Report 1

Paweł Matykiewicz

University of Adam Mickiewicz in Poznan, Faculty of Social Sciences, Department of Philosophy

Abstract

In this report we try to achieve similar results as in [2]. We compare chaotical properties of the network with discrete and continuous output function using **Delphi 5.0**.

Key words: associative memory, neural networks, chaotic dynamics, Nagumo-Sato Neuron Model

1 Introduction

We focus on the difference between two network models. One is constructed with "Chaotic Neuron Model" and the other "Devil's Staircase Neuron Model" [1]. The only difference between them is in the output function. Nonetheless it gives enormous difference in behavior of these two neural networks. In this report we focus only on the look of the dynamics depended from initial conditions. In [2] we see that chaotic dynamic comes from chaotic neuron model and that it has good searching properties. First thought should be that devil's staircase neuron model will not give such a searching properties as chaotic one. We take short look at that phenomenon.

2 Model

We use discrete-time neuron model which was developed by Nagumo and Sato in 1972. They found fractal properties of their Heaviside output function

Email address: pm@felix.fizyka.amu.edu.pl (Paweł Matykiewicz).

neuron model. In 1990 chaotical properties were described of the continuous output function neuron model [1]. We use both of these output functions:

$$f^C(u) = \frac{1}{1 + \exp(\frac{-u}{0,015})} \tag{1}$$

$$f^D(u) = \begin{cases} 1 & \text{if} \quad u > 0 \\ 0 & \text{if} \quad u \leq 0 \end{cases} \tag{2}$$

We want to see the same numerical experiments which were conducted in [2] but with f^D function.

2.1 Network Model

We build a simple auto-associative neural network composed of 100 devil‘s staircase neurons or chaotic neurons. Here are equations defining our network:

$$x_i(t+1) = f^C(\eta_i(t+1) + \zeta_i(t+1)) \tag{3}$$

$$x_i(t+1) = f^D(\eta_i(t+1) + \zeta_i(t+1)) \tag{4}$$

$$\eta_i(t+1) = k_f \eta_i(t) + \sum_{j=1}^{100} w_{ij} x_j(t) \tag{5}$$

$$\zeta_i(t+1) = k_r \zeta_i(t) - \alpha x_i(t) + a \tag{6}$$

All parameters and equations are the same as in [2]. Namely k_f, k_r are refractory decay parameters for feedback and refractoriness, α, a are refractory scaling parameter and threshold respectively. We put $k_f = 0,2$, $k_r = 0,9$, $\alpha = 8$ and $a = 2$. The feedback interconnections w_{ij} are determined by:

$$w_{ij} = \frac{1}{4} \sum_{p=1}^{4} (2x_i^p - 1)(2x_j^p - 1) \tag{7}$$

where x_i^p is the ith component of the pth stored pattern. All $w_{ii} = 0$. Stored patterns are shown in figure 1. For initial conditions we use the same patterns or perturbed versions (same as in [2]). Initial conditions $\eta_i(0)$ and $\zeta_i(0)$ are determined (in continuous as well as in discrete version of f) by equations [private e-mail with prof.Adachi]:

$$g(u) = -0,015 \ln(\frac{1}{u} - 1) \tag{8}$$

$$\eta_i(0) = g(0,9x_i^{\rho} + 0,5) \tag{9}$$

$$\zeta_i(0) = a \tag{10}$$

Of course $g = (f^C)^{-1}$ and $g(1) = g(0) = \infty$ so we use 0,95 and 0,05 as initial conditions. f^D is not a surjection nor injection so we have to use g to load $\eta_i(0)$ in the devil's staircase neural network.

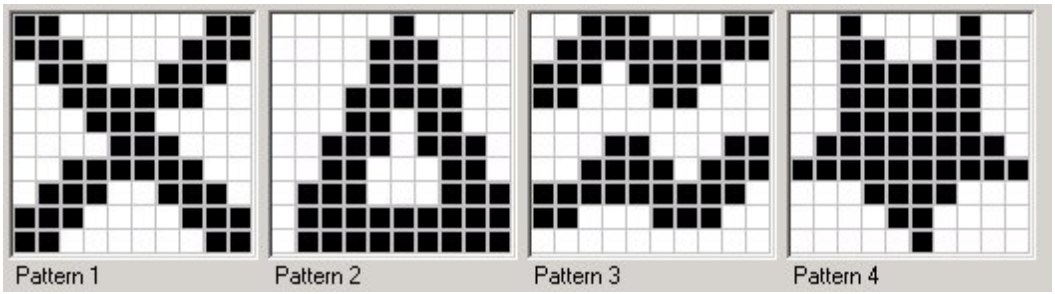

Fig. 1. The 4 stored patterns are displayed in the form of a 10×10 matrix. Each square is 1 or 0 output neuron. The QE values are -393, - 390.5, -384.5, -381, -285.5 respectively.

