CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **00**:1–8
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Function aproximation using experimental artifical neuron model

Michal Chovanec, Lukáš Čechovič

Faculty of management of science and informatics, Žilinská Univerzita, SK

SUMMARY

In this paper we introduce experimental neuron model, using inputs signal's multiplications. Some approximation abilities test's were done and further applications introduced. Copyright © 2016 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: neural network, approximation, neuron model, McCulloch-Pitts neuron

1. INTRODUCTION

Feed forward neural network can be used for approximation of any continuous function [1], [2], [3] Some problems are also know [4] - using of many neuons, or layers - which is serious problem for real time applications [5], especially real time control, robotics, image recognition or virtual agents. Other problem is gradient backpropagation which can stay in local minima [6]. Some techniques, like momentum can help in this problem, or stochastic optimization algorithms.

Some common neural network applications [7] can be found in :

- Stock Market Prediction
- Medical Diagnosis
- Process Control
- Prediction
- Classification
- Change and Deviation Detection
- Knowledge Discovery
- Response Modeling
- Time Series Analysis
- Staff Scheduling
- Personnel Profiling

To reduce neurons and layers count we modify neuron model to multiplication ability. On presented experiments we can see approximation abilities differences on neuron model topology.

2. PRECEDING EXPERIMETS

The main idea of neuron model is to separate input space using plane. Approximation ability can be proven using [1]. Common neuron model McCulloch-Pitts can be described as 1.

*Correspondence to: michal.chovanec@fri.uniza.sk, lukas.cechovic@fri.uniza.sk

$$y(n) = \varphi(\sum_{i=0}^{N-1} x_i(n)w_i(n)) \tag{1}$$

where
$x(n)$ is input vector
$w(n)$ is weight vector
$y(n)$ is neuron output
and $\varphi(g)$ is activation function

Mostly used activation function are sigmoid, tahn, linear, rectifier, and step functions are on 1.

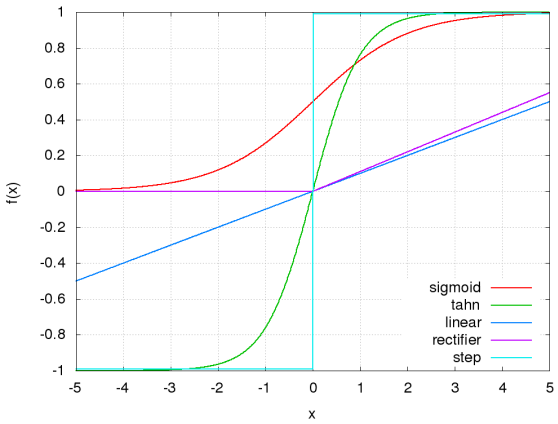

Figure 1. Activation functions

Consider following problem : We need to approximate multiplexer function with inputs A B and select S (figure)2).

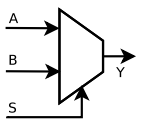

Figure 2. Multiplexer

Using boolean logic we can write

$$y = A\neg S + BS \tag{2}$$

When $A, B, S \in \mathbb{R}$ and $S \in \langle 0, 1\rangle$ we can write nothing else than

$$y = A(1 - S) + BS \tag{3}$$

Other problem : Consider simple two wheels robot, with four distance sensors, on figure 3. We can define robot state by following vector $R(n) = (r(n), l(n), \theta(n), s_0(n), s_1(n), s_2(n), s_3(n))$, where
$r(n)$ is right motor input
$l(n)$ is left motor input
$\theta(n)$ is robot orientation angle
$s_0(n)$ is front obstacle distance sensor output
$s_1(n)$ is left obstacle distance sensor output
$s_2(n)$ is rear obstacle distance sensor output

DOI: 10.1002/cpe

$s_3(n)$ is left obstacle distance sensor output

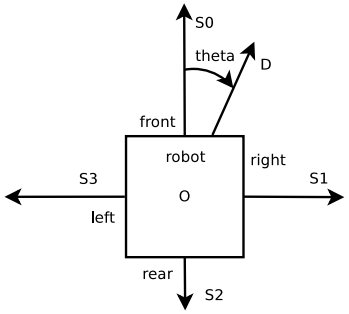

Figure 3. Robot

Our goal is determine prediction in G samples : $R(n+G)$. We can estimate exact physical model of robot and use it for modeling output (citovat tradicne modelovanie). Simplified robot model can be written as rotation and moving. Situation for front sensor is describen in equation 4.

