

BackPropModel

Cheol Han

University of Southern California

A supplement to Prof. Arbib's CS 564, Brain Theory and Artificial Intelligence

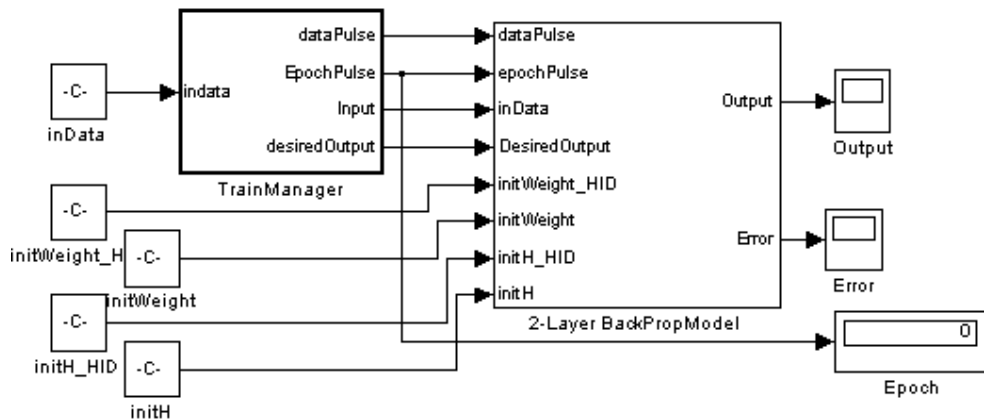
University of Southern California, Fall 2005.

Preface

It is strongly recommended to understand the document, "the General Guide to Design a NSL-MatLab model" before reading this.

This BackPropModel is different from other time-based models; every each step of simulation is evoked by each row of a data set. In other words, this is a sample-based discrete system with training set.

Thus, some parameters should be set for this kind of system. First of all, in the window of "Simulation->Simulation parameters", Change the type to "fixed-step", then set the "fixed step size" with some small value. ie. 0.01 And simulation "stop time" should be set as "MaxEpoch*dataSize*fixed step size" even though simulation will stop whenever the overall error per epoch is below the stopError.

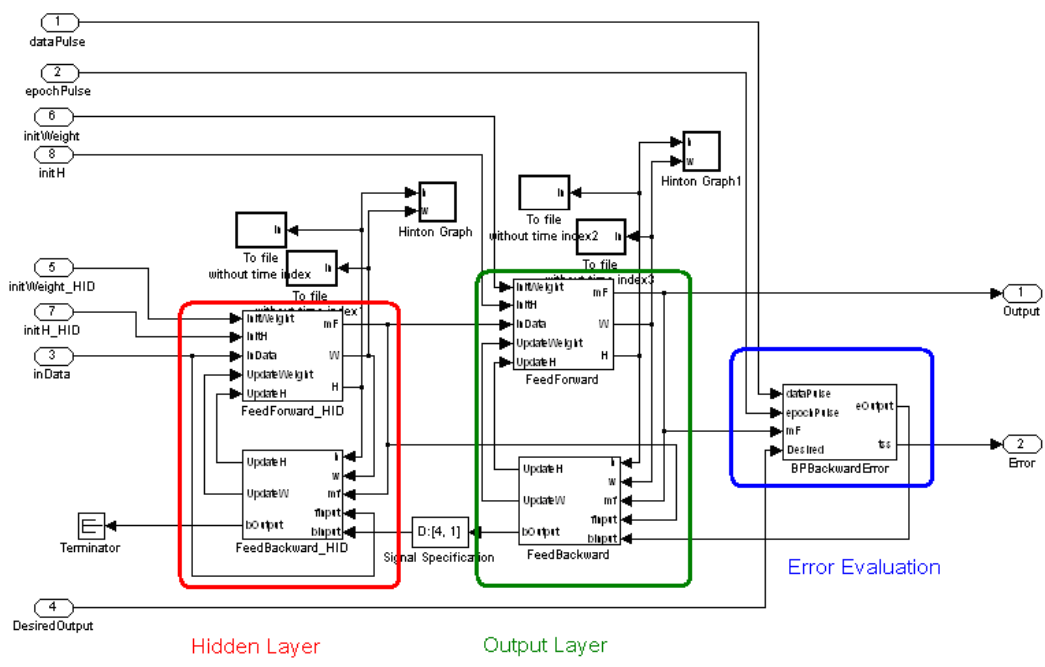


Sample-based System

The key concept of this sample-based discrete system is the "Memory Block." It is updated every fixed time step; ie. it is updated every 0.01 sec. The gold way to construct this kind of system is add "memory" into each sub block.