2.2 Methods of analysis on the Network Model

In order to analyze Neural Network we use listed below equations:

$$QE(t) = -\frac{1}{2}\sum_{i \neq j}\sum_{j} w_{ij}x_i(t)x_j(t) - \sum_{i} ax_i(t); \tag{11}$$

$$Dist(t) = \sqrt{\sum_{i=1}^{100}\{[\eta_i(t) - \eta_i(t_{end})]^2 + [\zeta_i(t) - \zeta_i(t_{end})]^2\}} \tag{12}$$

$$H_p(t) = \sum_{j=1}^{100} |x_j(t) - x_j^p| \tag{13}$$

QE function gives us an idea of long-term behavior of the network [2]. We use $Dist$ to analyze periodicity [2]. We can use $H_p(t) \times H_q(t)$ plot to reconstruct attractor in two dimensional phase space. In our simulation $p = 1$ and $q = 3$.

3 Implementation

We have implemented both neural networks in two different application but with the same graphical interface. We use **Delphi 5.0** object capabilities (Tn2

is a class). There is only difference in `Tn2.x` procedure. Hart of the application is `Tn2.etazeta` procedure which loads η and ζ vectors.

```
procedure Tn2.etazeta (axvector:vector;var aetavector:vector;var
azetavector:vector);//internal state

var i,j:integer;
    sum:real;
    fetavector:vector;
    fzetavector:vector;

begin
fetavector:=aetavector;
fzetavector:=azetavector;

for i:=0 to n-1 do
  begin
  fzetavector[i]:=kr*azetavector[i]-alfa*axvector[i]+a;
  end;
for i:=0 to n-1 do
  begin
  sum:=0;
  for j:=0 to n-1 do
    sum:=sum+wmatrix[i,j]*axvector[j];
  fetavector[i]:=kf*aetavector[i]+sum;
  end;
aetavector:=fetavector;
azetavector:=fzetavector;
end;


procedure Tn2.initIS (axvector:vector;var aetavector:vector;var
azetavector:vector); //initial conditions

var i:integer;

begin for i:=0 to n-1 do
  begin
  azetavector[i]:=a;
  aetavector[i]:=g(axvector[i]*0.9+0.05)-a;
  end;
end;
```

```
function Tn2.g(u:real):real;

begin
result:=-smalleps*Ln((1/u)-1);
end;
```

In the `neuro431c.exe` we have:

```
function Tn2.f(u:real):real; //continuous output function

begin
result:=1/(1+exp(-u/smalleps));
end;


procedure Tn2.x (var axvector:vector);//output pattern

var i:integer;

begin
for i:=0 to n-1 do
  axvector[i]:=f(etavector[i]+zetavector[i]);
end;
```

And in the `neuro431d.exe` we have:

```
procedure Tn2.x (var axvector:vector);//output pattern-discrete
version

var i:integer;

begin

for i:=0 to n-1 do
 if (etavector[i]+zetavector[i])>0 then axvector[i]:=1
 else axvector[i]:=0;
end;
```

It should be noticed that `neuro431d.exe` and `neuro431c.exe` make `timeseries.ts` file which is used to evaluate *Dist*. Files `ptX-Y.ptn` are patterns for initial conditions where `X` is pattern number and `Y` is Hamming distance in perturbed

version of patterns in figure 1. They are the same as in [2]. To run whole simulation one need to press F1 - Ignition, F2 - Load patterns, F3 - Writes parameters, F4 - Starts calculations, CTRL+T - time interval.

4 Simulation results

All images are taken from `neuro431d.exe` or `neuro431c.exe`. From figure 2 and 4 we can see that there is not much difference in the transient phase. Also attractors in figure 3 seems to look similar.

