Hypercube Neuron

Wojciech Krzysztof Fiałkiewicz

Abstract: Classic perceptron is limited to classifying only data which is linearly separable. In this work it is proposed neuron that does not have that limitation. Using hypercube architecture it is possible to describe any logic function. Neuron using that kind of architecture can discover any logic function from train data. It is also possible to write into that kind of neuron any logic rule.

Keywords: Neuron, Hypercube, Multidimensional Activation Function, Neural Network, Hypercube Neuron.

1. Introduction.

In [1] was introduced neuron based on hypercube architecture, in this paper it is presented enhancement of that neuron.

Some may argue that using hypercube to represent neuron leads to overcomplicated design, but it is known fact that artificial neural networks with many hidden layers are hard to train with back propagation algorithm. It may be the case that creating more complex neurons will allow to create neural networks with less hidden layers but with the same informational capacity.

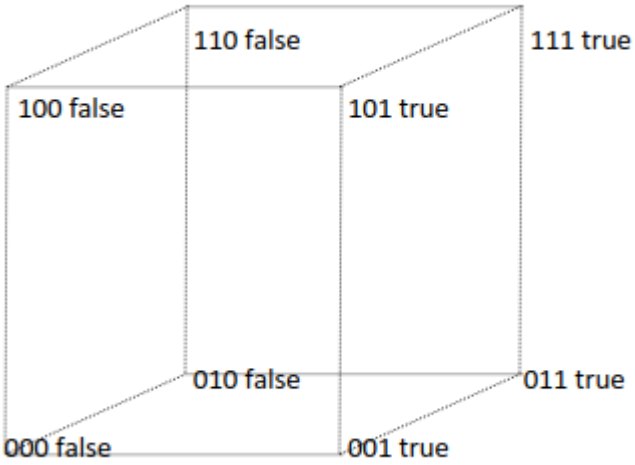
2. Logic functions.

Logic function can be described by its pattern by enumerating values for all possible sets of arguments . Pattern of logic function XOR is as follows : (false, false; false), (false, true; true), (true, false; true), (true, true; false).

Pattern of logic function can be described with use of hypercube architecture. For three dimensional function described by pattern :

Pattern P = (false, false, false; false),
(false, false, true; true),
(false, true, false; false),
(false, true, true; true),
(true, false, false; false),
(true, false, true; true),
(true, true, false; false),
(true, true, true; true).

There is hypercube that visualizes that logic function shown in Graph 2.1.



Graph 2.1 Three dimensional hypercube of logic function described by pattern P.

3. Linearly separable patterns of logic functions.

Classic perceptron can only be trained data sets that are linearly separable. Table (3.1) shows amount of logic functions with linearly separable patterns for given arguments amount (data from [2]), as well as total amount of logic functions for the same amount of arguments.

arguments amount	linearly separable	total functions	percent
2	14	16	87,5
3	104	256	40,625
4	1882	65536	2,871704102
5	94572	4294967296	0,002201926
6	15028134	1,84467E+19	8,14677E-11
7	8378070864	3,40282E+38	2,46209E-27
8	17561539552946	1,15792E+77	1,51664E-62
9	144130531453121000	1,3408E+154	1,075E-135

Table 3.1 Amount of logic functions for given arguments amount.

Last column of Table (3.1) shows percent of logic functions that classic perceptron can learn. In this paper is proposed neuron that can learn any logic function.

4. Neuron with single input.

In neuron input value is x, activation function is f(x), w is weight of its single input. Neurons exit value is equal to f(w*x).

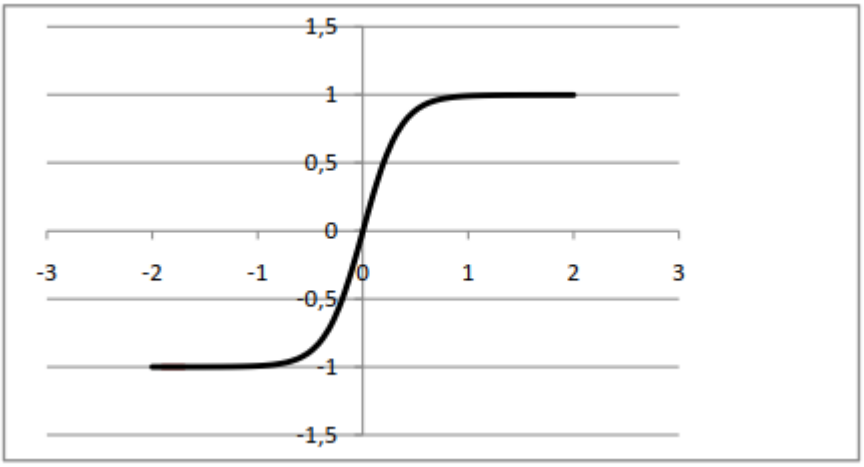
f(x) is defined as follows :