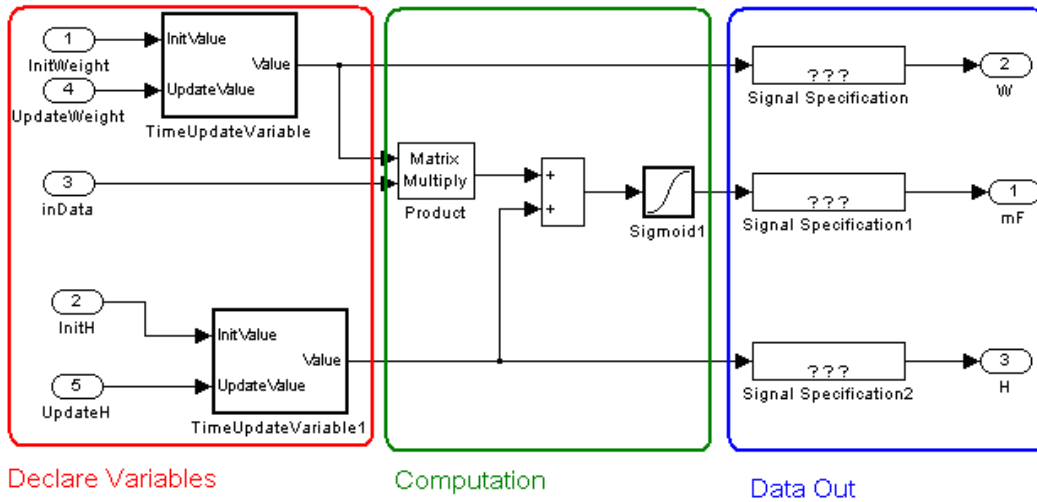
However, notice that other algebraic operation can be computed in the continuous sense. The point is that the value is updated by the fixed time step. Of course, this memory can be used in the continuous system. In this case, update can be done irregularly. Thus it should be careful. Further detail can be found in the document, “the General Guide to Design a NSL-MatLab model.”

This document does NOT explain how to copy and paste blocks into the model from our library. Instead, important concept and technique which are used in the model are shown. Moreover, it betrays how the general guide is applied when this model was constructed.



1. FeedForward Block (Evaluate Phase)

This is wide-used feed-forward model. It can handle any dimension of layer. The parameter “size” should be set as [OutSize, inSize].



As indicated in the above diagram, the variables are declared, initialized and updated in the leftmost part. Then, passing through the computational part, the valuable output is produced.

The first structure of this feedforward module is the red box. Notice that it includes updating of TimeUpdateVariables. The upper one is a weight matrix and the below one is a threshold matrix. Because these two matrixes are the memory of this feedforward module, timeupdatevariables are used.

The computational part consists of a matrix multiply and a sigmoid block. As shown below, the matrix multiplication betrays the neural network computation. In this case, an input vector should be a column vector. Our sigmoid block is dimension-free.

$$mp_p = \sum_s w_{sp} in_s$$

$$\begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{bmatrix} \begin{bmatrix} in_1 \\ in_2 \\ in_3 \end{bmatrix} = \begin{bmatrix} mp_1 \\ mp_2 \end{bmatrix}$$