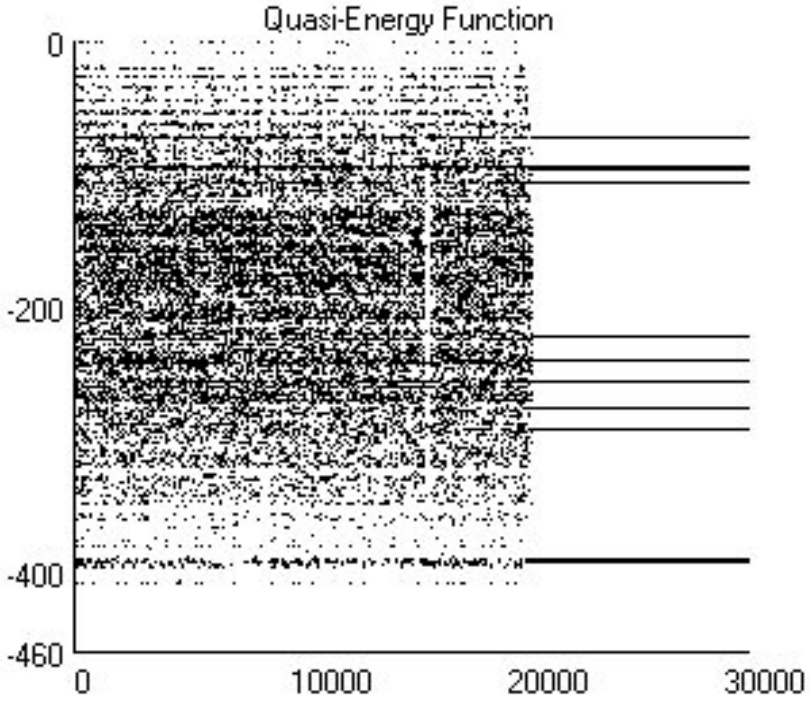

Fig. 2. Long-time behavior of QE function with initial pattern pt3.ptn in f^C case.

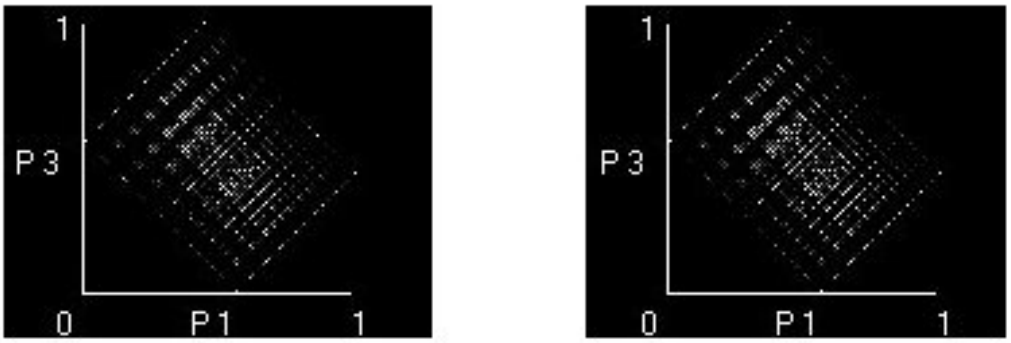

Fig. 3. From the left we see pt2-8.ptn in f^C and pt3.ptn in f^D case. The brighter a pixels is the oftener system is in that state.

