Special Neural Network Architectures for Easy Electronic Implementations

Bogdan M. Wilamowski
Auburn University, AMNSTC, Auburn, Alabama, USA
wilam@ieee.org

***Abstract*— An overview of various neural network architectures is presented. Depending on applications some of these architectures are capable to perform very complex operations with limited number of neurons, while other architectures, which use more neurons, are easy to train. There are neural network architectures which have very limited requirements for training or no training is required. The importance of the proper learning algorithm was emphasized because with advanced learning algorithm we can train these networks, which cannot be trained with simple algorithms. When simple training algorithms, such as EBP are used, neural networks with larger number of neurons must be used to fulfill the task.**

I. INTRODUCTION

The purpose of this presentation is not to give an introduction to neural networks but to give practical advice about its implementations. Most researchers who are trying to use neural networks are facing two major challenges:

(1) How to train this network and what software/algorithm to use
(2) What neural network architecture (topology) to employ and how many neurons to use.

The first challenge is relatively easy to solve. With little effort a neural network software can be found and used. What many people are not aware of is that not all popular algorithms can train every neural network. Surprisingly the most popular EBP (Error Back Propagation) algorithm [1,2] cannot handle more complex problems while other more advanced algorithms [3,4] can.

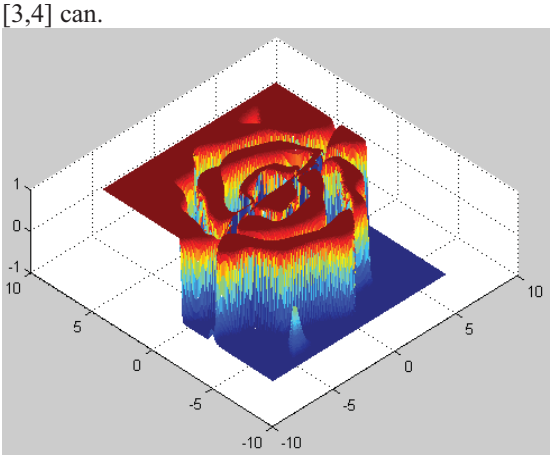


Fig. 1. Solution of the two spiral problem with NBN algorithm [4] using fully connected architecture with 8 neurons and 52 weights.

For example, training the popular test bench with Wieland two spiral problem can be solved (Fig. 1) with second order using cascade architecture with 8 neurons but in order to solve the same problem with the EBP algorithm (Fig 2) at least 16 neurons and weights in cascade architecture are needed. With only 12 neurons in cascade, the NBN algorithm can produce a very smooth response (Fig. 3) with less than 150 iterations but we were not able to solve this 12 neuron problem with EBP algorithm despite many trials with 1,000,000 iterations limit. More detailed information about the relationship between complexity of network topology and learning algorithms can be found at [5]. The conclusion is that with a better learning algorithm the same problem can be solved with simpler hardware.

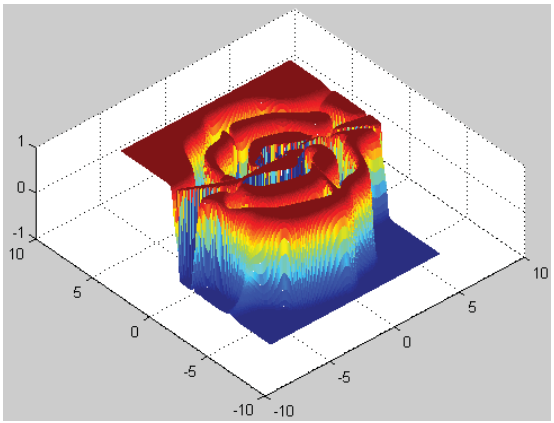


Fig. 2. Solution of the two spiral problem with EBP algorithm using fully connected architecture with 16 neurons and 168 weights.