$$\begin{aligned} s'_0(n) = min(&D(n) + H(\theta(n))s_0(n), \\ &D(n) + H(\theta(n) + \frac{\pi}{2})s_1(n), \\ &D(n) + H(\theta(n) + \pi s_2(n)), \\ &D(n) + H(\theta(n) + \frac{3\pi}{2})s_3(n)) \end{aligned} \tag{4}$$

Where $D(n)$ is robot position matrix, obtained from robots wheels speed.
$\theta(n)$ is rotation angle, calculated as speed diference of right and left motor $\theta(n) = b(r(n) - l(n))$
$H(\theta(n))$ is rotation matrix for 2D vector around θ angle.
When different motors (wheels count, diameter, power, etc.) or sensors (count, range, angles, etc.) are used, we need to modify model. Biological systems are based on self model estimating (citovoat biological learning) and robustness (citovat napr Mark Tildena), using senses confrontation with their model citovat. This solution is robust and durable to system modification.

There is possible to use neural network (similar like biological systems) to create robot model. We can see from 3 and 4 there is not simple to process multiplicaiton with rotation vector using McCulloch Pitts neuron model, because of inputs multiplication.

Other view to this problem is "variable weights problem" - once, when network is trained, it is computing only exactly learned function - modifying function parameters is impossible without modifying weights.

Consider simple function 5 with parameter a. From some reason, we need to modify this parameter, depending on other input $a = x_1$.

$$y = tanh(aw_0x_0) \tag{5}$$

Using neuron model from 1 it's impossible to this in single layer.

3. NEURON MODEL

We can simply solve proposed problems, adding multiplication matrix into neuron model

$$y(n) = \varphi(\sum_{i=0}^{N-1} x_i(n)w_i(n) + \sum_{j=0}^{N-1}\sum_{i=j}^{N-1} x_j(n)x_i(n)v_{ji}(n)) \tag{6}$$

DOI: 10.1002/cpe

or more generally

$$y(n) = \sum_{i=0}^{N-1} x_i(n)w_i(n) + \sum_{j=0}^{N-1}\sum_{i=j}^{N-1} x_j(n)x_i(n)v_{ji}(n) + \sum_{j=2}^{N-1}\sum_{i=0}^{N-1} x_i^j(n)q_{ji}(n) \tag{7}$$

where
$v(n)$ is matrix for two inputs multiplication
$q(n)$ is matrix for input power series multiplication

In 7 we replace $\varphi(x)$ with power series - this can be done beacuse of [8] and [9].

3.1. Neuron learning

For basic learning, we can use gradient method. We can define error as $e(n) = y_r(n) - y(n)$, where $y_r(n)$ is required output corresponding to input $x(n)$.

Basic gradient learning can be written as

$$w_i(n) = w_i(n-1) + \eta e(n)x_i(n) \tag{8}$$

Considering product $x_j(n)x_i(n)$ and $x_i^j(n)$ is nothing else than other input we can write for testing neuron

$$v_{ji}(n) = v_{ji}(n-1) + \eta e(n)x_j(n)x_i(n) \tag{9}$$

$$q_{ji}(n) = q_{ji}(n-1) + \eta e(n)x_i^j(n) \tag{10}$$

To avoid local minima, some stochastic algorithm can be used - well know is simulated annealing. First we need to define fitness function for all required outputs as

$$\lambda(n) = \lambda(n-1) + \sum_{i=0}^{Y-1} e_i(n)^2 \tag{11}$$

and network parameters $N(n) \in \{\lambda(n), w(n), v(n), q(n)\}$. We also mark best neural network as N and some testing neural network as N', where $\lambda_N \leq \lambda_{N'}$

$$N(n+1) = \begin{cases} N(n)' & if \lambda'_N \leq \lambda_N \vee pT(n) > rnd(0,1) \\ N(n) & otherwise \end{cases} \tag{12}$$

4. EXPERIMENTAL RESULTS

In this part we describe our experimental results on function approximation ability.

4.1. Testing functions

Few function approximation tests were done - each experiment ID correspond to one of testing function from 13. We choose four categories of functions

- Fuzzy logic - first two, using Łukasiewicz logic (citovat)
- Continuous nonlinear functions, with many multiplications
- Xor problem, with $sgn(x)$
- Four randomly initialized neural networks

Our goal was to cover principal approximation from spread set's of problems.

