REALIZATION OF DAUBECHIES TRANSFORM USING LATTICE STRUCTURE

J. Stolarek
Technical University of Lodz
Institute of Computer Science
ul. Wolczanska 215, 90-924 Lodz, Poland
e-mail : stolarek@ics.p.lodz.pl

1. Introduction

Wavelet transforms play an important role in processing, compression and analysis of signals [1, 7, 8]. Wavelet transform is linear, similarly to DFT, DCT, DST and DHT. However it uses different basis functions. Wavelet functions, unlike sinusoidal functions used by other mentioned transforms, are precisely located in time domain, thus allowing to obtain detailed information about local characteristics of a signal. However, there is no single basis function for all wavelet transforms. Many wavelet families are known. It is important, that chosen wavelet family closely corresponds to characteristics of analysed signal. One of the most frequently used wavelet functions are the Daubechies wavelets, especially the simplest Daubechies 4 that allows efficient implementation and straightforward interpretation of results. Thanks to their properties, Daubechies wavelets are well suited for processing natural signals. However, it is possible to synthesize wavelet function more suitable for particular task using adaptive methods [1, 8]. Neural networks offer the possibility to adaptively synthesize wavelet functions. Especially promising are the fast multilayer linear neural networks that are able to realize wide class of linear transforms [2, 6].

This paper presents analysis of new approach to wavelet synthesis, based on lattice structure proposed in [9]. Main goal is practical verification of this approach, by showing that neural network based on proposed lattice structure is able to learn existing Daubechies wavelet transforms.

2. Lattice structure

Lattice structure for realization of wavelet transforms was proposed in [9]. Basic element of neural network based on this structure is presented in Fig. 1a. This element (further referred to as the “neuron”) has two inputs, two outputs and four different weights. Therefore it can be treated as two independent ordinary linear neurons, with two inputs and one output each. Operation performed by this element can be treated as a 2-by-2 matrix multiplication:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{, where} \quad P = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad . \tag{1}$$

Network architecture is presented in Fig. 1b. Weights of all neurons within one layer are identical. Neurons on the edge of a layer are wrapped around, which is equivalent to assuming that transformed signal is periodic. Neurons in the network are sparsely connected, i.e. the number of connections between layers is small, unlike the multilayer perceptron, where neurons in adjacent layers are connected all-to-all.

Network's ability to perform particular wavelet transform depends on number of layers. In case of three-layer network in Fig. 1b, each output depends on six inputs, which means that this network is suitable for realization of 6-tap transform. In general, to calculate m-tap transform, m / 2 layers are necessary. Network shown in Fig. 1b has eight inputs and eight outputs, however it doesn't mean that it is capable of processing only eight-element signals. Actual implementation of network allows to operate on any signal of even length. It is possible thanks to identical weights of neurons in one layer.

Same network can be used for calculating the inverse transform. To achieve this, network direction should be reversed (inputs become outputs) and weight matrices, denoted in Equation 1 as P, should be replaced with inverse matrices:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = P^{-1} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \text{, where} \quad P = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad . \tag{2}$$

Network was taught using backpropagation algorithm based on signal flow graph method [3, 5], which

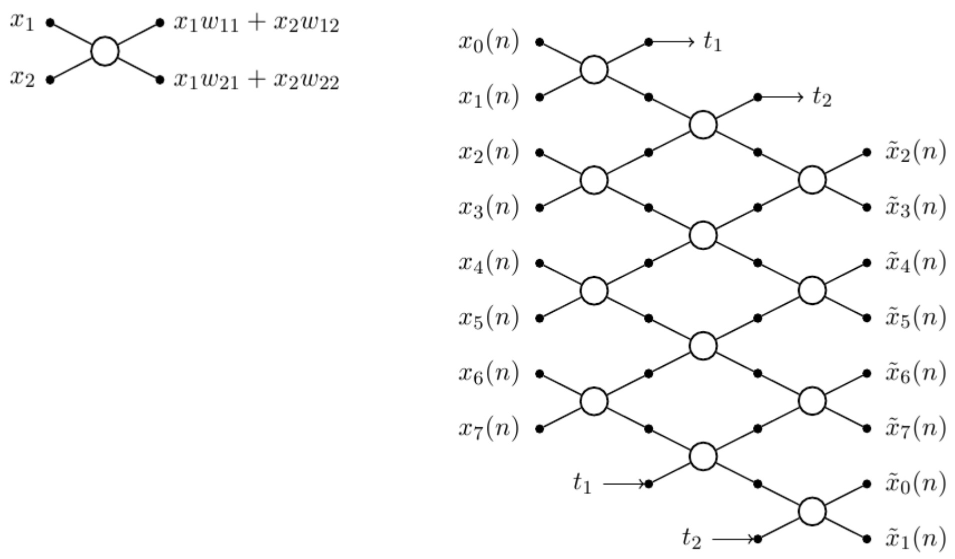

Figure 1: a) Basic structural element of network b) Structure realizing 6-tap wavelet transform

allowed for straightforward gradient calculation of error function in respect to weights. Weights modification was performed according to steepest descent algorithm [4]:

$$w_{n+1}=w_n-\eta\nabla E(w) \quad , \tag{3}$$

where w_n is weights vector in n-th iteration, η is the learning step and $\nabla E(w)$ is error function gradient calculated in respect to network weights.

