Demonstrating the PRM over the toy problems of Section 2 of the paper "Merging race models and adaptive networks: A parallel race network"

Denis.Cousineau@UMontreal.ca

The purpose of this notebook is to demonstrate the use of the Parallel Race model sample programs. PRMod has no learning capabilities and the delay matrix and threshold vector must be provided. PRMod starts with random delays and learn by exposure to input-output pairs. The command ERROR[%] plots the percent of errors made by the network.

See more at http://prelude.psy.umontreal.ca/~cousined/papers/07-PRM

0- Loading the packages and defining a simple noise function.

```
SetDirectory["c:\Mes documents\Papers\
    XX-Submitted-XX\\07-n-minnetwork\TON_program"];
```

```
<< tools.m;
<< stimuli.m;
<< PRNet.m;
<< PRmod.m;
```

```
Tools loaded correctly.

Stimuli loaded correctly; use ? Stimuli for a list.

PRNet loaded correctly; use ? PRNet for more

PRMod loaded correctly; use ? PRMod for more
```

```
noise[x_] := noisypairs[x, ExponentialDistribution[1], 0, 0, 0, 1000];
AppendTo[Attributes[noise], HoldAll];
```

1: The Detection problem

In the Detection problem, there is only one input that is either present (0) or absent (late, 100). When present, the accumulator A must be triggered, or else, the B accumulator must fire (using clocks nodes).

a) The input-output paires:

```
NiopairsDetect // TableForm
```

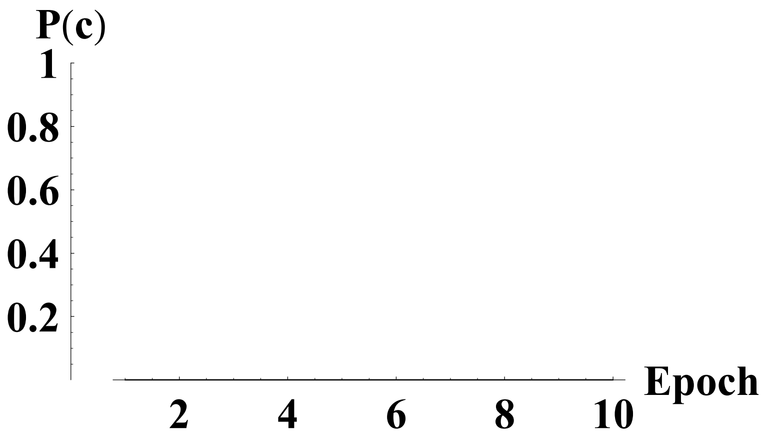
```
        0
100     1

        1
0       0
```

b) A simple test with no noise.

```
myD = {{0, ∞}, {∞, 1.}};
myK = {1, 1};
myD // MatrixForm
```

$$\begin{pmatrix} 0 & \infty \\ \infty & 1. \end{pmatrix}$$

```
PRMod[NiopairsDetect, myD, myK];
ERROR[%];
```

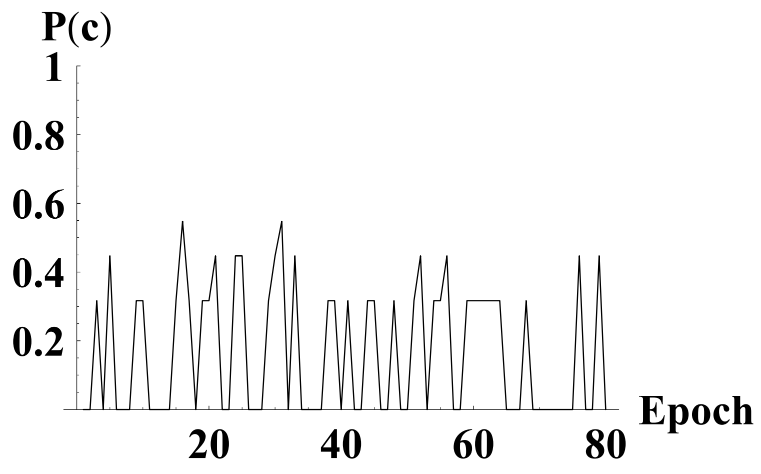


c) test with noisy inputs

```
noise[NiopairsDetect] // TableForm
```

```
                0
103.872         1

                1
0.249194        0
```