All functions have two inputs, in range $\langle -1, 1, \rangle$ and single output. For correct neural network learning, one additional input was used as bias, with constant value 1.0.

DOI: 10.1002/cpe

Last four testing functions were randomly initialized feed forward neural networks, with different activation function's and different count of hidden layers $\langle 1, 3\rangle$, always with 32 neurons, with single output and three inputs - two for random input from $\langle 1, 3\rangle$ and one for bias.

$$
\begin{aligned}
y &= \neg(x_0 x_1) \\
y &= ((x_0 x_1) \vee x_1)\neg(x_0 x_1) \\
y &= x_0 x_1 \\
y &= sin(5.0 x_0 x_1) \\
y &= 0.25(cos(x_0 x_1 + 0.731) - cos(2.3 x_0 - 4.45 x_1) \\
&\quad - cos(sin(3.17 x_0 - 2.787 x_1))sin(3.5435 x_0) \\
&\quad + sin(x_1 + 5.0 x_0 x_1)) \\
y &= sin(3.3214 x_0)sin(3.742 x_1) \\
y &= sgn(x_0 x_1) \\
y &= \frac{x_0 + x_1}{2.0} \\
y &= nn(linear, 1) \\
y &= nn(tanh, 1) \\
y &= nn(tanh, 2) \\
y &= nn(tanh, 3)
\end{aligned}
\tag{13}
$$

4.2. Neural network conditions

Each neural network have some parameters, with difference only in neuron model type and hidden layers count 14, and for Kohonen layer 15. In our experiments, these topology was testeted : feed forward neural network's with 1 or 2 hidden layers, Kohonen network function approximation, and finally feed forward neural network's with input preprocessing in Kohonen network.

Each network configuration had limited weight range, and output range $y_{max} \in \langle -1, 1\rangle$. There was always 100000 learning iterations in each trial.

$$
\begin{aligned}
neuron\_type &= \{ \\
&\quad NEURON\_TYPE\_LINEAR, \\
&\quad NEURON\_TYPE\_TANH, \\
&\quad NEURON\_TYPE\_TESTING\} \\
init\_vector &= seetext \\
weight\_range &= 4.0 \\
init\_weight\_range &= weight\_range/4 \\
\eta &= 1.0/1000.0; \\
y_{max} &= 4.0
\end{aligned}
\tag{14}
$$

$$
\begin{aligned}
neurons\_count &= 8; \\
weight\_range &= 1.0; \\
\eta &= 1.0/100.0; \\
y_{max} &= 1.0;
\end{aligned}
\tag{15}
$$

DOI: 10.1002/cpe

4.3. Results

Each network was tested on 12 different functions. And each function was running 16-teen times - trials, to show indenpendency from initial weight values. These result can be seen on fig 4. We can see, only little average results fluctuations.

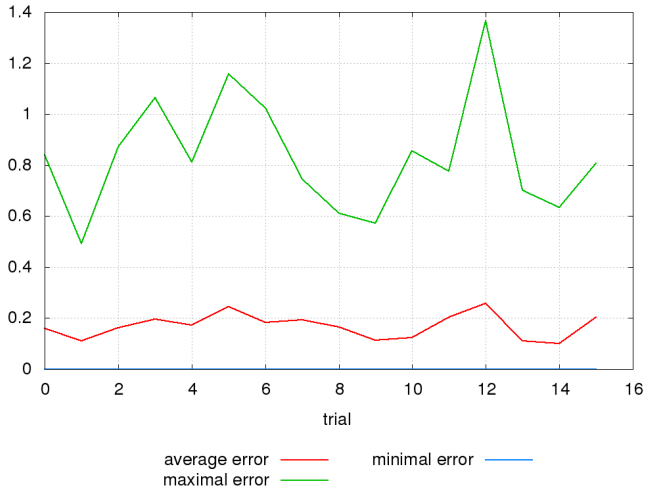

Figure 4. Trials run example

Summary results, for all network configurations, resulting averaging all 16 trials can be seen on fig 5. Testing neuron with one hidden layer configuration (cyan color) have in all experiments best error results. Second best results achieved McCulloch-Pitts neuron with one hidden layers. For two hidden layers, backpropagation learning algorithm have pure rusults in all cases - in next experiments is necessary to use some stochastic optimization insted of backpropagation.

Partial results, for each experiment can be seen on figures 6, 7, 8, 9. Each of these figures represents different neuron model, and is resulting minimum, maximum and average error for 16 trials for each approximating function.

