2023 20th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE) | 979-8-3503-0676-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/CCE60043.2023.10332880

FPGA Simulation for Computing Pseudoinverse Matrices

Gerardo Marcos Tornez Xavier
Energy and Electronics
Academic, UPIEM-IPN,
Mexico City, Mexico
gtornezx@ipn.mx

Luis Martín Flores Nava
Electrical Engineering
Department, CINVESTAV-IPN,
Mexico City, Mexico
lmflores@cinvestav.mx

Felipe Gómez Castañeda
Electrical Engineering
Department, CINVESTAV-IPN,
Mexico City, Mexico
fgomez@cinvestav.mx

José Antonio Moreno Cadenas
Electrical Engineering
Department, CINVESTAV-IPN,
Mexico City, Mexico
jmoreno@cinvestav.mx

***Abstract*— Many scientific and engineering processing models of big data deal with classification tasks. In particular, current intelligent approaches work efficiently for them, as they use neural techniques with shallow architectures. The Extreme Learning Machine (ELM), which is a one-hidden layer feedforward neural network, is capable of achieving high accuracy in these tasks. Moreover, one of its weighting matrices is the pseudoinverse of the hidden data. In this sense, this work presents meaningful results of computing the Moore-Penrose pseudoinverse for the ELM, using a recurrent model. The numerical simulations were based on an FPGA framework used to design a pseudoinverse-computing core.**

Keywords— Pseudoinverse, Extreme Learning Machine, FPGA Simulation.

I. INTRODUCTION

At present, efficient solutions for dealing with big data in classification tasks are given via the machine learning scheme. The Extreme Learning Machine, which is a one-hidden layer neural network, belongs to this scheme [1]. In the ELM model, the entries of the weighting matrix between the input layer and the hidden layer are random; meanwhile, the Moore-Penrose (M-P) pseudoinverse defines the entries of the matrix between the hidden and the output layers. Moreover, the solution of the pseudoinverse matrix is a research topic ongoing, whose formulation can be made either with linear algebra analysis [2] or with neural network models [3, 4]. Also, the hardware implementation topic draws the acceleration issue of M-P matrices computation as necessary for real-time systems [5-7]. In this context, we use a low-complexity recurrent neural model as a numerical model for computing the M-P matrix in hardware. This goal is achieved with the FPGA Xilinx simulation software.

This manuscript is divided as follows. Section II recalls the fundamental properties of the M-P pseudoinverse of full-rank matrices. Section III reviews the concepts for establishing first-order recurrent neural models that are suitable for computing the M-P pseudoinverse. In section IV, the Simulink platform simulates a recurrent neural model, showing acceptable accuracy results with floating point arithmetic. Section V introduces the results in Matlab along with the FPGA software system. The section VI, presents timing diagrams based on a standard FPGA development board for a specific case of the pseudoinverse computation and the section VII presents a Discussion. The article ends with the Conclusion section.

II. MOORE-PENROSE PSEUDOINVERSE

This section introduces the Conditions and Formulations for the M-P pseudoinverse using the following definitions: the M-P pseudoinverse of A, which is full-rank is denoted as A^{+}, with $A \in \mathbb{R}^{mxn}$, and $A^{+} \in \mathbb{R}^{nxm}$; A^{T} is the transpose of A and $(\bullet)^{-1}$ denotes the inverse.

Conditions:

The M-P pseudoinverse satisfies (1) through (4).

$$AA^{+}A = A \tag{1}$$

$$A^{+}AA^{+} = A^{+} \tag{2}$$

$$(AA^{+})^{T} = AA^{+} \tag{3}$$

$$(A^{+}A)^{T} = A^{+}A \tag{4}$$

Formulations:

The M-P pseudoinverse is computed with either (5) or (7).

$$A^{+} = (A^{T}A)^{-1}A^{T} \quad if\ m > n \tag{5}$$

$$A^{+} = (A)^{-1} \quad if\ m = n \tag{6}$$

$$A^{+} = A^{T}(AA^{T})^{-1} \quad if\ m < n \tag{7}$$

If only one equation in the set of Conditions is satisfied, the pseudoinverse is called the generalized inverse.

