Documentation for GRU and LSTM network implementation in Ptolemy II

This documentation explains the way a GRU and LSTM network are implemented in Ptolemy II, how actors behave and the shape of the input data.

GRU Network

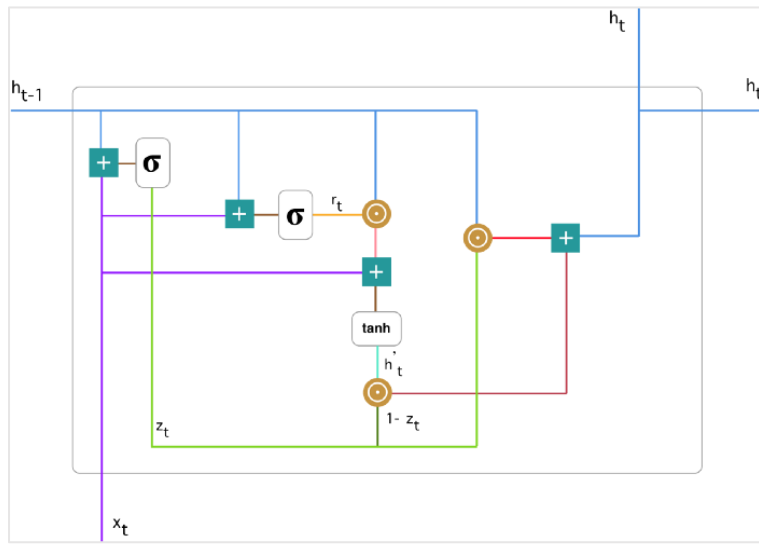The GRU network architecture is shown in *Figure 1*:

Figure 1: GRU network architecture

The equations between the gates are:

$$z_t = \sigma(x_t W_z + b_{xz} + h_{t-1} U_z + b_{hz})$$

$$r_t = \sigma(x_t W_r + b_{xr} + h_{t-1} U_r + b_{hr})$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh(x_t W_h + b_{xh} + r_t \odot (h_{t-1} U_h + b_{hh}))$$

where:

$z_t \rightarrow Update\ Gate$
$r_t \rightarrow Reset\ Gate$
$h_t \rightarrow Hidden\ State\ at\ timestep\ t$

[1]

$x_t \rightarrow Input\ sequence$
$\sigma \rightarrow Sigmoid\ activation\ function$
$tanh \rightarrow tanh\ \ activation\ function$
$\odot \rightarrow Element-wise\ multiplication\ (Hadamard\ product)$

W, U, b, correspond to the *weights* and *biases* of the input data and *z, r, h* to the GRU gates. *Table 1* shows the input data dimensions.

	Input (*x*) (1xA)			Hidden (*h*) (1xB)		
Βάρη (weights)	W_z (AxB)	W_r (AxB)	W_h (AxB)	U_z (BxB)	U_r (BxB)	U_h (BxB)
Biases	b_{xz} (1xB)	b_{xr} (1xB)	b_{xh} (1xB)	b_{hz} (1xB)	b_{hr} (1xB)	b_{hh} (1xB)

Table 1: GRU network – Input data dimensions

The GRU network implementation in Ptolemy II is shown below:
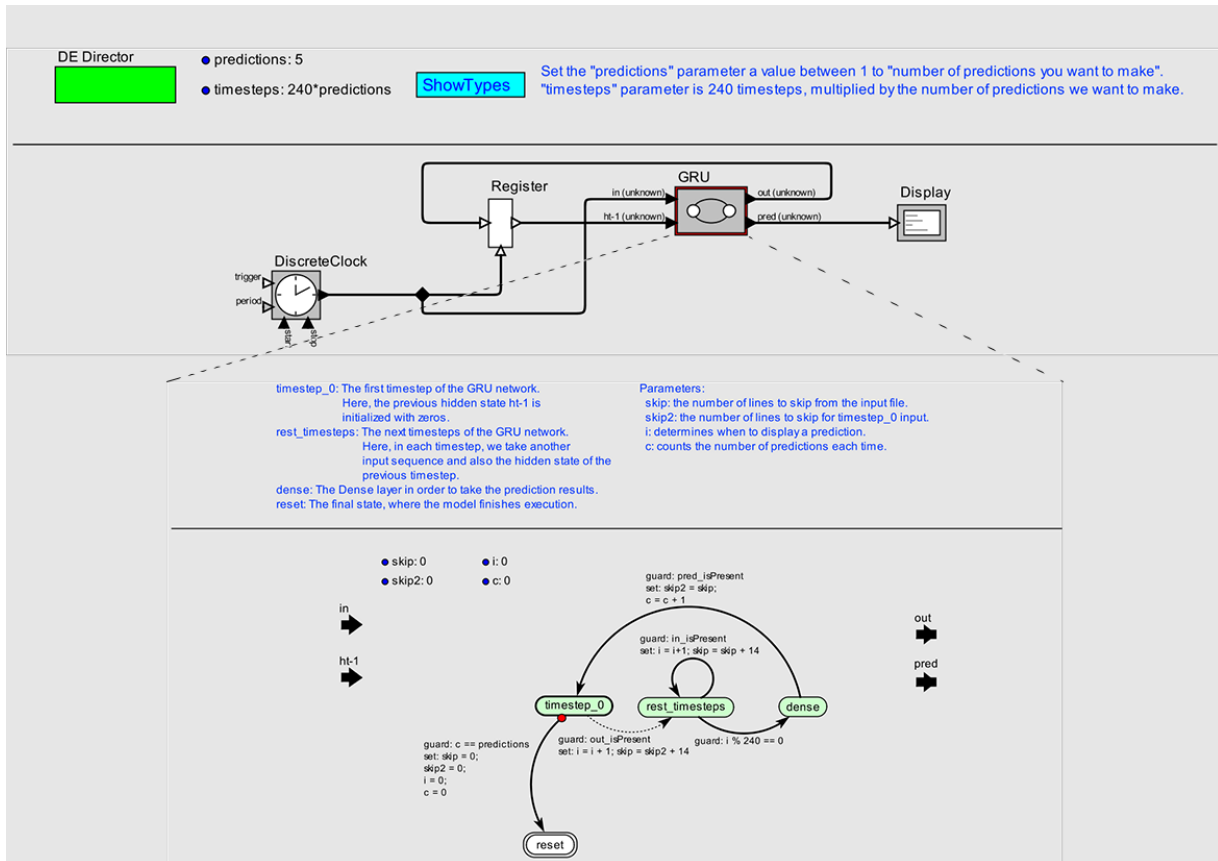

Figure 2: GRU network in Ptolemy II

[2]

The main actor of this model is *GRU*, which is a *ModalModel* actor. ModalModels are a class of Finite State Machines (FSMs), where in each state a new model is implemented that has its own director. The transition from one state to another happens when the *guard* expression is true.

