Functionality of Extreme Learning Machines as Logical Operators

Benjamin Walleshauser

Correspondence: Benjamin Walleshauser (wallesbt@clarkson.edu)

Abstract. Extreme learning machines are a type of single layer feed forward network that utilize randomly chosen input and bias weights and have output weights which are determined via a simple linear method. In this study, it is shown that extreme learning machines can be accurately used to return logical operations, such as AND, OR, XOR, and NOT. This paper showcases this ability by creating models comprised of extreme learning machines that run Tic-Tac-Toe and Conway's Game of Life.

1 Introduction

Since their discovery in 2006 (Huang et al., 2006b), extreme learning machines (ELMs) have become a popular form of feed forward neural network due to their simplicity and associated quick training times. Extreme learning machines differ from traditional single layer feed forward networks (SLFNs) as the input and bias weights of the ELM are randomly selected, with only the output weights being trained via a simple inverse operation. This drastically reduces training time, while still producing results that are competitive with machine learning methods that are trained more extensively (Bonakdari and Ebtehaj, 2016; Peng et al., 2013; Bhat et al., 2008). Researchers have applied extreme learning machines to tasks from time series forecasting (Xing et al., 2019; Zhang et al., 2017) to object recognition (de Chazal et al., 2015; Rong et al., 2014; Sreekanth et al., 2014) with success. The ability for the extreme learning machine to be able to approximate various functions has been explained by universal approximation theory (Huang et al., 2006a), which can be understood heuristically as the ELM piecing together random functions in a manner that fits the observed data set.

Single layer ELMs are essentially a weighted sum of the outputs from a layer of neurons with randomly selected internal synaptic weights. Therefore if the extreme learning machine is able to model fundamental logical operators, this is indicative of how the brain can perform complex computations with very minimal adjustment to the synaptic weights, as the logic gates can be arranged in an organized manner that allows for more advanced computation. It has been shown in the past that a single perceptron (which is a rough model for a biological neuron) is capable of modelling several logical operators (Adesola, 2015). Therefore our goal is to rather show how a network of neurons with randomly chosen internal weights (i.e., an ELM) can still give rise to computation by being able to mimic logical operators.

2 Extreme Learning Machines

The single layer extreme learning machine is defined as follows (Huang et al., 2006b). The input data on the j^{th} time step $\mathbf{x}_j = [x_{j,1}\ x_{j,2}\ \dots\ x_{j,dx}]^\intercal$ is transformed by the input weights $\mathbf{w}_i = [w_{i,1}\ w_{i,2}\ \dots\ w_{i,dx}]$, where it is then summed with a

bias term b_i. The values of the input and bias weights are sampled from the same normal distribution $N(0, \sigma)$. The value of $\mathbf{w}_i * \mathbf{x}_j + b_i$ is then transformed by activation function $g(s)$, which is commonly chosen to be the hyperbolic tangent function, the sigmoid function, or the radial basis function (Wang et al., 2021). After being transformed by the activation function this is effectively the output of a single perceptron, which is then weighted appropriately depending on the output node it is going to $\beta_i = [\beta_{i,1}\ \beta_{i,2}\ \dots\ \beta_{i,dy}]$. The resulting output $\mathbf{y_j} = [y_{j,1}\ y_{j,2}\ \dots\ y_{j,dy}]$ is then the summation of the aforementioned process $i = 1$ to $\tilde{N}$ times, hence corresponding to $\tilde{N}$ neurons.

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i * \mathbf{x}_j + b_i) = \mathbf{y}_j \tag{1}$$