III. NEURAL MODELS FOR MOORE-PENROSE PSEUDOINVERSE

In early works, neural network models were proposed to compute the M-P pseudoinverse of matrices [8-9], where their numerical complexity can become rich. From this background, we observe the work by J. Wang [10], who presented an analog recurrent network with minimal complexity. Wang's network is dynamic, whose matrix-valued variable *X(t)* approaches the Moore-Penrose pseudoinverse of the matrix *A* when *dX/dt* is zero. This system is based on a scalar-valued function *E=E(t)*,which reduces its magnitude as the course of time advances into infinity and, is defined as:

979-8-3503-0676-7/23/$31.00 ©2023 IEEE

$$E = \|AX(t) - I\|_F^2 \qquad (8)$$

$\|\bullet\|$ denotes the Frobenius norm and I is the identity matrix. Moreover, using the negative of the gradient of E along with the equation: $dX/dt = -(\eta/2)\, dt/dX$, leads to the set of ordinary differential equations, which is given in (9), where η is a constant.

$$\frac{dX(t)}{dt} = -\eta(A^T AX(t) - A^T) \qquad (9)$$

This equation solves matrix X for matrix A, whose dimension satisfies $m>n$. This is the case for most of the practical matrices in the ELM model under consideration. Moreover, replacing A^T by $(A^T)_j$ and X by x_j, their respective columns, (9) solves for x_j.

IV. Simulink Analysis

This section deals with simulations using Simulink, whose precision was set for floating-point.

The computing implementation of (9) is given in Figure 1, where the vector $(A^T)_j$ is the column of the matrix A^T with index j, with $j=1, \ldots, n$. This system contains a matrix-vector multiplier, one adder, and an integrator. The adder subtracts $A^T A x_j$ from the vector $(A^T)_j$ and produces the time derivative of x_j namely, dx_j/dt. The Simulink representation of Figure 1 for a 9xn matrix is shown in Figure 2, where the integrator is built with a one-sample delay or z^{-1}, and an input scaler with a $1\text{x}10^{-3}$ factor. This value establishes the differential in time. Figure 3 shows 9 blocks of Figure 2 to compute the vector $(A^T)_j$. Figure 4 shows the Simulink simulation of x_j with $j=1$, i.e. corresponding to the first column of matrix A, whose entries are included below.

$A =$

0.6294	-0.6848	0.3115	0.4121	-0.1225	-0.4479	0.5025	0.6814	-0.2967
0.8116	0.9412	-0.9286	-0.9363	-0.2369	0.3594	-0.4898	-0.4914	0.6617
-0.7460	0.9143	0.6983	-0.4462	0.5310	0.3102	0.0119	0.6286	0.1705
0.8268	-0.0292	0.8680	-0.9077	0.5904	-0.6748	0.3982	-0.5130	0.0994
0.2647	0.6006	0.3575	-0.8057	-0.6263	-0.7620	0.7818	0.8585	0.8344
-0.8049	-0.7162	0.5155	0.6469	-0.0205	-0.0033	0.9186	-0.3000	-0.4283
-0.4430	-0.1565	0.4863	0.3897	-0.1088	0.9195	0.0944	-0.6068	0.5144
0.0938	0.8315	-0.2155	-0.3658	0.2926	-0.3192	-0.7228	-0.4978	0.5075
0.9150	0.5844	0.3110	0.9004	0.4187	0.1705	-0.7014	0.2321	-0.2391
0.9298	0.9190	-0.6576	-0.9311	0.5094	-0.5524	-0.4850	-0.0534	0.1356

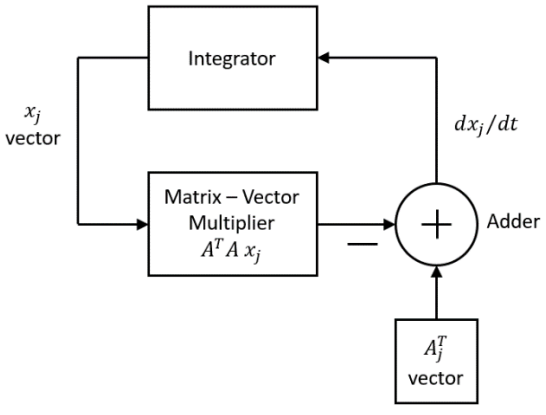

Fig. 1. Computing representation of equation (9), using vector x_j.

