energies

MDPI

Article

A Learning Probabilistic Boolean Network Model of a Smart Grid with Applications in System Maintenance

Pedro Juan Rivera Torres [1,2,3,*], Chen Chen [4], Jaime Macías-Aguayo [5], Sara Rodríguez González [1], Javier Prieto Tejedor [1], Orestes Llanes Santiago [6], Carlos Gershenson García [7] and Samir Kanaan Izquierdo [3]

1 Department of Computer Science and Automatics, Universidad de Salamanca, Patio de las Escuelas 1, 37006 Salamanca, Spain
2 St. Edmund's College, University of Cambridge, Mount Pleasant, Cambridge CB3 0BN, UK
3 Escuela Técnica Superior de Ingeniería Industrial de Barcelona, Universidad Politécnica de Cataluña, Av. Diagonal, 647, 08028 Barcelona, Spain
4 Department of Computer Science and Technology, University of Cambridge, Cambridge CB3 0FD, UK
5 Center for Transportation and Logistics, Massachusetts Institute of Technology, 1 Amherst Street, MIT Building E40-376, Cambridge, MA 02139, USA
6 Departamento de Control y Automática, Instituto Superior Politécnico José Antonio Echeverría (CUJAE), Marianao, La Havana 19390, Cuba
7 School of Systems Science and Industrial Engineering, Binghamton University, Binghamton, NY 13902, USA
* Correspondence: pedro.rivera@usal.es

Abstract: Probabilistic Boolean Networks can capture the dynamics of complex biological systems as well as other non-biological systems, such as manufacturing systems and smart grids. In this proof-of-concept manuscript, we propose a Probabilistic Boolean Network architecture with a learning process that significantly improves the prediction of the occurrence of faults and failures in smart-grid systems. This idea was tested in a Probabilistic Boolean Network model of the WSCC nine-bus system that incorporates Intelligent Power Routers on every bus. The model learned the equality and negation functions in the different experiments performed. We take advantage of the complex properties of Probabilistic Boolean Networks to use them as a positive feedback adaptive learning tool and to illustrate that these networks could have a more general use than previously thought. This multi-layered PBN architecture provides a significant improvement in terms of performance for fault detection, within a positive-feedback network structure that is more tolerant of noise than other techniques.

Keywords: fault detection and isolation; machine learning algorithms; probabilistic Boolean networks; probabilistic Boolean network modeling; smart grids; complex network modeling

Citation: Rivera Torres, P.J.; Chen, C.; Macías-Aguayo, J.; Rodríguez González, S.; Prieto Tejedor, J.; Llanes Santiago, O.; García, C.G.; Kanaan Izquierdo, S. A Learning Probabilistic Boolean Network Model of a Smart Grid with Applications in System Maintenance. *Energies* **2024**, *17*, 6399. https://doi.org/10.3390/en17246399

Academic Editor: Ahmed Abu-Siada

Received: 14 October 2024
Revised: 28 November 2024
Accepted: 16 December 2024
Published: 19 December 2024

1. Introduction

Probabilistic Boolean Networks (PBNs) [1] have been employed in the scientific literature not only for modeling gene regulatory networks (GRNs), but also for engineered systems such as smart power networks and manufacturing systems. PBNs are stochastic versions of Kauffman's Boolean Network (BN) model [2,3], in which a GRN is modeled using a graph in which every gene is represented by a binary node that can assume the expressed or unexpressed state. There is also a Boolean function (called a predictor) that specifies the value of each node based on its current state. PBNs are not as deterministic as their BN counterparts. For every node in the network, there are one or more predictors, and each has a probability of occurrence, essentially resulting in a tree of BNs that are selected based on probability.

Controlling these networks has become an important field of research in systems biology, since therapy (radiation, gene therapy, chemotherapy) may be thought of as an intervention that controls the evolution of a GRN [4,5]. BNs and PBNs have certain states, either a single state (fixed point) or set of states (cyclic) that repeat, towards which the

network tends to evolve. These states are called attractors, and they represent the network's behavior in the steady state. These attractors can be seen as a form of knowledge stored within the network. Previous work has demonstrated that these states can be equated in a manufacturing system to the states that a machine may be in, either its normal operating mode or the different types of faults or failures it may experience, making these states equivalent to an intervention in the form of predictive or preventive maintenance [6–8].

Machine learning has been used to control PBN models and direct their "learning" using deep reinforcement learning [5]. In [9,10], the capacity of PBNs to perform basic reinforcement learning was studied. In this paper and based on the learning processes inherent to neural networks, we propose an architecture variant for PBNs to learn with the aim of preserving a state without external control, which constitutes the first contribution of this paper. The proposed architecture is parametrized, enabling the adjustment of these parameters to obtain better network performance results, which is the second contribution of this work. The presented results demonstrate the effectiveness of the proposed method and open new possibilities, as there is no precedent of a similar method.

Regarding maintenance and the detection of faults in smart grids, the literature highlights several contemporary challenges. One of these challenges is the real-time identification of issues in smart grids. Examples include detecting anomalies in real time [11], predicting hardware failures in sequential data environments in real time [12], and detecting faults in noisy environments in real time [13]. Another challenge lies in autonomous detection of anomalies in smart grids. This task has become increasingly difficult due to complex grid conditions (e.g., unpredictable renewable energy sources), massive amounts of heterogeneous data to process, and unusual and rare faults that are hard to detect [14]. A third challenge comes from the limited scalability of communication architectures, which makes it challenging to handle the large volume of data coming from multiple sensors [15]. This is due to the growing complexity of smart grids, which is exacerbated by various environmental factors [16]. Other challenges include the detection of cyber-security intrusions and the integration of legacy monitoring infrastructure in the smart environment [15]. Together, these challenges constitute an area of active research where multi-faceted solutions have been proposed.

Most of the proposed solutions to improve fault detection and location in smart grids are impedance-based, analytical (e.g., correlation, likelihood, Markov analysis, etc.), or learning-based [15]. Among these, learning-based solutions that are commonly known as Machine Learning (ML) algorithms have grown in popularity in the context of smart grid applications. For instance, ML algorithms such as Isolation Forest, one-class support vector machines (SVMs), artificial neural networks, and random forests have been successfully applied and compared in the context of anomaly detection [11]. Similarly, long short-term memory (LSTM) networks (a type of ANN) have been applied for real-time fault prediction in sequential data environments [12]. For a comprehensive summary of ML techniques used in smart grid operations, the reader is referred to [17]. Apart from ML techniques, analytical methods constitute the second most popular set of methods for fault detection and location.

Despite the extensive coverage of analytical and ML techniques for anomaly detection and fault prediction and detection, the application of PBNs in this context has been limited. This study bridges this gap by pioneering the use of PBNs in combination with ML to model the complex dynamics of smart grids. In doing so, this study shows how PBNs can support decisions about maintenance of smart grids, hence extending the toolkit available for practitioners in the field. More broadly, this study contributes to the family of mixed approaches involving both analytical and learning-based methods, a strategy described in the literature that appears to provide better results [15].

2. Materials and Methods

In this section, we discuss the relevant techniques used in this study and introduce the proposed method and the experiments performed to measure its performance.

2.1. Probabilistic Boolean Networks

PBNs are state-transition dynamical system models used extensively in gene regulatory network modeling. Proposed by Shmulevich and Dougherty [1], they are a non-deterministic extension of Kauffman's Boolean Network (BN) model. Figure 1a presents an example of a PBN, and Figure 1b presents a constituent Boolean Network of a PBN, where the dark circles represent attractors (states that repeat).

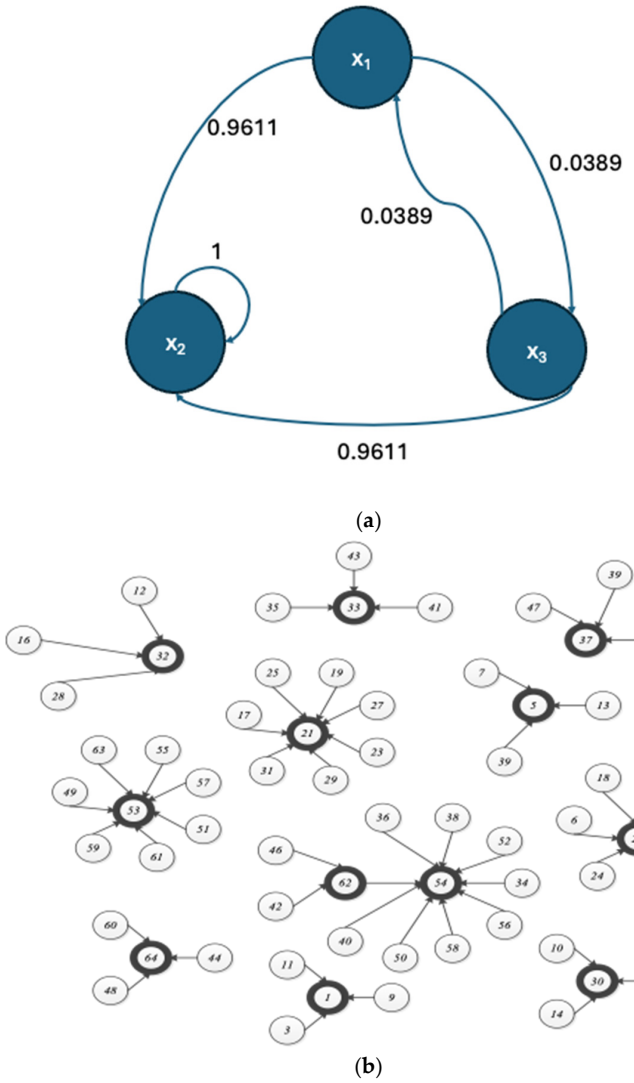

Figure 1. (**a**) A Probabilistic Boolean Network. (**b**) Transition diagram of a constituent BN of a Probabilistic Boolean Network.