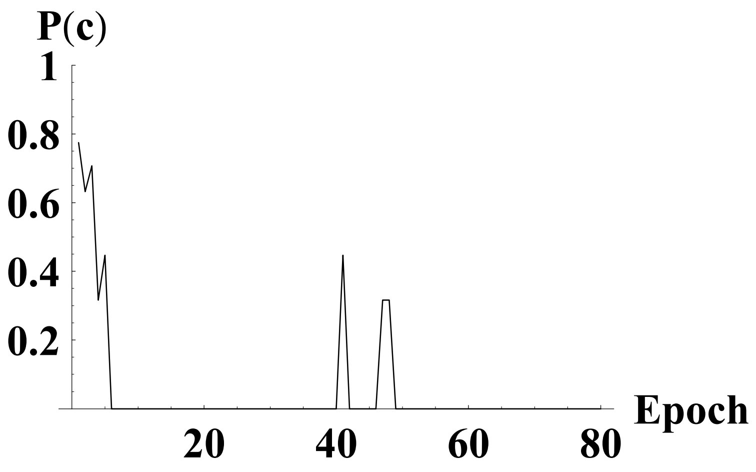
```
θ = 2;
PRMod[noise[NiopairsDetect], θ myD, myK, 800];
ERROR[%];
```



d) test with noise and learning

```
PRNet[noise[NiopairsDetect], 800, 0];
ERROR[%];
```

```
============================================
Starting Speeds D and threshold K are:
⎛0.00590126  0.0535011⎞
⎝ 0.0553557  0.0679895⎠2x2
{1, 1}

============================================
All done; the final matrix D and threshold K are:
⎛0.00590126  0.553501⎞
⎝  5.55536    5.06799⎠


{1, 1}
```



2: The 1D problem

The 1D problem has two actual inputs (plus the clock, of course) but one of the input is irrelevant and should be ignored by using large delays.

a) Input-output pairs

```
Niopairs1D // TableForm
```

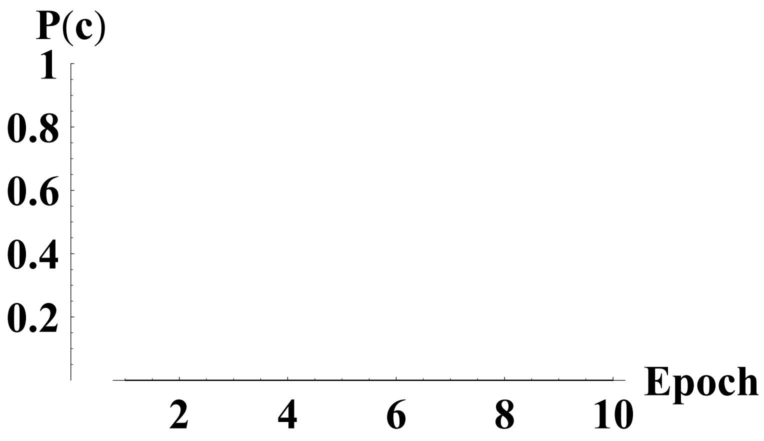
```
100      0
100      1

100      1
0        0

0        0
100      1

0        1
0        0
```

b) A simple test with no noise

```
myD = {{∞, 1.}, {0, ∞}, {∞, 1.}};
myK = {1, 1};
myD // MatrixForm
```

```
⎛∞  1.⎞
⎜0   ∞⎟
⎝∞  1.⎠
```

```
PRMod[Niopairs1D, myD, myK];
ERROR[%];
```