weight matrix input output

where $w_{\text{input index, output index}}$

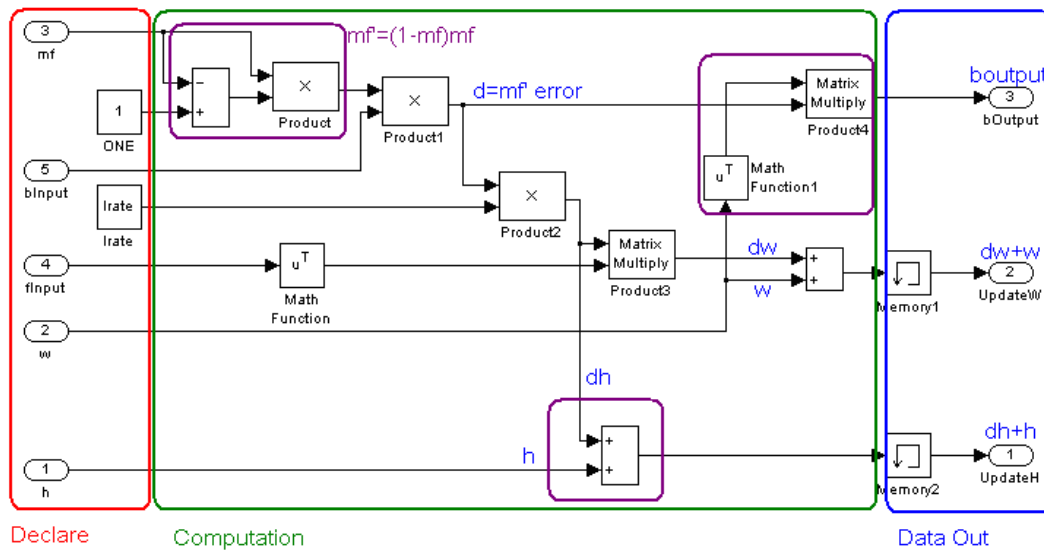
$$mf_p = \text{sigmoid}(mp_p + h_p)$$

$$\mathbf{mf} = \text{sigmoid}(\mathbf{mp} + \mathbf{h})$$

Just before out-ports, the signal specification blocks are found. Those look trivial but important because we treat matrixes. Basically, MatLab transforms m by 1 matrix(column vector) and 1 by m matrix(row vector) into a vector whose length is m. After this transformation, MatLab usually succeeds to re-transform it to a column vector or a row vector. However, sometimes it fails; as an example, if there are complex loops, it is so ambiguous whether this vector is a column vector or a row vector. So, it is strongly recommended to set the form of matrix like this using a signal specification block.

2. FeedBackward Block (Update Phase)

In contrast with FeedForward model, this FeedBackward model is the update algorithm of the specific back propagation algorithm. Surely, it can handle any dimension of layer without setting size information. The learning rate is the only value it needs. The default value is 0.2



Even though the direction in the overall 2-layer diagram is reversed, the signal flow can be in the right direction; optionally, it should be. Computational part is the most important in backpropmodel but easy to construct if the update equation is known as below. One careful point is a matrix multiply.

While integrate all inputs into matrixes, the multiplication operation can be confused. Notice that the shown equation is only for “one neuron” not for “one layer.” Additionally, because almost all the book does not concern the matrix representation of a layer, the notation does not distinguish real matrix multiplication from multiplying a scalar to a matrix. Below, the type of multiplication is clarified; the only two real matrix multiplications are shown below. Binput is computed outside; in case of output layer, the BPBackwardError module computes for the output layer and in case of hidden layer, output layer does.

$$\delta_q = mf'_q \cdot binput_q$$

$$\bar{\delta} = \mathbf{mf}' \mathbf{binput} \quad (\text{element-wise})$$

$$mf'_q = mf_q(1 - mf_q) \quad (\text{in case of sigmoid})$$

$$\mathbf{mf}' = \mathbf{mf}(1 - \mathbf{mf}) \quad (\text{element-wise})$$

$$\Delta h_q = \eta \delta_q$$

$$\Delta \mathbf{h} = \eta \bar{\delta} \quad (\text{scalar multiplication})$$

$$h_q(t+1) = h_q(t) + \Delta h_q$$

$$\mathbf{h}(t+1) = \mathbf{h}(t) + \Delta \mathbf{h}$$

$$\Delta w_{pq} = \eta \delta_q \mathit{finput}_p \quad (\text{scalar multiply scalar})$$

$$\Delta \mathbf{w} = \eta \bar{\delta} \mathbf{finput}^T \quad (\text{matrix multiplication})$$

