Fast Orthogonal Neural Networks

Bartłomiej Stasiak and Mykhaylo Yatsymirskyy

Institute of Computer Science, Technical University of Łódź
ul. Wólczańska 215, 93-005 Łódź, Poland
basta@ics.p.lodz.pl, jacym@ics.p.lodz.pl

Abstract. The paper presents a novel approach to the construction and learning of linear neural networks based on fast orthogonal transforms. The orthogonality of basic operations associated with the algorithm of a given transform is used in order to substantially reduce the number of adapted weights of the network. Two new types of neurons corresponding to orthogonal basic operations are introduced and formulas for architecture-independent error backpropagation and weights adaptation are presented.

1 Introduction

Linear neural networks represent linear transforms of input signals. One layer of linear neurons is capable of learning an arbitrary linear transform [1, 2], which involves determining of $O\left(N^2\right)$ weights, where N is the dimension of the transformed space.

For special types of linear transforms, including discrete Fourier transform (DFT), discrete cosine transform (DCT), discrete sine transform (DST) and discrete Hartley transform (DHT), a symmetry-based factorization of their matrices leads to reduction in computational complexity [3–5]. Following the factorization scheme in neural network architecture it is possible to obtain a fast multilayer linear network with sparsely connected layers, containing $O\left(N \log\left(N\right)\right)$ weights [6, 7]. One of the substantial advantages of such an approach is the possibility of efficient hardware implementations, based on the existing DSP architectures [8].

In this paper we consider a new approach to constructing and teaching neural networks of this type, based on the orthogonality of basic operations in the underlying algorithms. A notion of a basic operation orthogonal neuron (BOON) is introduced and two types of BOONs are presented.

The main contribution of the paper is a method of BOON-based network teaching in an architecture-independent way, applicable to a wide class of known orthogonal transforms. It is also shown that the application of BOONs leads to a two-fold or a four-fold reduction in the number of weights and to an increase in the stability of the learning process. The fast cosine transform, type II and its variant with tangent multipliers [9] have been chosen to demonstrate the network construction, but other known fast orthogonal transforms (e.g. [10]) may be also easily realized.

2 Fast Two-stage Orthogonal Transforms Algorithms

In homogeneous two-stage algorithms of fast orthogonal transforms, best suited for hardware implementation, two main types of basic operations are typically used: trivial operations (addition/subtraction) and non-trivial ones involving multiplications by twiddle factors. In the next part of the paper two variants of the fast cosine transform, type II will be presented as examples of a homogeneous two-stage algorithm construction.

2.1 Fast Cosine Transform, Type II

Let $x(n)$ be an N-point real sequence, where $n = 0, 1, ..., N-1; N = 2^m; m \in \mathbb{N}$. The discrete cosine transform, type II of $x(n)$ is defined as [11]:

$$L_N^{II}(k) = \mathrm{DCT}_N^{II}\{x(n)\} = \sum_{n=0}^{N-1} x(n)\, C_{4N}^{(2n+1)k} \quad , \tag{1}$$

where $n, k = 0, 1, ..., N-1; C_K^r = \cos(2\pi r/K)$.

The basic computational procedure of the fast cosine transform, type II (FCT2) may be given as [9]:

$$\begin{aligned} L_N^{II}(0) = L_1(0), L_N^{II}(N/2) &= \sqrt{2}/2 \cdot L_2(0) \quad , \\ L_N^{II}(k) &= C_{4N}^k L_1(k) + S_{4N}^k L_2(N/2-k) \quad , \\ L_N^{II}(N-k) &= -S_{4N}^k L_1(k) + C_{4N}^k L_2(N/2-k) \quad , \\ k &= 1, 2, ..., N/2-1 \quad , \end{aligned} \tag{2}$$

where $L_1(k) = \mathrm{DCT}_{N/2}^{II}\{a(n)\}, L_2(k) = \mathrm{DCT}_{N/2}^{II}\{b(n)\}$ and the sequences $a(n)$ and $b(n)$ are formed from the input sequence $x(n)$ as follows:

$$\begin{aligned} a(n) &= x(2n) + x(2n+1) \quad , \\ b(n) &= (-1)^n (x(2n) - x(2n+1)) \quad , \\ n &= 0, 1, ..., N/2-1 \quad . \end{aligned} \tag{3}$$

Formulas (2) and (3) may be applied recursively, leading to a homogeneous FCT2 algorithm, concisely described in the form of a directed graph (Fig. 1, 2).

2.2 FCT2 Algorithm with Tangent Multipliers (mFCT2)

Multiplying and dividing the formulas (2) by C_{4N}^k enables an additional optimization [9] by means of cumulating the coefficients C_{4N}^k as a single multiplication performed as the last step of the transform (Fig. 3, 4).