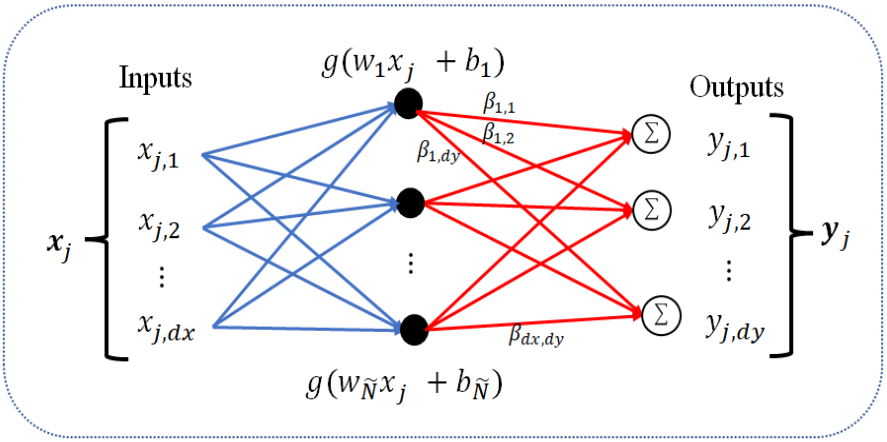

Figure 1. Architecture of the extreme learning machine.

For training of the extreme learning machine, a matrix $\mathbf{H} \in \mathbb{R}^{N \times \tilde{N}}$ is compiled for each of the N samples of the input data $\mathbf{x} \in \mathbb{R}^{dx \times N}$ which will then be used to find the output weights.

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1 * \mathbf{x}_1 + b_1) & \dots & g(\mathbf{w}_{\tilde{N}} * \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \ddots & \vdots \\ g(\mathbf{w}_1 * \mathbf{x}_N + b_1) & \dots & g(\mathbf{w}_{\tilde{N}} * \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix} \tag{2}$$

Therefore with the use of the matrix $\mathbf{H}$, Eq. (1) can be expressed compactly by Eq. (3) for the training phase.

$$\mathbf{H}\beta = \mathbf{y} \tag{3}$$

As it is desired that the extreme learning machines output target values $\mathbf{y} = \mathbf{t}$ (where $\mathbf{y}, \mathbf{t} \in \mathbb{R}^{dx \times N}$), the values of the output weights β are chosen such that it minimizes the following (where λ is the regularization parameter and $||\cdot||$ denotes the Euclidean norm),

$$\beta = \arg\min_{\hat{\beta}}(||\mathbf{H}\hat{\beta} - \mathbf{t}||^2 + \lambda||\hat{\beta}||^2) \tag{4}$$

This in practice is known as a ridge regression, which includes the term $\lambda||\hat{\beta}||^2$ to prevent over fitting (Afkham et al.; Yu et al., 2013). The solution to Eq. 4 is then given by Eq. 5.

$$\beta = (\mathbf{H}^\intercal\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^\intercal\mathbf{t} \tag{5}$$

Therefore with the trained output weights, the value of $\mathbf{y}_j$ simply becomes a function of the value of the input $f(\mathbf{x}_j)$.

2.1 Training

The extreme learning machines are trained to replicate the following common logical operators, AND, OR, NOT, and XOR. As the AND, OR, and XOR operators have only two inputs and the NOT operator has only one input, their respective values of dx are 2 and 1 respectively - with all four sharing a value of $dy = 1$ (as they all have only one output). To train an extreme learning machine to a given logical operator $\mathcal{L} : \mathbb{R}^{dx} \rightarrow \mathbb{R}^{dy}$, input data $\mathbf{x} \in \mathbb{R}^{dx \times N}$ is generated from a random sequence of $0's$ and $1's$ perturbed with noise, denoted by $\mathbf{x} = \mathbf{r} + \mathbf{z}$, where $\mathbf{r} \in \mathbb{R}^{dx \times N}$ is a matrix containing randomly selected values of 0 and 1 and $\mathbf{z} \in \mathbb{R}^{dx \times N}$ is the added noise. Therefore the target data is then $t_j = \mathcal{L}(r_{j,1}\ ,\ r_{j,2})$ when $dx = 2$ and $t_j = \mathcal{L}(r_j)$ when $dx = 1$. The noise term $\mathbf{z}$ which has values sampled from a uniform distribution $U(-\epsilon, \epsilon)$, is introduced to prevent model divergence when the logic gates are connected in series, as the ELM operator will learn to treat account for all of the points within the epsilon neighborhood surrounding the operational points.

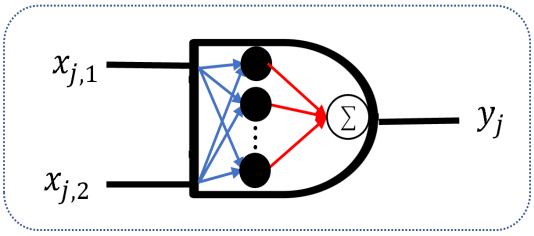

Figure 2. The AND gate modelled by an extreme learning machine.