$$\begin{bmatrix} \Delta w_{11} & \Delta w_{21} & \Delta w_{31} \\ \Delta w_{12} & \Delta w_{22} & \Delta w_{32} \end{bmatrix} = \eta \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \begin{bmatrix} \mathit{finput}_1 & \mathit{finput}_2 & \mathit{finput}_3 \end{bmatrix}$$

$$= \eta \begin{bmatrix} \delta_1 \mathit{finput}_1 & \delta_1 \mathit{finput}_2 & \delta_1 \mathit{finput}_3 \\ \delta_2 \mathit{finput}_1 & \delta_2 \mathit{finput}_2 & \delta_2 \mathit{finput}_3 \end{bmatrix}$$

$$w_{pq}(t+1) = w_{pq}(t) + \Delta w_{pq}$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}$$

Binput for the output layer

$$error_q = desiredOutput_q - actualOutput_q$$

error = desiredOutput - actualOutput

Binput for a hidden layer

$$boutput_p = \sum_q \delta_q w_{pq} \quad (\text{scalar multiply scalar})$$

boutput = w^T δ (matrix multiplication)

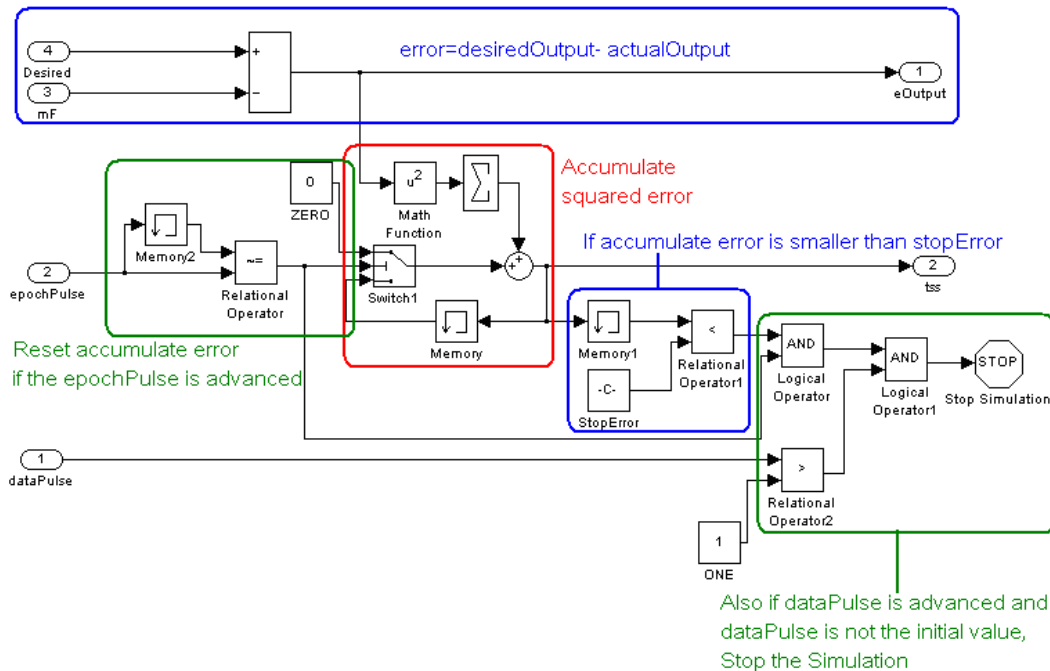
$$\begin{bmatrix} boutput_1 \\ boutput_2 \\ boutput_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} w_{11}\delta_1 + w_{12}\delta_2 \\ w_{21}\delta_1 + w_{22}\delta_2 \\ w_{31}\delta_1 + w_{32}\delta_2 \end{bmatrix}$$

One careful point is that memory blocks are put before out-ports. This is because the updated value should be used in the next sample not this current sample. If there were not a memory block, the loop for updating is trapped infinitely, and the weight matrix and threshold matrix will not be converged. In the update equations, this is indicated with time index (t+1) and (t). Notice again. These memory block in the update feedbackward module are crucial.

3. BPBackwardError

A tricky technique is used in order to reset the error signal and synchronized to the discrete time. To synchronize all systems, two kinds of pulses are utilized: dataPulse and epochPulse. The advantage of this technique is explained in the TrainManager and, these special signals are provided by the TrainManager.