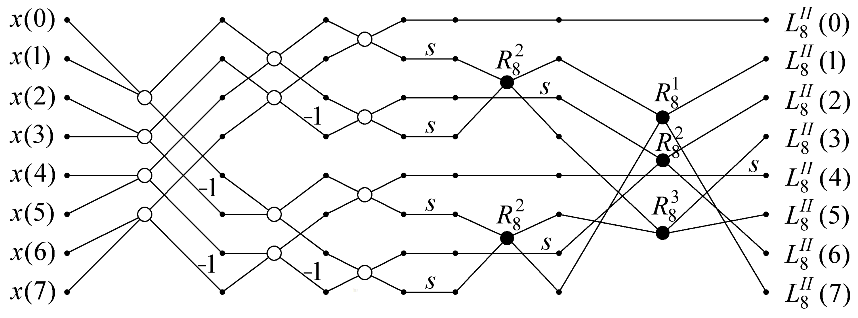

Fig. 1. Directed graph of the FCT2 algorithm for $N = 8$, $s = \sqrt{2}/2$

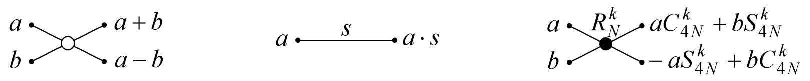

Fig. 2. Basic operations of the FCT2 algorithm

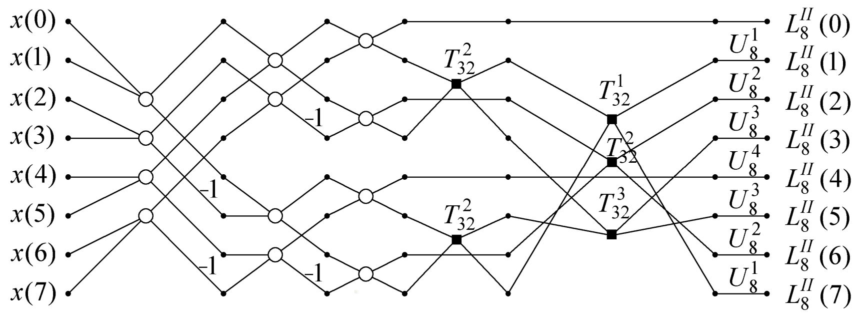

Fig. 3. Directed graph of the mFCT2 algorithm for $N = 8$

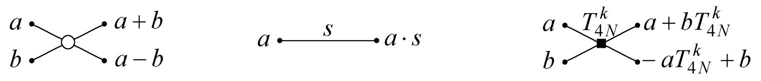

Fig. 4. Basic operations of the mFCT2 algorithm

3 Orthogonal Neural Networks

3.1 Orthogonal Basic Operations

Based on the diagram in Fig. 1 a neural network architecture where each non-trivial basic operation is replaced by two neurons [6, 7] may be built. Both neurons corresponding to a single basic operation have two inputs, i.e. the number of weights to adapt equals 4. A basic operation may be therefore seen as a 2-by-2 matrix multiplication

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_4 \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ , where } P_4 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad . \tag{4}$$

However, considering the matrix representation of the third operation in Fig. 2 we notice that only two weights are actually needed

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_2 \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ , where } P_2 = \begin{bmatrix} u & w \\ -w & u \end{bmatrix} \quad . \tag{5}$$

The significant difference lies in the orthogonality of matrix P_2. In fact, P_2 satisfies even more restrictive condition: not only are its rows/columns orthogonal, but they are also of equal norm (the orthogonality condition itself would imply adapting three independent weights).

Taking into account the explicit target values of P_2 elements, $u = C_{4N}^{k}$; $w = S_{4N}^{k}$, it is also possible to express the weights of a basic operation as functions of one parameter α, e.g. $u = \cos(\alpha)\,; w = \sin(\alpha)$. This is equivalent to defining the rows/columns of P_2 as orthonormal vectors. Such an approach would, however, result in the necessity of trigonometric functions computations in the course of the learning process, which is undesirable.

The solution to the last inconvenience may be obtained by implementing the neural network on the basis of the mFCT2 algorithm. This implies considering

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P_1 \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ , where } P_1 = \begin{bmatrix} 1 & t \\ -t & 1 \end{bmatrix} \quad , \tag{6}$$

according to the third operation presented in Fig. 4.

3.2 Teaching Methods

The main practical issue resulting from the application of matrices P_2 or P_1 affects the methods of neuronal weights adaptation. The classical definition of a neuron should be modified here to reflect the relationship between the outputs of the basic operation. We would either talk about two associated neurons, orthogonal to each other, or simply about a basic operation orthogonal neuron (BOON) with two outputs. It is also worth noting that the matrix P_4 does not require any special treatment, as its rows may be seen as representations of classical independent neurons with two inputs. As gradient backpropagation methods seem the best choice for teaching the considered types of neural networks [7], a proper algorithm suited for the special forms of matrices P_2 and P_1 is necessary.

