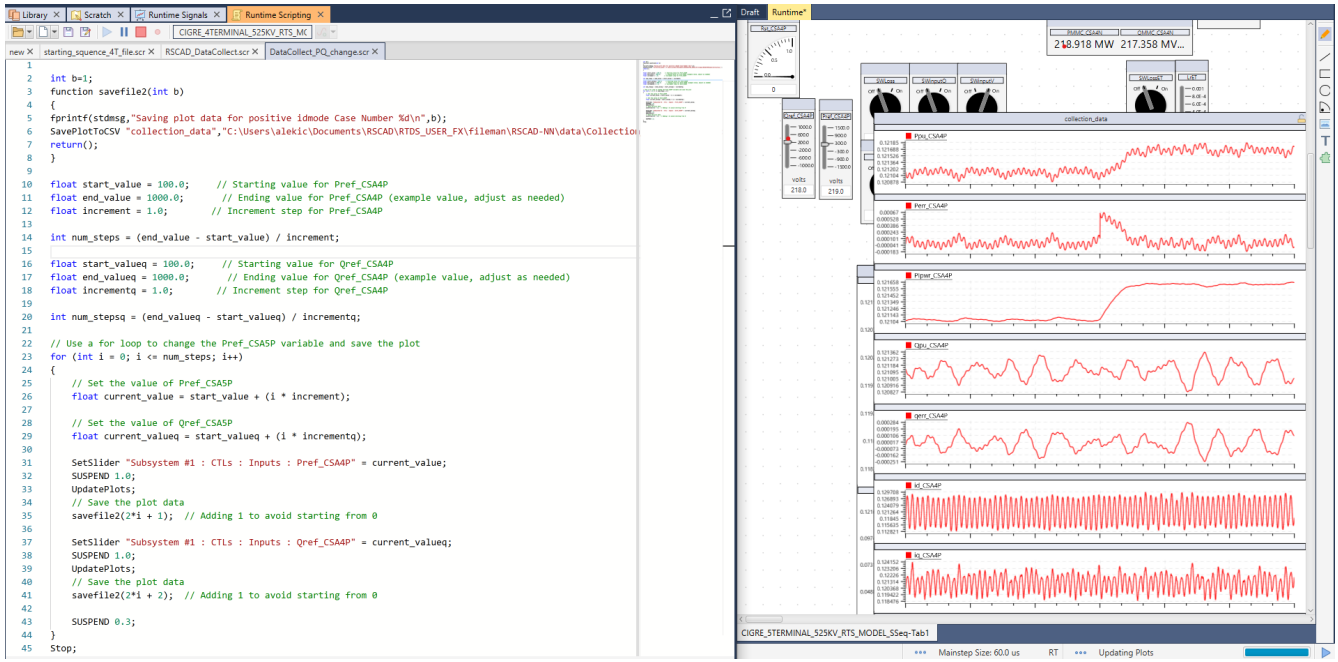


Figure 4.5: Running script.

In the file `trainNN.py` are given examples for training NN of type ANN with multiple layers. At the end of the training, it is possible to print weights and biases and these values can be directly pasted to the `.c` file of each layer.

Similarly for LSTM, also the values can be pasted inside the `.c` file of the RSCAD LSTM definition.