The following metaparameters are chosen for the model $g(s) = tanh(s)$, $N = 4{,}000$, $\tilde{N} = 200$, $\lambda = 1e-4$, $\sigma = 30$, and $\epsilon = 5e-5$. Though the ELM utilizes random weights with success, the magnitude of these weights can significantly effect model results (Dong and Li, 2021; Dudek, 2016). Therefore value of σ was optimized, as it was observed that results were

best for a value of $\sigma \gtrsim 15$. The value of the regularization parameter λ was also tuned, and it was found that results were the best when $\lambda \gtrsim 10^{-7}$, which is indicative that the ridge regression was ultimately necessary as the regression would simplify to ordinary least squares when $\lambda = 0$. The value of the perturbance was chosen to be $\epsilon = 5e-5$, as this was only about $2-3$ orders of magnitude greater than the exact values returned by the ELM, and hence is quite a large guard band against signal divergence.

2.2 Results

To present results for each of the four logical operators trained, a truth table is provided with the output being the corresponding trained ELM output, along with a figure for each ELM operator depicting the surface created, where it is seen to intersect the same operational points described by the truth table.

Table 1. Resulting truth table returned by the various extreme learning machines operating as logical operators.

$x_{j,1}$	$x_{j,2}$	OR_{ELM}	AND_{ELM}	XOR_{ELM}	NOT_{ELM}
0	0	5.7291e-08	-5.8191e-08	-6.0687e-08	-
0	1	1.0000	-1.5527e-07	1.0000	-
1	0	1.0000	-6.9759e-08	1.0000	-
1	1	1.0000	1.0000	-3.1476e-08	-
0		-	-	-	1.0000
1		-	-	-	3.4806e-07

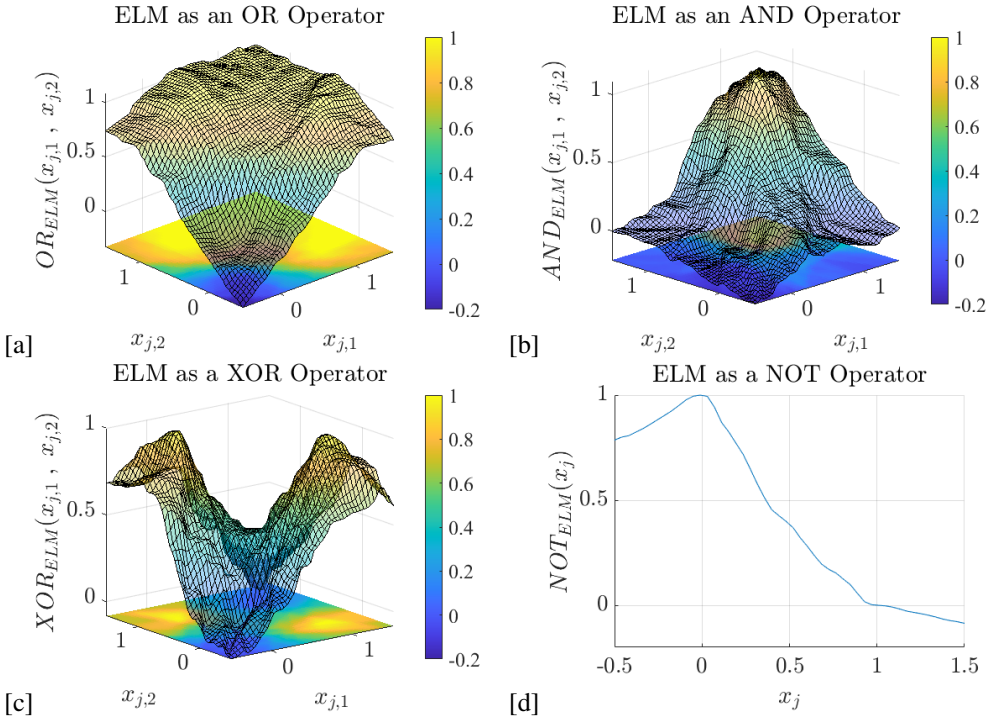

Figure 3. Surface formed by the various extreme learning machines operating as logical operators. The following operators are depicted (a) OR, (b) AND, (c) XOR, and (d) NOT.

