Parallel Race Network

Author: Denis Cousineau,
Denis.Cousineau@UMontreal.CA

Note: double-click on the arrow on the right-side to view the content of a section; use shift-e

0- Overview of the network

■ 0.0- History

First version TON.m	**©1998: implemented using division for noise.**
Second version TON-prod.m	**©1999: using production ^ for noise**
Third version TON3.m	**©1999: using sum * for the noise functions.**
Fourth version TON-3+uniform.m	**©2000: final sum version, with uniform noise in the delays.**
Fifth version PRNet.m	**©2000: same as above, but with some cleaning...**
Sixth version PRNet.m	**©2002: increased performance by 5%...**

■ 0.1- Neural networks with firing mechanics

The network presented here uses a different mechanic: there is two layers, one layer is the detector lay communicates to the decidors through firing of impuls. What is important is to coordinates *when* each dete

■ 0.2- Not an additive transmission rule

Here's an allegory of what does a decidor in a standard, competitive network: "He ask the question to After a while, he then weighted the opinion of each by the importance of the persons, and based on the overal

■ 0.3- Not a competitive network

A network with a race-based decision rule is no longer a competitive network. Detectors have to orga other detectors to respond, in case they saw an input. they also implement a special type of input, the time-ou

■ 0.4- A cooperative network using race-based decision rule is an accumulator mode

If the firing of each detector is a random variable, then each decidor will receive an increasing amoun processes, the decidors will learn to adopt larger and further apart thresholds. This is formally equivalent to a system, and variability and the form of the variability are free parameters.

1- Initialization section

```
SetDirectory["c:\mes documents\Papers\XX-Submitted-XX\\07-N-MinNetwork\TON
```

```
<< 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
```

2- Obtaining help for the various functions

■ 2.1- Help can be obtained using the ? name.

```
? PRNet
```

```
The function PRNet is a network that learns to associate inputs with outpu
  architecture. Unlike neural net, it does so using a Min respond rule and
  Usage:
 PRNet[iopairs];  provides learning errors on input/outputs pairs. for exe
  iopairs1D]; (see ? Stimuli for help on pairs) learns to discriminate inp
 PRNet[iopairs, numIter, Debug]; Provides a number of iteration  (default
  none with -Infinity). Debug = 0 outputs begin and end values of the matr
  debug information for trials # larger than Debug. Debug =-1 outputs the
  analysis). Debug =-2 outputs the RT (for distribution analysis), and Deb
 PRNet[iopairs, numIter, Debug, NbreTimeout, omega, Psi, c]; specifies  th
  include in the simulation, it also provides a learning rate omega for th
  learning rate Psi for the change in delays (omega should  always be a ra
  with the Floor function).   Finally, c is a criterion above which no inp
 Use ERROR[PRNet[iopairs]] for a graph of the Percent correct.
 Use Drift[PRNet[iopairs, 800, -1]] for the averages of the firing speed.
```

■ 2.2- Stimuli used in the simulations

Stimuli are simple lists of Input/output couples. For example:

```
NiopairsOr // MatrixForm
```

$$\begin{pmatrix} \begin{pmatrix} 100 \\ 100 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 100 \\ 0 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 100 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix}$$

■ 2.3- The list of all stimulus pairs are:

```
? Stimuli
```

```
List of i/o pairs of Stimuli are:
 iopairsOr:  a 1 in either or both input results in response A;
 iopairs1D:  a 1 in the second dimension of the input gives a A;
 iopairsAnd:  a conjunction of 1 is necessary for response A to be made;
 iopairsXor:  a 1 in either but not both input yields a A
 N*           are the same, but with times instead of binary values
 use noisypairs to add noise to these pairs;
 use redundantpairs  to duplicate the inputs.
```

■ 2.4- Stimuli can be made redundant

```
redundantpairs[NiopairsOr, 5] // MatrixForm
```

$$\begin{pmatrix} \{100, 100, 100, 100, 100, 100, 100, 100, 100, 100\} & \{0, 1\} \\ \{100, 0, 100, 0, 100, 0, 100, 0, 100, 0\} & \{1, 0\} \\ \{0, 100, 0, 100, 0, 100, 0, 100, 0, 100\} & \{1, 0\} \\ \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0\} & \{1, 0\} \end{pmatrix}$$