PBNs are computational models used to represent and analyze the dynamics of complex systems. It is a discrete-time dynamical system characterized by a set of Boolean variables and their probabilistic transitions.

Let $V = \{x_1, x_2, \ldots, x_N\}$ be a set of Boolean variables, where each x_i can take values of 1 or 0 (ON and OFF). A Probabilistic Boolean Network is defined as a graph G(V, F), where V represents the set of nodes and F represents a list of predictor function sets F = ($F_1, \ldots, F_n$); $F_i = \left\{ f_1^{(i)},\ f_{2,}^{(i)}, \ldots,\ f_{l(i)}^{(i)} \right\}$, $l(i)$ is the number of predictors, $f_{l(i)}^{(i)} : \{0,\ 1\}^v \rightarrow \{0,\ 1\}$, and v is the number of input nodes for each x_i. A context of realization of a PBN at any point in time is defined by a set of predictor functions that are applied. With D possible contexts, there exist D vector functions $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_N$, and every $\boldsymbol{f}_j = \left(f_1^{(i)}, \ldots, f_{l(i)}^{(i)} \right)$, where $1 \leq j \leq N$, and $1 \leq j_i \leq l(i)$, $f_{j_i}^{(i)} \in \boldsymbol{F_i}$. A context of the PBN is a mapping of the node

values into the new node values, $\boldsymbol{f_j} : \{0,1\} \rightarrow \{0,1\}$, and they represent the constituent Boolean Network that governs during a particular time. For each node, there is a probability of selection of a specific predictor and a probability of selecting a specific context or constituent BN, with $c_j^{(i)} = Pr\left\{\boldsymbol{F}^{(i)} = f_j^{(i)}\right\}$. If $\boldsymbol{F_i}$ represents a specific context, the total number of contexts in a PBN (or the number of constituent BNs) is the product of all the predictors for each node. In a two-node PBN with two predictors per node, D, the total number of contexts would therefore be four. The probability of selecting a particular context would then be the product of the selection probabilities for each context. For a complete definition of a Probabilistic Boolean Network, please refer to [1,18].

2.2. Machine Learning

Machine learning (ML) [19] is a field within the artificial intelligence umbrella that is centered on the development of algorithms and systems that can learn or improve their performance as a function of the information they consume. The data are analyzed and patterns are classified, with the knowledge obtained used for the improvement of an assigned task. Depending on the situation, ML algorithms can function with more or less human intervention. There are four main types of ML strategies: supervised [20], unsupervised [21], semi-supervised [22], and reinforcement learning [23].

In supervised learning, the computer has a labeled dataset that allows it to learn a human task. This is the simplest ML grouping because it tries to mimic human learning. With unsupervised learning, the data are not labeled, and these algorithms try to extract information out of the dataset or previously unidentified patterns. There are many means through which this can be achieved, including clustering, where a computer finds similar data points in a set and groups them (creating clusters). In density estimation, the computer discovers "knowledge" when it tries to find how the dataset is distributed. In anomaly detection, the computer identifies data points within this dataset that are significantly different from the rest. In dimensionality reduction algorithm analysis, a computer analyzes a dataset and summarizes the data so they can be used to make predictions or trends. In semi-supervised learning, a computer has a partially labeled dataset and performs its task by using the labeled data to understand the parameters and interpret unlabeled data. With reinforcement learning, a computer observes its environment and uses that data to identify the "ideal" behavior that will minimize risk or maximize reward. This iterative process requires a reinforcement signal to help identify the best course of action.

In this study, the discussion is centered around the use of positive (feed-forward) feedback (where the information regarding the outcome of an action previously performed is used to increase or decrease the probability of the same action being performed or not if the same conditions hold in the future) for the direction of the learning process, as algorithms that use such strategies, such as the genetic algorithm [24] and ant colony optimization [25] are more similar to the PBN model. The most recognizable method to perform machine learning is the use of artificial neural networks (ANNs) [26], which can use both negative and positive feedback, as in the case of feed-forward networks (convolutional neural networks are an example) [27]. Within the field of machine learning, ANNs are a group of ML models that were conceived using the principles that govern biological neural networks and neuronal organization [28]. They are at the center of the deep learning algorithms [29]. Simplifying the connectivity of a real brain (which is recurrent, i.e., with feedback), an ANN is formed by interconnected layers of neurons, grouped in levels (layers); a layer is a set of neurons with inputs that come from the previous layer (from the input data in the case of the first layer) and whose outputs are identical to the input of another layer. The first layer receives the input data that feeds the ANN, and the layers between input and output (without any feedback). The hidden layers are named this way because the input and output values are unknown. ANNs used for deep learning [29] contain many hidden layers. ANNs are constructed with interconnected layers of nodes that contain an input layer (which receives the initial data or features that are fed unto the network, and where each input has an associated numerical value or feature), one or

more hidden layers (the layers that perform computations on the input data and extract higher-level features from it via mathematical transformations), and an output layer (which produces the network's predictions or outputs that are based on the input data and the learned weights).

Schematically, an artificial neuron (a node of the network) has weighted inputs and biases. The weights that are associated with each input determine the strength of the connections between neurons, and during the learning process, the weights are adjusted to optimize the performance of the network. The bias term allows additional fine tuning of the activation function, a threshold (the neuron applies an activation function to the sum of the weighted inputs that produce the output and introduces non-linearities to the network, permitting the modelling of complex relationships between input and output), and an output, which could be connected to another neuron. The sum of their inputs multiplied by their associated weights determines the "neural impulse" that is received. The activation function returns a value that is sent as the output. Computation flows forward through the network in a process called forward propagation, in which the outputs of the neurons in a layer serve as the inputs to the neurons of the next, until the output layer is reached, which produces the final predictions. Activation functions can be linear or non-linear.

Figure 2 illustrates a neuron and shows the inputs x_1 to x_n, the bias a, and the activation function f applied to the weighted sum of inputs.

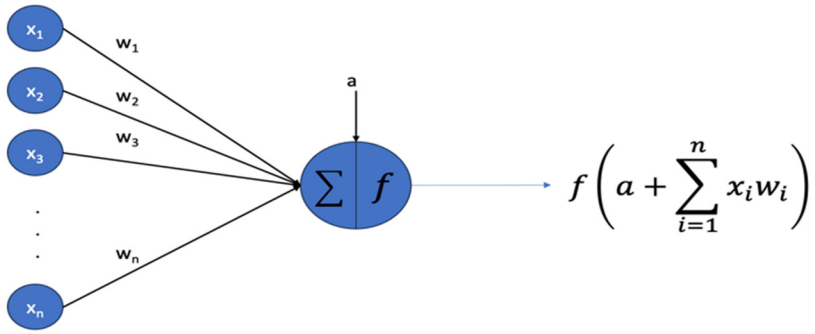

Figure 2. A neuron with inputs labeled from x_1 to x_n, their weights, a bias, and the activation function.

Training an ANN consists of adjusting each of the weights of the inputs such that the values in the output layer are close to the known data. These networks are trained by algorithms such as backpropagation, which involves comparing the predicted outputs of the network with the desired outputs and calculating an error measure as the difference between the predicted and target values. The error is propagated backwards through the network, adjusting the weights and biases using optimization algorithms such as gradient descent. The iterative nature of the process minimizes the error, helping the network to learn and make more accurate predictions [30].

2.3. Intelligent Power Routers

With this research, we apply a new modeling architecture for smart grids, using a device called an Intelligent Power Router. The IPR [31] was developed as a simple device (with four main components) that can manage power intelligently in a complex environment. It is an architecture for distributed control and decentralized coordination. This device allows systems for electrical power generation, transmission, and distribution to be more resilient, by allowing the grid to respond to emergencies, attacks, or demand by effectively handling the available resources. A complete description of the device and its components, along with its reliability calculations, is available in [31] and an application with PBNs is presented in [9,10]. The IPR is composed of a data router, software that defines the behavior of each device and its response to demand, emergencies, etc., and a set of switches that connect the device to different points of the power network; it has been used to model manufacturing systems and smart-grids [6–10].

2.4. The Proposed Architecture: Probabilistic Boolean Network Learning

In this study, we employed a system modeled as a PBN: a model of the Western System Coordinating Council (WSCC) nine-bus system that employs the Intelligent Power Router (IPR). The WSCC nine-bus model is a test case used to represent an equivalent small system comprising nine buses, six transmission lines, three generators, and three loads, which is simple to control. It was designed for the study of power systems control, stability, and load flow analysis. It includes three two-winding power transformers and has been used in both steady-state and dynamic stability studies, serving as a good reference for various simulations in educational and research contexts. It is commonly used in research for studying fault analysis, load flow solutions, and system dynamics, as it provides a balanced level of difficulty without being excessively large. This model was built into the PRISM model checker [32], using PRISM's 4.7 programming language. The advantage of using this model checker and its language was that we were able to perform experiments and simulate events in a single environment. PRISM has been used for programming and analyzing the formal correctness of several PBN models [6–10,33].