There are two blue bullets on the right side of the DE Director. The *predictions* parameter defines the number of predictions. The *timesteps* parameter defines when the model stops executing. Because the implementation involves RNN networks, the input is a timeseries sequence of data. At each timestep, a batch of the sequence is introduced into the network. Therefore, the termination of the model execution depends on the size of the sequence multiplied by the number of predictions that will be made:

$$timesteps = input\_sequence\_size \cdot predictions$$

The *Register* actor defines the recurrency of the GRU network. The GRU output goes to the Register actor and then this actor sends it as input to the GRU network.

In the lower part of the *Figure 2,* there are three states of the ModalModel:
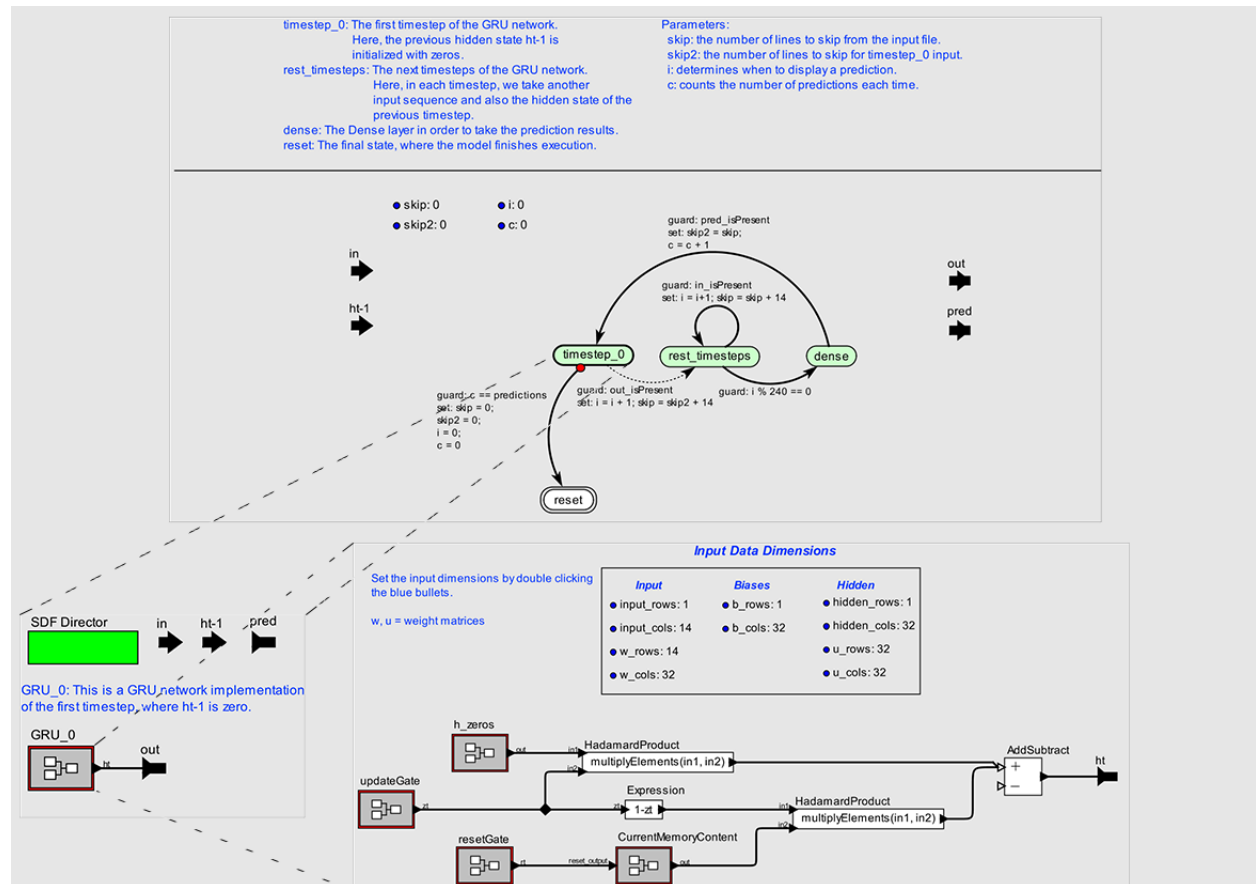
timestep_0 state

Figure 3: GRU – timestep_0 state

[3]

In this state, a GRU network is implemented for the 1st timestep (the 1st batch of the input sequence), where the previous hidden state h_{t-1} is initialized with zeros. Inside the *GRU_0* actor, the GRU network architecture is shown. The *CurrentMemoryContent actor implements the part:*

$$\tanh(x_t W_h + b_{xh} + r_t \odot (h_{t-1} U_h + b_{hh}))$$

of the h_t equation.

h_zeros actor in initialized with zeros (*1xB dimensions)*. The content of the GRU gates are presented in the next figure:
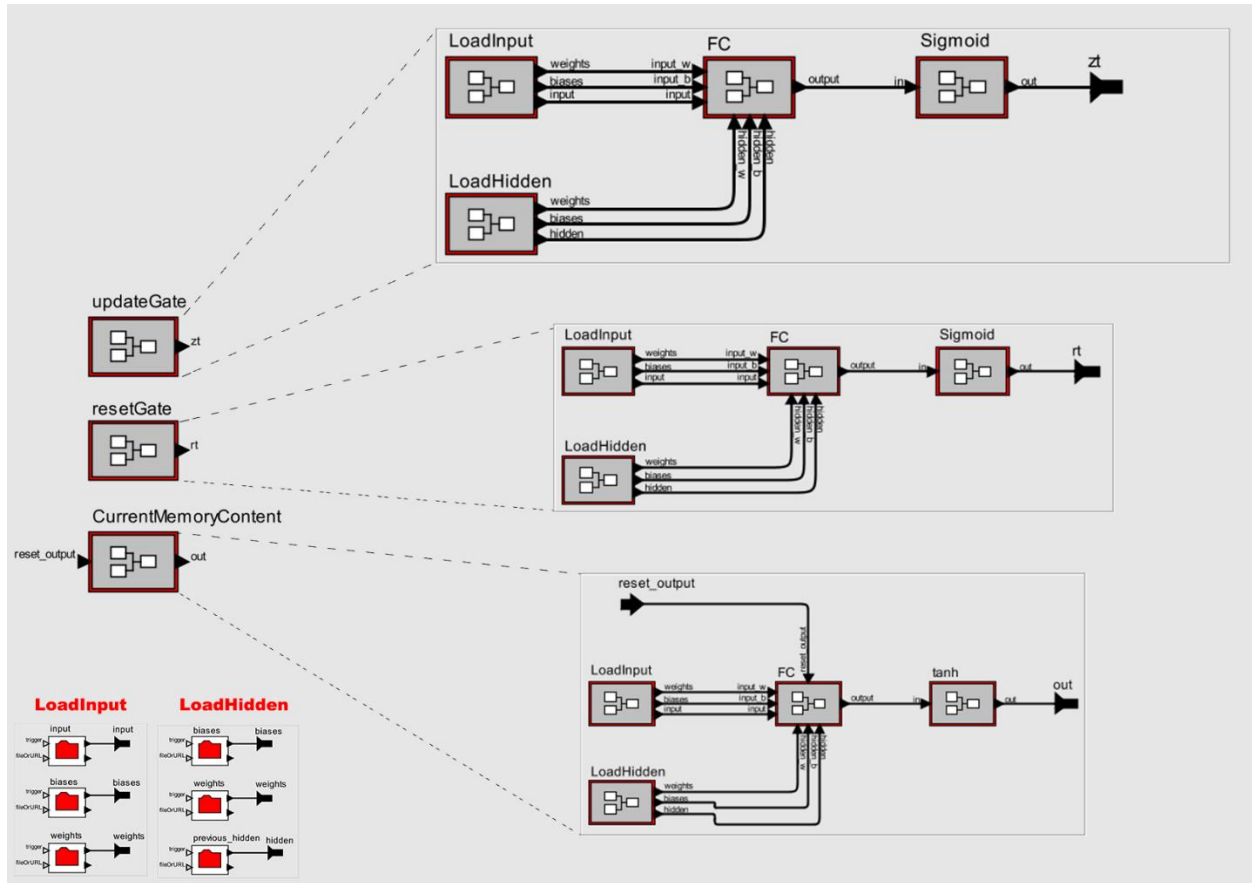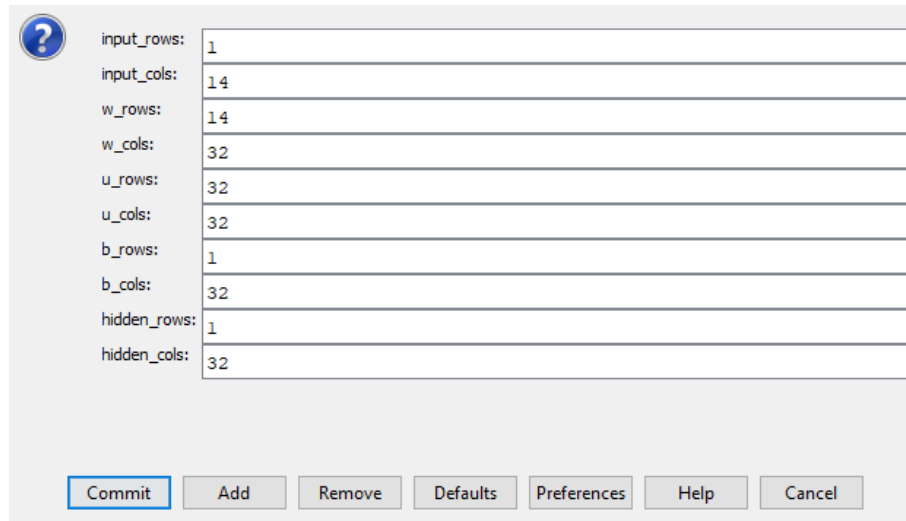