Via Table 1 and Fig. 3 it is evident that the extreme learning machine is able to form a function which when inputted values $x_{j,1}$ and $x_{j,2}$ it returns the corresponding value associated with the logical operator with practical perfection. The reason for choosing such a large value of $\tilde{N}$ is the increased complexity of such function, allowing it to not only intersect the points where the logical operator is satiated but to also flatten out near the points, hence accounting for the additive noise. This ability to flatten is quantified numerically by calculating the magnitude of the gradient of each ELM operator at each of the four operational points. For the function to effectively keep the signal from diverging, the magnitude of the gradient should be less than 1, as that indicates that even in the direction of maximal change at the point, the difference between the evaluated output and the ideal value (0 or 1) is less than the original perturbance. As the NOT operator only has one input, the magnitude of the derivative is calculated.

$$\frac{\partial \mathcal{L}_{ELM}}{\partial x_{j,1}}(x_{j,1}, x_{j,2}) \approxeq \frac{\mathcal{L}_{ELM}(x_{j,1}+\epsilon, x_{j,2}) - \mathcal{L}_{ELM}(x_{j,1}, x_{j,2})}{\epsilon} \tag{6}$$

$$\frac{\partial \mathcal{L}_{ELM}}{\partial x_{j,2}}(x_{j,1}, x_{j,2}) \approxeq \frac{\mathcal{L}_{ELM}(x_{j,1}, x_{j,2}+\epsilon) - \mathcal{L}_{ELM}(x_{j,1}, x_{j,2})}{\epsilon} \tag{7}$$

$$|\nabla \mathcal{L}_{ELM}(x_{j,1}, x_{j,2})| = \sqrt{\left(\frac{\partial \mathcal{L}_{ELM}}{\partial x_{j,2}}(x_{j,1}, x_{j,2})\right)^2 + \left(\frac{\partial \mathcal{L}_{ELM}}{\partial x_{j,1}}(x_{j,1}, x_{j,2})\right)^2} \tag{8}$$

Table 2. Resulting truth table returned by the various extreme learning machines operating as logical operators.

$x_{j,1}$	$x_{j,2}$	$\|\nabla OR_{ELM}\|$	$\|\nabla AND_{ELM}\|$	$\|\nabla XOR_{ELM}\|$	$\|\nabla NOT_{ELM}\|$
0	0	0.0487	0.0310	0.0614	-
0	1	0.0038	0.1480	0.0596	-
1	0	0.034	0.0325	0.0349	-
1	1	0.0232	0.0701	0.0459	-
0		-	-	-	0.0139
1		-	-	-	0.0395

It is seen via Table 2 that the magnitude of the gradient for each of the logical operators at the 4 operational points is well below 1, hence indicating that the resulting function does flatten out and accounts for the added noise. In general, the ability for the ELM to function as a logical operator is to be expected, as it has been proved that a SLFN with $\tilde{N}$ hidden nodes and randomly chosen input and bias weights can learn $\tilde{N}$ observations (Huang, 2003; Tamura and Tateishi, 1997). Therefore it should not be a surprise that our trained ELMs can learn to return 4 different logical combinations (when inputted with exact values of 0 or 1) with the rather generous value of $\tilde{N} = 200$ hidden nodes. But our results prove that the ELM is not only able to output these values with precision, but is also able to keep the signal from diverging by flattening near the operational points. A much smaller value of $\tilde{N}$ could be chosen (which would still accurately output exact values at the operational points as seen in Table 1) and transform the output by an activation function (such as a shifted unit step) to account for the noise, though this does not follow the traditional ELM framework described above. To showcase the ability of a large number of the extreme learning machines to operate in a circuit, two examples ran by ELMs are presented in the proceeding two sections: Conway's Game of Life and tic-tac-toe.

3 Tic-Tac-Toe

The use of ELM as logic gates is explored by providing the computational power to host a game of tic-tac-toe. The game of tic-tac-toe (also commonly known as noughts and crosses) is a game where two players take turns placing markers in a 3×3 box, with the ultimate goal of completing three of their marks in the same row, column, or diagonal. The same previously trained ELM logic gates are utilized in a circuit which was implemented with the use of Simulink. A schematic of the circuit can be found below in Fig. 4.