■ 2.5- Stimuli can be altered or made noisy

```
? noisypairs
```

```
noisypairs modifies the input so that noise is introduced.  However, the
  will never be lower than lower bound (default 0) and never be large than
 noisypairs[iopairs] adds uniformly distributed noise in the range -.5..+.
 noisypairs[iopairs, distributionfunctionwithparameters] adds noise to the
  named as the second parameters  (you must supply parameters).  Ex: noisy
  0,1] adds normal standardized noise to the input. You can use in your di
  variable StimValue, which represents the stimulus value.  For example, n
  0,StimValue]] adds noise to the inputs whose standard deviation is equal
 noisypairs[iopairs, distribution, zero, one] is used to provide  alternat
  excitation (instead   of 1). e.g. providing 0.2 as zero parameter will r
 noisypairs[iopairs, distribution, zero, one, lowerbound, upperbound] is
  truncate the inputs in the range [lowerbound, upperboud]. Default is [0,
 noisypairs[iopairs, distribution, zero, one, lowerbound, upperbound, nonc
```

3- Using a deterministic model, PRMod

■ 3.1- First example: using a 1D problem (without and with noise).

```
DD = {{∞, 1.}, {0, ∞}, {∞, 1.}};
KK = {1, 1};
DD // MatrixForm
Mean[PRMod[Niopairs1D, DD, KK]] // N
```

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

```
0.
```

■ 3.2- Varying Delays to keep the mean errors at about 0.05

```
MyD[x_] := {{∞, x}, {0, ∞}, {∞, x}};
KK[x_] = {x, x};
MyD[XX] // MatrixForm
```

$$\begin{pmatrix} \infty & XX \\ 0 & \infty \\ \infty & XX \end{pmatrix}$$

```
{"η", "ρ", "{Errors for delays 2 power x (0..4)}"}
Table[{eta, rho,
    Table[Mean[PRMod[
       noisypairs[redundantpairs[Niopairs1D, rho], ExponentialDistribution[
       redundantdelays[MyD[2^x], rho, 1], KK[rho], 100, -∞, rho]] // N,
    {x, 0, 4}]},
  {rho, 1, 4, 3},
  {eta, 1 / 2, 2, 3 / 2}] // ColumnForm
```

```
{η, ρ, {Errors for delays 2 power x (0..4)}}
{{1/2, 1, {0.55, 0.01, 0., 0., 0.}}, {2, 1, {0.54, 0.17, 0.07, 0.02, 0.}}}
{{1/2, 4, {0.59, 0.04, 0., 0., 0.}}, {2, 4, {0.46, 0.41, 0.23, 0.03, 0.01}}}
```

■ 3.3- Doing the same test for AND problems

```
MyD[x_] := {{0, x}, {0, x}, {∞, x}};
KK[x_] = {2 x, x};
MyD[XX] // MatrixForm
```

$$\begin{pmatrix} 0 & XX \\ 0 & XX \\ \infty & XX \end{pmatrix}$$

```
{"η", "ρ", "{Errors for delays 2 power x (0..4)}"}
Table[{eta, rho,
    Table[Mean[PRMod[
       noisypairs[redundantpairs[NiopairsAnd, rho], ExponentialDistributior
       redundantdelays[MyD[2^x], rho, 1], KK[rho], 100, -∞, rho]] // N,
    {x, 0, 4}]},
  {rho, 1, 4, 3},
  {eta, 1/2, 2, 3/2}] // ColumnForm
```

```
{η, ρ, {Errors for delays 2 power x (0..4)}}
{{1/2, 1, {0.76, 0., 0., 0., 0.}}, {2, 1, {0.72, 0.19, 0.06, 0.03, 0.}}}
{{1/2, 4, {0.75, 0.01, 0., 0., 0.}}, {2, 4, {0.73, 0.29, 0.13, 0.05, 0.}}}
```

■ 3.4- More advance example: XOR provokes no errors

```
DD = {{0, 1.}, {0, 1.}, {2, ∞}, {2, ∞}};
KK = {2, 1};
DD // MatrixForm
PRMod[NiopairsXor, DD, KK, 100, -∞, 2]
```