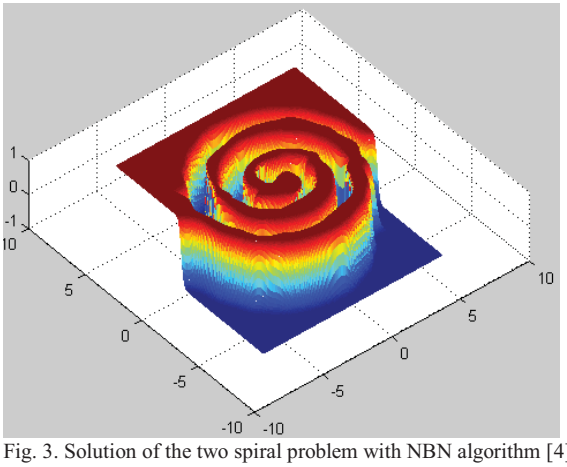


Fig. 3. Solution of the two spiral problem with NBN algorithm [4] using fully connected architecture with only 12 neurons and 102 weights

The second challenge about neural network architecture is more difficult. In most cases neural network architectures are selected by trial and error process. Often success depends on a lucky guess. This presentation may provide some hints for neural network architecture/topology selection. The selected architecture may depend on many factors such as:

- How to get maximum performance with minimal hardware?
- What topology should be selected so network can be easily trained?
- How to make network less sensitive to element tolerances?
- Is it possible to have neural networks which need not to be trained or can only very simple training algorithm be used?

In the following chapters we will discuss these issues.

II. How to get maximum performance with minimum hardware

Another test bench for neural networks is the parity-N problem. The simplest parity-2 problem is also known as the XOR problem. The larger the problem, the more difficult it is to solve it. The parity N problem can be solved analytically and the number of required neurons depends on the neural network topology [6].. In the case of the most popular MLP (Multi Layer Perceptron) architecture with one hidden layer there are at least 9 neurons required and 8*9+9 = 81 weights (Fig. 4). When connection across layers are allowed (Fig. 5) then the same problem can be solved only with 5 neurons and the total number of weights is 4*9+8+4+1=49. For cascade connection (Fig. 6) then only four neurons are required and the total number of weights is 9+10+11+12=42. For the pipeline architecture (Fig. 7) the parity 8 problem can be solved with 3 neurons and 8+2+2=12 weights.

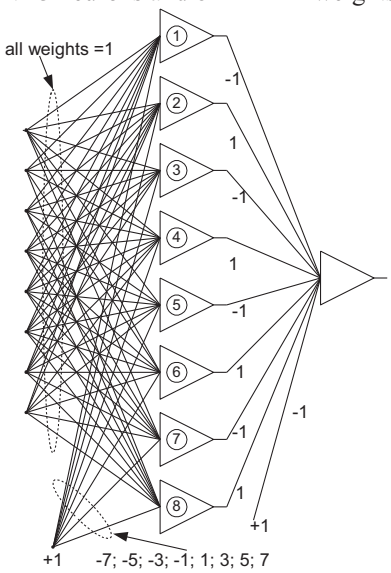


Fig. 4. Parity-8 problem with feedforward bipolar neural network with one hidden layer.

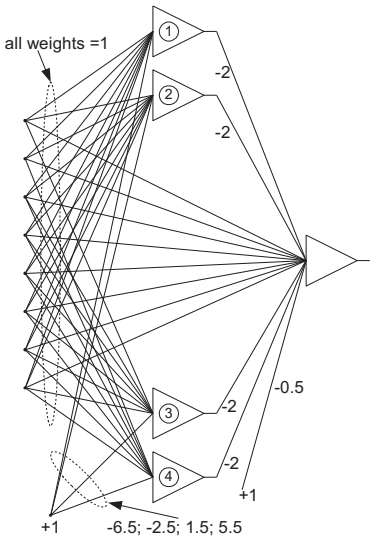


Fig. 5. Fully-connected layered bipolar neural network with one hidden layer for the parity-8 problem.

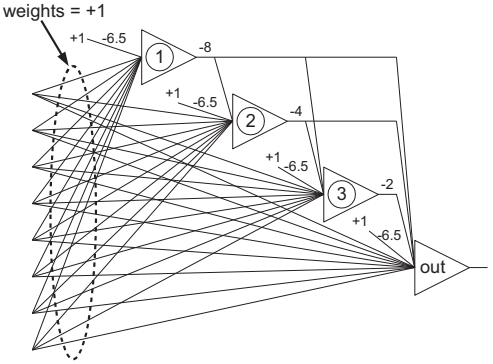


Fig. 6. Bipolar implementation of a fully connected cascade neural network for the parity-8 problem.

