RSCAD Neural Network Library

April 2024

THIS IS STILL MISSING ALOT 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 fine 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

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.

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:

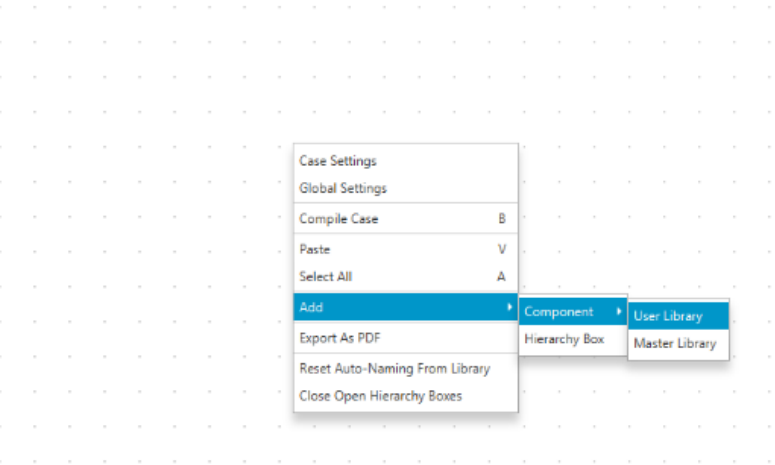

Figure 1.1

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

