Back Error Propagation Simulator (BPS)
Brief Tutorial

by

Emilio A. Apey

September 1989

Table of contents

1) Introduction

The BPS program simulates a multilayer neural network using back error propagation as learning algorithm. This simulator is intended for educational purposes and depends heavily on user interaction. It is important for the user to be familiar with the back error propagation algorithm before using BPS; in this way, the user can understand what the simulator is doing.

In this brief "tutorial" most of the features of BPS will be demonstrated by building a simple example. Figure 1, overview of BPS, can be very useful in becoming familiar with BPS.

Also, it is highly recommended for the user to draw the topology of the network before using the simulator, indicating the layers numbers and the unit numbers. This will be helpful when the program asks for information about layers and units. To have an idea of a network topology built by BPS, see figure 2.

2) Walk-through an example

A simple example is to teach a network the XOR (exclusive OR) mapping, which is the following

input	**output**
0 0	**0**
0 1	**1**
1 0	**1**
1 1	**0**

For this example there are four "training patterns," one for each mapping that the network has to learn. Each training pattern consist of an "input vector" and an "output vector."

The first thing to do, before entering BPS, is to write a "training set file" and a "productions file" using an editor. The format of the training set file is very flexible; the restrictions are that an input vector has to precede its corresponding output vector, there must be at least one blank character separating each number, and there can be no blank lines.

Figure 1 overview of BPS

Now, for BPS to give meaningful results the user has to be careful that the dimension of the input vectors matches the number of units in the input layer of the network, and that the dimension of the output vectors matches the number of units in the output layer. If they do not match, BPS will seem to be working normally, but the results will be meaningless. Here are three training set file formats that could be used in this example

```
0.0  0.0  0.1
0.0  1.0  0.9
1.0  0.0  0.9
1.0  1.0  0.1
```

Format 1

```
0.0  0.0
0.1
0.0  1.0
0.9
1.0  0.0
0.9
1.0  1.0
0.1
```

Format 2

```
0.0
   0.0
      0.1
0.0
   1.0
      0.9
1.0
   0.0
      0.9
1.0
   1.0
      0.1
```

Format 3

The productions file is similar to the training set file, but the output vectors are omitted; the network has to produce them. As with the training set file, the format is flexible, but there can be no blank lines.

```
0.0  0.0
0.0  1.0
1.0  0.0
1.0  1.0
```

Format 1

```
0.0
0.0
0.0
1.0
1.0
0.0
1.0
1.0
```

Format 2

```
0.0
   0.0
0.0
   1.0
1.0
    0.0
1.0
   1.0
```

Format 3

For the XOR example the training set file will be named XOR.DAT and the productions file will be name XOR.PRO.

From the training patterns it can be seen that the network needs to have two input units (the dimension of an input vector) and one output unit (the dimension of an output vector). The number of hidden layers and hidden units is up to the user; it is good practice to keep networks as small as possible because it takes less time to train them,

since there are fewer weights to update. However, if they are too small they might not be able to learn all the training patterns; this is one of the artistic aspects of neural networks. The simplest topology that can learn the XOR mapping is the following

Figure 3 XOR network topology

Now it is time to get the program running and start doing something with this network. To run the program in the VAX 8530 type BPS. If you have the PC version, select the executable version that best fit your computer (bps286.exe, bps8088.exe) and type the name without the extension.

```
gmuvax> bps
```

Once the initial message appears, hit return to enter the main menu.

3) Main menu

In this menu is where the user spends most time, here networks are defined and then they are trained. Under this menu there are three other menus the production menu, the edit and view menu, and the transfer function menu; they will be described later on.

3.1) Build/load network (b)

The first thing to do when using BPS is to define a network, because many functions depend on the network topology. To define a network choose "b" (for build) and answer the questions. Remember to hit the return key after each entry.

Supposed that you have been training a network for a while and then decide to create or load a new network. BPS will ask if you want to save the old network. Then, BPS will ask if you want to load a network from your working directory

Do you want to load a network from file? (y/n) n

To load a network answer "y"; then BPS will ask you for the name of a previously defined network. The network is loaded and BPS returns to the main menu.

In the case that you want to define a new network, then answer "n". Next BPS will ask if you want to initialize the network with random weights.

Do you want random weights? (y/n) y

When the answer is "y", BPS will completely connect all the units of adjacent layers and the initial weights of the edges will be in the range that you will soon specify. When the answer is "n", BPS will not connect any unit in the network at all; it will create the layers and units, but there will not be any edges. To connect the units you would go to the Edit and View menu and add edges between the units that you want to be connected. So, at this point, BPS would create only a skeleton of the network.