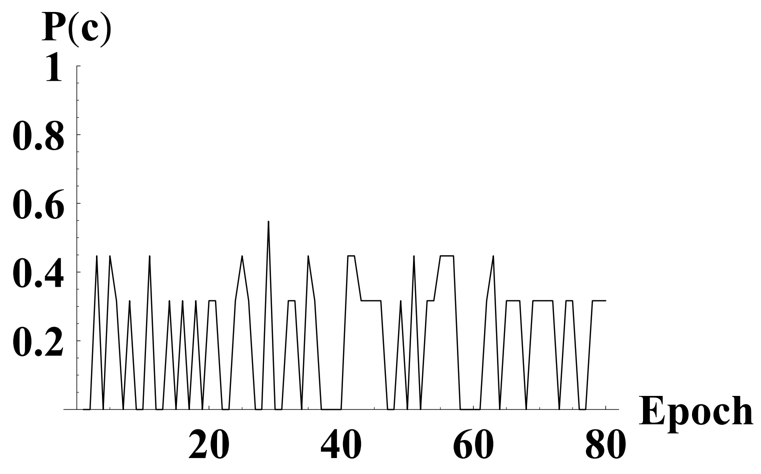


c) test with noisy inputs

```
noise[Niopairs1D] // MatrixForm
```

```
⎛ ⎛100.726 ⎞  ⎛0⎞ ⎞
⎜ ⎝100.191 ⎠  ⎝1⎠ ⎟
⎜ ⎛103.826 ⎞  ⎛1⎞ ⎟
⎜ ⎝0.439998⎠  ⎝0⎠ ⎟
⎜ ⎛0.413144⎞  ⎛0⎞ ⎟
⎜ ⎝101.413 ⎠  ⎝1⎠ ⎟
⎜ ⎛0.960479⎞  ⎛1⎞ ⎟
⎝ ⎝0.675308⎠  ⎝0⎠ ⎠
```

```
θ = 2;
PRMod[noise[Niopairs1D], θ myD, myK, 800];
ERROR[%];
```

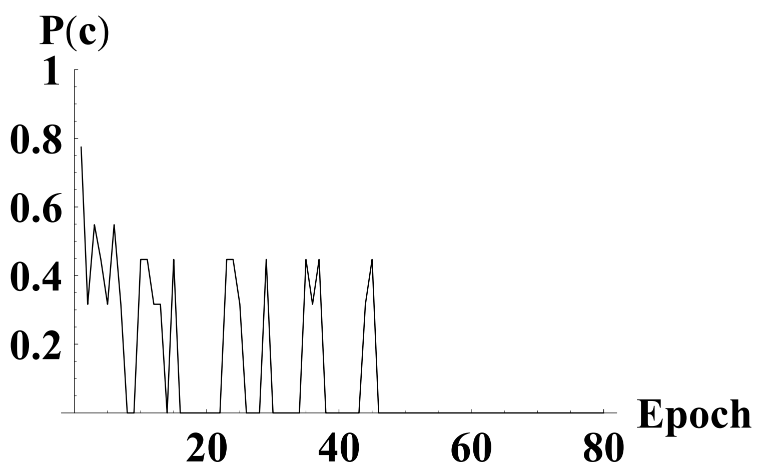


d) test with noise and learning

```
PRNet[noise[Niopairs1D], 800, 0];
ERROR[%];
```

```
============================================
Starting Speeds D and threshold K are:
⎛0.0140157  0.0981703⎞
⎜0.0401236   0.002351⎟2x3
⎝0.0926825  0.0791982⎠
{1, 1}

============================================
All done; the final matrix D and threshold K are:
⎛  4.01402   3.59817⎞
⎜0.0401236   2.00235⎟
⎝  5.59268    5.0792⎠
```

$\left\{\frac{9}{5}, 1\right\}$



3: The Identification problem

The Identification problem has two actual inputs (plus the clock, of course) but -this is the new thing- three outputs, A, B, or C.