For the selection probabilities of the predictors of the model, an unbiased approach was used as a starting point (note that this is unrelated to the concept of bias in ANNs), where the probabilities were all balanced, each predictor receiving equal probability with respect to the other (each node had a 0.5 probability of producing a '0' or '1'). The values of the probabilities for each predictor were set at the start of the execution, and at that time, the input values for the first level were chosen (as per the patterns previously mentioned). In this method, the output values of each layer are based on the input values selected and the values resulting from the application of the predictor functions. The values of the output layer's input correspond to the output of the inner layer just before. At such a point, the value of the output of each node in the output layer is compared with the value from the input pattern that the user would like the network to learn (the value for the node that the user wants). If that value is correct, the probability of the selected predictor for that node is strengthened (increased), and the value of the predictor(s) that produce an incorrect result are lessened by a small factor *delta* that is constant and set at the beginning of the execution. For the internal layers, there are no values that can be used to compare each layer's output, and an immediate assessment of the correctness of the output of each internal layer cannot be performed. For this reason, a correction factor is introduced at the output of all layers. We cannot compare these values, but we can perform a computation to measure the correctness of each of the output node values. The correction factor is a linear function that has a value of 1 if all node values are correct, −1 if all node values are incorrect, and 0 if the number of correct values is equal to the number of wrong values. The delta factor is then multiplied by this correction factor, which is added to the probability of the output of every node.

Figure 3 describes the process of modeling a system as a Probabilistic Boolean Network, which was employed previously in [6–10,34].

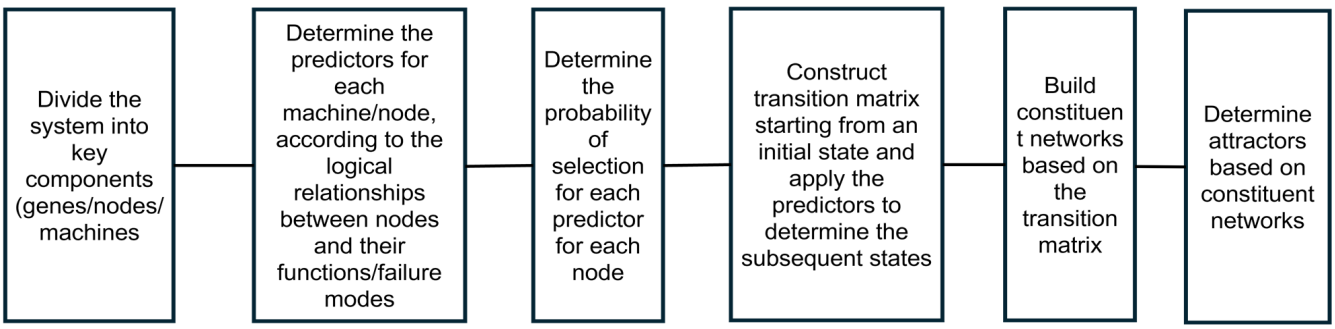

Figure 3. Method for characterizing a system as a PBN.

2.5. Description of the Method

The proposed architecture is a PBN as defined by Shmulevich and Dougherty in [1], where each node represents an Intelligent Power Router used in every bus of the WSCC nine-bus test system, but with a multi-layered feed-forward structure where each layer is a PBN. Each node has a set of predictor functions that determine the next state of the node, based on the influence of the network's previous state. All connections are fixed and based on the physical structure of each of the networks presented; they are intentionally connected as they follow each of their specific connection structures. This structural modification was carried out in PRISM, representing a feed-forward PBN that differed from the model presented in [9,10]. The network was composed of four layers: an input layer, two middle layers, and an output layer. Figure 4 illustrates the architecture of the solution.

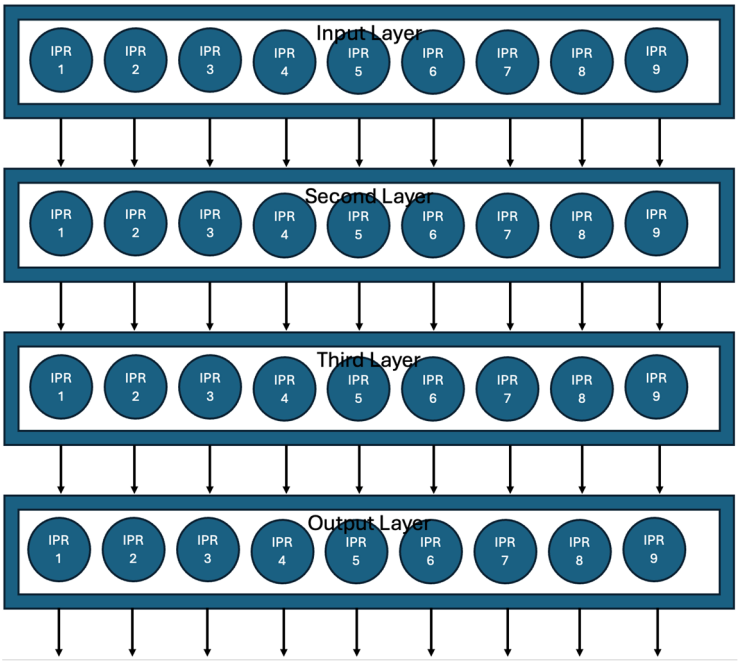

Figure 4. Architecture of the proposed learning PBN.

Before the input level, the values of the probabilities for the predictors are set by the user. The initial values of the nodes are also defined by the user. The first layer is the input level. The output of each layer becomes the input of the next layer, until the output layer is reached. The values of the nodes after the input layer are determined using the values of the previous layer's output as an input. In this state, the value of the node is compared with the desired state's value for the node after applying the predictor functions. If the result is correct, the probability of that predictor function is increased by adjusting the parameters employed to increase or decrease the probability of occurrence of the predictors based on their suitability to support the network's learning of the desired state.

2.6. Description of the Experiments

To verify the system's performance, a performance index (PI) was established. The PI was defined as follows:

$$\mathrm{PI}(t) = \frac{\sum_{\tau=t-n+1}^{t} nocb(\tau)}{n} \quad (1)$$

where *nocb* is the number of correct bits after each iteration τ, and $n \leq Min\{\overline{n}, \overline{\tau}\}$, where $\overline{\tau}$ is the number of iterations with the same input pattern, and $\overline{n}$ is a user defined constant. This gives a measure of the algorithm's performance and allows the user to track whether the system is learning. PI(t) gives the average number of correct bits that have appeared in the last n iterations, where n is the smallest value between a user-defined constant and

the iterations that have been performed with the same input pattern. Two $\overline{n}$ values were selected: $\overline{n}$ = 100 and $\overline{n}$ = 200. In addition, a satisfaction index (SI) was introduced, where an input pattern is sustained until the performance index is equal to this satisfaction index. A complete set of inputs is sought (cycling through each possible combination of inputs). A complete set is the set of all possible inputs of the network, which in this case contains 2^9 different patterns of input. Once a complete set has been achieved, the value of the SI is adjusted to a higher level of satisfaction. The SI represents the PI that the user is satisfied with. The values used for SI are between 0.55 to 0.85. The number of iterations required to achieve a complete set of inputs in the network is not constant. Testing with the equality function is equivalent to performing an intervention in the system immediately after a fault or failure is detected (based on the condition of the asset, as in predictive maintenance).

These networks use positive feedback instead of backpropagation. The transition probabilities are adjusted through the PI, SI, and correction factor to select the constituent BN that holds the attractor state that contains the network pattern to be learned. Here the parameter *delta* is introduced as a constant parameter at the start of the execution, as a correction factor that allows the probability of occurrence to be increased or decreased for each predictor. The *delta* factor chosen was *delta* = 0.01, and the correction factor introduced in the inner and output layers was given a value as described in the previous section.

Simulations were performed using the PRISM model checker following the method described in Figure 2 but programmed as a positive-feedback PBN set up with an input layer, two middle layers, and an output layer, as per Figure 3. In the PRISM simulations presented, the algorithm considered the number of steps performed and calculated the number of iterations taken to produce a certain satisfaction level and to calculate the number of complete input sets.

The PRISM IPR model is a PBN that follows a structure similar to the structure from [10], modeled as a set of nodes that represent the device's components and the predictor functions that provide a description of the physical and logical relationships they have, with a set of sets of Boolean predictor functions that determine the next state of each node through the relationships each component have with the components connected to it, and their state in the current time step. PBNs satisfy the Markov property. PRISM modules contain variables representing the system and its components and statements in the PRISM language that express the logical relationships between the elements of the system (the predictors). With labeled transitions, the modules can synchronize and connect the output of each layer as an input to the next. The probability of occurrence of each predictor is affected in two ways; for what we call a biased PBN, one of the node predictors (in the case of this model, every node has two predictors; refer to [10] for specific details) of the IPR's PBN with a higher probability of occurrence (0.1) is favored, and for an unbiased PBN, each predictor has the same probability of occurrence as the others (0.5).