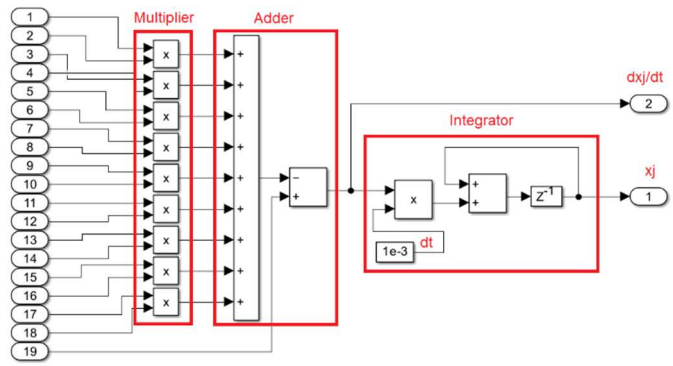

Fig. 2. Simulink representation of Figure 1 for a 9xn matrix.

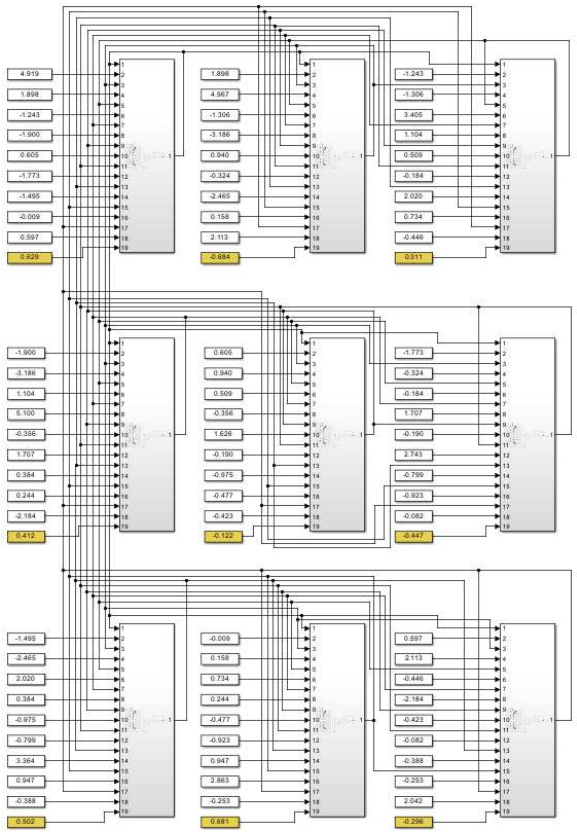

Fig. 3. Simulink using 9 blocks of Fig. 2 to create vector $(A^T)_j$.

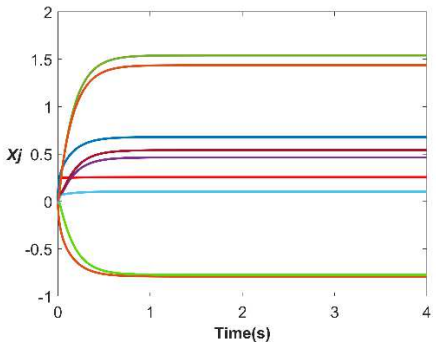

Fig. 4. Simulink simulation x_j with j=1, the first column the A^T.

Introducing the following columns of A, the next columns of the inverse matrix A^+ can be computed. The numerical result for the whole A^+ is below.