a) Input-output pairs of stimuli

```
NiopairsIdent // TableForm
```

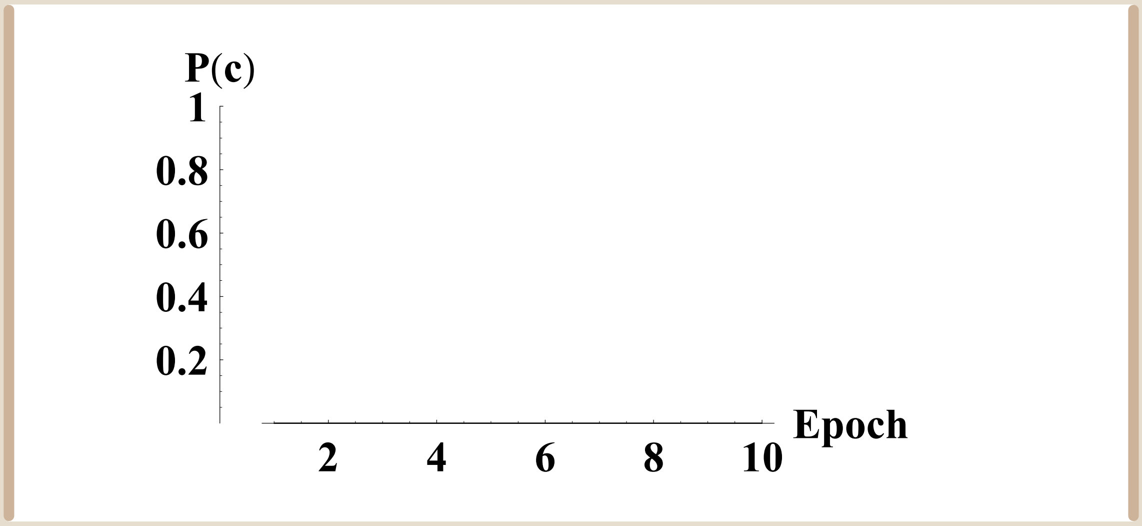
```
            0
100         0
100         1

            0
100         1
0           0

            1
0           0
100         0
```

b) simple test with no noise,no learning

```
myD = {{0, ∞, ∞}, {∞, 0, ∞}, {∞, ∞, 1.}};
myK = {1, 1, 1};
myD // MatrixForm
```

$$\begin{pmatrix} 0 & \infty & \infty \\ \infty & 0 & \infty \\ \infty & \infty & 1. \end{pmatrix}$$

```
PRMod[NiopairsIdent, myD, myK];
ERROR[%];
```

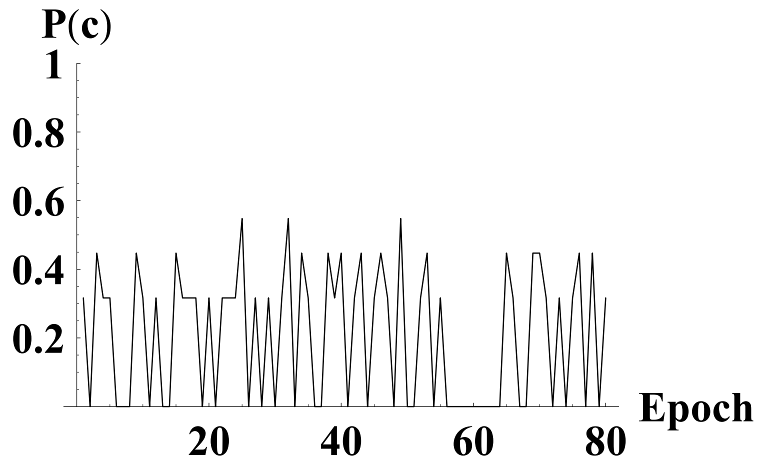


c) test with noisy inputs

```
noise[NiopairsIdent] // TableForm
```

```
             0
100.746      0
102.455      1

             0
100.919      1
2.5969       0

             1
1.82571      0
101.715      0
```