Notice that the training phase stops if the overall error per epoch is below "StopError"



The value error is introduced in the previous section. Check how memory block is also used for accumulating squared error per epoch.

4. TrainManager

This provided TrainManager is a very primitive version. Currently, it provides with only the function of a patcher of data set: named as Pusher. The further explanation of Splitter is inside it. Pusher has the parameters: size of data and maximum # of epoch.

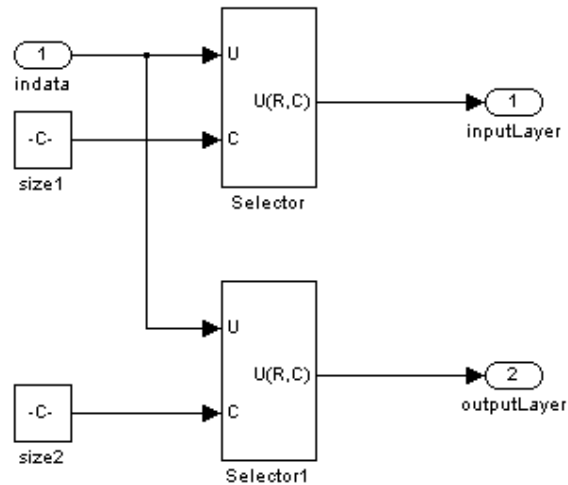
It includes for loop of NSL scheduler. However, you cannot see the explicit for or while subsystem. If so, the model should be in the TrainManager. Instead, the technique in the VLSI design is applied in order to detach the Train Manager and a model. In that field, the memory should be synchronized by the clock generator. Hereby, as setting the simulation parameter with a fixed time step, the clock generator function is absorbed in the MatLab. In addition to this, memory block takes an important role. Moreover, to avoid providing any Train dependant parameter like maxEpoch to

the model, additional iteration information is delivered to other blocks with dataPulse and EpochPulse. Handling this information is explained in the BPBackwardError.

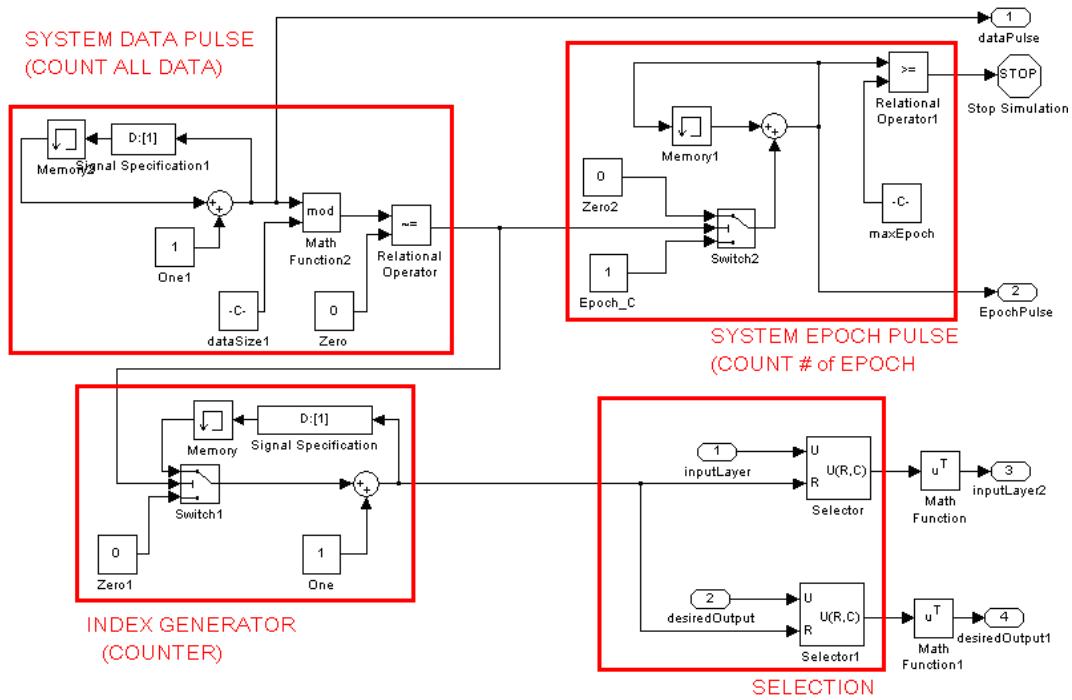
4.1.Splitter

The splitter distinguishes inputs and outputs. In many cases, the data set's each row has each instance including inputs and outputs. In that case, the input and output should be separated. Otherwise, data can be connected directly to Pusher.