Figure 4: GRU gates

In each gate, there is an *FC* actor, which implements the operations between the weights and biases of each gate.

LoadInput and *LoadHidden* actors are used to load the .csv data files (weights, biases, input sequence). *previous_hidden* LineReader actor in loaded with a .csv file of *B zeros.* This procedure of loading input data files is done for every gate of the GRU network. Each gate has different weights and biases, but the input sequence is the same. Note that in the LineReader actor *input*

[4]

of all three gates, the *numberOfLinesToSkip* parameter has the value *skip2*. The usefulness of this parameter will be commented below.

Inside the *GRU_0* actor there is a frame which defines the dimensions of the input data. These dimensions can be modified by *double-clicking* on the *GRU_0* actor (*Figure 5*).

Figure 5: GRU_0 – double click

The output of this state is the hidden state for the next timestep.
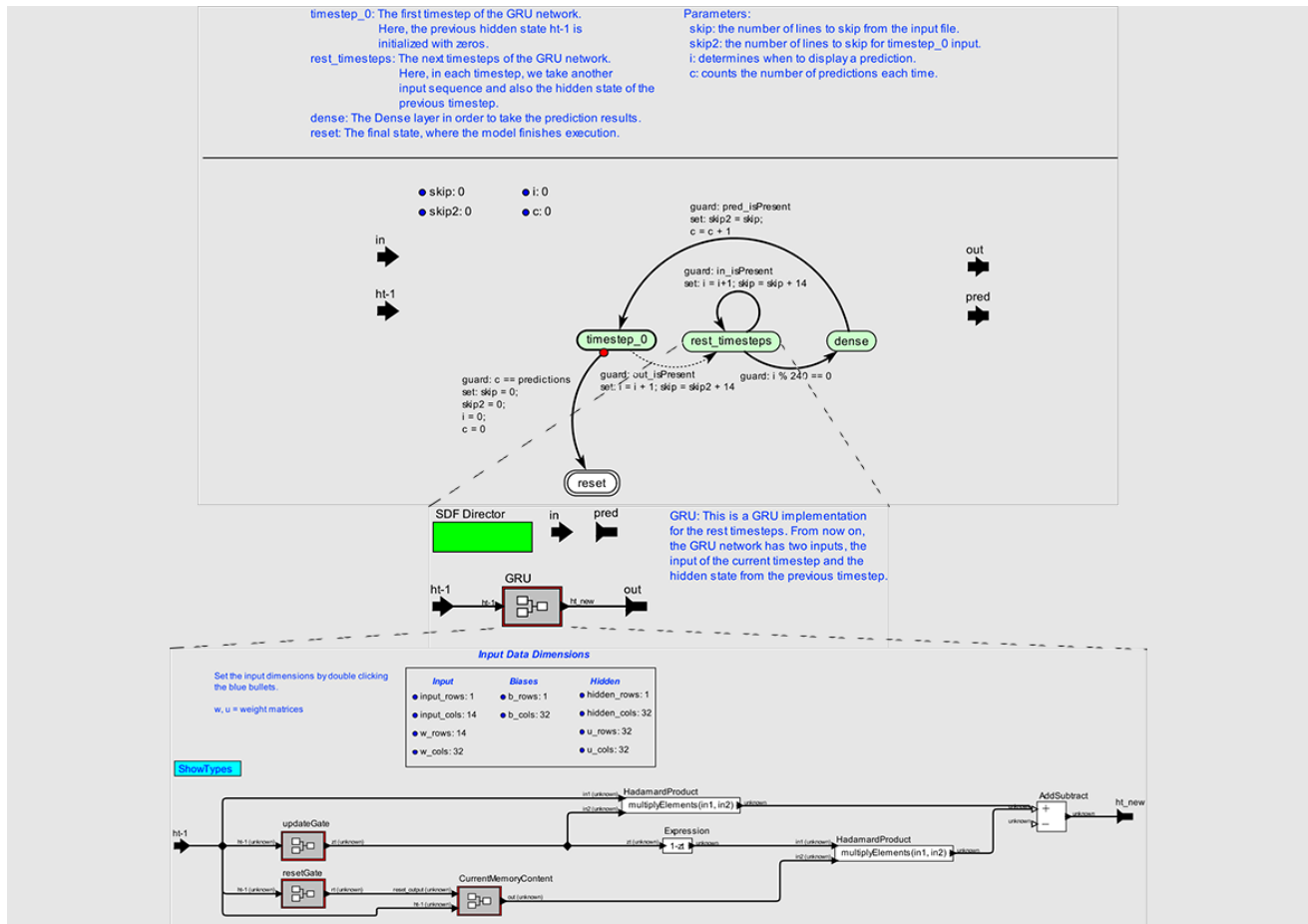
rest_timesteps state

Figure 6: GRU – rest_timesteps state

In this state, a GRU network is implemented for the rest timesteps of the model. Now, except from the next batch of the sequence, the previous hidden h_{t-1} comes in as input at the network.