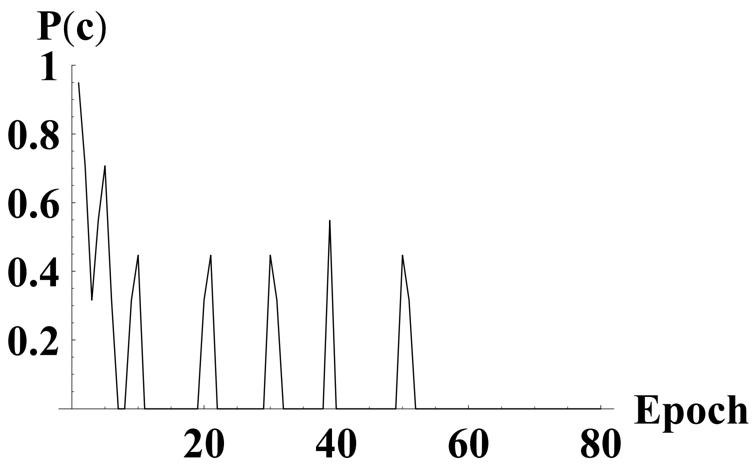
```
θ = 2;
PRMod[noise[NiopairsIdent], θ myD, myK, 800];
ERROR[%];
```



d) test with noise and learning

```
PRNet[noise[NiopairsIdent], 800, 0];
ERROR[%];
```

```
=============================================
Starting Speeds D and threshold K are:
⎛0.0166923  0.0241408  0.00411427⎞
⎜0.0239465  0.0523135  0.00474248⎟3x3
⎝0.0423116  0.0397665   0.098257 ⎠
{1, 1, 1}

=============================================
All done; the final matrix D and threshold K are:
⎛0.0166923   0.524141   0.504114⎞
⎜ 0.523946  0.0523135  0.504742⎟
⎝  6.04231    6.03977   5.59826⎠


{1, 1, 1}
```



4: The AND problem

The AND problem consists in deciding whether a conjunction of input is present or not. For that reason, the accumulator size for the conjunction response must be 2.

a) Input-output pairs of stimuli

```
NiopairsAnd // TableForm
```

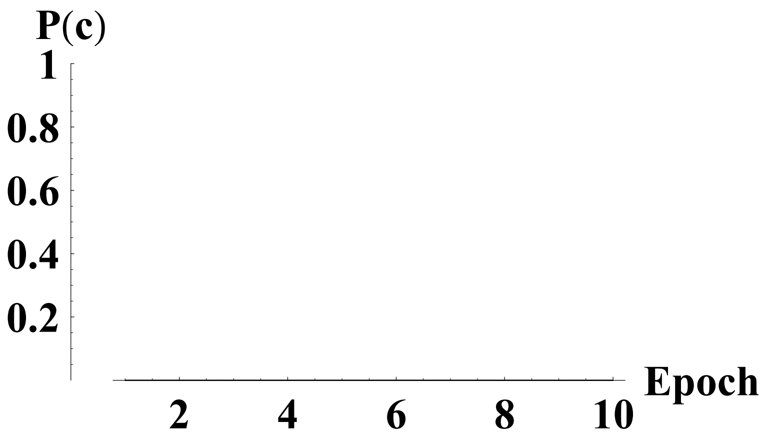
```
100      0
100      1
100      0
0        1
0        0
100      1
0        1
0        0
```

b) simple test with no noise,no learning

```
myD = {{0, 1.}, {0, 1.}, {∞, 1.}};
myK = {2, 1};
myD // MatrixForm
```

$$\begin{pmatrix} 0 & 1. \\ 0 & 1. \\ \infty & 1. \end{pmatrix}$$

```
PRMod[NiopairsAnd, myD, myK];
ERROR[%];
```