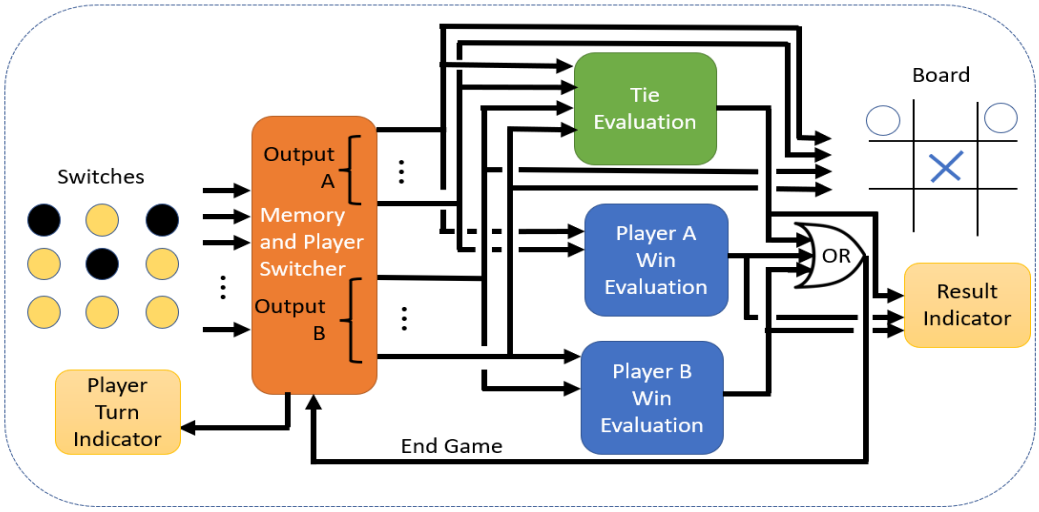

Figure 4. Schematic of the circuit used to play tic-tac-toe.

The player indicator displays which player it's turn it is, who then must select where they would like to place their mark by toggling one of the switches to the on position. This value is then sent to the input memory block, which holds the input. The basis for this specific block was heavily influenced by the circuit made available online by Ngoc Long (Long, 2021). This block consists of sub-blocks such as D Latches and a demultiplexer, which are all solely functions of the trained ELM logic gates [1] . The memory block also computes which player's turn it is by determining whether the sum of the current inputs is even or odd, which is used to appropriately store the next input to player A or B.

This output for each player is then sent to both a win and tie evaluator. The win evaluators basically check to see if a player meets one of the win conditions (i.e., full row, column, or diagonal). Meanwhile the tie evaluator simply checks to see if the board is full. If a player has won or there is a tie, this result is indicated next to the board and the memory block is disabled, hence not allowing players to continue selecting squares. By playing the game, the circuit is seen to perform as expected - once again verifying the efficacy of the logic gates arranged in series.

4 Conway's Game of Life

Conway's Game of Life is a discrete dynamical system consisting of a grid of cells who's "life" and "death" is governed by a set of rules. The game is widely known for the complex structures which appear out of the ostensibly simple laws which dictate the evolution of the system (Peña and Sayama, 2021). These rules are described as follows. For a cell a_0 which is alive on the t^{th} time step ($a_0^t = 1$): if the number of alive neighbors $n \leq 1$ the cell will be dead on the next time step $a_0^{t+1} = 0$ (hence simulating the consequences of solitude), if $2 \leq n \leq 3$ the cell will remain alive on the next time step $a_0^{t+1} = 1$, and if $n \geq 4$

[1] There are a couple other blocks included (such as the built in hold block) to make the model compatible with Simulink.

the cell dies on the next time step $a_0^{t+1} = 0$ (hence simulating the consequences of overpopulation). Meanwhile if the cell is dead on the t^{th} time step ($a_0^t = 0$), it can only become alive on the next time step if $n = 3$.