$$\begin{pmatrix} 0 & 1. \\ 0 & 1. \\ 2 & \infty \\ 2 & \infty \end{pmatrix}$$

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
```

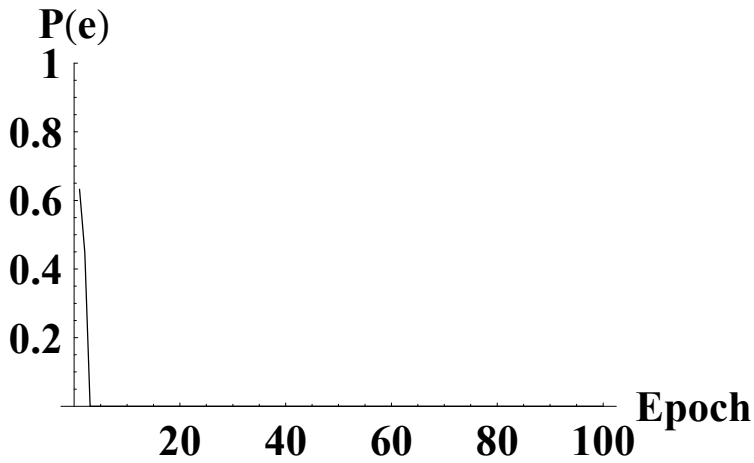
4- Examples of learning with PRNet

■ 4.1- Simple examples with PRNet (OR, 1-D)

PRNet is a very simple network provided in this package. Firing time is deterministic, and there is no
can make appropriate judgments.

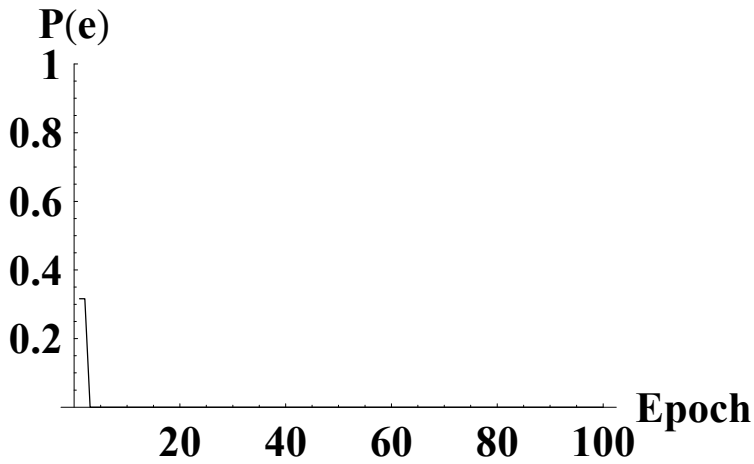
PRNet learns 1-D problems

```
ERROR[PRNet[Niopairs1D]];
```



PRNet easily learns OR

```
ERROR[PRNet[NiopairsOr]];
```



■ 4.2- More examples with PRNet (AND, XOR)

PRNet also implements threshold. It is still deterministic. It learns to adjust threshold and firing time ouptut pairs. For PRNet, AND and XOR problems are basically of the same complexity, as both requires thre

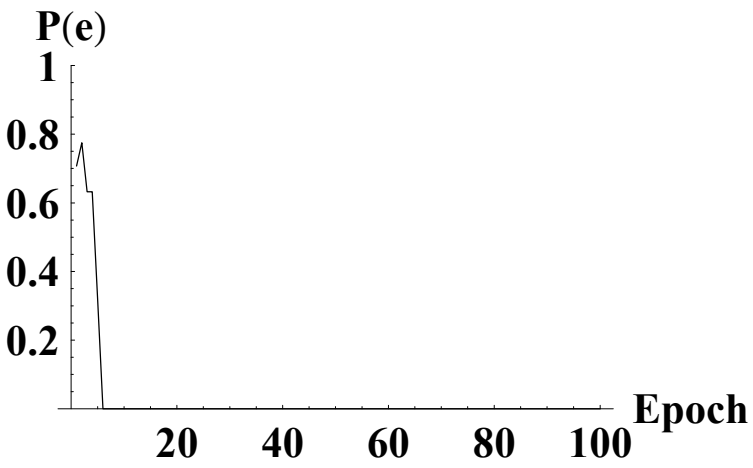
PRNet learns AND and shows the firing speed (Debug is 0)

```
ERROR[PRNet[NiopairsAnd, 1000, 0, 1]];

==========================================
Starting Speeds D and threshold K are:
⎛0.0879805  0.0996249⎞
⎜0.0174586   0.015247⎟2x3
⎝ 0.093916  0.00171373⎠
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
⎛1.58798  1.59962⎞
⎜1.01746  2.01525⎟
⎝2.09392  2.00171⎠