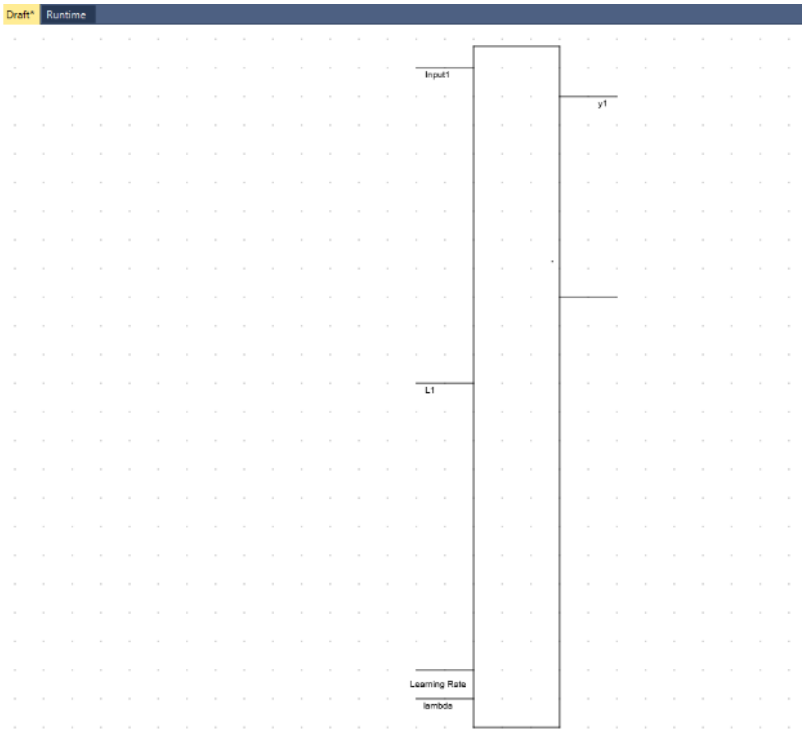

Figure 1.2

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

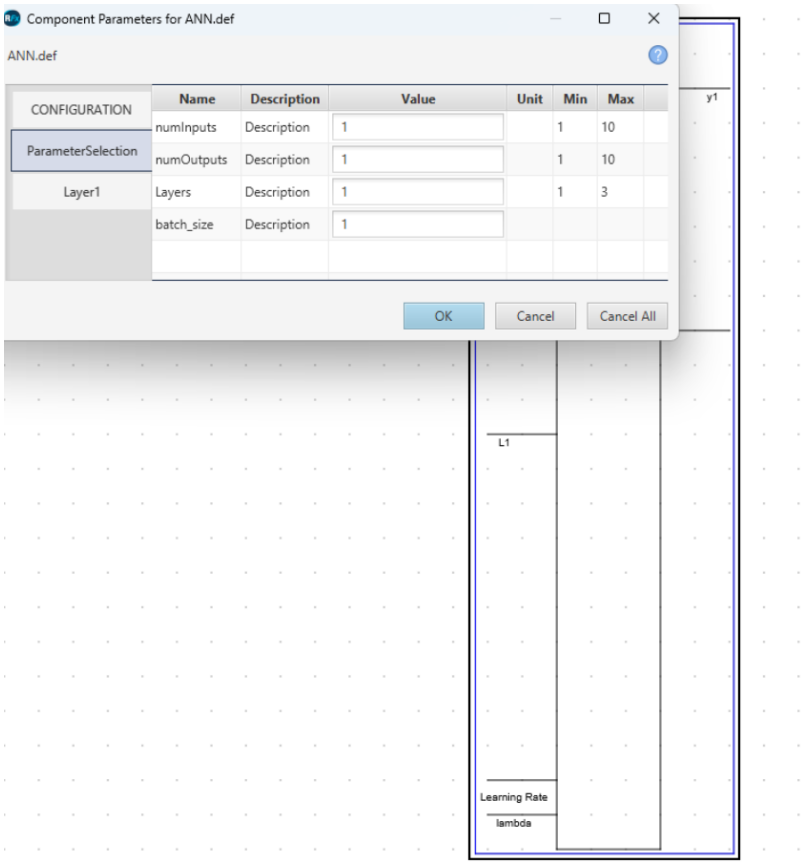

Figure 1.3

here 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, by increasing the number of layers more tabs will show up:

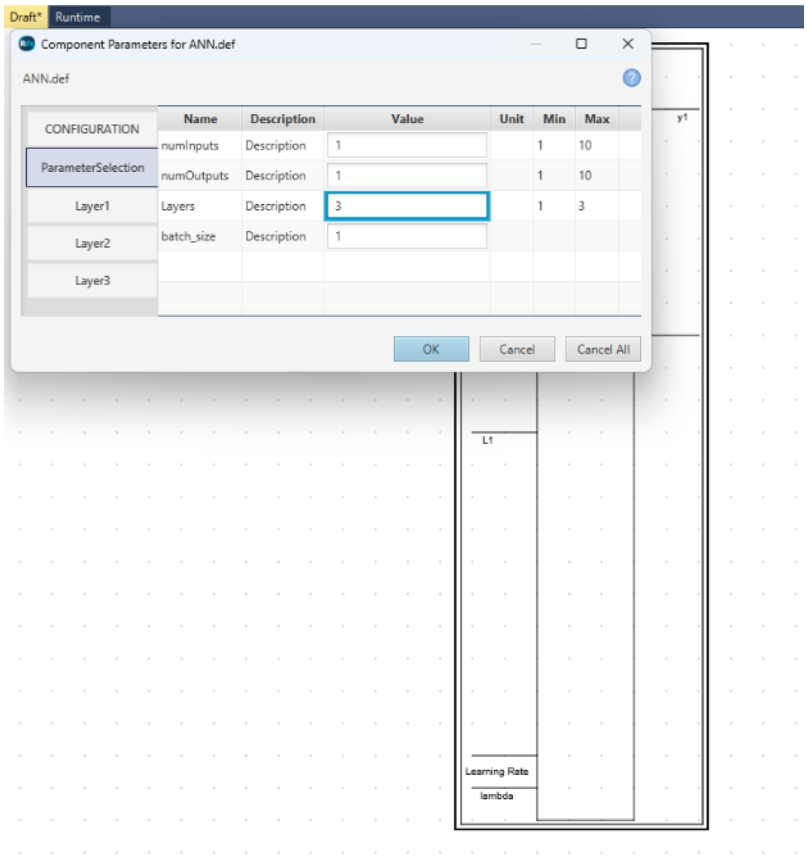

Figure 1.4

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:

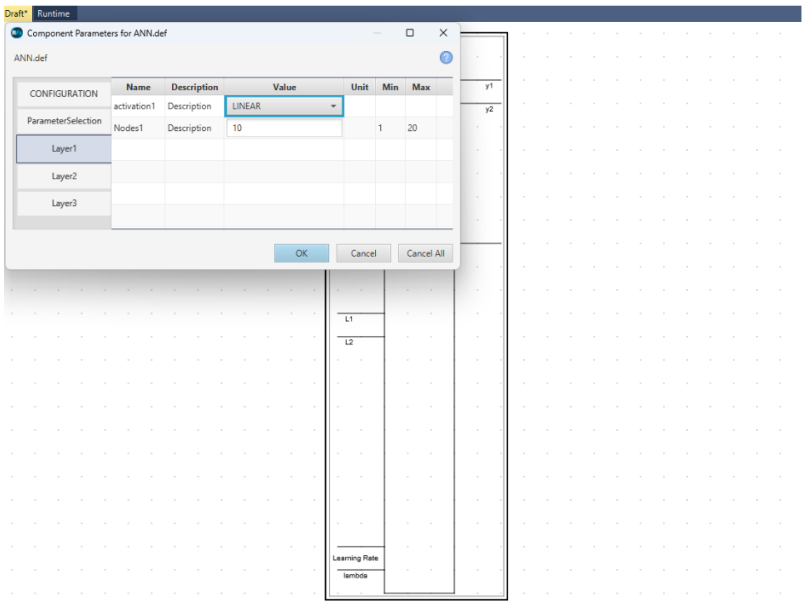

Figure 1.5

- 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:

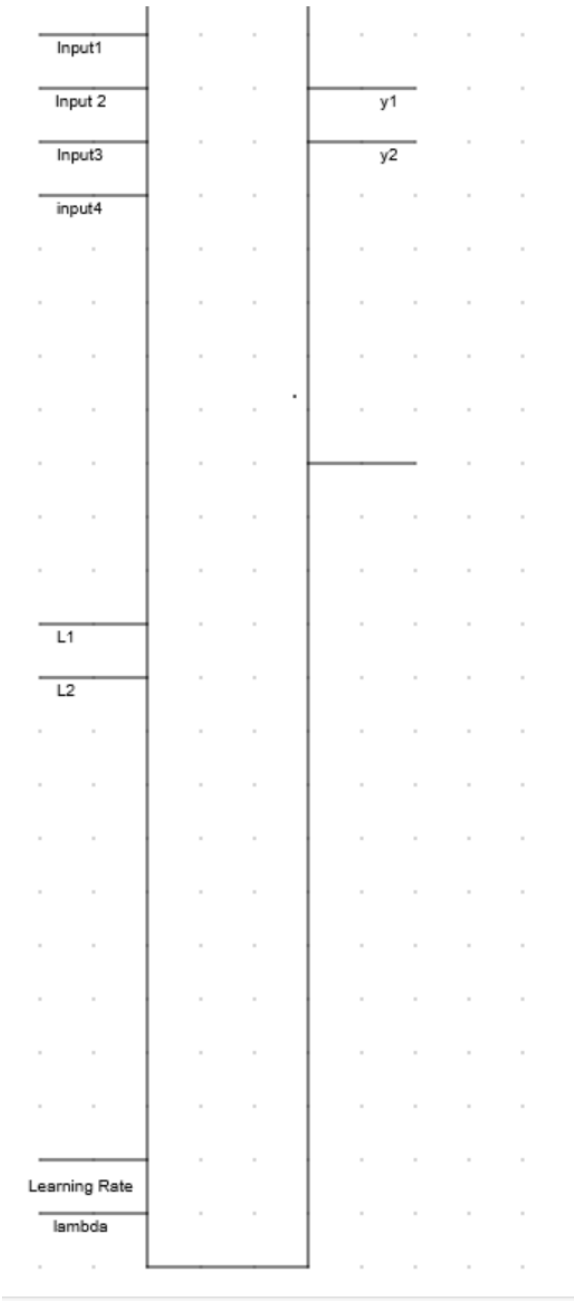

Figure 1.6

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.