The PI is used to monitor the behavior of the proposed architecture, and it calculates the average number of correct input values during the last n iterations, where n is the smallest constant value between a user-defined value and the number of iterations that have been performed with the same pattern in the input nodes of the network. In these experiments, the SI was the performance level of the architecture that we were satisfied with. An input pattern remains fixed until the PI reaches a specific SI and a complete set of inputs is found, after which the SI is set to a higher value.

3. Results and Discussion

The results of the experiments designed to measure the performance of the proposed architecture and a comparison between the previous PBN modeling and the proposed method are presented in this section.

In Table 1, the results of the experiments performed in PRISM to assess the model's learning capacity are presented. An initial SI was set at 0.55 and incremented after a complete set of inputs until the target SI was reached. The value of $\overline{n} = 100$ and δ = 0.01.

Table 1. Performance of the algorithm learning the equality function using a satisfaction index (unbiased PBN, $\delta = 0.01$, $\overline{n} = 100$).

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NIs	NCSs	NIs	NCSs	NIs	NCSs	NIs	NCSs
0.65	90	80	--	--	--	--	--	--
0.75	97	98	22	11	--	--	--	--
0.85	198	469	77	133	54	41	--	--
0.95	270	675	92	693	144	180	4	12

The predictor probability was not biased, and all predictors had the same probability of occurrence. In the following tables, NIs stands for number of iterations, representing the quantity of iterations required to reach a certain SI, and NCSs stands for number of complete sets, i.e., the number of complete sets that were needed to reach that SI. The experiments started with an SI of 0.55 and increased to an SI of 0.85 by intervals of 0.1. The results for the negation function are similarly presented in Table 2.

Table 2. Performance of the algorithm learning the negation function using a SI (unbiased PBN, $\delta = 0.01$, $\overline{n} = 100$).

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	25	10	--	--	--	--	--	--
0.75	120	239	20	6	--	--	--	--
0.85	147	331	22	10	121	122	--	--
0.95	313	542	25	18	140	147	164	155

In the experiments reported in the preceding tables, the initial SI was incremented until the final SI of 0.85 was reached. In the NI columns, the tables document the numbers of iterations required to attain specific SI levels, and in the NCS columns, they show the numbers of complete sets of inputs required to obtain those SI values. As an example, in Table 1, starting from an SI of 0.55 and incrementing the SI by 0.10 after n = 100, the network reached SI = 0.85 after 270 iterations, and after that number of iterations, 675 complete sets of inputs were found.

The next set of experiments did not adjust the other parameters (*delta*, n) but changed the selection probabilities of the predictors. The results of these experiments are presented in Tables 3–6. The networks were biased, favoring one of the two predictors for node by 0.1.

Table 3. Performance of the Algorithm learning the equality function using a Satisfaction Index for a biased PBN on component reliabilities, $\delta = 0.01$, $\overline{n} = 100$, with biased predictors (first bias).

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	13	4	--	--	--	--	--	--
0.75	22	20	31	6	--	--	--	--
0.85	34	55	35	17	18	11	--	--
0.95	222	1233	69	278	20	88	40	30

Table 4. Performance of the Algorithm learning the equality function using a Satisfaction Index for a biased PBN on component reliabilities, $\delta = 0.01$, $\overline{n} = 100$, with biased predictors (second bias).

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	96	64	--	--	--	--	--	--
0.75	108	101	175	78	--	--	--	--
0.85	223	611	626	516	115	35	--	--
0.95	139	276	368	511	135	76	216	260

Table 5. Performance of the Algorithm learning the negation function using a Satisfaction Index for a biased PBN on component reliabilities, $\delta = 0.01$, $\overline{n} = 100$, with biased predictors (first bias).

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	96	59	--	--	--	--	--	--
0.75	108	88	175	76	--	--	--	--
0.85	223	261	626	509	115	30	--	--
0.95	632	1277	627	511	135	75	216	186

Table 6. Performance of the Algorithm learning the negation function using a Satisfaction Index for a biased PBN on component reliabilities, $\delta = 0.01$, $\overline{n} = 100$, with biased predictors (second bias).

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	49	9	--	--	--	--	--	--
0.75	134	211	118	103	--	--	--	--
0.85	224	496	172	206	125	3	--	--
0.95	228	1027	976	226	271	506	153	86

In the case of the equality function, for the first bias, the network required fewer iterations to reach the desired SI in the SI = 0.5 column but produced fewer complete sets. For the SI = 0.65 column, we cannot reach a definitive conclusion, because for the SI range 0.65–0.95 there were fewer iterations in two results, but also fewer complete sets, while for the 0.65–0.75 range, the unbiased network required fewer iterations but also produced fewer complete sets. In these cases, there seems to be an indication that the selection of the probabilities affected the transitions, but we cannot reach a conclusion in terms of performance. For the second bias, the network seems to have been unable to learn the equality function at a final SI of 0.95 in the first three cases but learned in the range 0.85–0.95. For the negation function, the network learned in all ranges and with both biases, requiring more iterations in almost every instance but producing more complete sets, except for the range 0.85–0.95 with the second bias.

In the next set of experiments, the effect of varying the parameters $\overline{n}$ and δ was evaluated. Tables 7 and 8 present the performance after changing the $\overline{n}$ parameter for the unbiased case, where $\overline{n} = 200$.

Table 7. Performance of the algorithm learning the equality function using a Satisfaction Index for an unbiased PBN, $\delta = 0.01$, $\overline{n} = 200$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	14	10	--	--	--	--	--	--
0.75	42	77	19	4	--	--	--	--
0.85	84	266	56	86	51	23	--	--
0.95	109	756	189	705	342	517	56	90

Table 8. Performance of the algorithm learning the negation function using a Satisfaction Index for an unbiased PBN, $\delta = 0.01$, $\overline{n} = 200$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	33	16	--	--	--	--	--	--
0.75	37	31	79	33	--	--	--	--
0.85	64	116	85	45	38	12	--	--
0.95	180	934	271	498	70	448	124	87

In these experiments, for almost all ranges, the network required fewer iterations and also produced fewer complete sets, except for the final SI = 0.95. For the negation function, the results were different. For the initial SI of 0.55, fewer iterations were required, with one exception (final SI = 0.65), but the NCSs varied. For the initial SI 0.65, the network with $\overline{n} = 100$ required fewer iterations, but the network with $\overline{n} = 200$ produced more complete sets. The exact opposite results were obtained for the initial SI = 0.75; where fewer iterations were observed in the $\overline{n} = 200$ network, but the NCS results were different; final SI = 0.85 had a higher NCS value than final SI = 0.95. Both networks learned both functions in the cases presented. The performance for the unbiased case with a delta parameter of 0.001 and $\overline{n} = 100$ is presented in Tables 9 and 10. The performance of the algorithm learning the equality function using a satisfaction index for an unbiased PBN, $\delta = 0.001$, $\overline{n} = 200$ is presented in Table 11, while the performance of the algorithm learning the negation function using a satisfaction index for an unbiased PBN, $\delta = 0.001$, $\overline{n} = 200$ is presented in Table 12.

With these parameters, the networks required fewer iterations in most cases but also produced fewer complete sets, for both functions. The results of the experiments for the biased case with parameters of $\delta = 0.0001$, $\overline{n} = 100$ are presented below on Tables 13 and 14.

Table 9. Performance of the Algorithm learning the equality function using a SI for an unbiased PBN, $\delta = 0.001$, $\overline{n} = 100$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	21	2	--	--	--	--	--	--
0.75	65	119	8	1	--	--	--	--
0.85	87	263	20	23	43	27	--	--
0.95	88	479	92	332	102	50	152	67

Table 10. Performance of the Algorithm learning the negation function using a Satisfaction Index for an unbiased PBN, $\delta = 0.001$, $\overline{n} = 100$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	11	8	--	--	--	--	--	--
0.75	50	62	28	30	--	--	--	--
0.85	63	129	40	46	82	44	--	--
0.95	314	840	748	1331	88	131	168	130

Table 11. Performance of the algorithm learning the equality function using a Satisfaction Index for an unbiased PBN, $\delta = 0.001$, $\overline{n} = 200$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	8	5	--	--	--	--	--	--
0.75	26	41	12	12	--	--	--	--
0.85	33	49	57	99	22	16	--	--
0.95	131	866	62	141	28	309	427	185

Table 12. Performance of the algorithm learning the negation function using a Satisfaction Index for an unbiased PBN, $\delta = 0.001$, $\overline{n} = 200$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	62	54	--	--	--	--	--	--
0.75	70	70	27	15	--	--	--	--
0.85	73	91	76	113	42	46	--	--
0.95	222	1043	86	138	87	126	83	107

Table 13. Performance of the algorithm learning the equality function using a Satisfaction Index for an unbiased PBN, $\delta = 0.0001$, $\overline{n} = 100$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	9	12	--	--	--	--	--	--
0.75	10	11	68	34	--	--	--	--
0.85	59	112	85	100	107	55	--	--
0.95	91	1220	472	1119	139	152	53	44

In general, for both functions, adjusting the delta parameter to 0.0001 resulted in fewer iterations and produced fewer complete sets, with exceptions mainly in the final SI = 0.95.