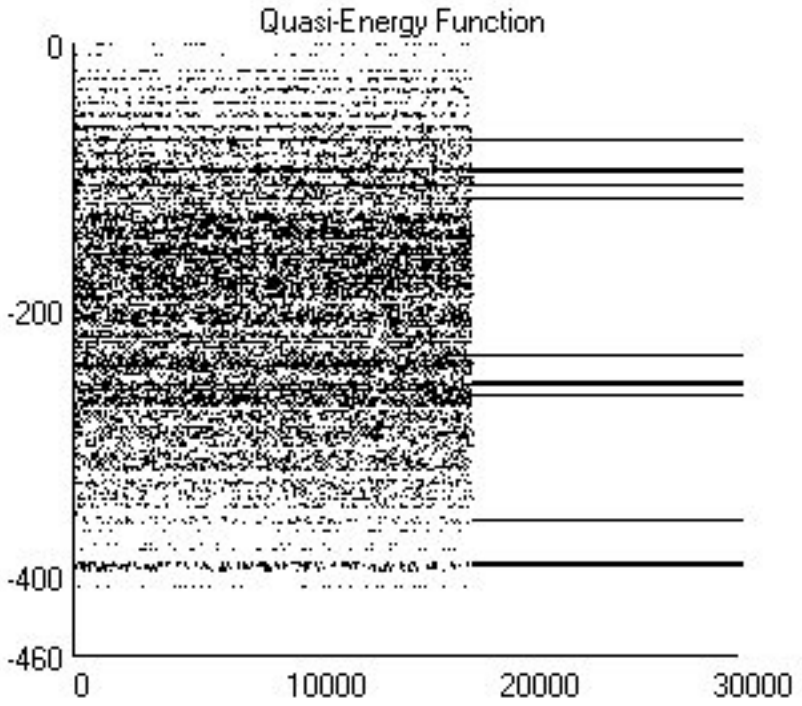

Fig. 4. Long-time behavior of QE function with initial pattern pt1-8.ptn in f^D case.

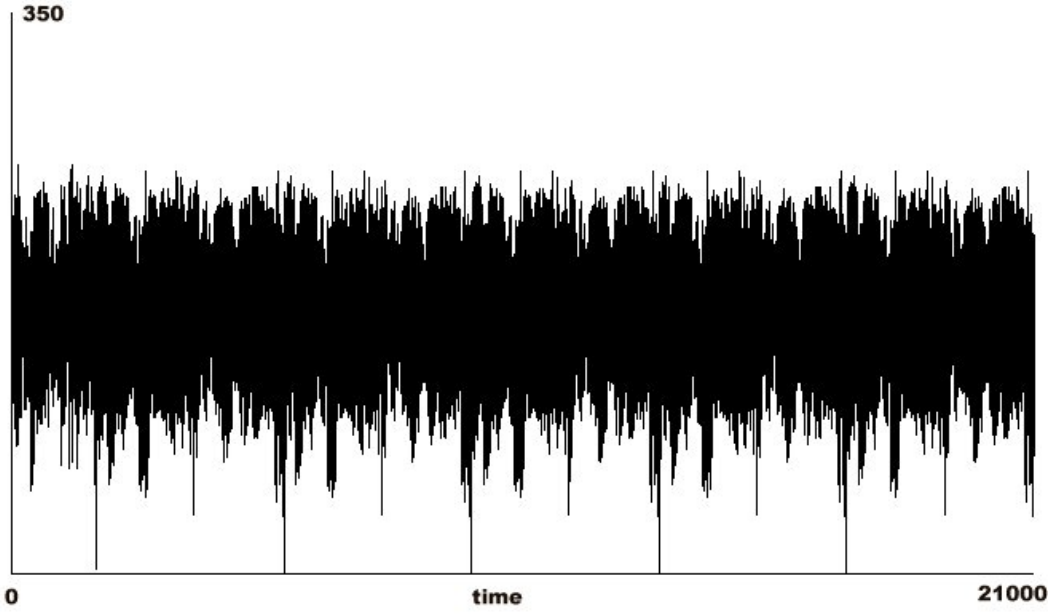

Fig. 5. Plot of *Dist* function with initial pattern pt4-4.ptn in f^D case.

5 Summary

pattern number	H-distance 0	H-distance 4	H-distance 8
1	Tr $\sim$ 17500	Pr $\sim$ 5000	Tr $\sim$ 19000
2	Tr $\sim$ 14000	Tr $\sim$ 11000	Tr $\sim$ 2500
3	Tr $\sim$ 10500	Tr $\sim$ 17600	Tr $\sim$ 17500
4	Tr $\sim$ 14000	Pr $\sim$ 4000	Tr $\sim$ 16000

pattern number	H-distance 0	H-distance 4	H-distance 8
1	Pr $\sim$ 9500	Nonperiodic	Tr $\sim$ 1000
2	Pr $\sim$ 9500	Tr $\sim$ 9000	Tr $\sim$ 10000
3	Tr $\sim$ 21000	Pr $\sim$ 9500	Tr $\sim$ 6000
4	Nonperiodic	Pr $\sim$ 9500	Tr $\sim$ 4000

After transient chaos we have period 20.

6 Conclusion

We see that the f^D function has the same searching abilities as f^C. f^D has fractal-like average firing rate [1] but there is no positive Lyapunov exponent although there are longer transient times in average and less periodic without transient chaos cases. In *Report 2* we will use f^D to solve TSP.

References

[1] Aihara K., Takabe T., Toyoda M., Chaotic neural networks, Phys. Lett. A, 144, 333-340, (1990)

[2] Adachi M., Aihara K., Associative dynamics in chaotic neural network, Neural Networks, 10, 83-98, (1997)