$$4.1\ f(x) = \frac{At-Ab}{2.0} * bsgm(x) + \frac{At+Ab}{2.0}$$

bsgm(x) is defined as follows :

$$4.2\ bsgm(x) = \frac{2.0}{(1+e^{-B*x})} - 1.0, \text{ where } B = 5{,}625$$

So At and Ab are respectively top and bottom asymptote of function f(x).



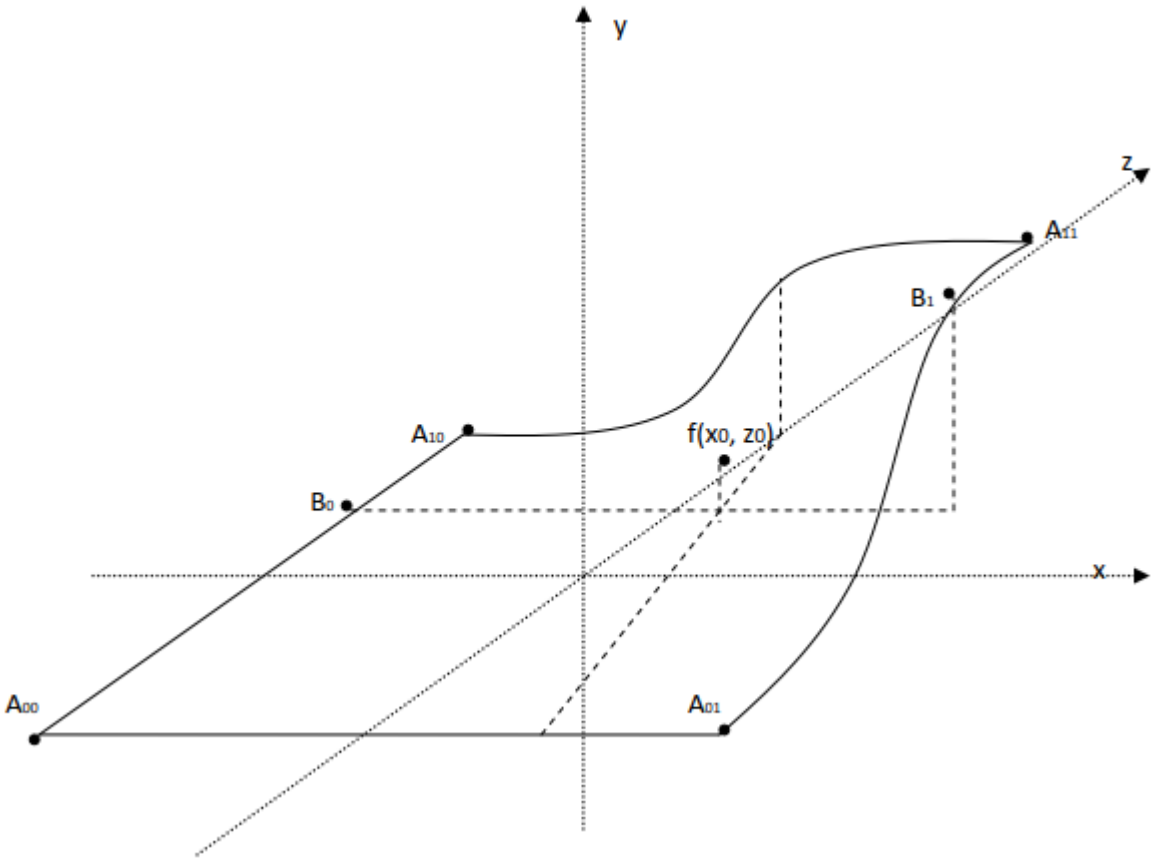
Graph of f(x) for At = 1.0 and Ab = -1.0.

Neuron with single input has three parameters that will be changing during learning process, they are At, Ab and w.

5. Neuron with two inputs.

In neuron with two inputs inputs are defined as x and z, weights on inputs as wx and wz, activation function f(x,z) is defined with use of four asymptote points A00, A01, A10 and A11 .