{2, 1}
```



PRNet also learns XOR problems, but two time-out nodes are needed

```
ERROR[PRNet[NiopairsXor, 1000, 0 , 2]];
```

```
==========================================
Starting Speeds D and threshold K are:
```

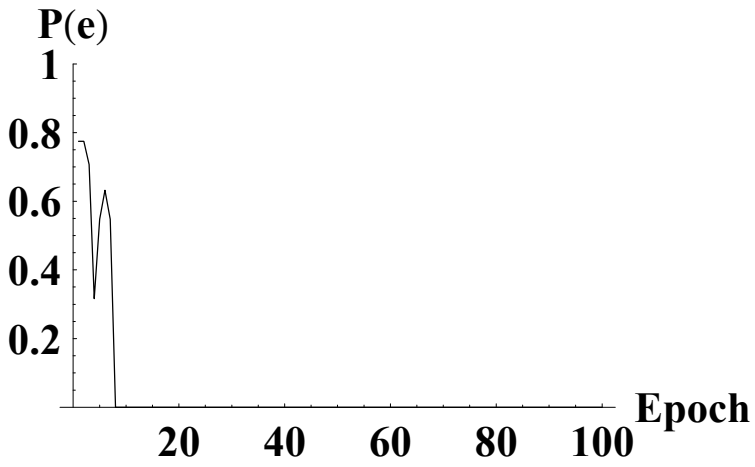
$$\begin{pmatrix} 0.0819375 & 0.0485969 \\ 0.0018176 & 0.0310202 \\ 0.053576 & 0.0250944 \\ 0.0380217 & 0.0980007 \end{pmatrix}_{2x4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 1.08194 & 1.5486 \\ 1.50182 & 1.53102 \\ 2.05358 & 2.52509 \\ 2.03802 & 2.098 \end{pmatrix}$$

```
{2, 1}
```



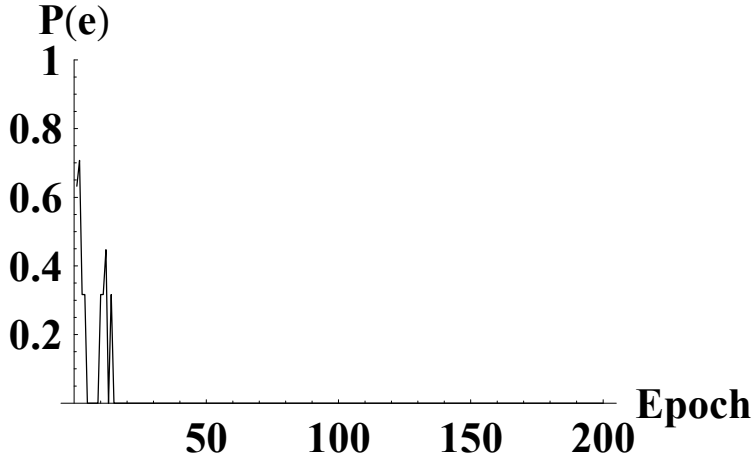
PRNet on XOR and the adjunction of redundant pairs

Redundant inputs signifies that there are multiple nodes that can detect the same input. Thus, each inp likelihood that fast evidence occurs, and thus should help the network make fast answer.

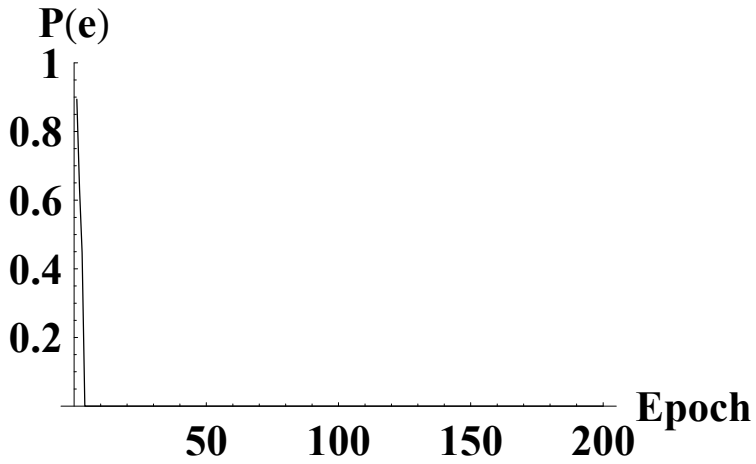
PRNet with various amount of redundancy learning XOR problems