The same logic as *timestep_0* is followed here. The only difference is inside *FC* actor and more specifically in the Hidden part, *hidden* is already a matrix from the previous state. So, it doesn't need to be converted. The differences of the *Hidden* actors for the two states are shown below:
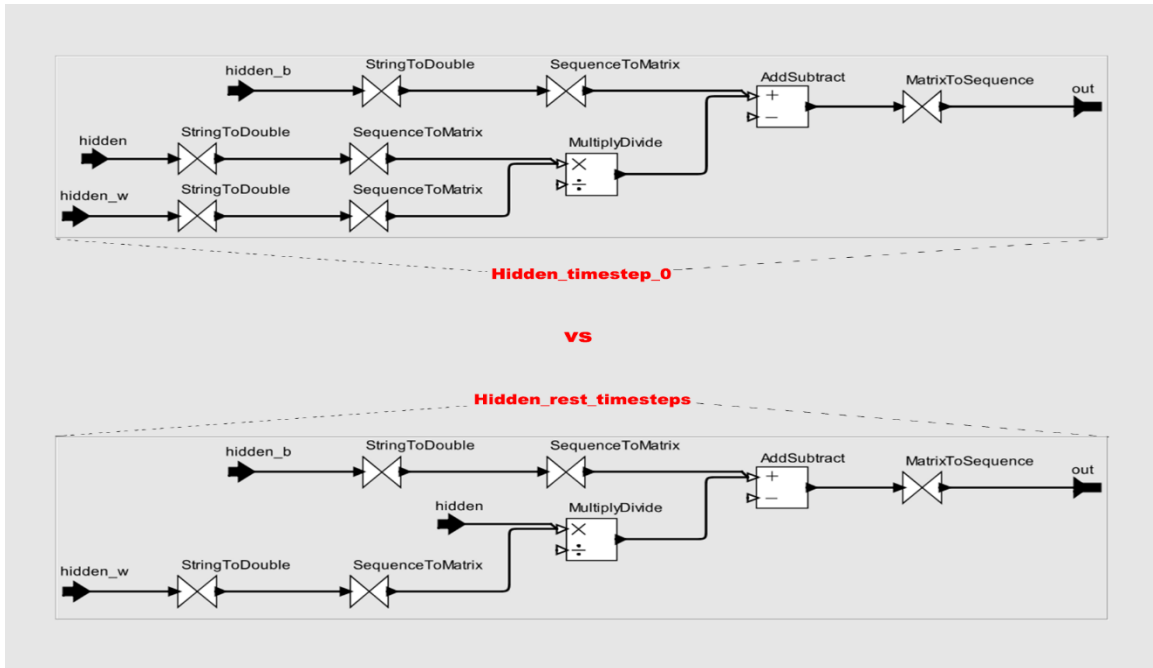
Figure 7: Hidden actors' comparison

Also, the *numberOfLinesToSkip* parameter is set to *skip*, not *skip2.*

The output of this state is the hidden state for the next timestep.
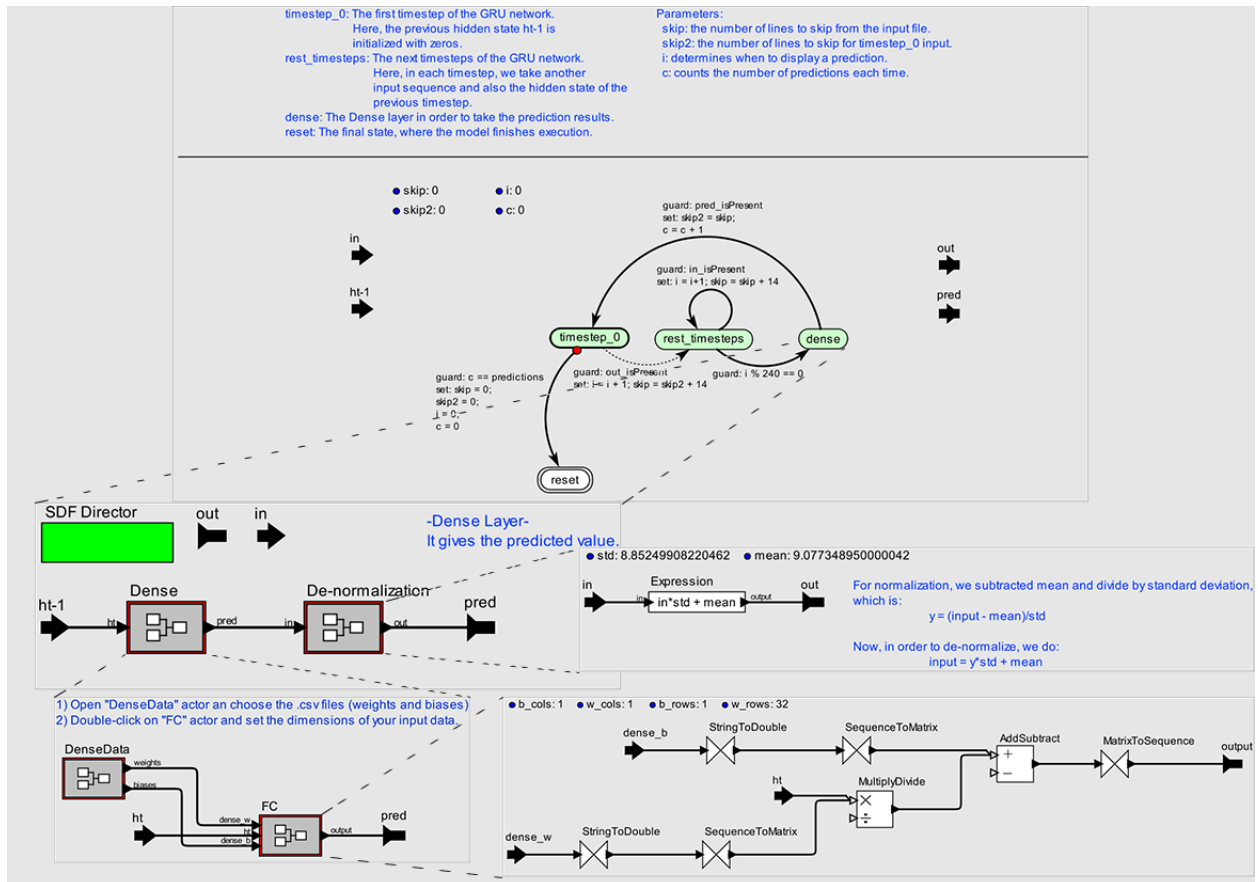
dense state

Figure 8: GRU – dense state

The *Dense* layer is implemented here. When the GRU network finishes all the batches from the input sequence, the last output from the GRU network enters the Dense layer and it gives the first prediction. A *De-normalization* actor is used in order to get the prediction at the desired measure (this can be omitted).