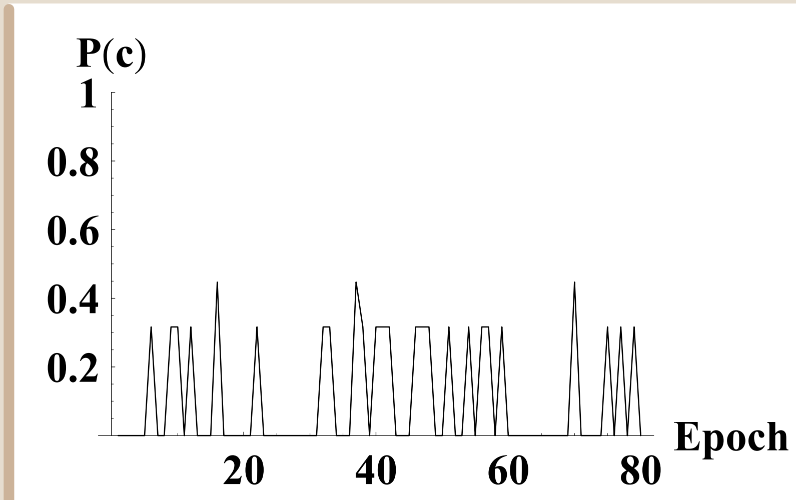


c) test with noisy inputs

```
noise[NiopairsAnd] // TableForm
```

```
100.152       0
100.088       1
100.041       0
0.216094      1
0.0709063     0
100.636       1
0.63039       1
1.18471       0
```

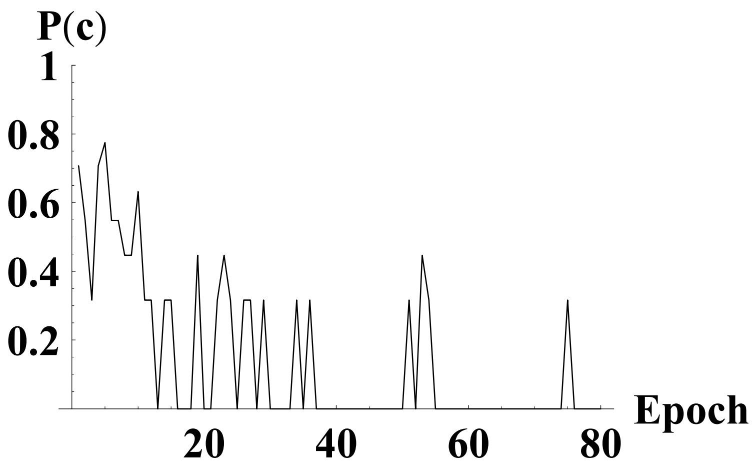
```
θ = 2.5;
PRMod[noise[NiopairsAnd], θ myD, myK, 800];
ERROR[%];
```



d) test with noise and learning

```
PRNet[noise[NiopairsAnd], 800, 0];
ERROR[%];
```

```
==============================================
Starting Speeds D and threshold K are:
⎛ 0.0893503   0.0878961⎞
⎜ 0.0060509    0.057663⎟2x3
⎝0.00321809   0.0681414⎠
{1, 1}

==============================================
All done; the final matrix D and threshold K are:
⎛1.58935   5.5879 ⎞
⎜1.50605  5.05766⎟
⎝7.00322  6.56814⎠


{2, 1}
```



5: The XOR problem

The XOR problem is a classic that can't be solved by two-layered neural networks. It involves the same response for none or both input present. It is also special since this problem absolutely requires a redundant clock.

a) input-output pairs

```
NiopairsXor // TableForm
```

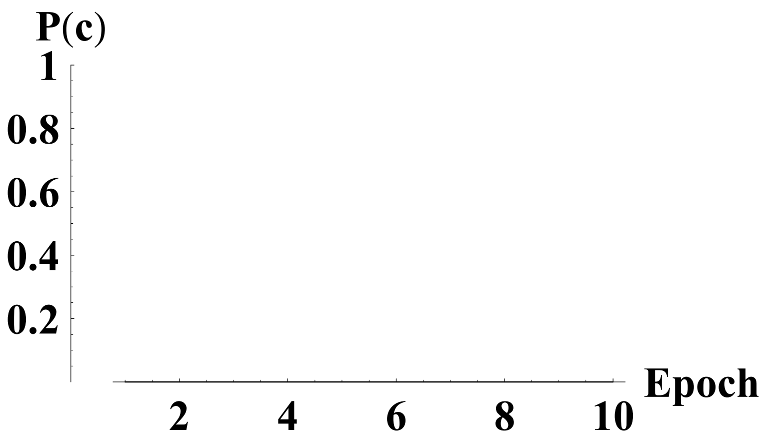
```
100      1
100      0

100      0
0        1

0        0
100      1

0        1
0        0
```