Supervised teaching was used to teach the network – for each input signal expected output was known. Error function minimized during learning process is:

$$E=\sum_{i=0}^{k-1}\frac{(y_i-d_i)^2}{k} \quad , \tag{4}$$

where y_i denotes signal value on i-th output of network, d_i denotes expected value for that output and k denotes number of outputs. This is standard error function [4] subjected to normalization, i.e. divided by number of outputs. If normalization isn't performed, the error function tends to achieve values proportional to signal length. This is disadvantageous and must be corrected by decreasing learning step η.

Knowledge of network weights after completed learning process allows to calculate coefficients of linear filter realized by network. Algorithm is following:

1. Starting from even-numbered input, choose 2n successive inputs, where n denotes number of layers.
2. Choose network output dependent on all selected inputs. Depending on chosen output, calculated coefficients will define low-pass or high-pass filter.
3. Treating network as a directed graph, for each selected input *i* determine all paths connecting this input with selected output.
4. For each path determined in point 3 calculate the product of all weights by which the signal following this path is multiplied during forward propagation.
5. Sum products determined in point 4. Calculated value defines the *i*-th coefficient of the filter.

It will be demonstrated, that during teaching process weights converge to such values, that they reconstruct existing Daubechies filters.

3. Experimental validation

Proposed neural network was implemented to verify it's abilities and performance. Existing wavelet transforms – Daubechies 4, 6 and 8 - were chosen for experiments. They were realized by two–, three– and four–layer networks.

Training set consisted of 1000 randomly generated signals and their transforms calculated using known Daubechies wavelet filter coefficients. Signals were 16-element vectors with each component chosen randomly from range [-1, 1]. Testing was carried out on different data set generated in the same way. Both training and testing data were generated with eight digit precision. Learning process was terminated after reaching error less than 10^{-10}. Assuming smaller error would be pointless, due to selected precision of teaching data. Network's initial weights were chosen randomly from range [-1, 1].

Tables 1, 2 and 3 show weights obtained in the learning process. Tables 4, 5 and 6 present coefficients of filters learned by the network compared with expected values, i.e. coefficients of low–pass Daubechies filters.

Values are given with four-digit precision, however they were calculated with eight-digit precision, which allows to calculate error with the same precision. Figure 2 shows, in logarithmic scale, changes of error during learning process. Table 7 shows number of teaching patterns presented to the network before it learned the transform i.e. reached error value below 10^{-10}, as well as values of normalized mean square error obtained on the testing set. From this table we can see, that learning time increases with number of layers. In all three test cases the network managed to learn given transform with the desired precision.

Network layer	Weights			
	w_{11}	w_{12}	w_{21}	w_{22}
1	0.8072	-0.4660	0.4985	0.8636
2	0.9686	0.2776	-0.2595	1.0363

Table 1: Network's weights after learning of Daubechies 4 transform

Network layer	Weights			
	w_{11}	w_{12}	w_{21}	w_{22}
1	0.9386	-0.3870	-0.3300	-0.8005
2	-0.5112	-0.7983	0.9633	-0.4486
3	-1.0462	0.1140	-0.1107	-1.0766

Table 2: Network's weights after learning of Daubechies 6 transform

Network layer	Weights			
	w_{11}	w_{12}	w_{21}	w_{22}
1	-1.1104	-1.1104	-0.3802	-1.1800
2	0.6622	-0.8678	0.9396	0.8096
3	0.1651	-0.7327	0.8062	0.2405
4	-0.7996	-0.0465	0.0367	-1.0122

Table 3: Network's weights after learning of Daubechies 8 transform

	Filter coefficients			
	h_0	h_1	h_2	h_3
actual	0.4829	0.8365	0.2241	-0.1294
learned	0.4829	0.8365	0.2241	-0.1294
error [$\cdot 10^{-4}$]	0.0550	0.0497	-0.1258	0.0609

Table 4: Daubechies 4 low-pass filter coefficients

	Filter coefficients					
	h_0	h_1	h_2	h_3	h_4	h_5
actual	0.3327	0.8069	0.4599	-0.1350	-0.0854	0.0352
learned	0.3326	0.8069	0.4598	-0.1350	-0.0854	0.0352
error [$\cdot 10^{-4}$]	-0.1215	-0.1248	-0.0483	0.0423	0.0994	-0.0602

Table 5: Daubechies 6 low-pass filter coefficients

	Filter coefficients							
	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
actual	0.2304	0.7148	0.6309	-0.0280	-0.1870	0.0308	0.0329	-0.0106
learned	0.2303	0.7148	0.6308	-0.0279	-0.1870	0.0308	0.0328	-0.0106
error [$\cdot 10^{-4}$]	0.1091	-0.0183	0.1347	0.0078	0.0864	-0.0038	-0.0776	0.0335