How does the transition happen from one state to another?

The *timestep_0* state is the *initial_state*, because it has the bold outline. When a value is generated at the *out* port, the guard *out_isPresent* is enabled and the execution of the model is proceeded to the *rest_timesteps* state. Upon transitioning, the *set* parameters are set. *i* parameter counts the number of batches at each timestep and the parameter *skip* is used in order to read the next batch of the sequence. The transition from *timestep_0* state to the *rest_timesteps* state is a *default transition*. This means that if no other non-default transition is enabled and the guard is true, then the model will choose this transition.

[8]

In the *rest_timesteps* state, there is a *self-transition*. The model stays here and keeps processing batches of data, until the input sequence is over. Again, *i* parameter is used to count the timesteps and *skip* parameter to pass the next batch on the network.

When the batches are over, the input sequence is over, so a prediction can be made. This is when the guard for the *dense* state is enabled.

As soon as the prediction is made, the prediction value is generated at the *pred* port and the guard of the transition to the *timestep_0* state is enabled. Here, *skip2* parameter is set equal to *skip*, in order to skip the current input sequence and get the next input sequence. If there is not another input sequence, then this parameter can be omitted. *c* parameter counts the predictions.

When the *c* parameter is equal to the *predictions* parameter, then the execution of the model goes to the *reset* state, in order to reset the parameters and finish execution. This transition is a *preemptive transition*, which means that the current state will not be iterated prior the transition.

As mentioned before, *skip* parameter is used in order to get the next batch of the sequence. So, it increases by the *size_of_batch (skip = skip + size_of_batch)*. On the other hand, *skip2* parameter is equal to the size of the input sequence, so it gets the full size of the sequence in order to skip it and read the next sequence (if there is one).

LSTM Network

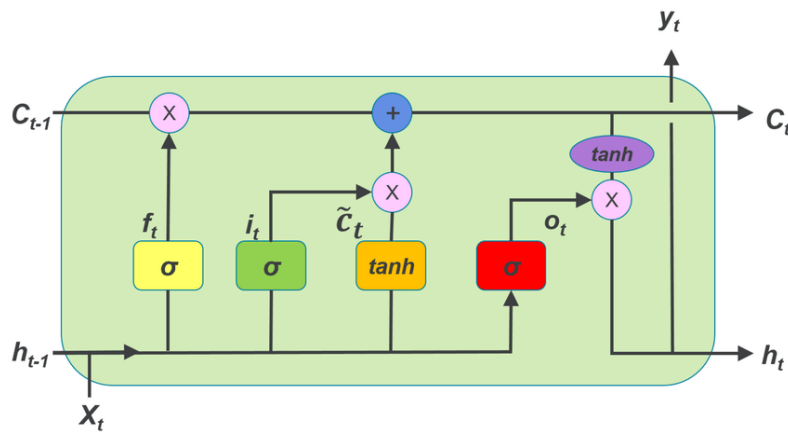The LSTM network architecture is shown in *Figure 9:*

Figure 9: LSTM network architecture

The equations between the gates are:

$$f_t = \sigma(x_t W_f + h_{t-1} U_f + b_f)$$

$$i_t = \sigma(x_t W_i + h_{t-1} U_i + b_i)$$

$$\tilde{c}_t = \tanh(x_t W_c + h_{t-1} U_c + b_c)$$

$$o_t = \sigma(x_t W_o + h_{t-1} U_o + b_o)$$

where:

$f_t \rightarrow Forget\ Gate$
$i_t \rightarrow Input\ Gate$
$\tilde{c}_t \rightarrow g\ Gate\ (ή\ candidate\ cell\ state)$
$o_t \rightarrow Output\ Gate$
$h_t \rightarrow Hidden\ State\ at\ timestep\ t$
$x_t \rightarrow Input\ sequence$
$\sigma \rightarrow Sigmoid\ activation\ function$
$tanh \rightarrow tanh\ activation\ function$

W, U, b, correspond to the *weights* and *biases* of the input data and *f, c, I, o* to the LSTM gates.
Table 2 shows the input data dimensions.

	Input (*x*) (1xA)				Hidden (*h*) (1xB)			
Βάρη (weights)	W_f (AxB)	W_c (AxB)	W_i (AxB)	W_o (AxB)	U_f (BxB)	U_c (BxB)	U_i (BxB)	U_o (BxB)
Biases	b_f (1xB)	b_c (1xB)	b_i (1xB)	b_o (1xB)	b_f (1xB)	b_c (1xB)	b_i (1xB)	b_o (1xB)

Table 2: LSTM network – Input data dimensions

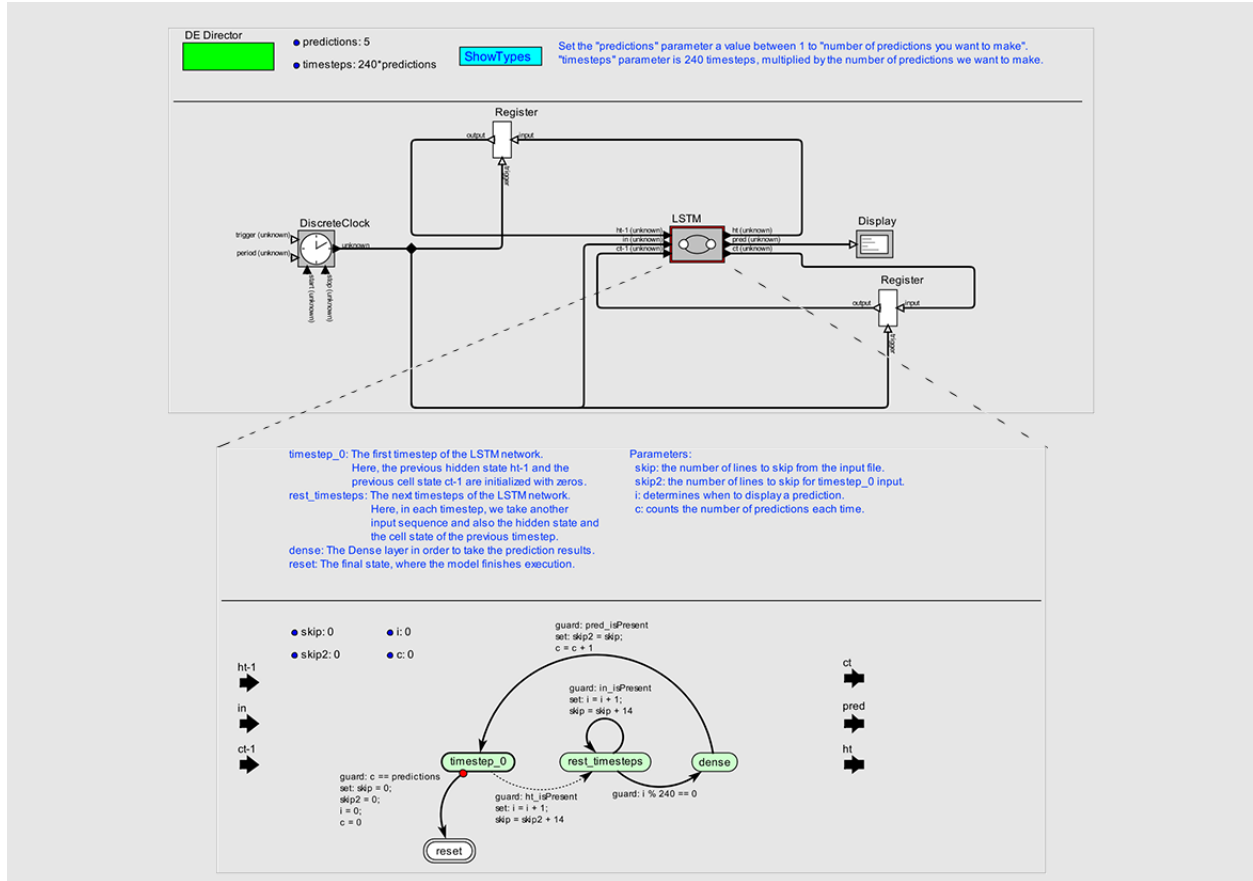The LSTM network implementation in Ptolemy II is shown below:

Figure 10: LSTM network in Ptolemy II

Again, the main actor of this model is the *LSTM* actor and there is also the *predictions* parameter, which defines the number of predictions and the *timesteps* parameter, which defines when the model stops executing. There are two *Register* actors now, because except from the previous hidden state, previous cell state also comes in as input at the LSTM network.

Again, there are three states:
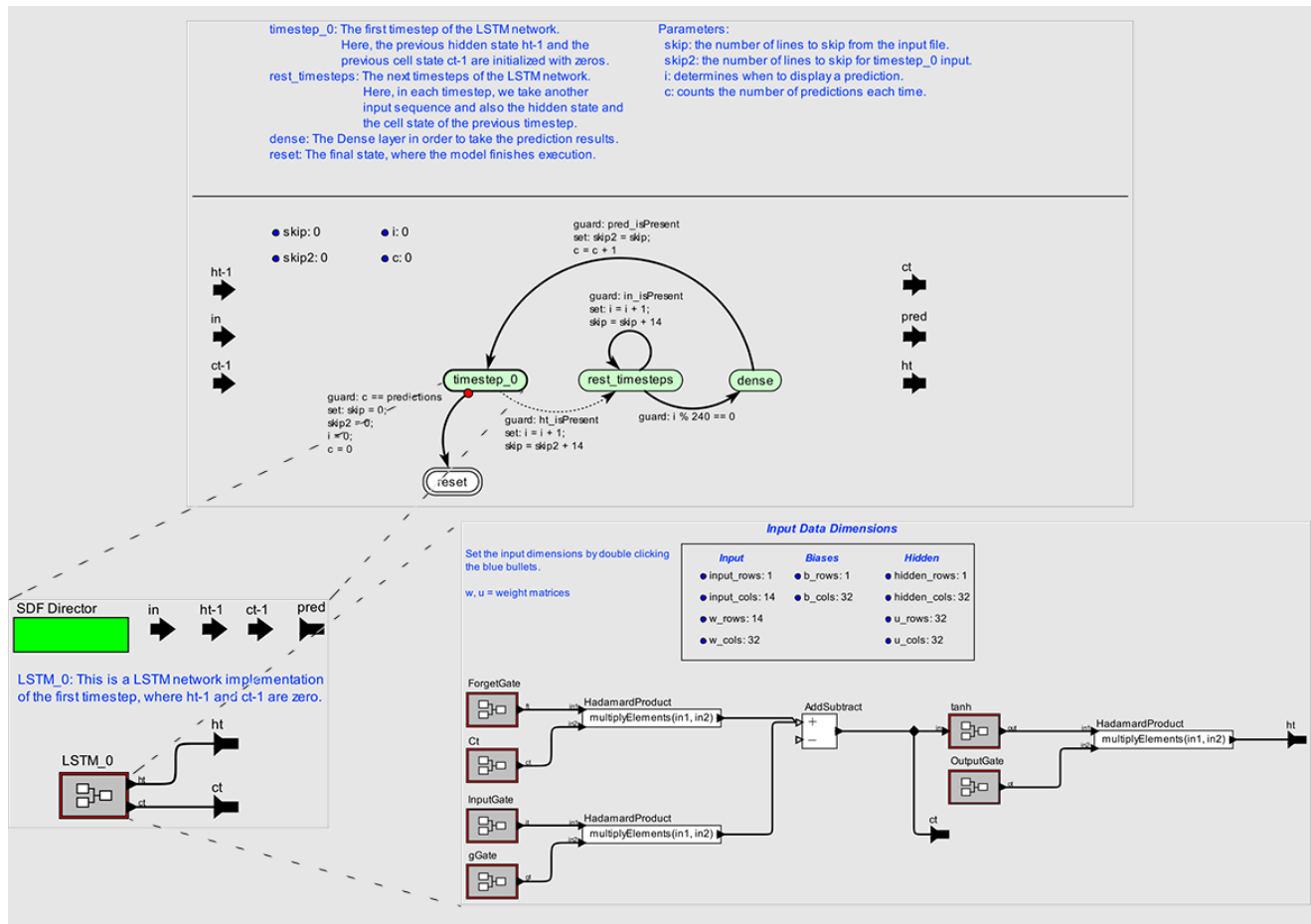
timestep_0 state

Figure 11: LSTM – timestep_0 state

In this state, an LSTM network is implemented for the 1st timestep (the 1st batch of the input sequence), where the previous hidden state h_{t-1} and the previous cell state c_{t-1} are initialized with zeros.