b) simple test with no noise, no learning

```
myD = {{0, 1.}, {0, 1.}, {2, ∞}, {2, ∞}};
myK = {2, 1};
myD // MatrixForm
```

```
⎛ 0  1. ⎞
⎜ 0  1. ⎟
⎜ 2  ∞  ⎟
⎝ 2  ∞  ⎠
```

```
PRMod[NiopairsXor, myD, myK, 100, -∞, 2];
ERROR[%];
```

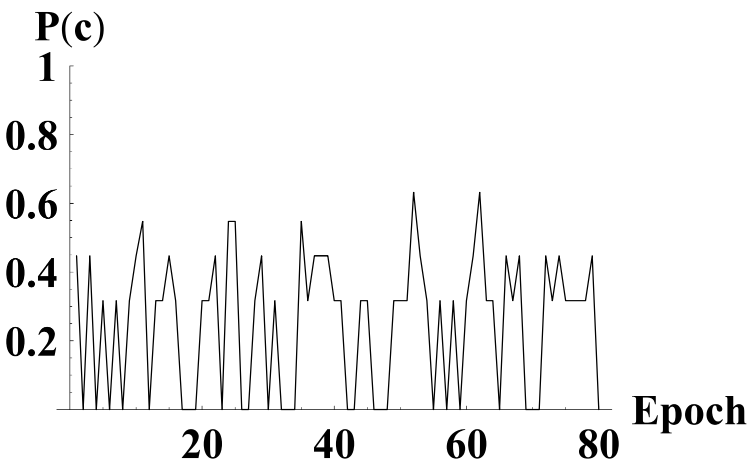


c) test with noisy inputs

```
noise[NiopairsXor] // TableForm
```

```
100.551      1
101.979      0

103.014      0
0.202654     1

0.702024     0
104.534      1

1.29716      1
0.928142     0
```

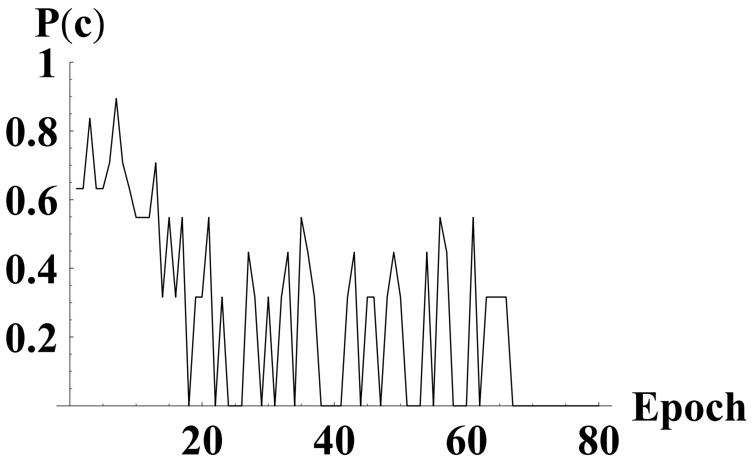
```
θ = 2;
PRMod[noise[NiopairsXor], θ myD, myK, 800, -∞, 2];
ERROR[%];
```



d) test with noise and learning

```
PRNet[noise[NiopairsXor], 800, 0, 2];
ERROR[%];
```

```
============================================
Starting Speeds D and threshold K are:
⎛0.0764874  0.0659359⎞
⎜0.0647003  0.0280941⎟
⎜0.0462327  0.0717618⎟2x4
⎝0.0816378  0.0107796⎠
{1, 1}

============================================
All done; the final matrix D and threshold K are:
⎛ 1.07649  6.06594⎞
⎜ 1.5647   5.02809⎟
⎜10.0462  10.5718 ⎟
⎝10.0816  10.5108 ⎠


{2, 1}
```



End of the solutions to Section 2.