```
Table[
    Print["Redundancy of: ", 2^i];
    ERROR[PRNet[redundantpairs[NiopairsXor, 2^i], 2000, -∞, 2^(i + 1), 1, 2]]
```

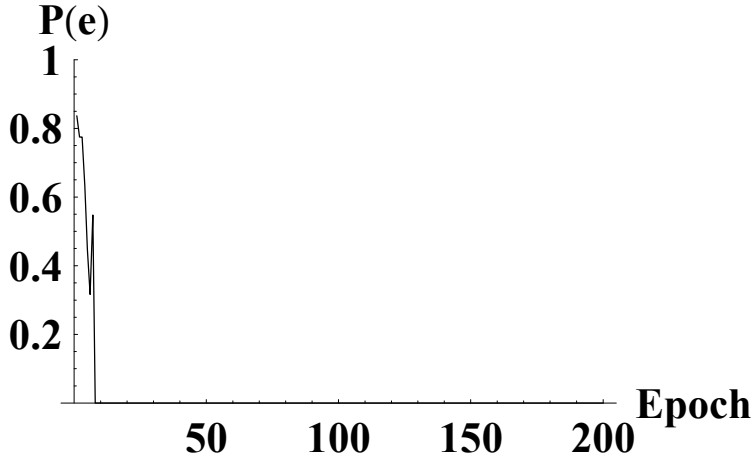
```
Redundancy of: 1
```



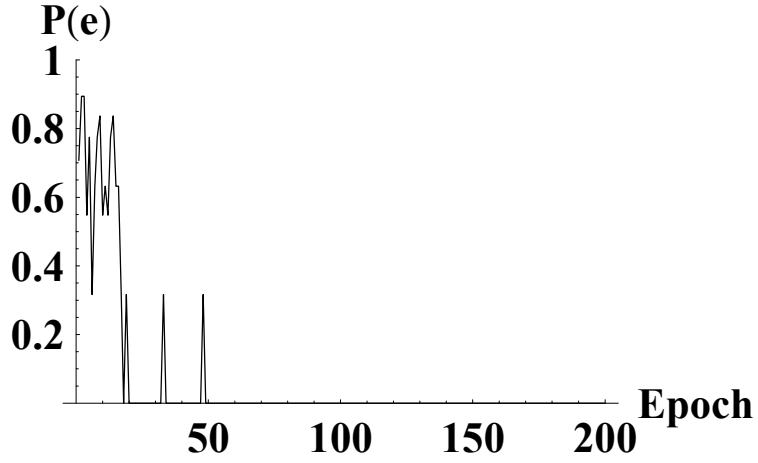

```
Redundancy of: 2
```



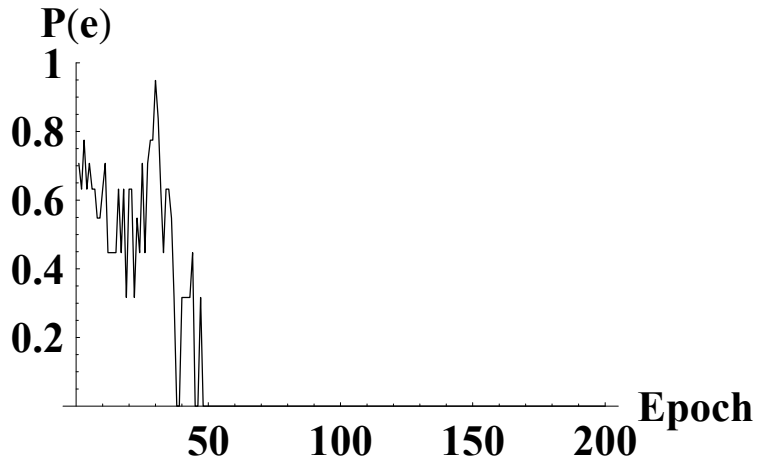

```
Redundancy of: 4
```




```
Redundancy of: 8
```




```
Redundancy of: 16
```



PRNet does have moderately stable Thresolds when there are redundant signals though (debug of -3

```
x = Table[PRNet[redundantpairs[NiopairsXor, 4], 2000, -3, 8], {10}];
x // N // MatrixForm
Apply[Plus, x] / Length[x] // N
```

$$\begin{pmatrix} 5.6 & 4. \\ 5.2 & 4. \\ 6.4 & 4. \\ 5.4 & 4. \\ 6.8 & 4. \\ 5.6 & 4. \\ 5.4 & 4. \\ 5.6 & 4. \\ 4.4 & 4. \\ 5.2 & 4. \end{pmatrix}$$

```
{5.56, 4.}
```

■ 4.3- Examples of PRNet learning noisypairs

Noisy inputs means that the input are no longer binary, but can vary on the exact time they each arrive

1- Uniform noisy patterns

PRNet learns a XOR having a small amount of noise (uniform noise, ±1) with or without redundancy.