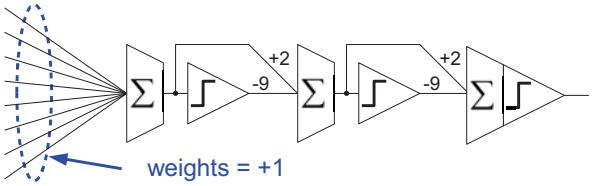


Fig. 7: Bipolar implementation of a pipeline neural network for the parity-8 problem.

The pipeline architecture is specific only to the parity-N problems; therefore, one may conclude that the cascade network (Fig. 6) is the most powerful, since it would require a minimum number of elements. At the same time, because of a long signal path across many nonlinear elements, the cascade architecture is more difficult to train. One may also notice that when in the feed forward neural networks all possible connections between neurons are implemented (fully connected network) then the resulting topology is the cascade architecture. However the most popular is MLP architecture primarily because it is easiest to write training software for MLP networks. For example very popular MATLAB Neural Network Toolbox is primarily designed for MLP networks. Surprisingly the MLP architecture is the least efficient.

III. WHAT TOPOLOGY SHOULD BE SELECTED SO NETWORK CAN BE EASILY TRAINED

The general rule is that less layers in the neural network then it is easier to train it. It is very easy to train one layer neural network where the solution can be obtained with only several iterations. With more layers the network is becoming less transparent and it is more difficult to train because signals have to pass more neurons with nonlinear activation functions. From this perspective it takes longer to find solution for MLP network with multiple layers than for the same network where connections across layers are allowed (Fig. 5). At the same time, this network can handle more complex tasks. The problem is that only limited number of software packages were developed to train arbitrarily connected neural networks such as SNNS [7] and NBN [8]. SNNS package has limited applications because it uses only the first order algorithms and its ability to train complex neural networks is limited.

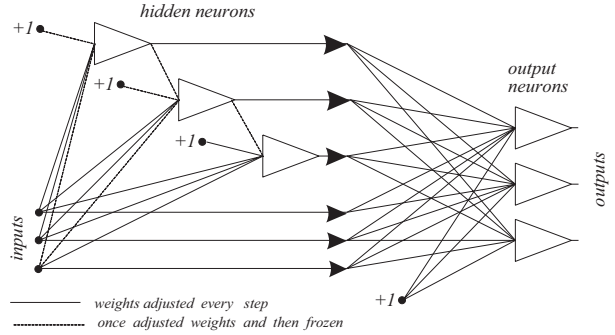


Fig. 8. Cascade correlation algorithm

There are some neural network architectures which are merged with training algorithms. The best example of such an approach is cascade correlation algorithm [9]. This unique algorithm is dedicated to train cascade architectures (Fig. 8) and the training process is relatively fast and simple. At each time only one neuron (one layer) is being trained in a very specific sequence. The network is being built during the training process by adding neurons and training them. This is one of very few algorithms where network architecture is being developed at the same time as it is trained. This algorithm is fast and easy and it is more powerful than EBP algorithm [9]. For example the two spirals problem, mentioned in the introduction, can be solved with 12-19 neurons and it requires at least 10 times less iteration than EBP algorithm [9]. Of course the cascade correlation algorithm is not as powerful as NBN [4], which can solve the same problem with only 8 neurons (Fig 1), but the computations are more advanced.

There are other neural network architectures, which require even less training. One of them is the FLN Functional Link Network [10] (Fig. 9) and another is Polynomial Neural Network (Fig 10). The latter one differs from the first one that only polynomial are being used as nonlinear terms. Both networks can be very easily trained because only one layer training is required and it has to be performed only once.

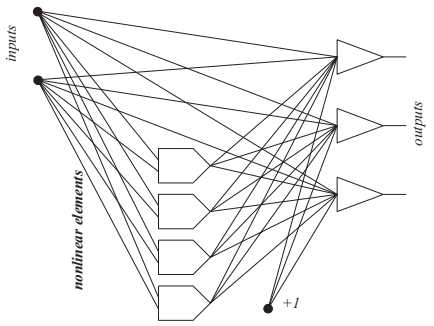


Fig. 9. Functional link network with arbitrary nonlinear terms.