1.2 LSTM

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

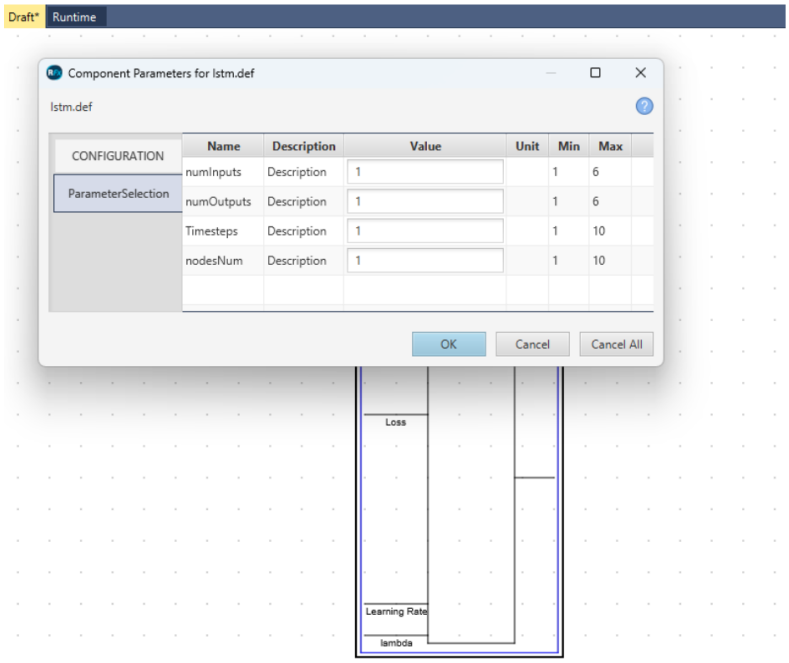

Figure 1.7

Here the parameters are as follows:

- 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.

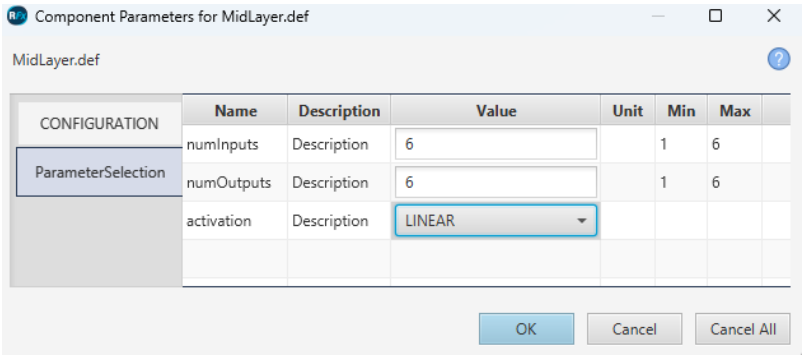

Figure 2.1

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.