DOI: 10.1002/cpe

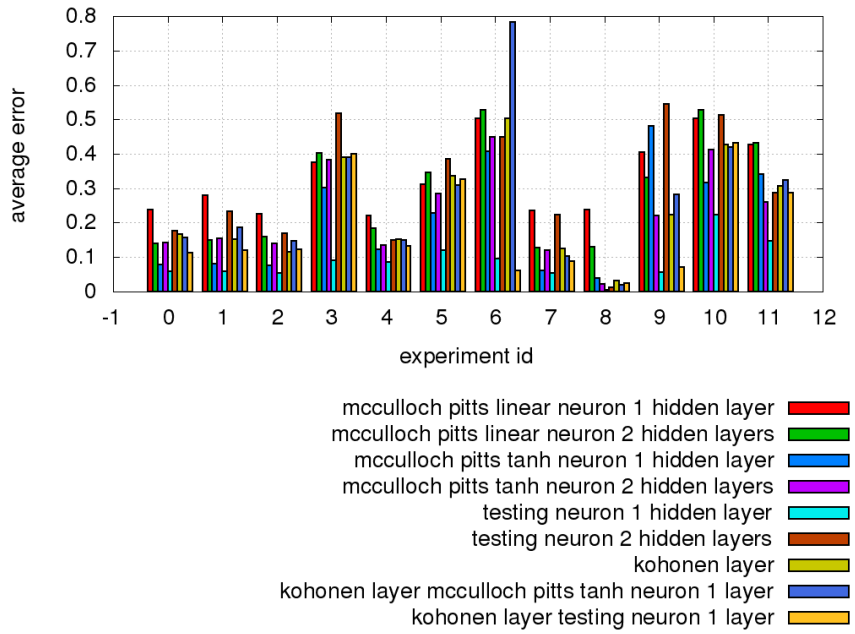

Figure 5. Average results

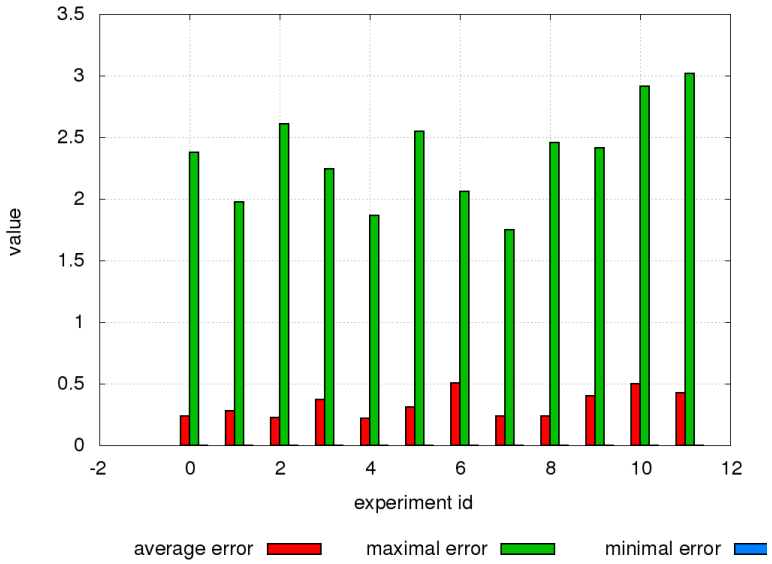

Figure 6. McCulloch-Pitts neuron with tanh activation function, single hidden layer

5. CONCLUSION

see ya

References

1. Kolmogorov's Theorem, http://neuron.eng.wayne.edu/tarek/MITbook/chap2/2_3.html
2. Silvia Ferrari, Member, IEEE, and Robert F. Stengel, Fellow, IEEE, Smooth Function Approximation Using Neural Networks, http://www.princeton.edu/ stengel/TNN2005.pdf