C_t actor in initialized with zeros (*1xB dimensions)*. The content of the LSTM gates is presented in the next figure:
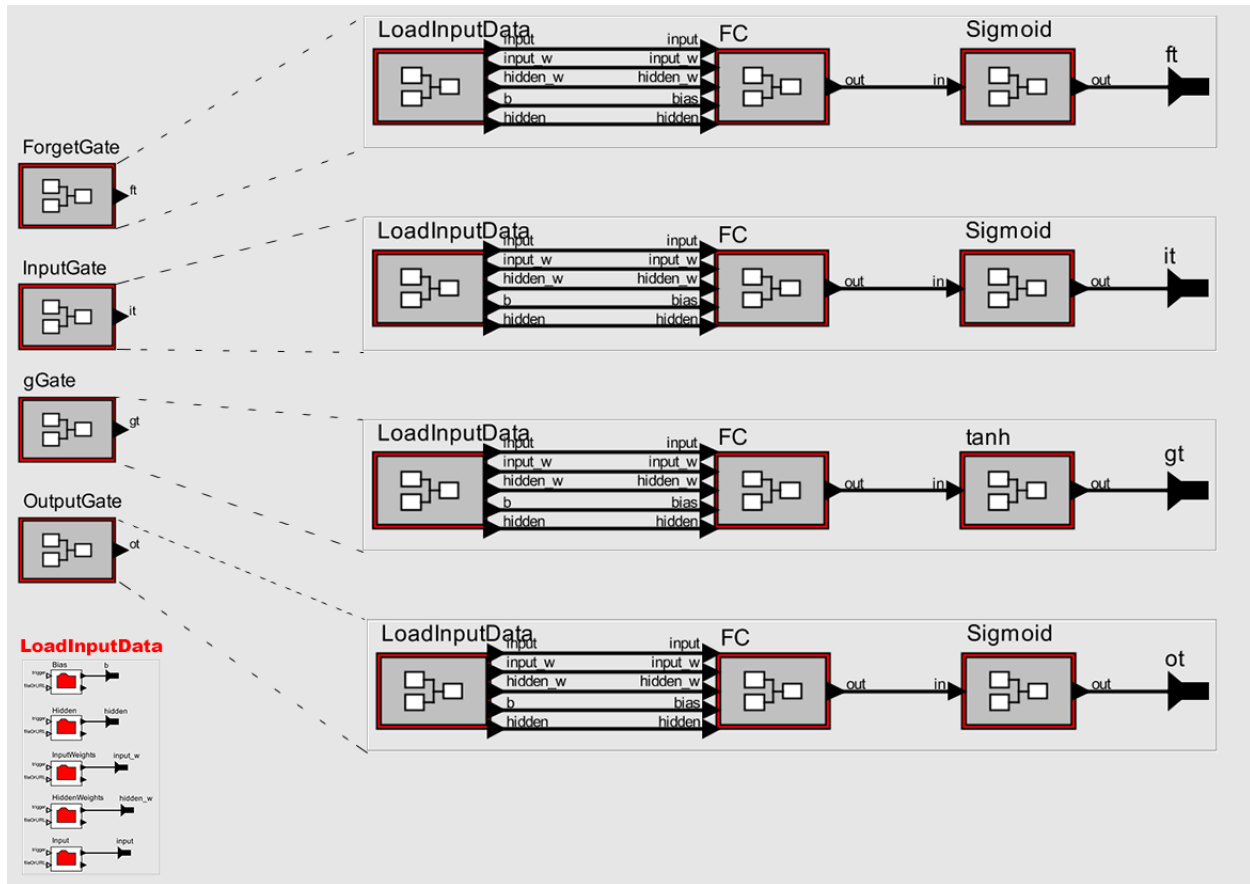
Figure 12: LSTM gates

LoadInputData actor is used to load the .csv data files (weights, biases, input sequence). *Hidden* LineReader actor in loaded with a .csv file of *B zeros.* This procedure of loading input data files is done for every gate of the LSTM network. Each gate has different weights and biases, but the input sequence is the same. Note that in the LineReader actor *Input* of all four gates, the *numberOfLinesToSkip* parameter has the value *skip2*.

Inside the *LSTM_0* actor there is a frame which defines the dimensions of the input data. Again, the parameters can be modified by *double-clicking* the *LSTM_0* actor.

The output of this state is the hidden state and the cell state for the next timestep.
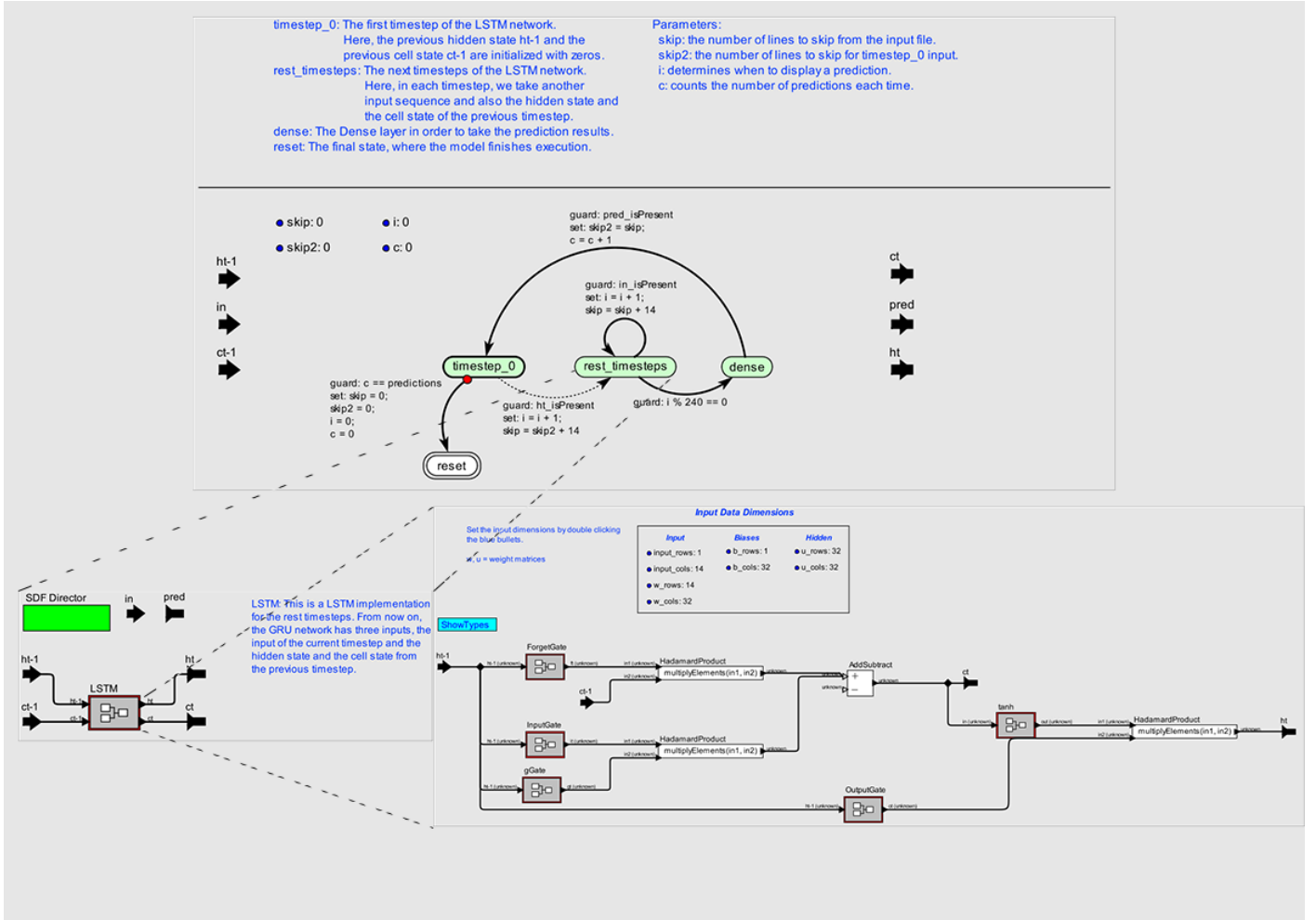
rest_timesteps state

Figure 13: LSTM – rest_timesteps state

In this state, an LSTM network is implemented for the rest timesteps of the model. Now, except from the next batch of the sequence, the previous hidden h_{t-1} and the previous cell state c_{t-1} comes in as input at the network.