```
ERROR[PRNet[noisypairs[NiopairsXor, UniformDistribution[-1, +1], 0, 0, 0, 10
```

```
==========================================
Starting Speeds D and threshold K are:
```

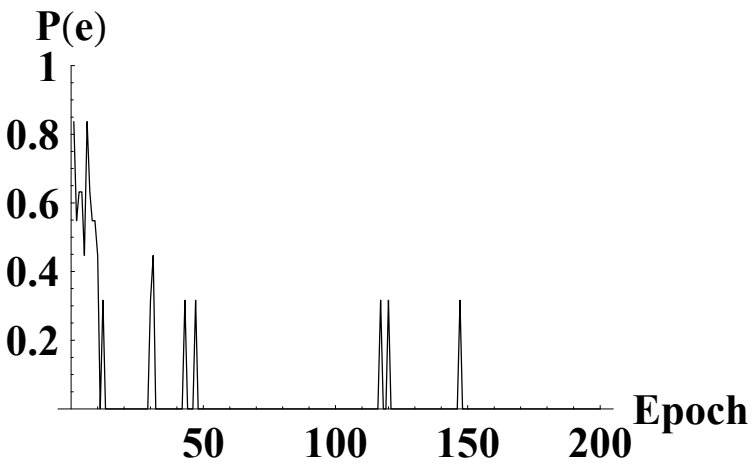
$$\begin{pmatrix} 0.00604743 & 0.0588206 \\ 0.0888191 & 0.0112477 \\ 0.0240777 & 0.0368502 \\ 0.0817369 & 0.043355 \end{pmatrix}_{2x4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 1.00605 & 2.55882 \\ 1.08882 & 2.01125 \\ 4.02408 & 5.03685 \\ 3.58174 & 5.04336 \end{pmatrix}$$

```
{2, 1}
```



or when the average activation state is 1 but the average spurious activation rate is closer (4). No le distribution and the distribution of spurious activites to place the second response's distribution.

```
noisypairs[iopairsXor, UniformDistribution[-1, +1], 4, 1, 0, 100] // MatrixFo
```

$$\begin{pmatrix} \begin{pmatrix} 4.59243 \\ 4.17219 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 4.42152 \\ 0.22123 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 1.79206 \\ 3.12759 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 1.03863 \\ 0.48446 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix}$$

```
ERROR[PRNet[noisypairs[iopairsXor, UniformDistribution[-1, +1], 4, 1, 0, 100
```

```
==========================================
Starting Speeds D and threshold K are:
```

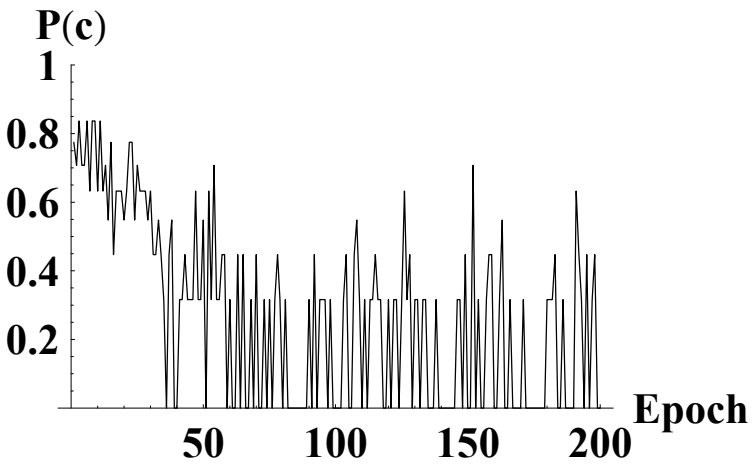
$$\begin{pmatrix} 0.0520316 & 0.0739552 \\ 0.0582439 & 0.0122965 \\ 0.0624284 & 0.0675759 \\ 0.0063125 & 0.0880735 \end{pmatrix}_{2\times 4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 1.45203 & 2.87396 \\ 1.15824 & 3.0123 \\ 4.96243 & 5.06758 \\ 5.00631 & 5.08807 \end{pmatrix}$$

```
{2, 1}
```



2-Normal noise

An example with a lot of random normal noise: errors remains because on some trials, the absence

```
ERROR[PRNet[noisypairs[iopairsXor, NormalDistribution[0, 1.0], 10, 1, 0, 100
```