Graph 5.1 shows neurons activation function skeleton for A00= 0, A01= 0, A10= 0, A11= 1.



Graph 5.1 – sample skeleton of two dimensional function.

Function f(x,z) is defined with use of four f(x) (4.1) functions with At and Ab respectively equal to (A01 ,A00), (A11, A01), (A11, A10) and (A10, A00).

Value of function f(x,z) in point (x0 ,z0) is calculated in following way :
Firstly is reduced dimension z by calculating B0 and B1 asymptote points defining one dimensional f(x) function with x argument. Secondly this function is used to obtain f(x0) value equal to f(x0 ,z0).

$$B0 = \frac{A10 - A00}{2.0} * bsgm(wz * z0) + \frac{A10 + A00}{2.0}$$

$$B1 = \frac{A11 - A01}{2.0} * bsgm(wz * z0) + \frac{A11 + A01}{2.0}$$

$$f(x_0) = \frac{B1-B0}{2.0} * bsgm(wx * x0) + \frac{B1+B0}{2.0}$$

$$f(x_0, z_0) = f(x_0) = \frac{1}{4}[A00 * [1 - bsgm(wz * z)] * [1 - bsgm(wx * x)] + A01 * [1 - bsgm(wz * z)] * [1 + bsgm(wx * x)] + A10 * [1 + bsgm(wz * z)] * [1 - bsgm(wx * x)] + A11 * [1 + bsgm(wz * z)] * [1 + bsgm(wx * x)]].$$

Neuron with two inputs has six parameters that will be changing during learning process, they are A00, A01, A10, A11 , wx and wz.

6. Neuron with N inputs.

Neuron with N inputs has N weights w1, w2, .., wn on its inputs. Its activation function is defined by 2^N asymptote points.

Calculating exit value of neuron with N inputs involves reducing in each step of algorithm one dimension of hypercube by calculating asymptote points for hypercube with lower dimensions.

Following algorithm can be used to calculate exit value of neuron with N inputs :

(6.1)
```
double ExitValue(double *Weights, double *Inputs, double *Points, int N)
{
	int iPairsAmountANDStepSize;

	iPairsAmountANDStepSize = 2^(N-1);
	for(int iDim=N-1;iDim>=0;iDim--)
	{
		for(int iPair=0;iPair<iPairsAmountANDStepSize;iPair++)
		{
			Points[iPair]=FuncBSGM(Points[iPair],
			Points[iPair+iPairsAmountANDStepSize],
			Weights[iDim]*Inputs[iDim]);
		}
		iPairsAmountANDStepSize /= 2;
	}
	return Points[0];
}
```

Where Weights is array of weights on neuron inputs, Inputs is array of input values that neuron will process, Points is array of asymptote points defining neuron activation function and FuncBSGM is function that calculates (4.1) function and takes Ab and At as parameter.

Exit value of neuron with N inputs can be specified as formula :

(6.2) $f(x_1, x_2, .., x_n) = \frac{1}{2^N}$ (A_0 * Fun(0, w_0, x_0, 0) * Fun(1, w_1, x_1, 0)* ... * Fun(N, w_N, x_N, 0) + A_1 * Fun(0, w_0, x_0, 1) * Fun(1, w_1, x_1, 1)* ... * Fun(N, w_N, x_N, 1) + ... + A_{2^N} * Fun(0, w_0, x_0, 2^N) * Fun(1, w_1, x_1, 2^N)* ... * Fun(N, w_N, x_N, 2^N))

Where x_0, x_1, .., x_N are neuron inputs values A_0, A_1 ... A_{2^N} are asymptote points defining neurons activation function and Fun(d, w, x, p) is defined as follows:

(6.3) Fun(d, w, x, p) = $\begin{cases}(1 - bsgm(w * x), if\ bit\ nr\ d\ of\ binary\ representation\ of\ p\ is\ 0.\\(1 + bsgm(w * x), if\ bit\ nr\ d\ of\ binary\ representation\ of\ p\ is\ 1.\end{cases}$

7. Supervised learning of neuron.

To change parameters defining neuron during learning process it is used gradient based algorithm. In this algorithm neurons error function is being minimized by modifying value of each of neuron parameter by its gradient, multiplied by learning parameter.

Learning change for each parameter of neuron is defined as follows :

$$(7.1)\ P = P - l * \frac{dE(P,\dots)}{dP}$$

Where P is neuron parameter that is being modified, l is learning parameter and E() is error function.