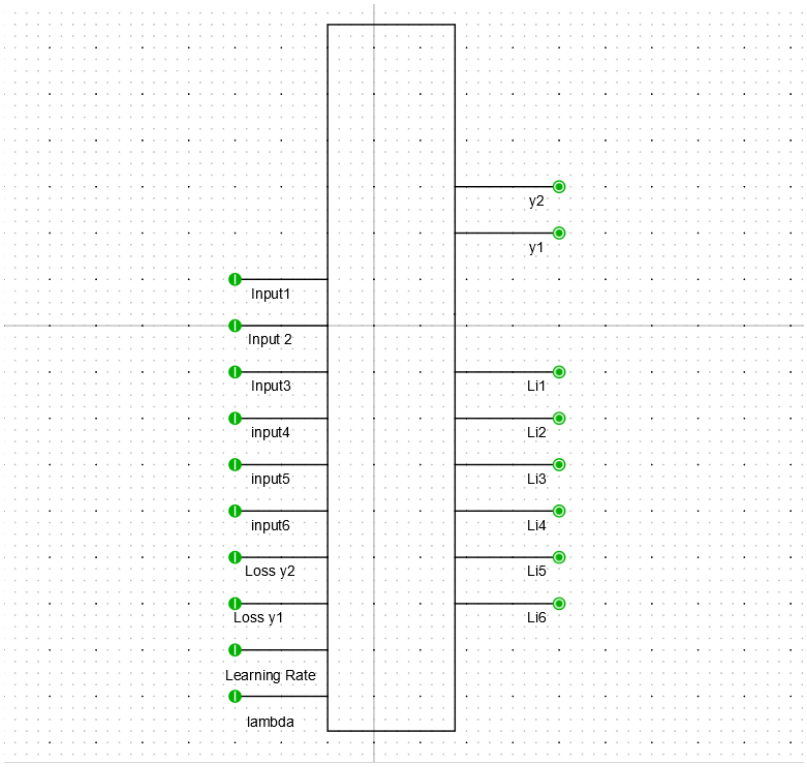

Figure 2.2

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.