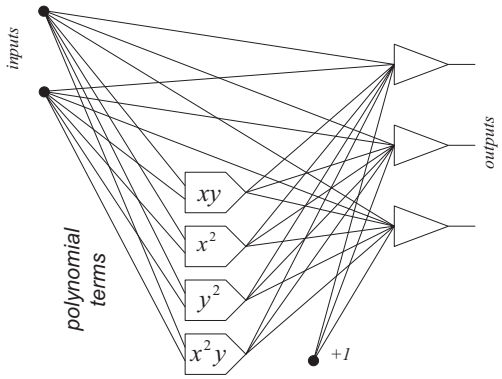


Fig. 10. Functional link network with polynomial nonlinear terms.

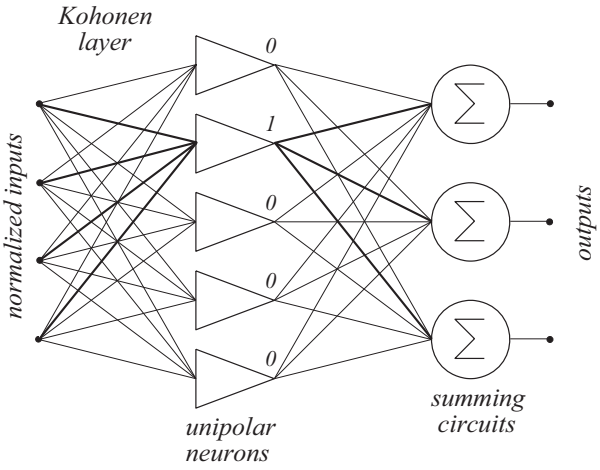


Fig. 11. Counterpropagation networks.

Another very interesting neural network architecture is the counterpropagation network [11,12] shown in Fig. 11. If the training patterns are normalized then number of neurons in the hidden layer must be equal to the number of patterns. Weights in the first layer are equal to the input patterns and weights in the output layer are equal to the output patterns. Therefore, no training process is required. This network has a generalization feature, which means that it may produce good results also for the patterns which were never stored in the network. In this case the network will generate an weighted average of the closest patterns which were stored in the network. A disadvantage of the counterpropagation network is that number of neurons in the hidden layer must be equal to number of training patterns and this number is sometimes excessively large. This limitation can be solved by storing not all patterns but only center of representative clusters

of patterns. Such network is shown in Fig. 12 and it has a nice name of LVQ -Learning Vector Quantization, but it is basically not much different than the counterpropagation network.

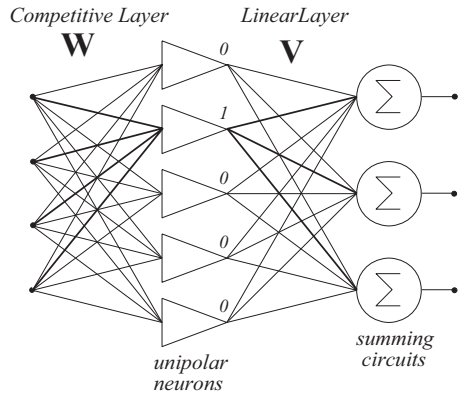


Fig. 12. LVQ Learning Vector Quantization networks.

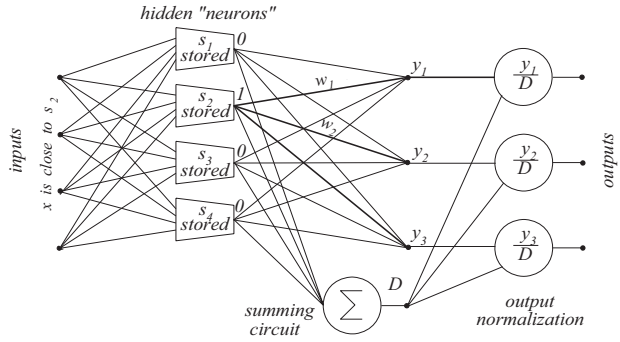


Fig. 13. RBF - Radial basis function networks.

Another network which needs only very limited training or no training at all is the RBF - Radial Basis Function network [13] (Fig.13). Also, in this case the number of processing units "neurons" is equal to number of patterns or number of clusters. Each of these "neurons" responds only to the input signals close to the stored pattern. The output signal h_i of the *i-th* hidden "neuron" is computed using the following formula.

$$h_i = \exp\left(-\frac{\|\mathbf{x}-\mathbf{s}_i\|^2}{2\sigma^2}\right) \quad (1)$$

The RBF can be used only when we have a significant computing power, because each processing unit "neuron" has to calculate the Euclidean distance, then calculate Gaussian function eq.(1) and finally several division operations must be performed. It would be very difficult to implement RBF networks in electronic hardware.