DOI: 10.1002/cpe

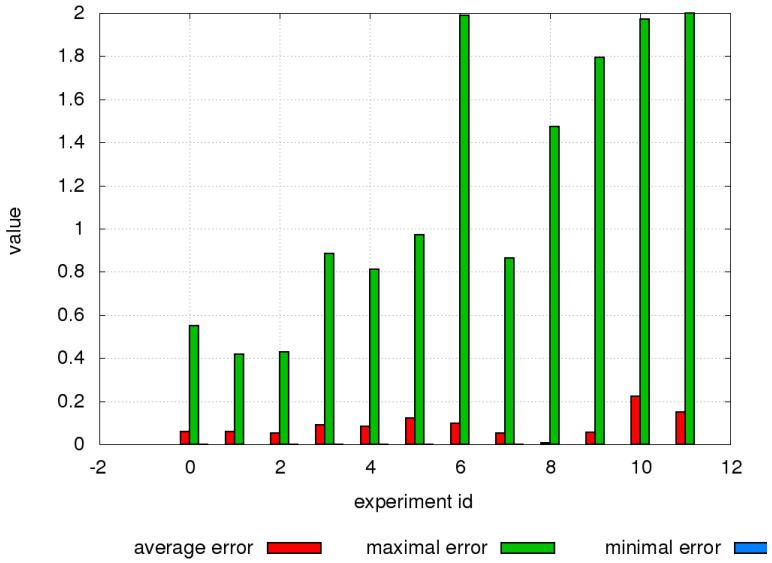

Figure 7. Testing neuron with tanh activation function, single hidden layer

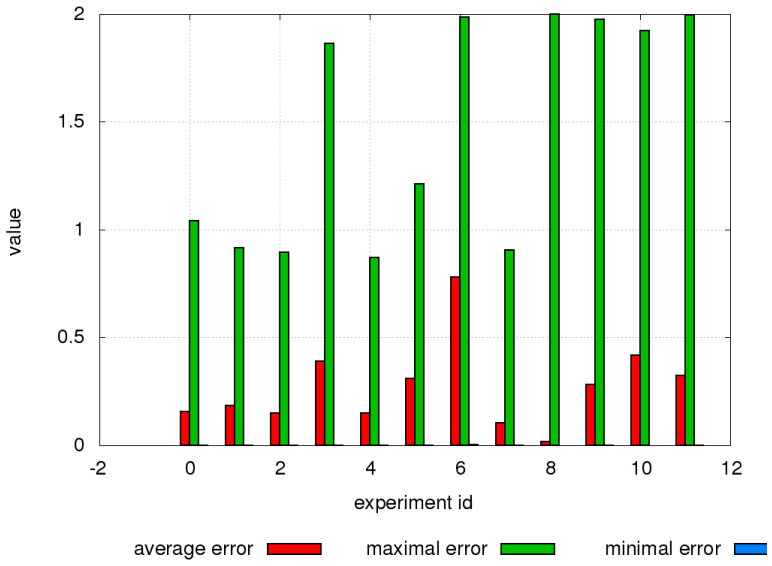

Figure 8. kohonen layer mcculloch pitts tanh neuron 1 layer summary

3. Yinyin Liu, Janusz A. Starzyk, Senior Member, IEEE, and Zhen Zhu, Member, IEEE, Optimized Approximation Algorithm in Neural Networks Without Overfitting http://www.ohio.edu/people/starzykj/network/research/Papers/Overfitting
4. Federico Girosi, Thomaso Poggio - Representation propherities of networks http://cbcl.mit.edu/people/poggio/journals/girosi-poggio-NeuralComputation-1989.pdf
5. Guang-Bin Huang, Senior Member, IEEE, Qin-Yu Zhu, and Chee-Kheong Siew, Member, IEEE - Real-Time Learning Capability of Neural Networks http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.2612&rep=rep1&type=pdf
6. R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996 - R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996 http://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf
7. neural network's applications http://www.alyuda.com/products/forecaster/neural-network-applications.htm
8. Taylor and Maclaurin Series https://cims.nyu.edu/ kiryl/Calculus/Section_8.7–Taylor_and_Maclaurin_Series/Taylor_and_Maclaurin_Series.pdf
9. TAYLOR'S THEOREM http://www.dcs.warwick.ac.uk/people/academic/Steve.Russ/cs131/NOTE26.PDF