Table 6: Daubechies 8 low-pass filter coefficients

Transform	No. of presented teaching patterns	Mean square error	
		minimal	average
Daubechies 4	27	$1.2488 \cdot 10^{-11}$	$6.9721 \cdot 10^{-11}$
Daubechies 6	36	$2.0228 \cdot 10^{-11}$	$1.4539 \cdot 10^{-10}$
Daubechies 8	80	$3.6894 \cdot 10^{-11}$	$1.7562 \cdot 10^{-10}$

Table 7: Learning time and result obtained on testing set

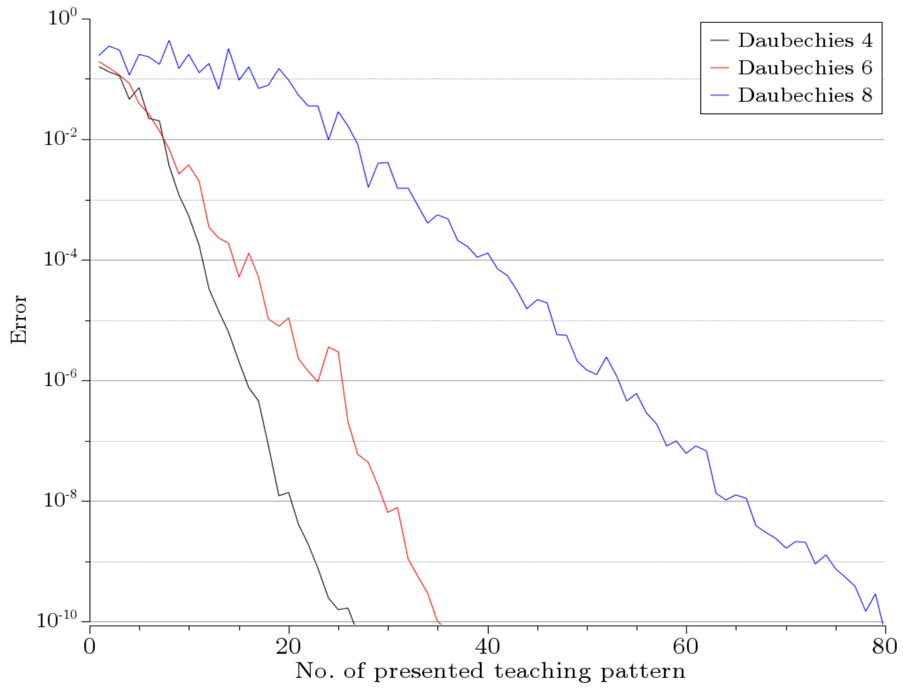

Figure 2: Error value during learning of Daubechies 4, 6 and 8 transforms

4. Conclusion

Experimental validation proved the ability of proposed neural network to learn given wavelet transforms belonging to Daubechies family. It was shown that with given signals and their transforms, network is able to recreate orthogonal filter bank coefficients used to perform wavelet transform. Obtained learning speeds show convergence of learning process in small number of iterations.

Obtained results are promising and allow to suspect, that proposed structure possesses potential ability to synthesize new orthogonal wavelet transforms, that would allow more effective processing of signals. More research should be carried out to precisely determine the network's abilities in this scope.

References

[1] Jan T. Białasiewicz. Falki i aproksymacje. WNT, 2000.

[2] Michał Jacymirski, Piotr Szczepaniak. *Neural realization of fast linear filters*. 4th EURASIP-IEEE Region 8 International Symposium on Video/Image Processing and Multimedia Communications, pages 153–157, 2002.

[3] Stanisław Osowski. *Signal flow graphs and neural networks*. Biological Cybernetics, 70(4): 387–395, February 1994.

[4] Stanisław Osowski. *Sieci neuronowe do przetwarzania informacji*. Oficyna Wydawnicza Politechniki Warszawskiej, wydanie drugie, 2006.

[5] Stanisław Osowski, Andrzej Cichocki. *Application of SFG in learning algorithms of neural networks*. International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, pages 75–83, August 1996.

[6] Bartłomiej Stasiak, Mykhaylo Yatsymirskyy. *Fast Orthogonal Neural Networks*. Lecture Notes in Computer Science, 4029:142–149, July 2006.

[7] James S. Walker. *A primer on wavelets and their scientific applications*. Chapman & Hall/CRC, 2nd edition, 2008.

[8] Mykhaylo Yatsymirskyy. *Synteza przekształceń falkowych dla mobilnych systemów e-gospodarki*. Monografia : Wybrane problemy elektronicznej gospodarki, pages 273–281. 2008.

[9] Mykhaylo Yatsymirskyy. *Struktury jednolite do syntezy przekształceń falkowych*, in printing.