A combination of 41 logic gates can used to determine whether the cell a_0 will be equal to one or zero on the next time step as it is a function of it's own current value and the value of the surrounding neighbors (Shard, 2013). Therefore rather than training the ELM on a single logic gate, we now train it to mimic the complete combinational logic circuit consisting of 41 logic gates in order to learn the rules of the Game of Life.

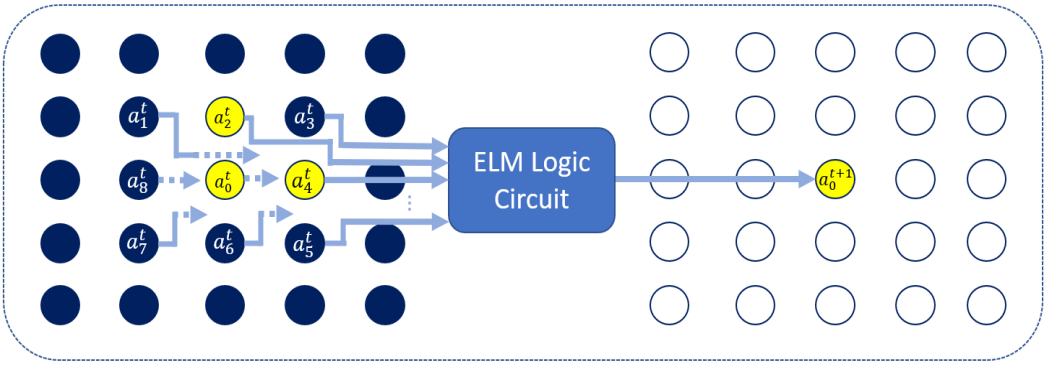

Figure 5. Figure depicting how the grid of cells evolves over time with the use of the trained ELM logic gates. Yellow corresponds to alive (1) and dark blue corresponds to dead (0).

The 9 dimensional input data is once again perturbed for noise resistance. As the combinational logic circuit is more complex, the value of $\tilde{N}$ is changed to $\tilde{N} = 2000$. To display the efficacy of the single ELM acting as the computational power for the game, the evolution of a glider is depicted below in Fig. 6.

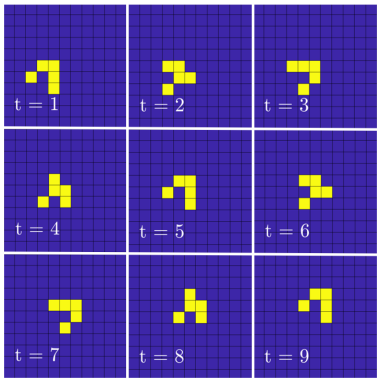

Figure 6. The evolution of a glider over nine time steps dictated by the ELM circuit. The yellow cells correspond to alive (1) and the dark blue squares correspond to dead (0).

The glider is seen to properly evolve over the time frame, as it begins it's ascent to the upright corner of the domain. From this simulation the ELM is seen to function properly, hence being able to learn complex combinational logic circuits.

5 Conclusion

The use of extreme learning machines is demonstrated to operate functionally as a logic gate. The extreme learning machine was seen to form a surface such that the operational points were intersected with near perfection and the surface flattened out near the points to account for the added noise during training. This effectively kept the signal from diverging, hence allowing us to connect the ELM logic gates into large circuits capable of providing the power to play the Game of Life and tic-tac-toe. These results can be of potential interest in the field of neuroscience, as if the ELM is capable of mimicking a logic gate, complex circuits consisting of these logic gates with minimally adjusted synaptic weights can be constructed and allow for advanced computation.

Code availability. The code is available online at https://github.com/BenWalleshauser/ELM-Logic .

References

Adesola: Artificial Neuron Network Implementation of Boolean Logic Gates by Perceptron and Threshold Element as Neuron Output Function, 2015.

Afkham, B. M., Chung, J., and Chung, M.: Learning Regularization Parameters of Inverse Problems via Deep Neural Networks, https://doi.org/10.48550/ARXIV.2104.06594.

Bhat, A. U., Merchant, S. S., and Bhagwat, S. S.: Prediction of Melting Points of Organic Compounds Using Extreme Learning Machines, Industrial & Engineering Chemistry Research, 47, 920–925, https://doi.org/10.1021/ie0704647, publisher: American Chemical Society, 2008.