IV SHALL WE USE NEURAL NETWORKS OR FUZZY SYSTEMS

For most problems we can use either neural networks or fuzzy systems. Both approaches perform function of nonlinear mapping, but they use a completely different philosophy. Fuzzy systems can be relatively easily designed using intuitive rules of operations. Neural networks need to be trained with sample patterns. One may notice that if these patterns are representing fuzzy rules then development neural networks and fuzzy systems could be very similar. Even for neural networks we need training process to find weights, while all fuzzy system parameters are extracted from fuzzy rules.

In case of fuzzy systems, the number of inputs are practically limited to two or three, while neural networks do not have limitations of number of inputs.

Neural networks are capable to produce control surfaces with significantly higher accuracy and in microcontroller applications are working faster with shorter number of instructions [13]

Because of very raw control surfaces of fuzzy controllers fuzzy systems are not used directly in the control loop but they are just used to adjust parameters of traditional PID controller.

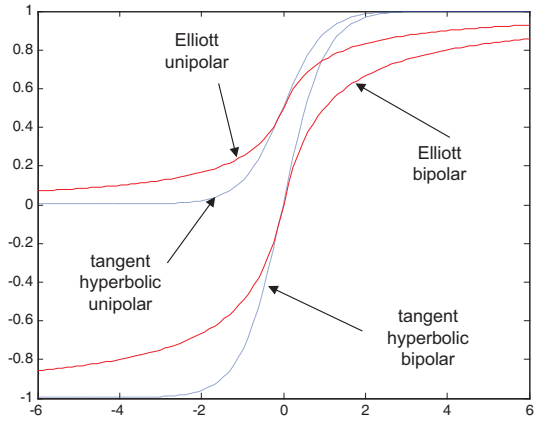


Fig. 14. Comparison of various activation functions.

It is not easy to implement neural networks on microcontroller because it would be difficult to compute traditional tangent hyperbolic function. However neural networks can use different sigmoid like function and very similar results can be obtained. When the tangent hyperbolic function is replaced by the Elliott (Fig. 14) function

$$f(net) = \frac{net}{1+|net|} \quad (2)$$

then the computations activation function becomes relatively simple. Neural networks are compared with fuzzy systems in microcontroller implementations [14]. With the Elliott function the neural network implementations resulted with shorter code, faster operation, and much more accurate results. Fig. 15 shows the comparison of several controllers for the same desired control surface implemented in the popular HC11 microcontroller, using various fuzzy and neural network architectures.