DOI: 10.1002/cpe

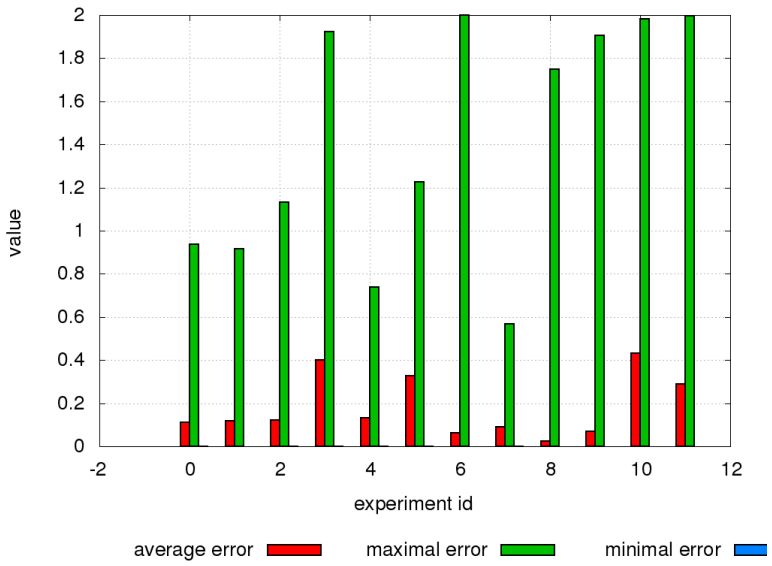

Figure 9. kohonen layer testing neuron 1 layer summary

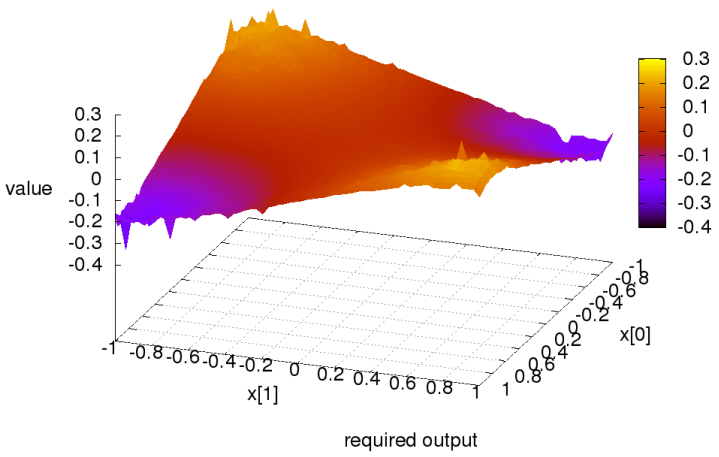

Figure 10. Required output example for experiment id 2

DOI: 10.1002/cpe

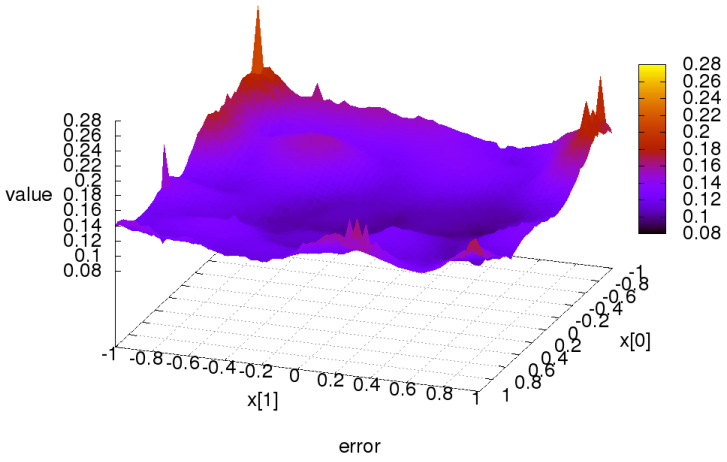

Figure 11. Error surface for McCulloch-Pitts neuron for experiment id 2

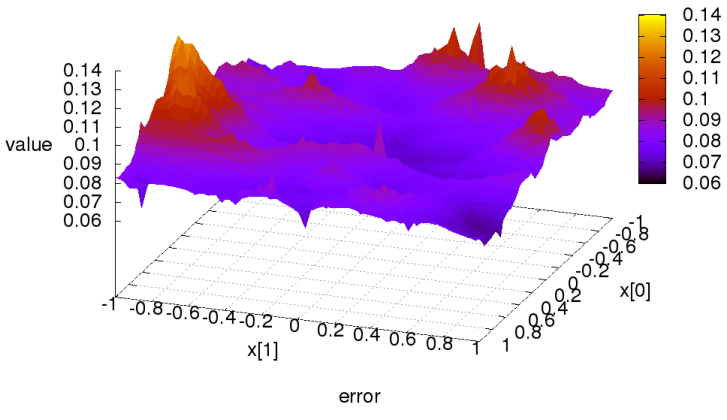

Figure 12. Error surface example for testing neuron for experiment id 2

DOI: 10.1002/cpe