```
==========================================
Starting Speeds D and threshold K are:
```

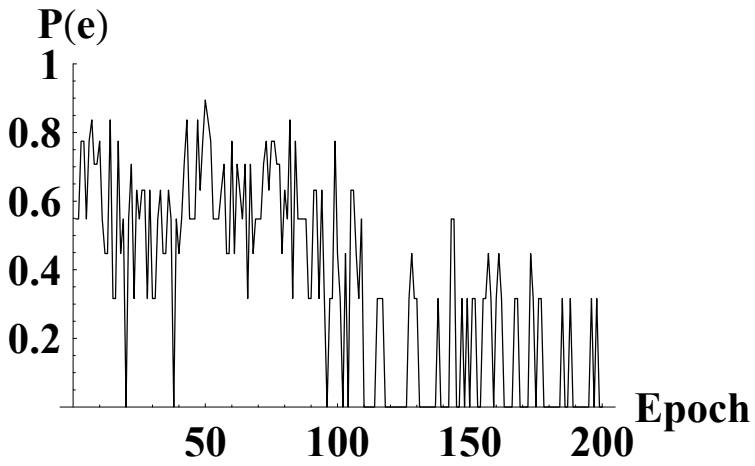
$$\begin{pmatrix} 0.0338908 & 0.0250264 \\ 0.020181 & 0.0843439 \\ 0.0995392 & 0.0115126 \\ 0.0611847 & 0.0632288 \end{pmatrix}_{2x4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 23.0339 & 23.025 \\ 21.0202 & 24.5843 \\ 27.5995 & 27.0115 \\ 29.5612 & 30.0632 \end{pmatrix}$$

```
{2, 2}
```



3-Exponential noise

Typical example from the simulations TABLE1 and TABLE2 (of the text)

```
r = 0;
n = 0.5;
ERROR[PRNet[noisypairs[redundantpairs[iopairsXor, 2^r], ExponentialDistrib
            10, 0, 0, 100], 2000, 0, 2 2^r]];
```

```
==========================================
Starting Speeds D and threshold K are:
```

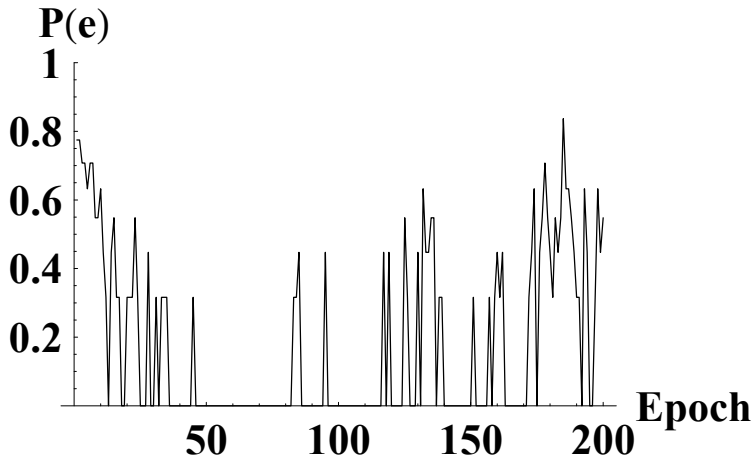
$$\begin{pmatrix} 0.036775 & 0.020199 \\ 0.0540181 & 0.060916 \\ 0.0844189 & 0.0573803 \\ 0.0330121 & 0.0536041 \end{pmatrix}_{2x4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 6.03678 & 7.0202 \\ 7.05402 & 5.56092 \\ 16.0844 & 16.0574 \\ 16.033 & 16.0536 \end{pmatrix}$$

```
{2, 2}
```



An example with exponential noise proportional to the input.

```
noisypairs[iopairsXor, ExponentialDistribution[1 / StimValue ],
            10, 1, 0, 100] // MatrixForm
```

$$\begin{pmatrix} \begin{pmatrix} 16.9656 \\ 21.6596 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 24.6554 \\ 1.53181 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 3.20127 \\ 14.7388 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 2.52895 \\ 1.24679 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix}$$

```
ERROR[PRNet[noisypairs[iopairsXor, ExponentialDistribution[1 / StimValue],
            10, 1, 0, 100], 2000, 0, 2]];
```