(7.2) $E() = (y - d)^2$, where y- exit value of neuron, d – desired exit value of neuron

Calculating gradient of any asymptote point is done with use of formula (6.2).

Gradient for asymptote point Am is as follows :

$$\frac{dE(Am,\dots)}{dAm} = \frac{1}{2^N} * 2 * (y - d) * Fun(0, w_0, x_0, m) * Fun(1, w_1, x_1, m) * \dots * Fun(N, w_N, x_N, m)$$

Where N is number of neuron inputs , Fun () is function (6.3), w0, w1, ..wn are neuron weights and x0, x1, ...,xn are neuron inputs.

Calculating gradients of weights is more complicated it requires modifying algorithm (6.1), to leave dimension of weight for which gradient is being calculated to be the last one to reduce. Following algorithm shows how it is accomplished :

(7.3) void LowerDimsToDim(int iDimNr, double *Weights, double *Inputs, double *Points, int N)

```
{
	int iPairsAmountANDStepSize;
	int iSecondPairsStart;

	iPairsAmountANDStepSize = 2^(N-1);
	for(int iDim = N-1;iDim>=0;iDim--)
	{
		if(iDim!=iDimNr)
		{
			for(int iPair=0;iPair<iPairsAmountANDStepSize;iPair++)
			{
				Points[iPair]=FuncBSGM(Points[iPair],
				Points[iPair+iPairsAmountANDStepSize],
				Weights[iDim] * Inputs[ iDim]);
			}
			if(iDim<iDimNr)
			{
				for(int iPair=0;iPair<iPairsAmountANDStepSize;iPair++)
				{
Points[iSecondPairsStart + iPair]=FuncBSGM(Points[iSecondPairsStart + Pair],
					Points[iSecondPairsStart +iPair+iPairsAmountANDStepSize],
					Weights[iDim] * Inputs[ iDim]);
				}
			}
		}else
		{
			iSecondPairsStart = iPairsAmountANDStepSize;
		}
		iPairsAmountANDStepSize /= 2;
	}
	Points[1] = Points[iSecondPairsStart];
}
```

When algorithm (7.3) is executed as a result it leaves two values in Points array, which are respectively Ab and At asymptote in dimension specified as iDimNr. Using those values it is possible to attain neuron exit value y with use of function (4.1).

$$(7.4)\ y = \frac{At-Ab}{2.0} * bsgm(Weights[iDimNr] * Inputs[iDimNr]) + \frac{At+Ab}{2.0}$$

So the gradient of weight number iDimNr is equal to :

$$(7.5)\ \frac{dy}{d(w_{iDimNr})} = \frac{At-Ab}{2.0} * \frac{d(bsgm(wiDimNr * xiDimNr))}{d(wiDimNr * xiDimNr)} * \text{xiDimNr.}$$

8. Open Source Implementations.

Implementation of Hypercube Neuron described in this paper can be found at [5]. That kind of neurons have limited amount of inputs, implementation of neural network that consitst of that neurons and is trainable with backpropagation algorithm can be found at [6]. Implemetation of neural networks with added modification to Hypercube Neuron that allows to construct neural networks that are fully connected and are trainable with backpropagation algorithm can be found at [7].

9. Future Work.

Hybrid learning systems based on neural networks presented in [3],[4] are combining imprinting symbolic knowledge into network structure and training that networks with back propagation algorithm with unclassified data. Neurons used in mentioned approaches to remember symbolic rules are realizing AND and OR logic functions. Neuron presented in this paper has ability to realize any logic function. Nets using that neurons can also be trained with back propagation algorithm to refine or discover new rules in data.

References.

[1] Fialkiewicz, Wojciech (2002). Reasoning with neural networks. M.Sc Thesis. Wroclaw University of Technology.

[2] Gruzling, Nicolle (2006). Linear separability of the vertices of an n-dimensional hypercube. M.Sc Thesis. University of Northern British Columbia.

[3] Geoffrey G. Towell , Jude W. Shavlik (1994). Knowledge-Based Artificial Neural Networks.

[4] Artur S. d'Avila Garcez , Luís C. Lamb , Dov M. Gabbay. A Connectionist Inductive Learning System for Modal Logic Programming.

[5] https://sourceforge.net/projects/hypercubeneuron

[6] https://sourceforge.net/projects/nnhn

[7] https://sourceforge.net/projects/fcnnhn