Considering a simple case of two connected layers shown in Fig. 5 and assuming that the basic operation matrix has a form defined by (5) we can explicitly express the outputs of the network as functions of its inputs

$$\begin{array}{l} y_1 = u_1^{(2)} v_1 + w_1^{(2)} v_2 \\ y_2 = -w_1^{(2)} v_1 + u_1^{(2)} v_2 \\ y_3 = u_2^{(2)} v_3 + w_2^{(2)} v_4 \\ y_4 = -w_2^{(2)} v_3 + u_2^{(2)} v_4 \end{array} \text{, where } \begin{array}{l} v_1 = u_1^{(1)} x_1 + w_1^{(1)} x_3 \\ v_2 = u_2^{(1)} x_2 + w_2^{(1)} x_4 \\ v_3 = -w_1^{(1)} x_1 + u_1^{(1)} x_3 \\ v_4 = -w_2^{(1)} x_2 + u_2^{(1)} x_4 \end{array} \tag{7}$$

and where the expressions $u_k^{(l)}$ and $w_k^{(l)}$ refer to the k-th operation of the l-th layer. Our goal is to minimize the error function given as:

$$E = \frac{1}{2} \sum_{i=1}^{N} (y_i - d_i)^2 \quad , \tag{8}$$

where $N = 4$ and d_i represents an expected value of the i-th output.

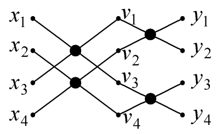

Fig. 5. Two layers of an orthogonal network, each containing 2 basic operation neurons

Substituting (7) into (8) and computing derivatives for the weights in the second and in the first layer we arrive at formulas defining the components of the gradient vector and the error vector for a single basic operation

$$\begin{bmatrix} \frac{\partial E}{\partial u} \\ \frac{\partial E}{\partial w} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \\ v_2 & -v_1 \end{bmatrix} \cdot \begin{bmatrix} e_1^{(n)} \\ e_2^{(n)} \end{bmatrix} \quad , \tag{9}$$

$$\begin{bmatrix} e_1^{(n-1)} \\ e_2^{(n-1)} \end{bmatrix} = P_2^{\mathrm{T}} \cdot \begin{bmatrix} e_1^{(n)} \\ e_2^{(n)} \end{bmatrix} \quad , \tag{10}$$

where P_2^{T} denotes the transpose of P_2.

The parameters v_1 and v_2 represent the inputs of the basic operation, the vector $\left[e_1^{(n)},\ e_2^{(n)} \right]^{\mathrm{T}}$ refers to the error values propagated back from the next layer and the vector $\left[e_1^{(n-1)},\ e_2^{(n-1)} \right]^{\mathrm{T}}$ defines the error values to be propagated back from the current layer to the previous one. As the matrix P_1 (6) is a special case of the matrix P_2 (5) for $u = 1$, the corresponding formulas, suitable for teaching an mFCT2-based network, can be derived from (9), (10) as follows:

$$\frac{\partial E}{\partial t} = \begin{bmatrix} v_2, & -v_1 \end{bmatrix} \cdot \begin{bmatrix} e_1^{(n)} \\ e_2^{(n)} \end{bmatrix} \quad , \tag{11}$$

$$\begin{bmatrix} e_1^{(n-1)} \\ e_2^{(n-1)} \end{bmatrix} = P_1^{\mathrm{T}} \cdot \begin{bmatrix} e_1^{(n)} \\ e_2^{(n)} \end{bmatrix} \quad . \tag{12}$$

The formulas (9) - (12) have a general meaning, i.e. they are applicable to a basic operation irrespective of its location in the network architecture. Moreover, no specific architecture is imposed as the information about the indexes of the interconnected basic operations' inputs/outputs is sufficient. Given the components of the gradient vector, any known gradient method may be successfully applied to minimize the error function of the network.

The numbers of multiplications (μ) and additions (α) for the matrices are:

$$\begin{array}{lll} \mu\left(P_4\right) = 8 & \mu\left(P_2\right) = 8 & \mu\left(P_1\right) = 4 \\ \alpha\left(P_4\right) = 2 & \alpha\left(P_2\right) = 4 & \alpha\left(P_1\right) = 3 \end{array} \quad . \tag{13}$$

These values concern gradient computation and error backpropagation only. As one of the most crucial parameters influencing the efficiency of gradient minimization algorithms is the number of the adapted weights, its two-fold (P_2) and four-fold (P_1) reduction will actually play the most important role in a global computational complexity improvement.

It should also be noted that learning of the inverse transform may be easily realized by changing all the matrices to their (properly scaled) transpositions.