$$A^+ = \begin{bmatrix} 0.2562 & 0.2968 & -0.1500 & 0.1980 & 0.0147 & -0.0488 & 0.1173 & -0.2974 & 0.2937 & 0.0971 \\ -0.7904 & 0.4896 & 0.1163 & -0.0046 & 0.3822 & 0.7591 & -0.6041 & -0.1227 & 0.7070 & -0.2176 \\ -0.7729 & -0.0344 & 0.0048 & 0.7420 & 0.1937 & -0.4597 & -0.5502 & -0.1545 & 0.4548 & -1.2498 \\ 0.4644 & -0.1970 & -0.1659 & -0.5161 & 0.1241 & 0.5989 & 0.5232 & 0.4525 & 0.3116 & 0.5636 \\ 1.5405 & -0.5253 & 0.5444 & -0.4952 & -0.5511 & 0.3442 & 1.4080 & 0.4560 & -0.6076 & 1.9704 \\ 0.1044 & 0.4593 & 0.3500 & -0.0088 & -0.2755 & -0.1443 & 0.3644 & -0.6849 & 0.0145 & -0.0011 \\ 0.5410 & 0.3608 & 0.1782 & -0.2957 & 0.1775 & 1.1925 & 0.5321 & -0.2824 & -0.0510 & 1.0388 \\ 0.6797 & -0.2791 & 0.3735 & -0.3649 & -0.0130 & -0.4826 & 0.2469 & -0.1380 & -0.2391 & 0.3604 \\ 1.4364 & -0.5440 & 0.0381 & -0.5990 & 0.0907 & -0.2082 & 1.7207 & 0.9087 & -0.6666 & 1.2437 \end{bmatrix}$$

Finally, it is worth noting that further simulations for matrices whose dimensions are diverse were realized using Matlab code, where the numerical precision keeps in floating-point.

V. MATLAB AND FPGA SIMULATION

The Matlab code to solve the equation (9) is listed in Algorithm 1. The pseudoinverse matrix of 10 matrices, whose dimension is *m*x*n*, was computed with this Matlab code. They have random entries in [-1, 1]. Table I contains information from both the Matlab and the FPGA simulation. The last row includes the equivalent formulations to get the values presented, where N can be approached by $5.66n^{2.88} + 6.32x10^5$. Moreover, T is the clock period, whose value is $1.1x10^{-6}$ sec. Figure 5 shows the FPGA system, which has one read-only RAM for storing A^TA and two write/read RAMs for storing temporarily x_j and $(A^T)_j$. The pulse signals were for the main clock, reset, internal reset, and stop.

Algorithm 1 to solve the equation (9)

```
1: A = rand(10,9)*2-1;
2: [m,n] = size(A);
3: At = A';
4: T = 400000;
5: xj = zeros(n,T);
6: dxj = zeros(n,T);
7: dt = 1e-3;
8: for col = 1:m
9:     for t = 1:T-1
10:        dxj(:,t) = -(A'*A*xj(:,t)) + At(:,col);
11:        xj(:,t+1) = xj(:,t) + dxj(:,t)*dt;
12:    end
13: xj(:,col) = xj(:,end);
14: end
```

This work considers using a Xilinx development board with an XC7A100T-CSG324 FPGA from the Artix-7 family, where the simulation software was Vivado. The processing blocks in Figure 1 namely, matrix-vector multiplier, integrator, and adder were simulated with this software. The strategy in [11] is applied to perform the matrix-vector multiplication, saving hardware resources.

TABLE I DATA FROM MATLAB AND FPGA SIMULATIONS

Matrix A $m \times n$	Clock cycles on FPGA	Number of iterations in Matlab to get $dX_j/dt \leq 1x10^{-12}$ (N)	Elapsed time to compute pseudoinverse on the FPGA, in sec. (T_E)	Number of DSP's
11×10	12	$0.50x10^6$	72.6	180
21×20	22	$0.70x10^6$	355.74	360
31×30	32	$0.75x10^6$	818.4	540
41×40	42	$0.90x10^6$	1,704.78	720
51×50	52	$1.20x10^6$	3,500.64	900
61×60	62	$1.50x10^6$	6,240.3	1080
71×70	72	$1.75x10^6$	9,840.6	1260
81×80	82	$2.20x10^6$	16,073.64	1440
91×90	92	$3.20x10^6$	29,469.44	1620
101×100	102	$4.0x10^6$	45,328.8	1800
	$n+2$	$\tilde{N} = 5.66n^{2.88} + 6.32x10^5$	$T_E = m(n+2)(T)(N)$	$(9n)(2)$