In the XOR example two edges will be added later on, an edge going from the first unit in the first layer to the first unit in the third layer, and an edge going from the second unit in the first layer to the first unit in the third layer (see fig. 3). But the rest of the network is fully connected, so at this point answer "y", you do want edges with random weights between the layers.

Then BPS will ask you for the lower and upper limit of the random weights. A good range is -0.34 and 0.34, that is -1/3 and +1/3.

Give lower limit of random interval -0.34
Give upper limit of random interval 0.34

To initialize the weights with numbers close to zero is good practice because if the weights are large it is very difficult, sometimes impossible, for BPS to change those weights.

Next, BPS will ask you how many layers of units you want. A typical back propagation network has three layers, but if you want to experiment with more layers BPS allows you to build networks with more layers. From figure 3 you can see that the network has three layers, then answer "3".

Enter number of layers 3

Now the number of units for each layer has to be specified. BPS will start asking the number of units for the first layer (input layer) up to the layer number specified in the total number of layers (output layer). Be careful that the number of units in the input layer and output layer matches the dimensions of the input and output vectors specified in the training set.

Number of nodes for layer no. 1 2
Number of nodes for layer no. 2 1
Number of nodes for layer no. 3 1

Since two units are needed for the first layer type "2"; for the second layer one unit is needed, so type "1"; and for the third layer one unit is needed; type "1".

BPS now returns to the main menu and it has given the default name of "unnamed" to the network. By now the network looks like this

Figure 4 XOR partial network topology

3.2) Specify training set (t)

To specify or change the training set, choose "t" (for training set) from the main menu. BPS will ask you for the name of the file where you wrote the training patterns.

Enter the file name of the training set xor.dat

3.3) Edit and View menu (e)

When you choose "e" (for edit and view) in the main menu the "edit and view menu" appears. In this menu you can inspect the network and modify it. The options on the left of the menu are to modify the network and the options on the right of the menu are to inspect the network.

Choose "a" (for activities and thresholds) to see the activities and thresholds of the units in the network. The screen looks like this

Layer	Node		Activity	Threshold
1	1	---->	0.000000	0.000000
1	2	---->	0.000000	0.000000
2	1	---->	0.000000	0.000000
3	1	---->	0.000000	0.000000

From this screen you can see that the first layer has two units (1,2), the second layer has one unit, and the third layer has one unit. Since this network was just defined, the activities and thresholds of the units are all zeros.

Now choose "w" to inspect the weights in the edges; this is one of the most used options in BPS. The screen will look like this

Layer	Node	Weight	Node	Layer
1	1	-0.20023	1	2
1	2	0.23874	1	2
2	1	0.18973	1	3

The layers and units on the left are the source units, and the layers and nodes on the right are the target units. From this screen you can see the weights between two units.

The XOR network needs two more edges, so choose "e" (for edge) to add a new edge. Edges in the network have direction; that is they come out of one unit and they

go into another unit; to add an edge you have to specify the source unit and the target unit. Since BPS handles multilayer networks, each unit location is defined by the number of the layer where the unit is and the unit's number in that layer; see figure 3 for layer and unit numbers. The following entries are to add an edge from the first unit in the first layer to the first unit in the third layer

Layer number of source unit 1
Number of source unit 1

Layer number of target unit 3
Number of target unit 1

Do you want a random weight? (y/n) y
Change random weight range? (y/n) n

The given weight was -0.15652

As you can see, BPS will ask you if you want a random weight; if you answer "n" BPS will ask you for a specific weight that you want to give to that edge. If you decide to add a random weight, you can change the range for the new weight.

Now that a new edge has been added, you can, if you want, check it by choosing "w" (for weights) and see if the new edge is really there.

To add the second edge from the second unit in the first layer to the first unit in the third layer choose "e" again; and respond to the prompts, but this time the unit number of the source unit is 2. After you have done this you can check the network by choosing "w" again. At this point the network topology should look like figure 3 above.

The remaining options in the edit and view menu work similarly. When modifying the network BPS will ask you about the location of the node(s) involved, information that you can get from a diagram of the network that you are building, like figure 3. The "t" (for training set) choice is to see the training set that you specified without having to leave BPS.

Now you can leave this menu by choosing "q"; this will put you back in the main menu.

3.4) Save network (s)

At this point that the network is complete and you may want to save it. When a network is saved a name is given to the network; also, all the parameters that the network has at that time are saved, including the number of epochs that it has been trained. In this way, when a network is loaded, all the parameters are loaded into BPS at the same time.

Name for file to save network xor211.net

The user can give any name to the network; XOR211.NET describes the network as with two input units, one hidden unit, and one output unit.