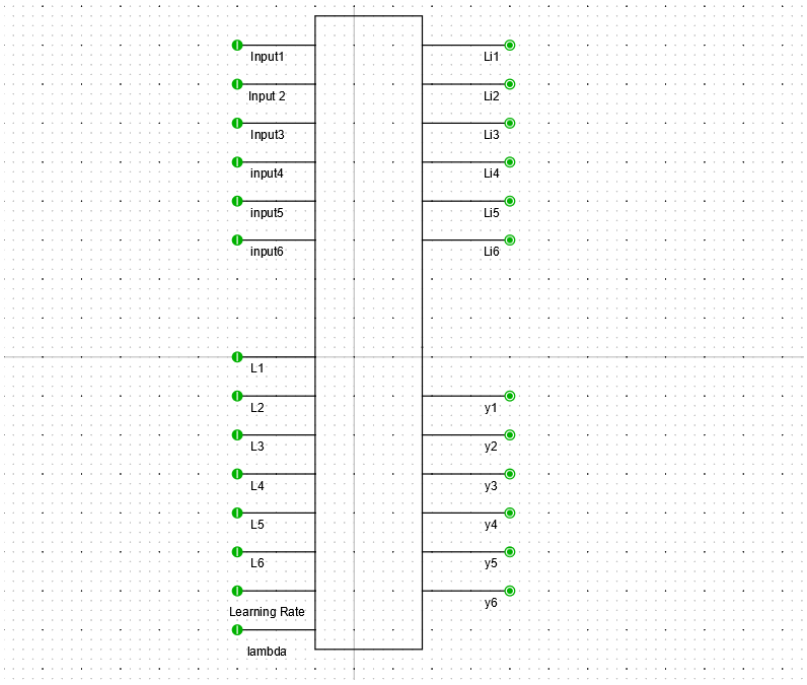

Figure 2.3

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

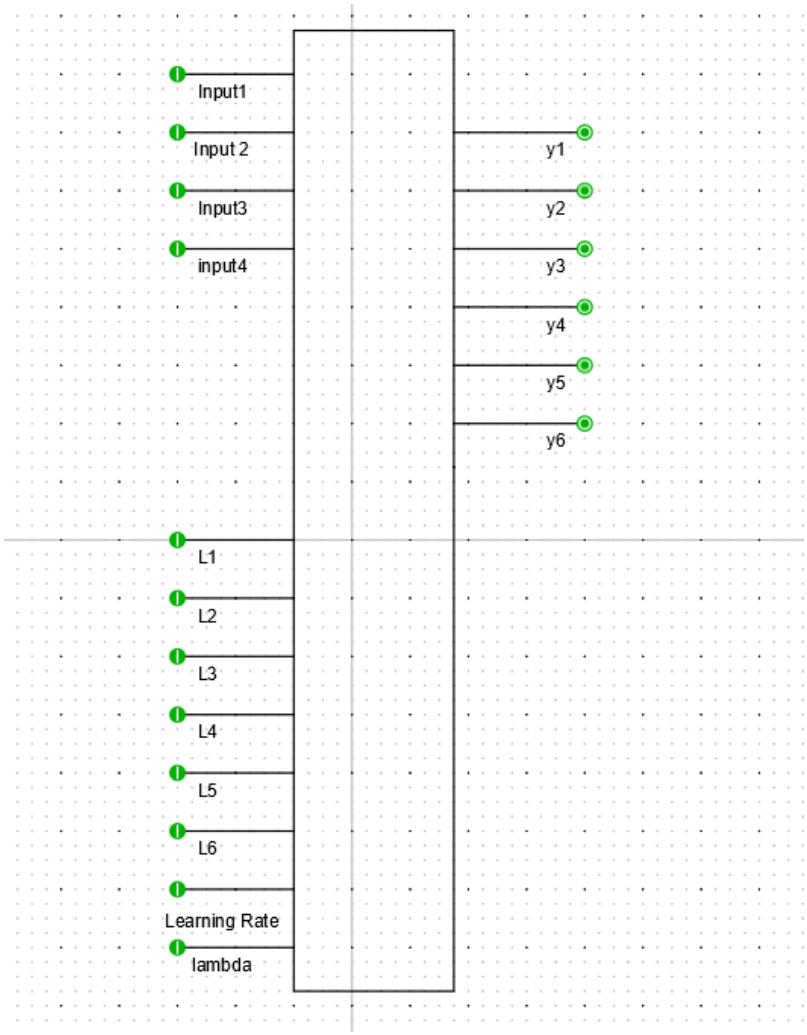

Figure 2.4

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:

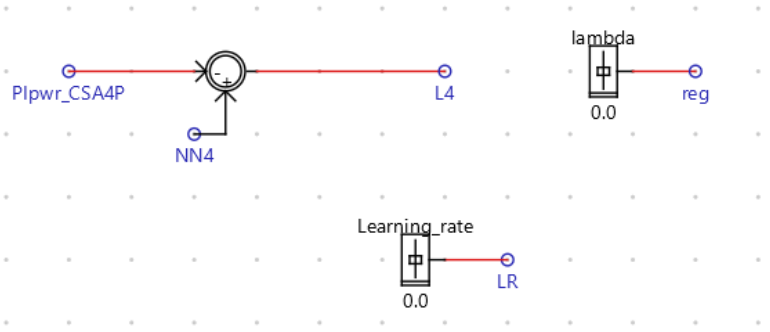

Figure 2.5

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

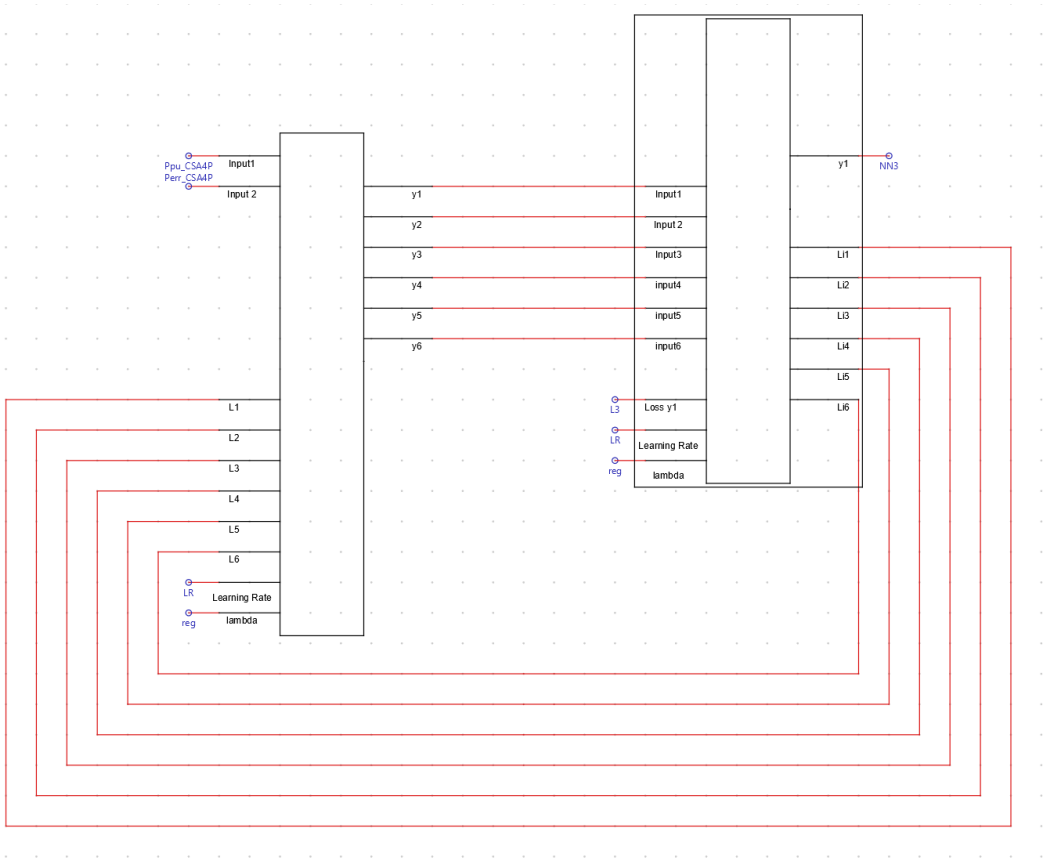

Figure 2.6

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.

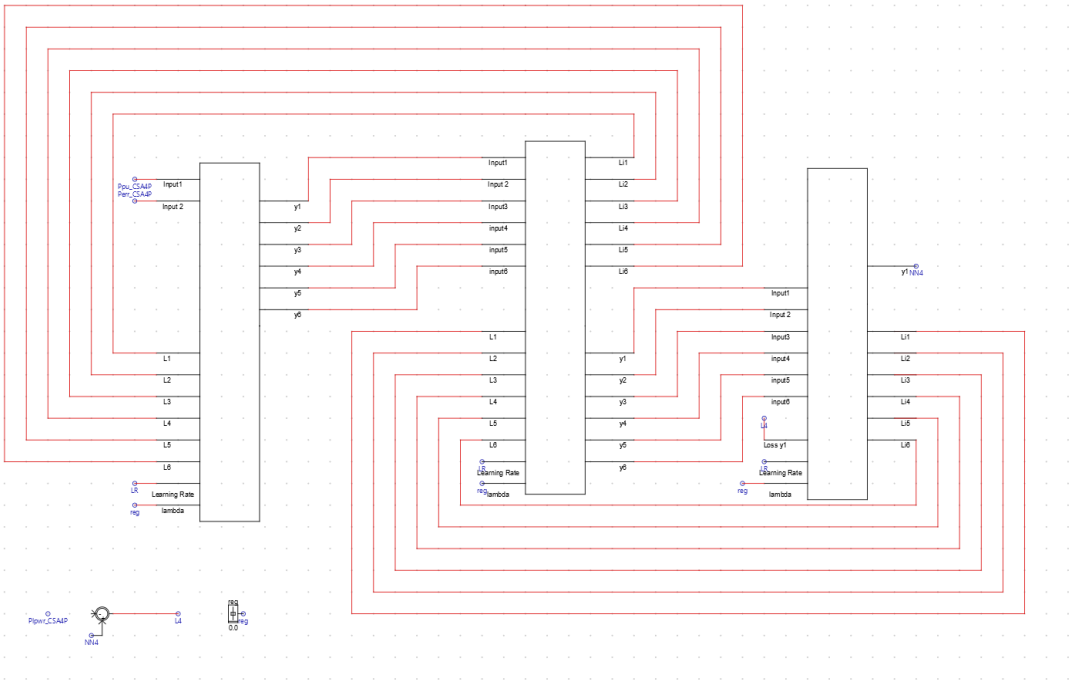

Figure 2.7

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.