3.3 Experimental Validation

The presented methods of teaching the BOONs were implemented within a framework developed for testing neural networks with arbitrary connections.

Three groups of tests were performed to compare the capabilities of a non-orthogonal FCT2-based network (type P_4 BOONs) and of two orthogonal networks: FCT2 and mFCT2-based (type P_2 BOONs and type P_1 BOONs). Several datasets, varied by the length and the number of input vectors, were used for all groups. The teaching was repeated ten times for each dataset, from a random starting point. The averaged results are presented in Tables 1, 2, 3, respectively.

Table 1. Results of FCT2-based non-orthogonal network training

N	P	Mean epochs	Epochs std	Mean time [s]	Time std	Weights
8	4	49	6.063	0.1329	0.0619442	20
16	8	119	23.7445	1.2891	0.245518	68
32	16	125	37.5847	8.2673	2.41184	196
64	32	172	81.5784	78.6579	37.2662	516

Table 2. Results of FCT2-based orthogonal network training

N	P	Mean epochs	Epochs std	Mean time [s]	Time std	Weights
8	4	25	1.96214	0.0782	0.0474443	10
16	8	46	3.74299	0.525	0.0564659	34
32	16	58	1.84662	3.9814	0.126828	98
64	32	71	2.5865	33.3281	1.16706	258

Table 3. Results of mFCT2-based orthogonal network training

N	P	Mean epochs	Epochs std	Mean time [s]	Time std	Weights
8	4	14	1.22066	0.0625	0.0394899	5
16	8	25	0.916515	0.297	0.0270222	17
32	16	24	0.538516	1.7124	0.04304	49
64	32	22	0	10.9764	0.1246	129

The first two columns contain the size (N) and the number (P) of random input vectors and std is the standard deviation. Target vectors for all the datasets were computed according to the formula (1).

The conjugate gradient method was applied as an algorithm of error function minimization [1] and the teaching was stopped when the error was lower than 1e-9. The tests were performed on a computer with Intel Celeron M, 1.40 GHz processor.

The teaching of the orthogonal networks proved to be a stable process in terms of the standard deviation of its length. The most interesting observation is the almost constant number of epochs in the case of mFCT2-based network. A closer examination revealed that the final state of the network was always similar for a given dataset and equal to the state obtainable by computing the weights values directly.

The relatively high mean time values for higher N result from the generality and flexibility of the framework which involves operations on large connection matrices. The comparison between the tables, however, shows a definite superiority of the orthogonal networks, which is particularly clear in the case of type P_1 BOONs.

4 Conclusion

A new method of constructing and teaching neural networks based on fast orthogonal transforms was presented. Respecting the orthogonality of the basic operations allowed to reduce the number of the adapted weights in comparison to the non-orthogonal network, increasing the efficiency and stability of the learning process. Owing to the generality of the presented solutions, the proposed BOONs may be used in the construction of neural networks realizing a wide class of known orthogonal transforms.

References

1. Osowski, S.: Neural networks for information processing. (in Polish) OWPW, Warsaw (2000)
2. Rutkowski, L.: Methods and techniques of artificial intelligence. (in Polish) Polish Scientific Publishers PWN (2005)
3. Wang, Z.: Fast algorithms for the discrete W transform and for the discrete Fourier transform. IEEE Trans. on Acoustics, Speech, and Signal Processing **32** (1984) 803-816
4. Yatsymirskii, M.N.: Fast algorithms for the discrete cosine transformation. Comput. Maths Math. Phys **33** (1993) 267-270
5. Egner, S., Püschel, M.: Automatic generation of fast discrete signal transforms. IEEE Trans. on Signal Processing **49** (2001) 1992-2002
6. Jacymirski, M., Szczepaniak, P.S.: Neural realization of fast linear filters. In: Proc. of the 4th EURASIP - IEEE Region 8 International Symposium on Video/Image Processing and Multimedia Communications. (2002) 153-157
7. Szczepaniak, P.S.: Intelligent computations, fast transforms and classifiers. (in Polish) EXIT Academic Publishing House, Warsaw (2004)
8. Rabiner, L.R., Gold, B.: Theory and application of digital signal processing. Prentice-Hall (1975)
9. Jacymirski, M.: Fast homogeneous algorithms of cosine transforms, type II and III with tangent multipliers. (in Polish) Automatics **7** AGH University of Science and Technology Press, Cracow (2003) 727-741
10. Yatsymirskyy, M.M.: Shifted in the time and frequency domains cosine and sine transforms fast algorithms with homogeneous structure. (in Russian) Izvestiya Vysshikh Uchebnykh Zavedenii, Radioelektronika **43**, Kiev (2000) 66-75
11. Ahmed, N., Rao, K.R.: Orthogonal transforms for digital signal processing. Springer-Verlag, New York (1975)