3.5) Find error (f)

If you want to know what is the current value of the network error function, choose "f" (for find error) from the main menu. The new error will be displayed in the main menu where it says "error." If you do not choose "f" and keep training the network, the error displayed in the main menu is not the present error, but it is the error when you requested it the last time; this operation was designed to be user activated.

Initially the error will be zero, but this does not means that the error of the network is zero; this is just an initialization.

3.6) Number of epochs (n)

This number has two different purposes, depending on which learning mode you choose to train the network. One of these modes is to let the learning algorithm to cycle until a specified error is reached; see next section for the other mode. "Cycle until converges" means that the network will be train until an user specified error is reached. In case that the user specified error can not be reached, BPS will exhaust the number of epochs specified when "n" was chosen. In other words, the network will be train until either it reaches an error or it uses all the specified epochs.

When you choose "n", BPS will ask you for the number of epochs to train the network.

Enter the number of epochs to train the network 1000

With what we have specified up to now we could attempt to train this network, but let's see the other options first.

3.7) Cycle until converges (c)

When "c" is choose, you can turn the "cycle until converges" mode ON or OFF. If you turn it ON, then BPS will ask you for the target error; if you turn it OFF, BPS will return to the main menu.

The other mode of learning is active when "cycle until converges" is OFF. It will train the network for the specified number of epochs, without paying attention to the error of the network.

For the XOR example leave "cycle until converges" OFF; so by now you do not need to choose "c".

3.8) Adjust learning parameters (a)

This is the fun part of BPS, adjusting the learning parameters and see with which set of parameters the network learns faster, or if it learns at all. The back propagation algorithm uses two parameters to adjust the weights in a network eta (learning rate) and alpha (momentum factor); see section 4. BPS has a speed up mechanism that adjusts these two parameters after each epoch; this mechanism introduces two new variables an upper bound for eta and an increasing rate for eta (see section 4).

These four variables can be set by the user. They default to conservative values, so most networks will slowly learn the patterns. For some topologies a larger bound for eta, 1.0, will allow the network to learn faster; instead for other topologies, a large eat would lead them to instability and their weights will become very large. By experience it has been found that if a network's weights after a few thousand epochs have become large (above 10) and the error is still large, probably that network will not learn the mapping. However, if after a few thousand epochs the network has not learn the mapping but its weights are still small (below 2), there is still a chance for the network to learn the mapping.

It is good practice to save a network as soon as you define it. In this way, if the network does not learn with a set of learning parameters, you can load the original network and try another set of learning parameters; this allows you to start from the same initial state. If the network still does not learn, them you can try another initial state (another set of random weights) by creating a new network. And if the network still does not learn, then you can try another topology add units or edges.

As you can see, there is some testing to be done with these networks; but the more experiments that you do, the more familiar you will become with their behavior.

When you choose "a" (for adjust learning parameters), you can change the values of these four variables eta, alpha, bound of eta, and increasing rate of eta. For the XOR problem the default values will be used; so once again, you do not need to choose "a".

3.9) Production menu (p)

Before training the network let's see what kind of output it is producing. To do this choose "p" (for production menu) from the main menu. In the production menu you can do productions (propagate input vectors through the network to obtain output vectors)

from a file or from the keyboard. You can also save those productions in a file. All these possibilities are shown in this menu.

To do productions from the keyboard choose "k"; BPS will give you the number of input units (dimension of input vector) and ask you for the input vector

Input vector 0.0 0.0

Output vector 0.52343

The terms of the input vector need to have a space in between, you can also hit return and give more terms in the next line. The output at this time will be around 0.5 because the network has not been trained yet. Try other input vectors and you will see that all of the output vectors are around 0.5.

Sometimes you want to give a large set of productions to the network; that is why the production file was created, because you will get tired of typing input vectors. So you can specify a production file where you have the set of productions that you want to do. To set the production file choose "p" and BPS will ask you

Enter file name xor.pro

BPS will remember this file, so you do not have to specify it every time that you enter the production menu. Sometimes you may want to have several productions file for the same network; to change the production file just choose "p" again and give the name of the new file. The "v" (for view) option is to view the production file, to check if it is the right set of productions that you want to test.

After you have specified a production file you can choose "f" (for productions from file) and get many productions at once.

Suppose that you want to save productions when you do them either from a file or from the keyboard. You can tell BPS to open a file to write the productions. To do this choose "s" (for save); this choice is used to open and close a file. If you want to open a file respond "y" and BPS will ask you

Save productions in a file? (y/n) y

Enter file name to save productions xor.out

If you respond "n" and a saving file was open, BPS will close the file. However, you do not need to close this file specifically; if you leave the production menu and a saving file was open, BPS will close it for you.

Now you know everything about the production menu, so you can choose "q" to return to the main menu.