```
==========================================
Starting Speeds D and threshold K are:
```

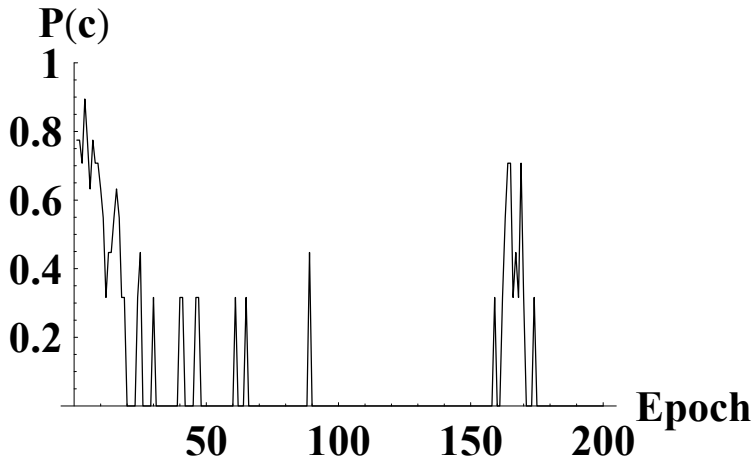
$$\begin{pmatrix} 0.0575214 & 0.0298609 \\ 0.0669312 & 0.0604461 \\ 0.0473552 & 0.0836134 \\ 0.0568408 & 0.0372173 \end{pmatrix}_{2x4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 1.55752 & 2.52986 \\ 1.56693 & 3.06045 \\ 12.0474 & 11.5836 \\ 12.0568 & 12.5372 \end{pmatrix}$$

```
{2, 2}
```



Example with noise Exponential, but added if signal, and removed if no signal.

```
ERROR[PRNet[noisypairs[iopairsXor, ExponentialDistribution[If[StimValue >=
            10, 1, 0, 100], 2000, 0, 2]];
```

```
==========================================
Starting Speeds D and threshold K are:
```

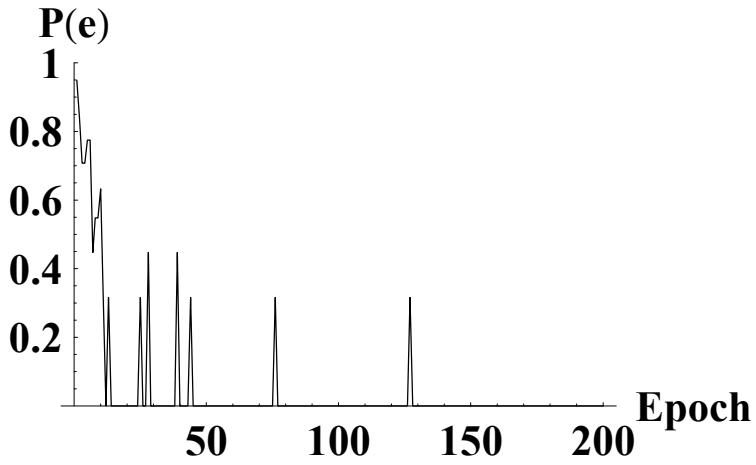
$$\begin{pmatrix} 0.0463204 & 0.0156203 \\ 0.0273773 & 0.0520927 \\ 0.0582799 & 0.0926817 \\ 0.00346589 & 0.0802775 \end{pmatrix}_{2\times 4}$$

```
{1, 1}

==========================================
All done; the final matrix D and threshold K are:
```

$$\begin{pmatrix} 2.04632 & 2.51562 \\ 2.52738 & 2.55209 \\ 4.55828 & 4.09268 \\ 6.00347 & 6.08028 \end{pmatrix}$$

```
{2, 2}
```



In these cases, the drift rates are proportional to the amount of noise introduced

```
x = Table[{noise, Drift[
                PRNet[noisypairs[iopairsXor, ExponentialDistribution[1 / noi
   {noise, 0.2, 2, 0.4}];
(*affiche les Δ drifts en pourcentage du min*)
Table[{x[[i]][[1]], (x[[i]][[2]] - Min[x[[i]][[2]]]) // MatrixForm}, {i, 1, Le
```

$$\begin{pmatrix} 0.2 & \begin{pmatrix} 0. & 1.22182 \\ 3.67154 & 4.45131 \end{pmatrix} \\ 0.6 & \begin{pmatrix} 0. & 0.951468 \\ 6.94553 & 6.98444 \end{pmatrix} \\ 1. & \begin{pmatrix} 0. & 0.496701 \\ 7.78485 & 7.75192 \end{pmatrix} \\ 1.4 & \begin{pmatrix} 0. & 1.49485 \\ 9.73778 & 9.52021 \end{pmatrix} \\ 1.8 & \begin{pmatrix} 0. & 0.474926 \\ 10.9796 & 10.7316 \end{pmatrix} \end{pmatrix}$$

End of notebook