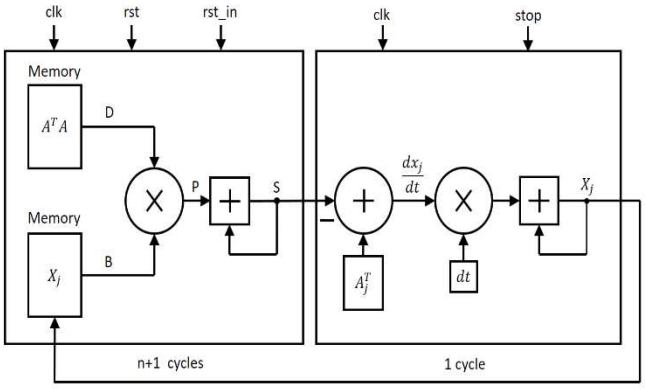

Fig. 5. Block diagram of the FPGA system.

VI. FPGA TIMING SIMULATION

Regarding the simulations of the FPGA, a sample of snapshots of timing is included in Figure 6. It belongs to the first column of matrix A^+, the pseudoinverse of the above matrix A. The values of dx_j/dt are around $1x10^{-12}$ at $600x10^6$ ns. This confirms the predicted values given in Table I. We also observed that this simulation reproduces the same values by Matlab simulation. This simulation verifies the efficacy of the proposed neural method, which is computed by the FPGA development system.

VII. DISCUSSION

The main goal of this work was focused on presenting an alternative numerical process for computing pseudoinverse matrices, which highly contrasts with current hardware accelerators [5, 7, 12]. The main difference comes from the usage of design frameworks that optimize FPGA resources using matrix transformations. Despite this fact, the numerical core in this work can be parallelized to improve its performance for moderate matrix dimensions.

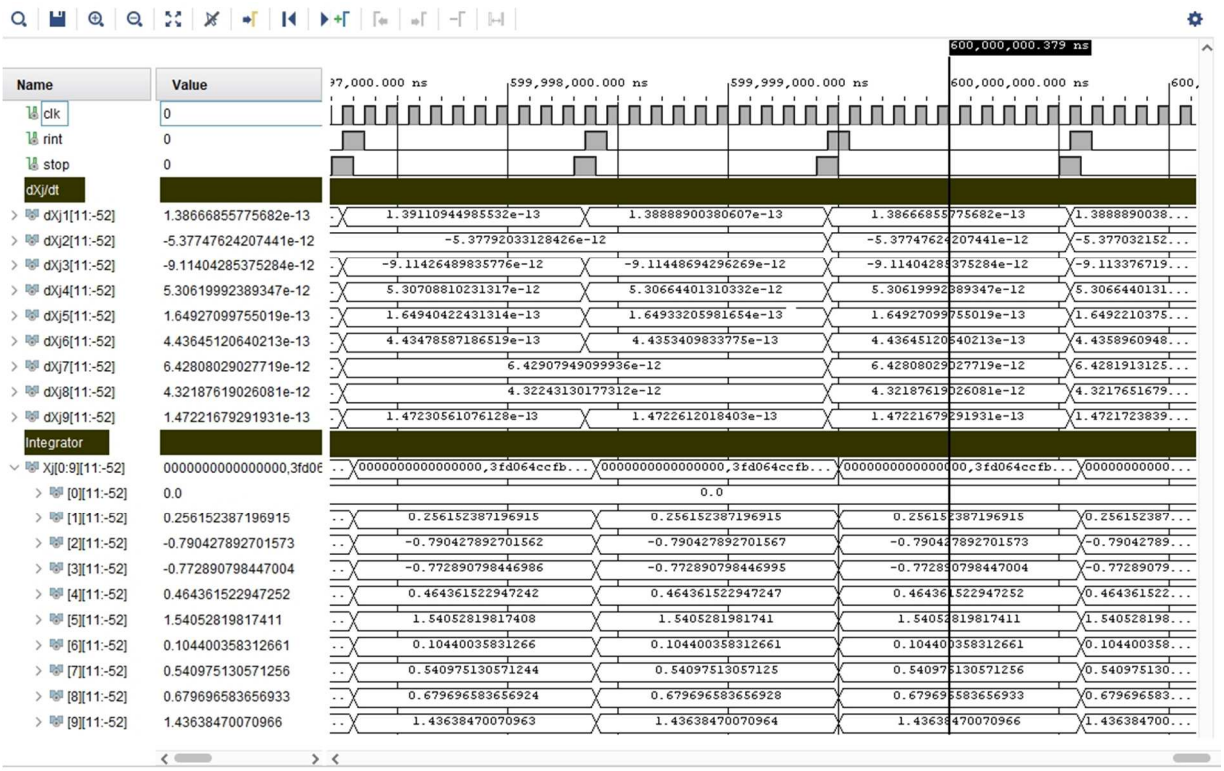

Fig. 6. Timing simulation snapshot.