The only information needed is how many inputs are and how many outputs are. The standard of this parameter is [size of input, size of output].

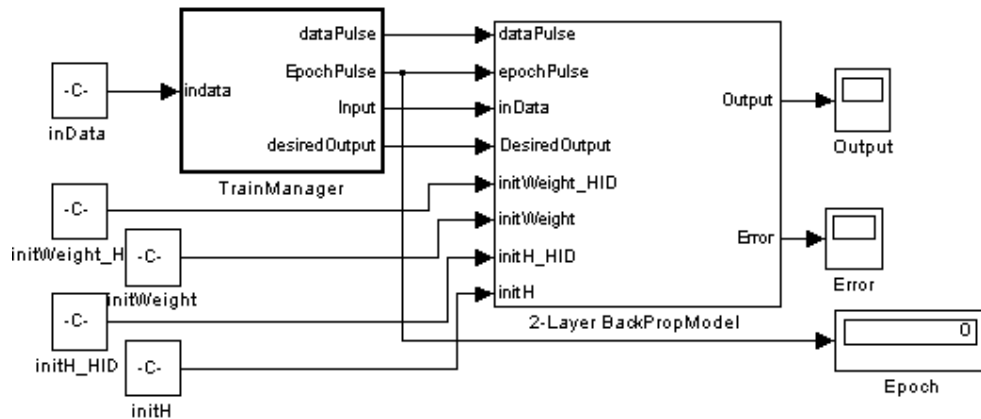


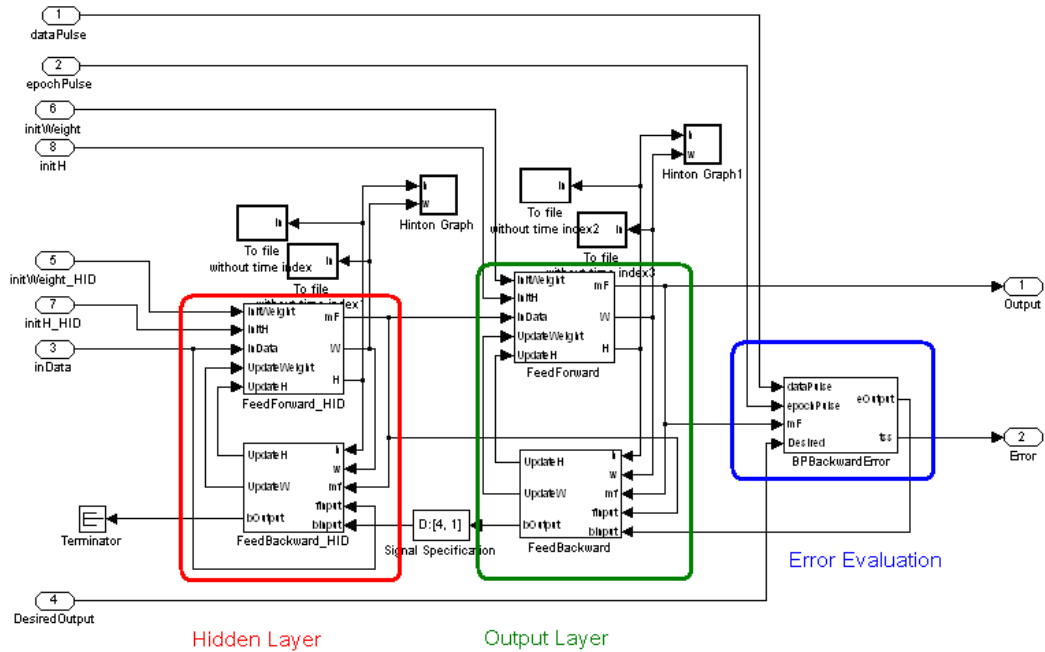
4.2.Date Pusher



Data Pusher's main responsibility is to produce a row of data set iteratively by dataPulse. Additionally, dataPulse and epochPulse are produced as indicators how much simulation is progressed. Basic components of this module are counters. Using a counter and relational logics, it represented nested "for" loops; the loop of system data pulse is nested in the loop of system epoch pulse.