For a more visual-friendly representation of the information in these tables, we include the following figures (Figure 5A–B1). For each of the tables presented (in order of appearance), there is a figure that presents the NIs and NCSs needed for a specific final SI.

Table 14. Performance of the algorithm learning the negation function using a Satisfaction Index for an unbiased PBN, $\delta = 0.0001$, $\overline{n} = 100$.

Final SI	Initial Satisfaction Index (SI)							
	SI = 0.55		SI = 0.65		SI = 0.75		SI = 0.85	
	NI	NCS	NI	NCS	NI	NCS	NI	NCS
0.65	28	11	--	--	--	--	--	--
0.75	44	63	20	12	--	--	--	--
0.85	136	293	41	51	12	4	--	--
0.95	398	1491	87	170	43	114	33	67

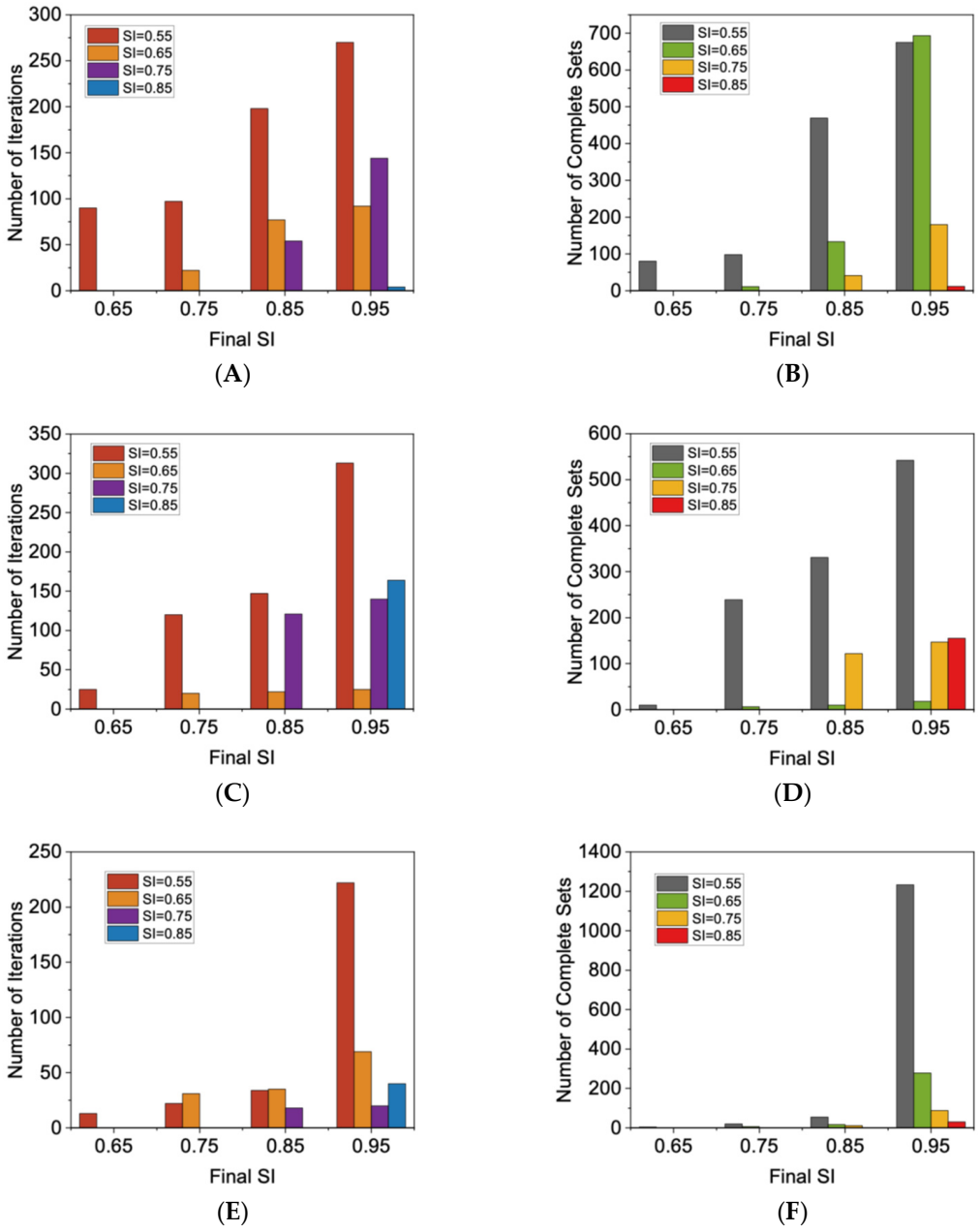

Figure 5. *Cont.*

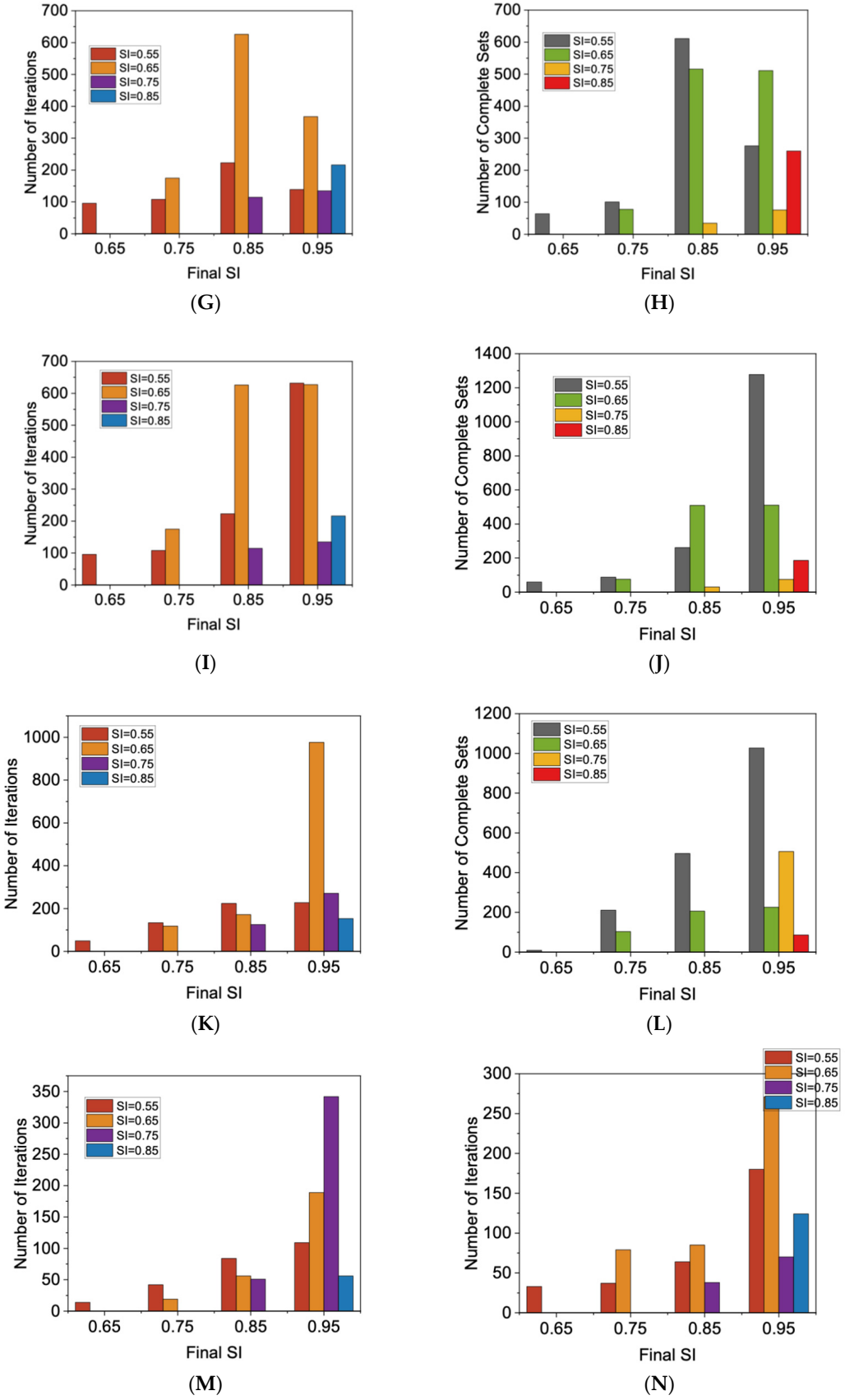

Figure 5. *Cont*.