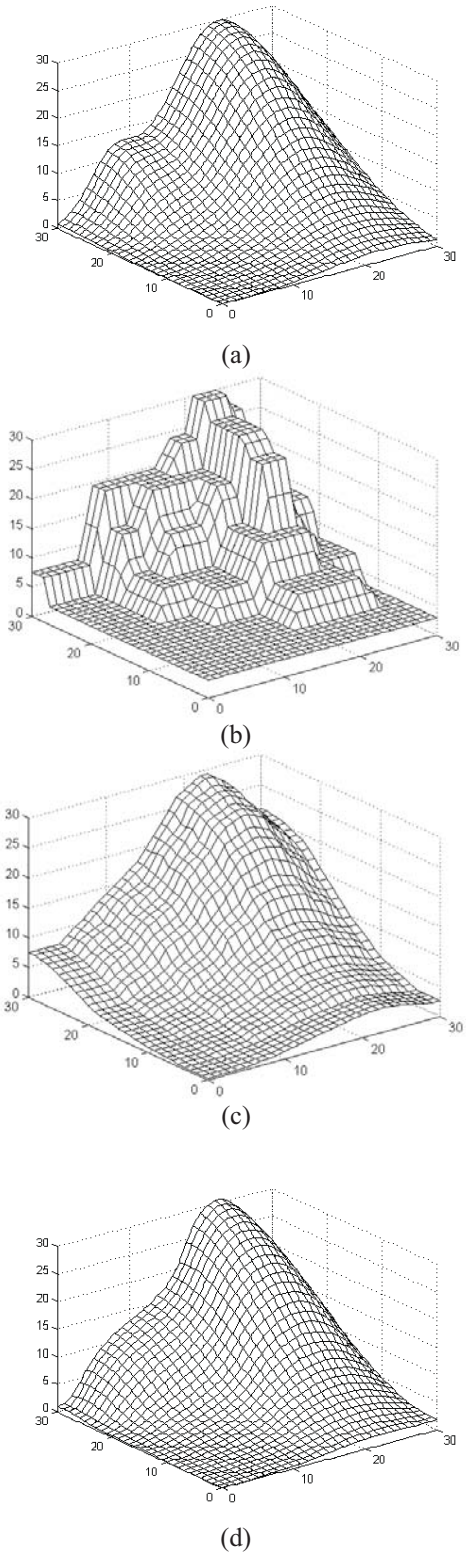


Fig. 15. Control surfaces for various controllers. (a). Required control surface, (b) Fuzzy Mamdani type with trapezoidal membership functions, (c) Fuzzy Tagagi-Sugeno type with triangular membership functions, (d) neural controller with six neurons in 2-1-1-1-1-1-1 architecture.

One may conclude that neural networks produce a smoother and more acurate surface, but suprisingly neural networks require shorter assembly code and it has shorter processing time [14].

Table 1. Error comparison for various types of fuzzy and neural controllers implemented on Motorola 68HC711E9 microcontroller to match the control surface from Fig 16 (a). All fuzzy systems had 7 membership functions on each input or output if applicable.

Approach used	Length of code (bytes)	Processing time (ms)	error MSE (%)
Mamdani fuzzy controller with trapezoidal	2324	1.95	9.45
Mamdani fuzzy controller with triangular	2324	1.95	6.71
Mamdani fuzzy controller with Gaussian	3245	39.8	5.85
TSK fuzzy controller with trapezoidal	1502	28.5	3.09
TSK fuzzy controller with triangular	1502	28.5	2.19
TSK fuzzy controller with Gaussian	2845	52.3	3.06
Neural network with 3 neurons in cascade	680	1.72	0.0578
Neural network with 5 neurons in cascade	1070	3.3	0.0093
Neural network with 6 neurons in one hidden layer	660	3.8	0.0302

V IMPLEMENTING NEURAL AND FUZZY SYSTEMS ON MICROCONTROLLERS

If Elliott function is used then it is relatively easy to implement neural networks on higher end microcontrollers such as HC11 [14]. As long the microcontroller can perform division operation the activation function can be easily calculated using Elliot functions. The results are of course not the same as in the case of tangent hyperbolic functions and often in order to obtain similar results slightly larger neural network can be used but all problems can be solved. Table 1 shows comparison of results for different types of neural and fuzzy systems implemented on 68HC711E9. The on-board features of the 68HC711E9 are 512 bytes of RAM and EEPROM and 12K bytes of UV erasable EPROM. The processor was used with an 8 MHz crystal, allowing an internal clock frequency of 2 MHz

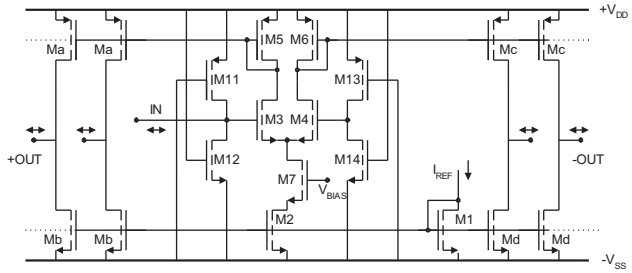
Neural networks can be also implemented on very simple microcontrollers, where there are no 16 bit multiplication instructions and there are no division or floating point operations. In this case, a special care must be taken in order to develop routine for activation function calculation such as Microchip Microcontroller (PIC) [15]. This very simple microcontroller allows for implementation of neural networks with up to 256 weights and practically unlimited number of neurons (less than 127). The obtained results are within couple percent accuracy. The PIC microcontroller programming was done in assembly language using pseudo floating point calculations and activation function approximating tangent hyperbolic function. For most of the tested problems the response time was around a couple

milliseconds. If the same PIC was programmed using C language the response time was reduced by the factor of 10.