5. 2-Layer BackPropModel and Overall diagram





This two layer BackPropModel has three parameters:

Size; size vector of neural network (for FeedForward Module),

Lerning Rate; (for FeedBackward Module),

StopError (for BPBackwardError Module)

The explanation of each parameter can be reached in the specified module.

In the two special blocks should be noticed. To prevent of warning message about dimension, signal specification block is used. It asserts the size of signal. Another is a terminator. It is a pseudo ender of unconnected output ends.

In the diagram, two hintonG blocks are added in order to show weight matrixes and threshold matrixes. Another display-related block is “to file without time index” block. Using this block we can track the finally trained weight matrixes and threshold matrixes at the end of simulation.

Here is a suggested simulation process.

- ① NslInit: To initialize NSL system
- ② **TrainMode**: Change NSL-mode to Train Mode

- ③ Set parameters in the diagram
- ④ Load xor_3_data.txt: load a training data set as specified in the model
- ⑤ Start simulation in simulink
- ⑥ **RunMode**: Change NSL-mode to Run Mode
- ⑦ Load xor_3_data.txt: load a verifying data set. In this case, it is same; however, a verifying data set is provided.
- ⑧ Change maxEpoch in the diagram to 1: verifying phase does not have to run many times. Because it is not a training phase, the matrixes memorized in the model are not changed.
- ⑨ Load bp.mat: load trained and valuable weight and threshold matrixes.
- ⑩ Change parameters in the diagram.
- ⑪ Start simulation in simulink
- ⑫ Get an error value in the scope or workspace

5.1 BpInit, a configuration script

Instead setting values in a simulink diagram, there is another easy and convenient way. Using the fact that every variable in the workspace can be used in the simulink as a constant, parameters can be defined in the workspace before simulation. Furthermore, this declaration can be written in .m file as a script.