In this sense, this work can not be compared fairly with other approaches even with other neural models, where their complexity is not suitable for hardware realization. The reason for them is purely academic and oriented to demonstrate stability and convergence issues due to their dynamic nature, using standard numerical systems [4, 13].

VIII. Conclusion

This work presented the main topics related to the computation of pseudoinverse matrices via an FPGA, which is based on a recurrent neural network. The numerical complexity of this neural model is low, leading to its execution in real-time with moderate latency for small matrix dimensions. However, for larger dimensions, it results in significant latency, which necessitates the use of an array of various FPGAs working in parallel to reduce this delay. In general, this approach is competitive compared to other works and is well-suited for the ELM model.

References

[1] Guang-Bin Huang,Qin-YuZhu and Chee-Kheong Siew, "Extreme learning machine: Theory and applications", Neurocomputing, pp. 489-501, 2006.

[2] J. C. A. Barata and M. S. Hussein " The Moore-Penrose Pseudoinverse. A Tutorial Review of the Theory", Brazilian Journal of Physics, pp. 146-165, 2012.

[3] Lin Li and Jianhao, " An efficient second-order neural network model for computing the Moore-Penrose inverse of matrices", IET Signal Processing, pp. 1-12, 2022.

[4] Wenqi Wu and Bing Zheng, " Improved recurrent neural networks for solving Moore-Penrose inverse of real-time full-rank matrix", Neurocomputing 418, pp. 221-231, 2020.

[5] A. Irturk, B. Benson, S. Mirzaei and R. Kastner, "An FPGA design space exploration tool for matrix inverion architectures", 2018 IEEE Symposium on Application Specific Processors. IEEE, June 2008.

[6] C. Y. Tan, C. Y. Ooi, H. S. Choo and N. Ismail, "Efficient hardware-accelerated pseudoinverse computation through algorithm restructuring for parallelization in high level synthesis", Int. J. Circuit Theory Appli., vol. 50, pp. 394-416, February 2022.

[7] J. V. France-Villora, A. Rosado-Muñoz, J. M. Martínez-Villena, M. Bataller-Mompean, J. F. Guerrero and M. Wegrzyn, "Hardware implementation of rea-time extreme learning machine in FPGA: Analysis of precision, resource occupation and performance", Comput. Electr. Eng., vol. 51, pp. 139-156, April 2016.

[8] G. Wu, J. Wang and J. Hootman, "A recurrent neural network for computing pseudoinverse matrices," Math. Comput. Modell., vol. 20, pp. 13-21, July 1994.

[9] Y. Zhang "Revisit the analog computer and gradient-based neural system for matrix inversion," Proceeding of the 2005 IEEE International Symposium on Mediterrean Conference on Control and Automation Intelligent Control. June 2005.

[10] J. Wang "A recurrent neural network for real-time matrix inversion," Appl. Math. Comput., vol. 55, pp. 89-100, April 1993.

[11] I. Sayahi, M. Machhout and R. Tourki, "FPGA implementation of matrix-vector multiplication using xilinx system generator", 2018 International Conference on Advanced Systems and Electric Technnologies (IC_ASET). March 2018.

[12] R. Wang, C. S. Thakur, G. Cohen, T.J. Hamilton, J. Tapson and A. Schaik "Neuromorphic Hardware Architecture Using the Neural Engineering Framework for Pattern Recognition", IEEE Trans. Biomed. Circuits Syst., vol 11, pp. 574-584, June 2017.

[13] Zhang N. and Zhang T. "Recurrent Neural Network for Computing the Moore-Penrose Inverse with Momentum Learning" Chinese Journal of Electronics Vol. 28 No.6, Nov. 2020.