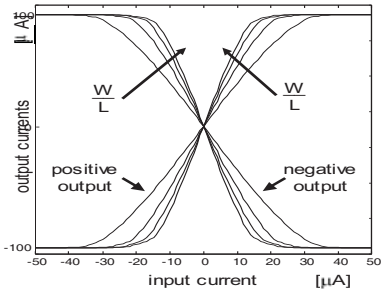
VI Implementing Neural and Fuzzy Systems in Analog Hardware

Implementations of neurons in VLSI chips are relatively simple, as long there is no need for weight adjustments. Note, that every differential pair generates a sigmoid type nonlinear function that is suitable for neural processing. In the case of bipolar transistor pair exact tangent hyperbolic function is generated. In the case of MOS implementations the MOS differential pair also sigmoid type of transfer function is produced (Fig. 16).

It is much more difficult to implement weights. A weight circuit is usually much more complicated than the neuron circuit. When a digital adjustment is required for each weight then the complicity of the weight circuit is similar to to the complexity of digital to analog converter. When fixed values of weights can be used, then the circuit can be significantly simplified. Each weight can be set by proper W/L ratios of output transistors as shown in Fig. 16(a). By taking signals from different outputs of the differential pairs, both positive and negative weights can be implemented.



(a)



(b)

Fig. 16 Neuron circuit with weights determined by W/L ratios of output transistors.

VII Conclusion

Various neural network architectures was described and compared. Depending on applications some of these architectures are capable to perform very complex operations with limited number of neurons, while other architectures, which use more neurons, are easy to train. There are neural network architectures which have very limited requirements for training or no training is required. The importance of the proper learning algorithm was emphasized because with advanced learning algorithm we can train these networks, which cannot be trained with simple algorithms. When simple training algorithms, such as EBP are used, neural networks with larger number of neurons must be used to fulfill the task.

References

[1] Rumelhart, D. E., Hinton, G. E. and Wiliams, R. J, "Learning representations by back-propagating errors", *Nature,* vol. **323**, pp. 533-536, 1986

[2] Scott E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *1988 Connectionist Models Summer School*, San Mateo, CA, 1988. Morgan Kaufmann.

[3] Hagan, M. T. and Menhaj, M., "Training feedforward networks with the Marquardt algorithm", IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989-993, 1994

[4] B. M. Wilamowski, N. J. Cotton, O. Kaynak, G. Dundar, "Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks," *IEEE Trans. on Industrial Electronics,* vol. 55, no. 10, pp. 3784-3790, Oct 2008

[5] Hao Yu and B. M. Wilamowski, "C++ Implementation of Neural Networks Trainer", 13-th *International Conference on Intelligent Engineering Systems, INES-09,* Barbados, April 16-18, 2009

[6] B. Wilamowski, D. Hunter, "Solving Parity-n Problems with Feedforward Neural Network," Proc. of the IJCNN'03 International Joint Conference on Neural Networks, pp. 2546-2551, Portland, Oregon, July 20-23, 2003

[7] Stuttgart Neural Network Simulator SNNS http://www.ra.cs.uni-tuebingen.de/SNNS/

[8] NNT - Neural Network Trainer http://www.eng.auburn.edu/~wilambm/nnt/

[9] S.E Fahlman,.and C. Lebiere, "The cascade- correlation learning architecture" nn D. S.Touretzky, Ed. *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Mateo, Calif., (1990), 524-532.

[10] Y. H. Pao, Adaptive Pattern Recognition and Neural Networks, Reading, Mass. Addison-Wesley Publishing Co. 1989

[11] Hecht-Nielsen, R. 1987. Counterpropagation networks. *Appl. Opt.* 26(23):4979-4984

[12] J.M. Zurada, *Artificial Neural Systems*, PWS Publishing Company, St. Paul, MN, 1995

[13] B. M. Wilamowski and R. C. Jaeger, "Implementation of RBF Type Networks by MLP Networks," *IEEE International Conference on Neural Networks*, Washington, DC, June 3-6, 1996, pp. 1670-1675.

[14] Bogdan Wilamowski and Jeremy Binfet " Microprocessor Implementation of Fuzzy Systems and Neural Networks," *International Joint Conference on Neural Networks (IJCNN'01),* pp. 234-239, Washington DC, July 15-19, 2001.

[15] N.J. Cotton, B.M. Wilamowski, and G. Dundar "A Neural Network Implementation on an Inexpensive Eight Bit Microcontroller" *12th INES 2008 -International Conference on Intelligent Engineering Systems,* Miami, Florida, USA, February 25-29, 2008, pp. 109-114.