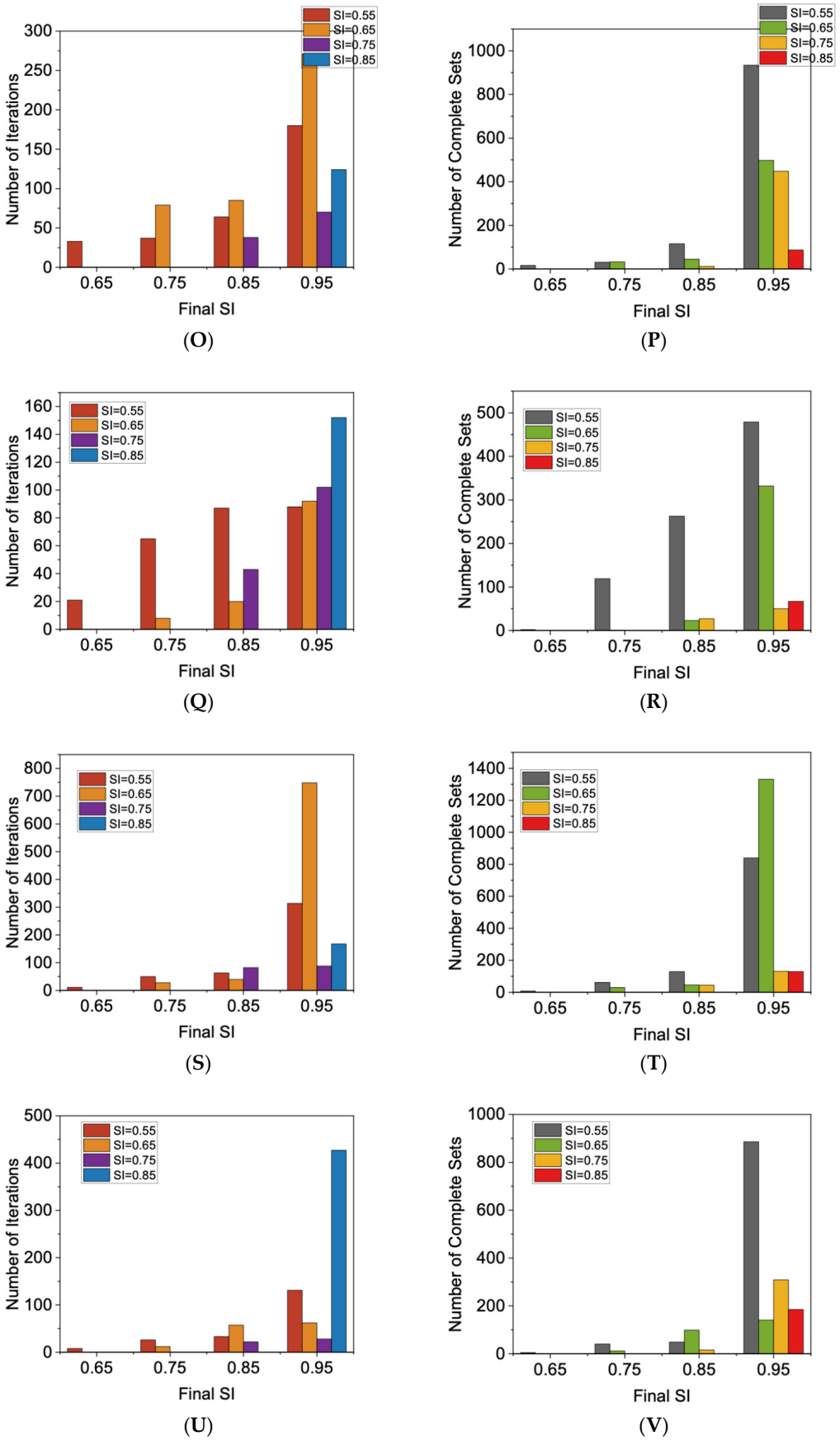

Figure 5. *Cont*.

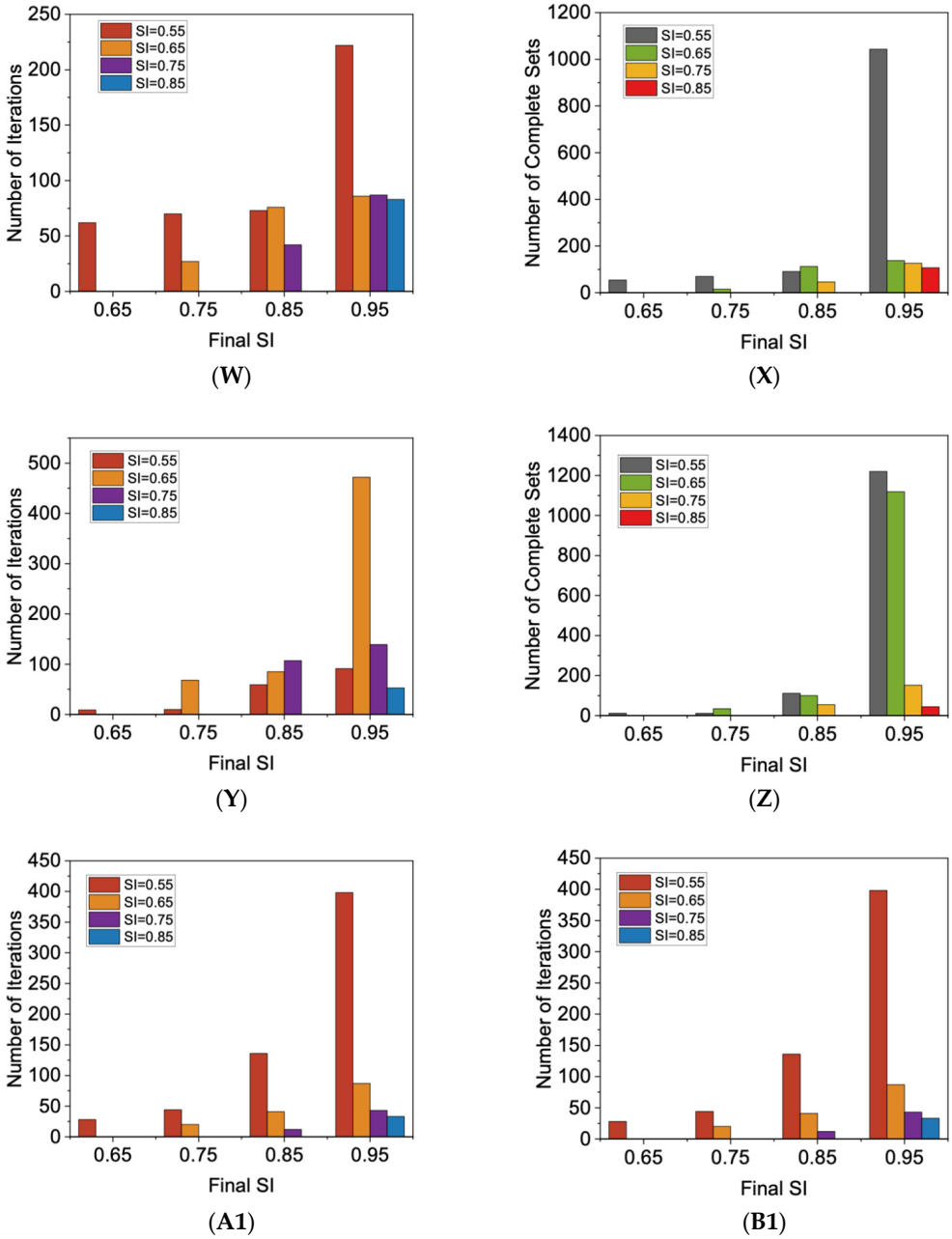

Figure 5. NI and NCS needed for a specific SI.

To ensure our assumptions were correct, an experiment to determine the maximum probability of occurrence of a fault was conducted using PRISM. A comparison was carried out between a PBN model of the WSCC 9 bus system without learning, which this group has used for other research in the past [10], and the model proposed in this study. If the Learning PBN model was indeed performing correctly, the maximum probability of a fault/failure occurring in the system as presented could be equivalent to performing predictive maintenance (PdM) in the system (thereby lowering the probability of occurrence of a fault/failure), compared with when the proposed system was learning the equality function (which in this setting implies that all of the system's components are in their normal operational mode).

Figure 6 presents the results of an experiment in PRISM to determine the maximum probability of occurrence of a failure of the PBN modeled system without learning.

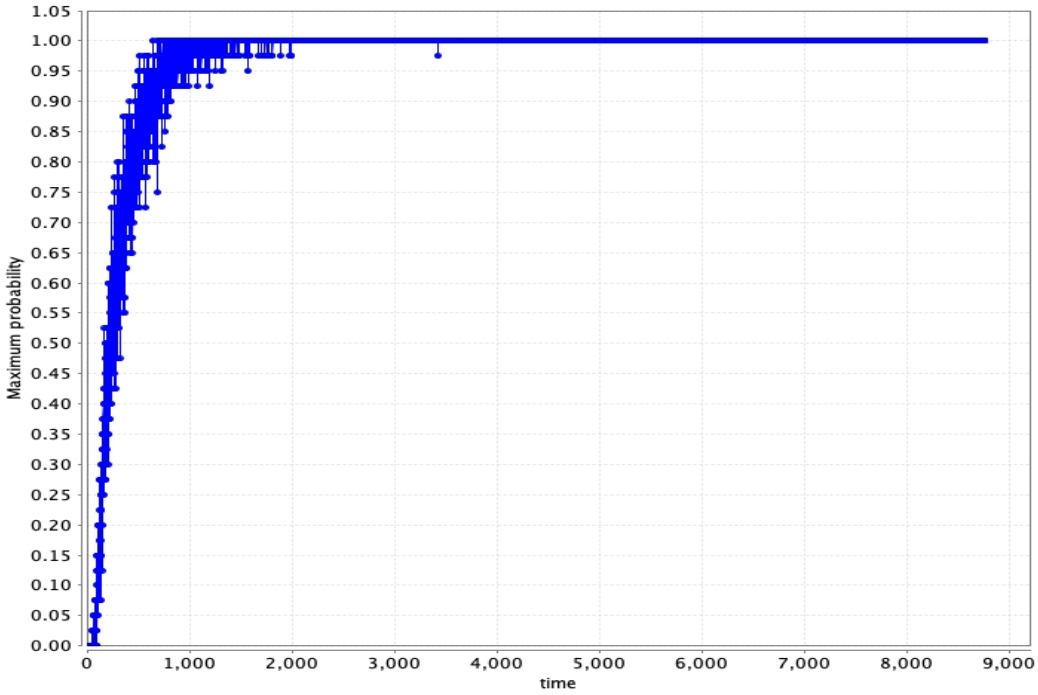

Figure 6. Maximum probability of fault/failure of the PBN model (no learning).