3.10) Learn (l)

Finally the network is ready to start the training. For this choose "l" (for learning) and depending on the state of the "cycle until converges" switch (see section 3.6), the learning will be in either of the two modes. Since we left this switch in OFF for this example, the learning will exhaust the specified number of epochs; the display while the network is being trained will look like this

```
20          eta   0.06        error   0.32012345
40          eta   0.10        error   0.32012201
60          eta   0.45        error   0.31999982
80          eta   0.60        error   0.31999873
 .            .                 .
 .            .                 .
```

From this display you can see how the epoch number increases, how eta changes and (hopefully) how the error decreases.

Probably for the first 1000 epochs the error will stay around 0.32, and may be up to 3000 epochs. But when the error starts decreasing it goes very fast. If the learning rate is high (around 0.95), the error will decrease up to a point and then stay there; if the learning rate is too high (above 1.0), the error will decrease rapidly up to a point and then it will start increasing. You may even get a running time error, "floating point excepting"; this is because the weights became extremely large. You can train the network for a while and then go to the "edit and view" menu to see how the weights have changed. Try training the same initial network with several learning parameters and see what kind of results you obtain.

These exercises will help you to get an idea of how the back propagation algorithm works so you can do bigger applications afterwards.

3.11) Transfer function options menu (m)

You enter this menu by choosing "m" from the main menu. This menu has two options to change the amplitude of the sigmoid function (M) and to shift the sigmoid function downward so it is symmetrical around the x-axis.

Figure 5 Sigmoid function with range 0.0, M and -M/2, M/2

By default the range of the sigmoid function is between 0.0 and 1.0. When you select "m" in this menu BPS will ask you for the new upper limit of the sigmoid function

```
Enter new value  2.0
```

Now the range of the sigmoid function is [0.0,2.0] instead of being in the range [0.0,1.0]. Note that to be able to train a network with this new amplitude, you have to define the training patterns to be in this range also.

The other option is to make the sigmoid function symmetrical with respect to the x-axis; this option can be switched ON and OFF by choosing "s" in this menu. Note that the range of the sigmoid function will be between -M/2 and M/2. For example, if you specify M = 2.0, then the range of the sigmoid function would be -1.0 and 1.0.

4) Formulas

To give you more insight into what BPS is doing while a network is being trained, this section describes some of the main formulas of the learning algorithm.

In these formulas p stands for training patterns. Then the summation over p means one epoch; all the training patterns have to go through. Symbols for the units are j, i, and k. The j is for the unit which parameters are being calculated; the i is for units in the previous layer, and the k is for units in the next layer.

Figure 6 units and subscripts

Weight adjustment

$$\Delta W_{ji}(t+1) = \eta \sum_p (\delta_{pj} \; O_{pi}) \quad + \quad \alpha \, \Delta W_{ji}(t) \qquad (1)$$

where

$\delta_{pj} = (T_{pj} - O_{pj}) (O_{pj}/M) (1 - O_{pj}/M)$ for output units.

$\delta_{pj} = (O_{pj}/M) (1 - O_{pj}/M) \sum_k (\delta_{pk} W_{kj})$ for hidden units.

O unit activity level.
T output unit target.
M sigmoid function amplitude.

A difference between formula 1 and the original back propagation formula is the summation of the products of the deltas and previous unit activation over all the training patterns, p. By doing this operation the weights will be updated only once per epoch of training. This is the first part of a two parts method proposed by Vogl to accelerate learning by back propagation.

Transfer function

$$O_{pj} = M \left(\frac{1}{1 + e^{- \left(\sum_i W_{ji} O_{pi} + \theta_j \right) / M}} - 1/2 \right)$$

Error function

$$\text{error} = \sum_p \left[1/2 \sum_j (T_{pj} - O_{pj})^{**2} \right]$$

Eta and alpha adjustment

C (t) = << $\Delta W_{ji}(t-1)$, $\Delta W_{ji}(t)$ >> Correlation.

if (C(t) < 0) then
  η --> 0.01
  α --> 0.00
else
  if (C(t) < (C(t-1) + 0.05 * C(t-1)) or η < eta bound) then
    η --> η + eta rate
    α --> α + 0.01
  endif
endif

The correlation gives an indication of how the weights are changing. If this term becomes negative, then eta and alpha are decreased drastically to 0.01 and 0.0 respectively. If the correlation is positive, then eta and alpha have passed the first test to be increased. The second test to increase eta and alpha is that if eta is below its maximum allowed value (its bound) and if the correlation is within the last correlation plus five percent.

The original correlation test was implemented by David Schreibman and consisted in checking if C(t) was either positive of negative; if it was negative then the values of eta and alpha were dropped to low values, and if it was positive eta and delta were increased. The additional test of checking if the correlation is increasing or decreasing was added to give more stability to the learning.