Bonakdari, H. and Ebtehaj, I.: A Comparative Study of Extreme Learning Machines and Support Vector Machines in Prediction of Sediment Transport in Open Channels, International Journal of Engineering, 29, 1499–1506, https://www.ije.ir/article_72820.html, publisher: Materials and Energy Research Center _eprint: https://www.ije.ir/article_72820_f94f2a0d3a84471bb90369eedfd02048.pdf, 2016.

de Chazal, P., Tapson, J., and van Schaik, A.: A comparison of extreme learning machines and back-propagation trained feed-forward networks processing the mnist database, in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2165–2168, https://doi.org/10.1109/ICASSP.2015.7178354, 2015.

Dong, S. and Li, Z.: Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, Computer Methods in Applied Mechanics and Engineering, 387, 114 129, https://doi.org/10.1016/j.cma.2021.114129, 2021.

Dudek, G.: Extreme Learning Machine as a Function Approximator: Initialization of Input Weights and Biases, pp. 59–69, https://doi.org/10.1007/978-3-319-26227-7_6, 2016.

Huang, G.-B.: Learning capability and storage capacity of two-hidden-layer feedforward networks, IEEE Transactions on Neural Networks, 14, 274–281, https://doi.org/10.1109/TNN.2003.809401, 2003.

Huang, G.-B., Chen, L., and Siew, C.: Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes, IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council, 17, 879–92, https://doi.org/10.1109/TNN.2006.875977, 2006a.

Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K.: Extreme learning machine: Theory and applications, Neurocomputing, 70, 489–501, https://doi.org/https://doi.org/10.1016/j.neucom.2005.12.126, 2006b.

Long, N.: Tic Tac Toe Simulator, https://circuitverse.org/users/75348/projects/tic-tac-toe-simulator, 2021.

Peng, X., Lin, P., Zhang, T., and Wang, J.: Extreme learning machine-based classification of ADHD using brain structural MRI data, PloS one, 8, e79 476–e79 476, https://doi.org/10.1371/journal.pone.0079476, publisher: Public Library of Science, 2013.

Peña, E. and Sayama, H.: Life Worth Mentioning: Complexity in Life-Like Cellular Automata, Artificial Life, 27, 105–112, https://doi.org/10.1162/artl_a_00348, 2021.

Rong, H.-J., Jia, Y.-X., and Zhao, G.: Aircraft recognition using modular extreme learning machine, Neurocomputing, 128, 166–174, 2014.

Shard: Circuit in Conway's Game of Life, https://math.stackexchange.com/q/298366, _eprint: https://math.stackexchange.com/q/298366 Published: Mathematics Stack Exchange, 2013.

Sreekanth, M., Rajesh, R., and SatheeshKumar, J.: Extreme Learning Machine for the Classification of Rainfall and Thunderstorm, Journal of Applied Sciences, 15, https://doi.org/10.3923/jas.2015.153.156, 2014.

Tamura, S. and Tateishi, M.: Capabilities of a four-layered feedforward neural network: four layers versus three, IEEE Transactions on Neural Networks, 8, 251–255, https://doi.org/10.1109/72.557662, 1997.

Wang, J., Lu, S., Wang, S., and Zhang, Y.-D.: A review on extreme learning machine, Multimedia Tools and Applications, https://doi.org/10.1007/s11042-021-11007-7, 2021.

Xing, Y., Ban, X., Liu, X., and Shen, Q.: Large-Scale Traffic Congestion Prediction Based on the Symmetric Extreme Learning Machine Cluster Fast Learning Method, Symmetry, 11, https://doi.org/10.3390/sym11060730, 2019.

Yu, Q., Miche, Y., Eirola, E., van Heeswijk, M., Séverin, E., and Lendasse, A.: Regularized extreme learning machine for regression with missing data, Neurocomputing, 102, 45–51, https://doi.org/https://doi.org/10.1016/j.neucom.2012.02.040, advances in Extreme Learning Machines (ELM 2011), 2013.

Zhang, R., Xu, M., Han, M., and Li, H.: Multivariate chaotic time series prediction based on improved extreme learning machine, in: 2017 36th Chinese Control Conference (CCC), pp. 4006–4011, https://doi.org/10.23919/ChiCC.2017.8027985, 2017.