For these experiments, time is expressed in hours. In the graphs, the y-axis represents the maximum probability of occurrence of a fault or failure of the modeled system. The experiments simulated the system running for one year (8760 h) to illustrate how the probability of failure increased without intervention in the system, in the form of maintenance. Maintenance can be reactive, proactive/calendarized (preventive), or predictive, intervening only when the system components require maintenance based on their condition, instead of relying on a calendar-based approach. In [8], maintenance was equated to the perturbation of a system, specifically to the mechanism of intervention in PBNs [35,36]. Preventive maintenance (PM) is a calendarized intervention that prevents the normal degradation of the reliability of a system. When examining the experiments performed, we observed that preventive maintenance delayed (but did not prevent) the failure of the system. Intervention in the form of PM or corrective maintenance involves perturbing the system after a certain (calendarized) period, through which the probability of a component's failure is altered. On the other hand, predictive maintenance (PdM) involves intervening in the system only when it is necessary or when the condition of the modeled component requires it. The aim is to intervene not based on a calendarized set of maintenance activities but only when the condition of the system requires it. The aim of PdM is to maintain a system's constant normal operating state by predicting the faults and failures of an asset, based on a set of parameters; once predicted, those responsible for the system's wellbeing can take the necessary steps to prevent these failures. The architecture proposed in this study can adjust the predictor functions of the PBN and determine the optimal set of predictors to allow the system operators to model the system accurately and use predictive analysis to study its evolution, guiding it and preventing failure. Prevention of such failures has several benefits, including reduced maintenance costs, fewer failures, reduced downtime, increased machine lifetime, and better safety, among others. This method can also be used to study the repercussions of certain unhealthy states. Furthermore, the user is able to see the repercussions and effects on the system of certain states that are part of the basin of attraction of a failure, such as system states where certain parts are operational but others with less priority remain at fault.

In Figure 6, the maximum probability of occurrence of a fault/failure is at its maximum at 638 h, with the probability reaching 100% at that time.

Figure 7 illustrates the maximum probability of occurrence of a fault/failure in the Learning PBN model.

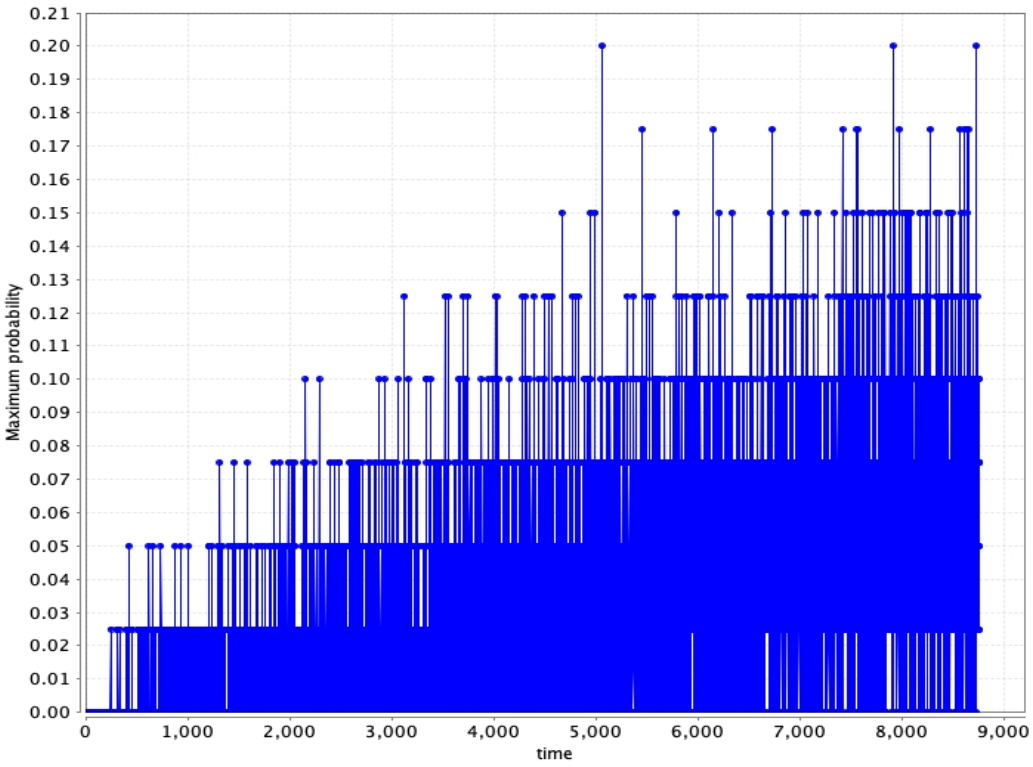

Figure 7. Maximum probability of fault/failure of the proposed Learning PBN model.

In this experiment, the maximum probability of occurrence of a fault or failure was found to be 20%, occurring three times in a year of operation, the first being after 5060 h. In our previous experiments, we evaluated the performance of the learning method regarding the equality and negation functions and learning to maintain an "all OK" state (normal operation), equivalent to intervening in the system whenever a fault or failure was detected, in order to maintain the normal operational mode of the system. It was evident that the Learning PBN significantly lowered the maximum probability of faults and failures occurring in a year of operation, from 100% after a year of operation to a maximum of 20% after a year.

4. Conclusions and Future Work

The novel architecture presented in this study is a significant improvement from the previous uses of PBNs in engineering modeling. In previous studies, PBNs were used for modeling; the proposed model can now learn without external control, to maintain a certain set of inputs. This expands the capacity to consistently increase the model's performance for its intended purpose, improving the predictive capacity of existing PBN models by adjusting the predictors throughout the learning model. However, in contrast to other learning techniques used in this area, the learning task in a PBN is not to change weights through backpropagation but to adjust the probabilities of the predictors affecting the network's transitions. Some possible advantages to the use of PBNs as learning tools are that positive feedback, like in the case of PBNs, may result in faster learning in some cases, and implementation in hardware, such as an ASIC, might be possible using PBNs. Negative feedback systems are sensitive to their training data, require more training information, and are less tolerant of noise and irregularity [37]. These initial experiments show that the network does learn the equality function, but also present new questions and opportunities for research. More work needs to be carried out in the selection of the

probabilities and in studying the effects of the selection of the parameters δ and $\overline{n}$. There is also a lot of work to be conducted in studying how the structure of the network, synchronicity, heterogeneity [38,39], and other factors affect this learning as well as attractor selection. Also, considering that the control mechanisms employed produced differences in performance, this initial study represents a first step in continued exploration. Currently, one of the main limitations we have is the development environment of the architecture. PRISM permits many interesting analyses, but models made using PRISM are not very scalable. We are currently in the process of determining which other programming environment may give us similar tools to those we have available in PRISM, but with the capacity to process more nodes. For this to become a more practical implementation, it would have to be able to address processing complexity and many nodes. PRISM has its own shortcomings in terms of parallel processing and memory management, but it is not meant to be the platform for a finished product. Also, since this approach is model-based, it has the same limitations as any other model-based fault detection technique.

Although we do not propose that PBNs are equivalent to ANNs, we do believe that these results merit further studies into PBNs and their learning capabilities, perhaps to derive an equivalency between them. The goal of this research is to present examples of practical PBN-modeled systems that demonstrate learning, so that we can understand them further and derive more theoretical conclusions, with the future aim of using PBNs as a learning tool and having an alternative for machine learning applications.

As part of possible future work, we can employ PBN models in the study of engineering systems under maintenance, specifically those under predictive maintenance. Models of these systems that incorporate learning can be used to improve maintenance strategies, allowing a reduction in the probability of faults and failures occurring in a system. The models can help to optimize maintenance schedules, condition monitoring of system elements, and the use of resources, possibly lowering the cost of system operation. Better models can be constructed through this method, as the architecture can start with unbiased predictors and optimize the evolution of the network by learning to change the probability of occurrence of the different predictors to sustain a learned state. The models that result from this exercise will be significantly improved by this architecture.

Author Contributions: Conceptualization, P.J.R.T., C.C., O.L.S. and C.G.G.; methodology, P.J.R.T., C.C. and J.M.-A.; software, P.J.R.T. and C.C.; validation, S.R.G., J.P.T. and S.K.I.; writing—original draft preparation, P.J.R.T., C.C., O.L.S. and J.M.-A.; writing—review and editing, S.R.G. and J.P.T.; supervision, S.R.G., J.P.T., S.K.I. and C.G.G.; project administration, P.J.R.T.; funding acquisition, P.J.R.T., S.R.G. and J.P.T. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the European Union's Horizon 2020 research and innovation program under the Maria Skłodowska-Curie grant agreement No. 101034371.