First, this script should declare “parameters” as global. If they are not declared globally, it does not appear at the workspace. Then initialize those values.

```
% 2 Layer BacpPropgationModule Init Script

% Declaration of parameters, Do not touch this
global sizeOfInput
global sizeOfOutput
global dataSetSize
```

```

global dataSet;
global maxEpoch;
global learningRate;
global stopError;
global saveFileName;
global sizeOfInputLayer;
global sizeOfHiddenLayer;
global sizeOfOutputLayer;
global whidden;
global hhhidden;
global woutput;
global houtput;

% Data set configuration
sizeOfInput=3;
sizeOfOutput=1;
dataSetSize=8;
dataSetFilename='xor_3_data.txt';

% Learning parameters configuration
maxEpoch=2000;
learningRate=0.2;
stopError=0.1;
saveFileName='bp.mat';

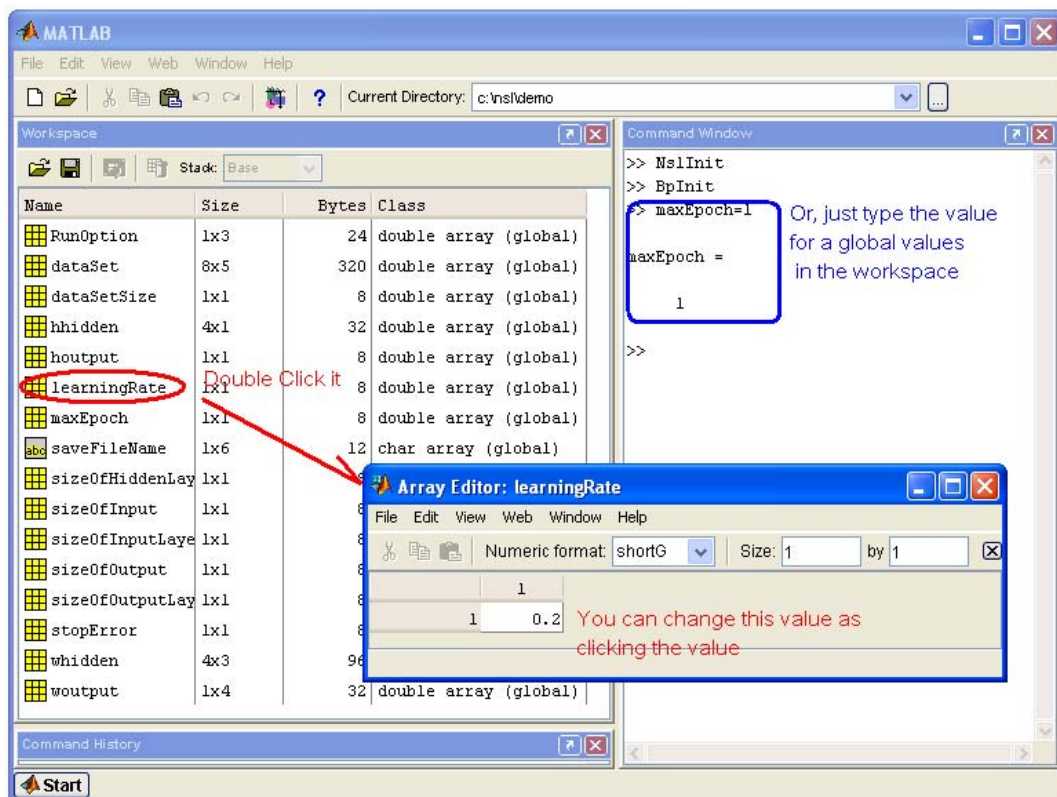
% Neural Network Settings
sizeOfInputLayer=3;
sizeOfHiddenLayer=4;
sizeOfOutputLayer=1;
% You can fill these values with your own value: whidden, hhhidden, woutput and houtput.
% Be careful of setting the size of them.
whidden=randn(sizeOfHiddenLayer,sizeOfInputLayer);
hhhidden=zeros(sizeOfHiddenLayer,1);
woutput=randn(sizeOfOutputLayer,sizeOfHiddenLayer);
houtput=zeros(sizeOfOutputLayer,1);

```

```

% Setting by NSL. Do not touch this!
eval(sprintf('load %s;', dataSetFilename));
dataSetFilename=dataSetFilename(1:(length(dataSetFilename)-4));
eval(sprintf('dataSet=%s;', dataSetFilename));
eval(sprintf('clear %s',  dataSetFilename));
clear dataSetFilename;

```



Using this script, the simulation process shown above is abbreviated like below. The advantage of this is to set parameters easily.

- ① NslInit: To initialize NSL system
- ② BpInit: To import parameters into the workspace
- ③ **TrainMode:** Change NSL-mode to Train Mode
- ④ Set parameters in the workspace: as clicking or typing in the command line

- ⑤ Start simulation in simulink
- ⑥ **RunMode**: Change NSL-mode to Run Mode
- ⑦ Load xor_3_data.txt: load a verifying data set. In this case, it is same; however, a verifying data set is provided.
- ⑧ Change maxEpoch in the diagram to 1: verifying phase does not have to run many times. Because it is not a training phase, the matrixes memorized in the model are not changed.
- ⑨ Load bp.mat: load trained and valuable weight and threshold matrixes.
- ⑩ Start simulation in simulink
- ⑪ Get an error value in the scope or workspace

5.2 How to speed up simulation

There are two ways to speed up. The first is to close the Hinton Graphs. The slowest part of this simulation is hintonG part. Initially, these graphs are instantiated whenever the simulation starts. However, you can close them without any error or warning. If the Hinton Graph for this simulation is necessary, you can use the command line function, hintonG. After finishing simulation, currently, trained weight matrixes and threshold matrixes are saved. As loading this information into the workspace, the finally trained matrixes are reachable in the command line. Using this value, the Hinton Graph can be shown.

Another method is using “accelerator mode” which is basically provided by MatLab. As clicking Simulink model window->simulation->accelerator, the model can be compiled into MEX file. This is what MatLab is proud of; they said its running speed is almost same with C. Waiting compile process, you can start simulation as same as in the normal mode.