The same logic as *timestep_0* is followed here. The only difference is inside *FC* actor and more specifically in the Hidden part, *hidden* is already a matrix from the previous state. So, it doesn't need to be converted. In order to pass the next batch of the input sequence, *skip* parameter is set at the *numberOfLinesToSkip* parameter in the *Input* FileReader actors.

The outputs of the *rest_timesteps* state are the hidden state and the cell state for the next timestep.
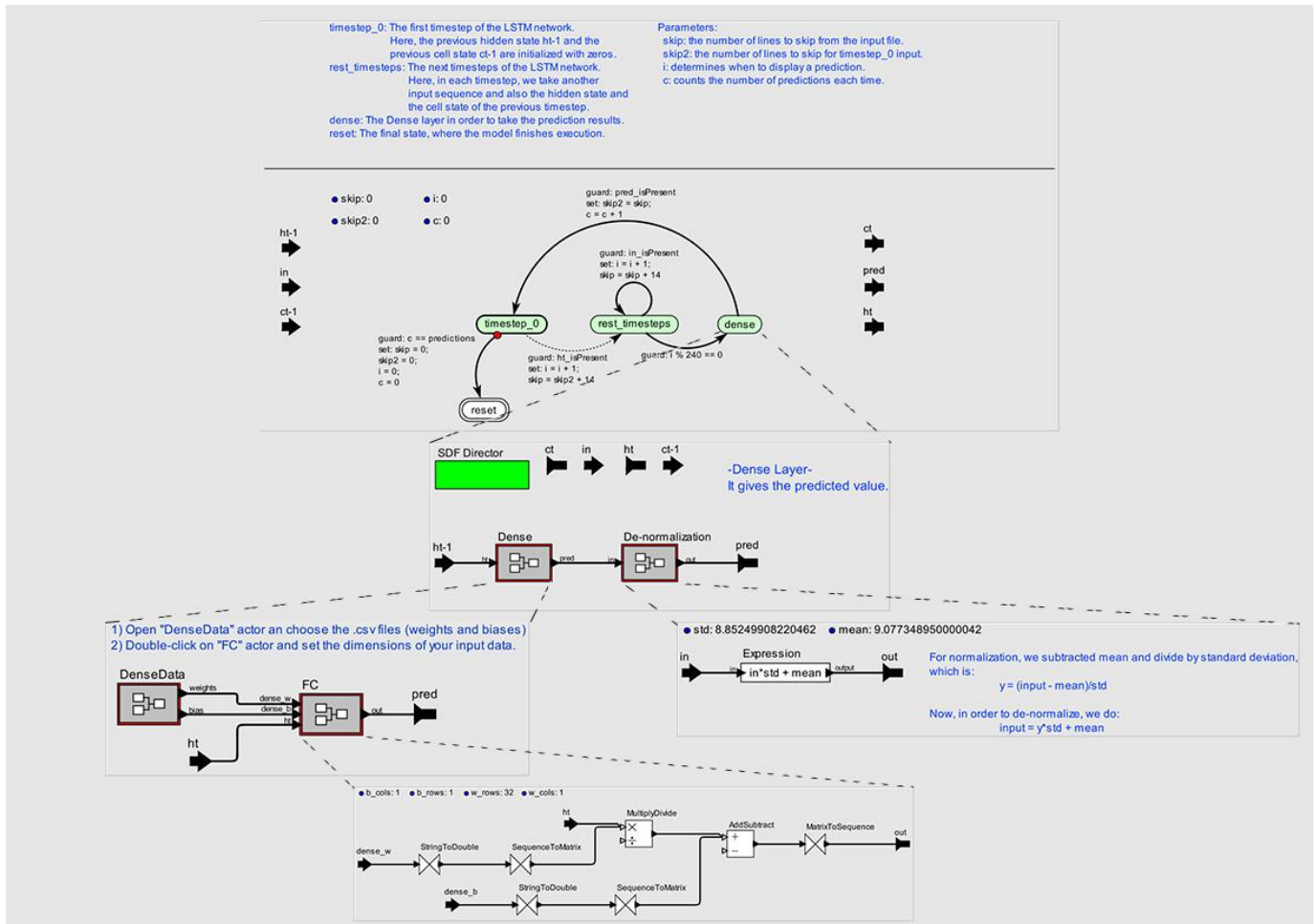
[14]

dense **state**

Figure 14: LSTM – dense state

The *Dense* layer is implemented here. When the LSTM network finishes all the batches from the input sequence, the last output from the LSTM network enters the Dense layer and it gives the first prediction. A *De-normalization* actor is used in order to get the prediction at the desired measure (this can be omitted).

The transitions between the states, the way guards enable, and the way parameters are modified, are exactly the same as in GRU implementation.

[15]

To sum up:

1) Both in GRU and LSTM networks, input files must be loaded on each gate separately using *LineReader* actors. Specifically:
   a) **GRU**
      i) *Right click on GRU ModalModel → Open Actor*
      ii) *Right click on timestep_0 state (or rest_timesteps or dense) → Look Inside*
      iii) *Right click on GRU_0 actor (GRU ή Dense) → Open Actor*
      iv) *For each gate updateGate, resetGate, CurrentMemoryContent (DenseData), right click → Open Actor*
      v) *Right click on LoadInput actor → Open Actor for the input data files*
         (1) *Double click on each of the FileReader actors and select the proper files*
      vi) *Right click on LoadHidden actor → Open Actor for the hidden state data files*
         (1) *Double click on each of the FileReader actors and select the proper files*

   For the h_zeros actor you can read a file with the proper number of zeros or define such a matrix using Ptolemy actors. In the above explanation, .csv file was used.

   b) **LSTM**
      i) *Right click on GRU ModalModel → Open Actor*
      ii) *Right click on timestep_0 state (or rest_timesteps or dense) → Look Inside*
      iii) *Right click on GRU_0 actor (GRU ή Dense) → Open Actor*
      iv) *For each gate ForgetGate, InputGate, gGate, OutputGate (DenseData), right click → Open Actor*
      v) *Right click on LoadInput actor → Open Actor for the input data files*
      vi) *Double click on each of the FileReader actors and select the proper files*

   For the C_t actor you can read a file with the proper number of zeros or define such a matrix using Ptolemy actors. In the above explanation, .csv file was used.

2) *LineReader* actors are used to read .csv files and not *CSVReader* or *FileReader*. This is because it is a less complicated and less time-consuming process than using the other two actors. LineReader actor reads a file line by line and gives it in *string* format. Therefore, our files must be in vector-column format. So, an array *(n x m)* must be converted to a vector *((n x m) x 1)*, <u>converting each row into a column</u>. Then, the actors *StringtoDouble* and *SequenceToMatrix* are used to convert the data to *double* format and then to a matrix of the desired dimensions.

3) All .csv files (or files of any other format for input data) must be in the same folder as the .xml file of the Ptolemy model.

4) In this repository, there is also the *userLibrary* about these models. *Copy-paste* the *userLibrary.xml* file in the userLibrary destination folder and open it with Ptolemy II.