Data Availability Statement: Data are unavailable due to third party privacy restrictions.

Acknowledgments: We would like to acknowledge Dr. Yaniv Proselkov for his insight and comments on the original version of this research work.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Shmulevich, I.; Dougherty, E.R. *Probabilistic Boolean Networks: Modeling and Control of Gene Regulatory Networks*; SIAM: Philadelphia, PA, USA, 2010.
2. Kauffman, S.A. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.* **1969**, *22*, 437–467. [CrossRef] [PubMed]
3. Kauffman, S.A. Homeostasis and differentiation in random genetic control networks. *Nature* **1969**, *224*, 177–178. [CrossRef] [PubMed]

4. Arnosti, D.N.; Ay, A. Boolean modeling of gene regulatory networks: Driesch redux. *Proc. Natl. Acad. Sci. USA* **2012**, *109*, 18239–18240. [CrossRef] [PubMed]
5. Papagiannis, G.; Moschoyiannis, S. Deep Reinforcement Learning for Control of Probabilistic Boolean Networks. In *Complex Networks and their Applications IX*; Studies in Computational Intelligence; Benito, R.M., Cherifi, C., Cherifi, H., Moro, E., Rocha, L.M., Sales-Pardo, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; Volume 944.
6. Rivera Torres, P.J.; Serrano Mercado, E.I.; Anido Rifón, L. Probabilistic Boolean Network modeling of an industrial machine. *J. Intell. Manuf.* **2018**, *29*, 875–890. [CrossRef]
7. Rivera Torres, P.J.; Serrano Mercado, E.I.; Anido Rifón, L. Probabilistic Boolean Network modeling and model checking as an approach for DFMEA for manufacturing systems. *J. Intell. Manuf.* **2018**, *29*, 1393–1413. [CrossRef]
8. Rivera Torres, P.J.; Serrano Mercado, E.I.; Llanes Santiago, O.; Anido Rifón, L. Modeling preventive maintenance of manufacturing processes with Probabilistic Boolean Networks with interventions. *J. Intell. Manuf.* **2018**, *29*, 1941–1952. [CrossRef]
9. Rivera Torres, P.J.; Gershenson García, C.; Sánchez Puig, M.F.; Kanaan Izquierdo, S. Reinforcement Learning with Probabilistic Boolean Networks. In Proceedings of the 19th Latin American Control Congress, La Habana, Cuba, 28 November 2022; Springer: Cham, Switzerland, 2022; Volume 2022, p. 15.
10. Rivera Torres, P.J.; Gershenson García, C.; Sánchez Puig, M.F.; Kanaan Izquierdo, S. Reinforcement Learning with Probabilistic Boolean Networks. *Complexity* **2022**, *2022*, 3652441. [CrossRef]
11. Omol, E.; Mburu, L.; Abuonji, P.; Onyango, D. Anomaly Detection in IoT Sensor Data Using Machine Learning Techniques For Predictive Maintenance In Smart Grids. *Int. J. Sci. Technol. Manag.* **2024**, *5*, 201–210. [CrossRef]
12. Goyal, H.R.; Almusawi, M.; Otero-Potosi, S.; Varshney, N.; Sharma, V.; Rao, A.K. Predictive Maintenance in Smart Grids with Long Short-Term Memory Networks (LSTM). In Proceedings of the 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE), Gautam Buddha Nagar, India, 9–11 May 2024; pp. 1370–1375. [CrossRef]
13. Jiang, H.; Zhang, J.J.; Gao, W.; Wu, Z. Fault Detection, Identification, and Location in Smart Grid Based on Data-Driven Computational Methods. *IEEE Trans. Smart Grid* **2014**, *5*, 2947–2956. [CrossRef]
14. Li, Q.; Deng, Y.; Liu, X.; Sun, W.; Li, W.; Li, J.; Liu, Z. Autonomous Smart Grid Fault Detection. *IEEE Commun. Stand. Mag.* **2023**, *7*, 40–47. [CrossRef]
15. Labrador Rivas, A.E.; Abrão, T. Faults in smart grid systems: Monitoring, detection and classification. *Electr. Power Syst. Res.* **2020**, *189*, 106602. [CrossRef]
16. Andresen, C.A.; Torsæter, B.N.; Haugdal, H.; Uhlen, K. Fault Detection and Prediction in Smart Grids. In Proceedings of the 2018 IEEE 9th International Workshop on Applied Measurements for Power Systems (AMPS), Bologna, Italy, 26–28 September 2018; pp. 1–6. [CrossRef]
17. Mahmoud, M.A.; Md Nasir, N.R.; Gurunathan, M.; Raj, P.; Mostafa, S.A. The Current State of the Art in Research on Predictive Maintenance in Smart Grid Distribution Network: Fault's Types, Causes, and Prediction Methods—A Systematic Review. *Energies* **2021**, *14*, 5078. [CrossRef]
18. Shmulevich, I.; Dougherty, E.; Kim, S. Probabilistic Boolean Networks: A Rule-Based Uncertainty Model for Gene Regulatory Networks. *Bioinformatics* **2002**, *18*, 261–274. [CrossRef] [PubMed]
19. Shalev-Shwartz, S.; Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*; Cambridge University Press: Cambridge, UK, 2014.
20. Cunningham, P.; Cord, M.; Delany, S.J. Supervised Learning, in Machine Learning Techniques for Multimedia. In *Cognitive Technologies*; Cord, M., Cunningham, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2008.
21. Greene, D.; Cunningham, P.; Mayer, R. Unsupervised Learning and Clustering, in Machine Learning Techniques for Multimedia. In *Cognitive Technologies*; Cord, M., Cunningham, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2008.
22. Zhu, X. Semi-Supervised Learning. In *Encyclopedia of Machine Learning*; Sammut, C., Webb, G.I., Eds.; Springer: Berlin/Heidelberg, Germany, 2011.
23. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Boston, MA, USA, 2018.
24. Holland, J.H. *Adaptation in Natural and Artificial Systems*; The University of Michigan Press: Ann Arbor, MI, USA, 1975.
25. Dorigo, M. Optimization, Learning, and Natural Algorithms. PM-AI & R Project. Ph.D. Dissertation, Politecnico di Milano, Milano, Italy, 1992.
26. Nunes da Silva, I.; Spatti, D.H.; Flauzino, R.A.; Liboni, L.H.B.; dos Reis Alves, S.F. *Artificial Neural Networks: A Practical Course*; Springer: Cham, Switzerland, 2016.
27. Yamashita, R.; Nishio, M.; Do, R.K.G.; Togashi, K. Convolutional neural networks: An overview and application in radiology. *Insights Imaging* **2018**, *9*, 611–629. [CrossRef]
28. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
29. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
30. Gershenson, C. Artificial neural networks for beginners. Sussex COGS Teaching Package. *arXiv* **2003**, arXiv:cs/0308031.
31. Irizarry-Rivera, A.; Rodríguez-Martínez, M.; Vélez, B.; Vélez-Reyes, M.; Ramírez-Orquín, A.R.; O'Neill-Carrillo, E.; Cedeño, J.R. *Intelligent Power Routers: Distributed Coordination for Electric Energy Processing Networks, Operation and Control of Electric Energy Processing Systems*; Momoh, J., Mili, L., Eds.; John Wiley and Sons: Hoboken, NJ, USA, 2010; pp. 47–85.

32. Kwiatkowska, M.Z.; Norman, G.; Parker, D. Prism 4.0: Verification of probabilistic real-time systems. *Comput. Aided Verif.* **2011**, *6806*, 585–591.
33. Kobayashi, K.; Hiraishi, K. Reachability analysis of probabilistic boolean networks using model checking. In Proceedings of the SICE Annual Conference 2010, Taipei, Taiwan, 18–21 August 2010; pp. 829–832.
34. Rivera Torres, P.J.; Serrano Mercado, E.I. Probabilistic Boolean Network Modeling as an aid for DFMEA in Manufacturing Systems. In Proceedings of the 18th Scientific Convention of Engineering and Architecture, La Habana, Cuba, 21–25 November 2016.
35. Shmulevich, I.; Dougherty, E.R. *Genomic Signal Processing*, 1st ed.; Princeton University Press: Princeton, NJ, USA, 2007; Volume 1.
36. Datta, A.; Choudhary, A.; Bittner, M.L.; Dougherty, E.R. External control in Markovian genetic regulatory networks. *Mach. Learn.* **2003**, *52*, 169–191. [CrossRef]
37. Frenkel, C.; Lefebvre, M.; Bol, D. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Front. Neurosci.* **2021**, *13*, 629892. [CrossRef]
38. Gershenson, C. Updating schemes in random Boolean networks: Do they really matter? In *Artificial Life IX Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*; Pollack, J., Bedau, M., Husbands, P., Ikegami, T., Watson, R.A., Eds.; MIT Press: Cambridge, MA, USA, 2004; pp. 238–243. [CrossRef]
39. López-Díaz, A.J.; Sánchez-Puig, M.F.; Gershenson, C. Temporal, structural, and functional heterogeneities extend criticality and antifragility in random Boolean networks. *Entropy* **2023**, *25*, 254. [CrossRef]