(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
02 January 2020 (02.01.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/005353 A1

(51) International Patent Classification:
B25J 9/16 (2006.01) *G06N 3/04* (2006.01)

(21) International Application Number:
PCT/US2019/024296

(22) International Filing Date:
27 March 2019 (27.03.2019)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/690,698 27 June 2018 (27.06.2018) US

(71) Applicant: **OHIO STATE INNOVATION FOUNDATION** [US/US]; 1524 North High Street, Columbus, Ohio 43201 (US).

(72) Inventors: **CANADAY, Daniel**; c/o Ohio State Innovation Foundation, 1524 North High Street, Columbus, Ohio 43201 (US). **GAUTHIER, Daniel**; c/o Ohio State Innovation Foundation, 1524 North High Street, Columbus, Ohio 43201 (US). **GRIFFITH, Aaron**; c/o Ohio State Innovation Foundation, 1524 North High Street, Columbus, Ohio 43201 (US).

(74) Agent: **WALDMAN, Jonathan M.** et al.; Meunier Carlin & Curfman LLC, 999 Peachtree St. NE, Suite 1300, Atlanta, Georgia 30309 (US).

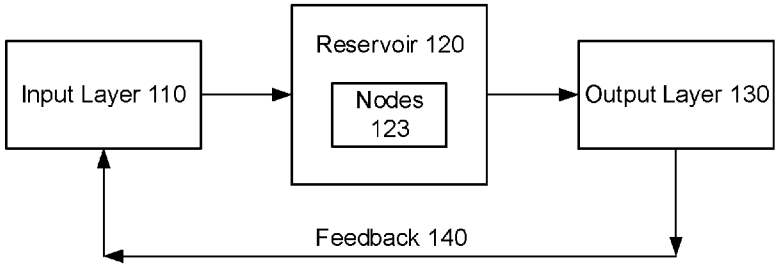
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*
— *of inventorship (Rule 4.17(iv))*

(54) **Title:** RAPID TIME-SERIES PREDICTION WITH HARDWARE-BASED RESERVOIR COMPUTER



100

FIG. 1

(57) **Abstract:** Reservoir computing systems and methods provide rapid processing speed by the reservoir and by the output layer. A hardware implementation of reservoir computing is based on an autonomous, time-delay, Boolean network realized on a readily-available platform known as a field-programmable gate array (FPGA). This approach allows for a seamless coupling of the reservoir to the output layer due to the spatially simple nature of the reservoir state and because matrix multiplication of a Boolean vector can be realized with compact Boolean logic. Embodiments may be used to predict the behavior of a chaotic dynamical system.

WO 2020/005353 A1

[Continued on next page]

Published:
— *with international search report (Art. 21(3))*

RAPID TIME-SERIES PREDICTION WITH HARDWARE-BASED RESERVOIR COMPUTER

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. provisional patent application No. 62/690,698, filed on June 27, 2018, and entitled "RAPID TIME-SERIES PREDICTION WITH AN FPGA-BASED RESERVOIR COMPUTER," the disclosure of which is expressly incorporated herein by reference in its entirety.

BACKGROUND

[0002] There is interest in the machine learning community in using recurrent neural networks (RNNs) for processing time-dependent signals. Under some mild assumptions, these types of networks are universal approximators of dynamical systems, similarly to how multilayer feedforward neural networks are universal approximators of static maps. Many machine learning and artificial intelligence tasks, such as dynamical system modeling, human speech recognition, and natural language processing are intrinsically time-dependent tasks, and thus are more naturally handled within a time-dependent, neural-network framework.

[0003] Though they have high expressive power, RNNs are difficult to train using gradient-descent-based methods. One approach to efficiently and rapidly train an RNN is known as reservoir computing (RC). Reservoir computing is a neural network approach for processing time-dependent signals that has seen rapid development in recent years. In RC, the network is divided into input nodes, a bulk collection of nodes known as the reservoir, and output nodes, such that the only recurrent links are between reservoir nodes. Training involves only adjusting the weights along links connecting the reservoir to the output nodes and not the recurrent links in the reservoir. This approach displays state-of-the-art performance in a variety of time-dependent tasks, including chaotic time-series prediction, system identification and control, and spoken word recognition, all with short training times in comparison to other neural-network approaches.

[0004] Thus, reservoir computers are well-suited for machine learning tasks that involve processing time-varying signals such as those generated by human speech, communication systems, chaotic systems, weather systems, and autonomous vehicles. Compared to other neural network techniques, reservoir computers can be trained using less data and in much less time. They also possess a large network component, called the reservoir, that can be re-used for different tasks.

[0005] Physical implementations of the RC technique using optical reservoirs have demonstrated high accuracy and processing speed at benchmark tasks. Recent implementations of reservoir computing using dedicated hardware have achieved much attention, particularly those based on delay-coupled photonic systems. These devices allow for reservoir computing at extremely high speeds, including the classification of spoken words at a rate of millions of words per second. There is also the potential to form the input and output layers using optics as well, resulting in an all-optical computational device. However, these devices are not well-equipped to handle tasks such as time-series prediction, which require the input and output layers to be coupled, where the output is fed back into the reservoir. For example, these approaches require rearrangement of the data injected into the reservoir, which causes delays in information processing. Often, this masking and the subsequent output-layer processing is done using an electronic device to maintain high performance, which limits their use in tasks such as time-series prediction at high speed.

SUMMARY

[0006] Reservoir computing systems and methods provide rapid processing speed by the reservoir and by the output layer. A hardware implementation of reservoir computing is based on an autonomous, time-delay, Boolean network realized on a readily-available platform known as a field-programmable gate array (FPGA). This approach allows for a seamless coupling of the reservoir to the output layer due to the spatially simple nature of the reservoir state and because matrix multiplication of a Boolean vector can be realized with compact Boolean logic. Embodiments may be used to predict the behavior of a chaotic dynamical system, as well as training the reservoir to learn the short-term behavior and the long-term behavior of a chaotic system. Additionally, the fading memory property is satisfied by using low connectivity and threshold nodes.

[0007] In an implementation, a reservoir computing device is provided. The reservoir computing device includes an input layer configured to receive at least one input signal; a reservoir comprising a plurality of nodes, wherein at least one node of the plurality of nodes is coupled to the input layer to receive the at least one input signal, wherein the reservoir comprises an autonomous time-delay Boolean network; and an output layer configured to output at least one output signal, wherein at least another node of the plurality of nodes is coupled to the output layer, wherein the input layer, the reservoir, and the output layer are comprised in integrated circuitry.

[0008] In an implementation, a reservoir computing device is provided. The reservoir computing device includes a reservoir comprising an autonomous time-delay Boolean network of

nodes; and a controller configured to provide input data to the reservoir and receive output data from the reservoir, wherein the reservoir and the controller are comprised in integrated circuitry and configured to provide time-series prediction.

[0009] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the detailed description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The foregoing summary, as well as the following detailed description of illustrative embodiments, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the embodiments, there is shown in the drawings example constructions of the embodiments; however, the embodiments are not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0011] FIG. 1 is block diagram of an implementation of a reservoir computing device;

[0012] FIG. 2 is an illustration of an example reservoir computing device;

[0013] FIG. 3 is an operational flow of an implementation of a method of reservoir computing;

[0014] FIG. 4 is an illustration of another example reservoir computing device;

[0015] FIG. 5 is an operational flow of another implementation of a method of reservoir computing;

[0016] FIG. 6 is block diagram of another implementation of a reservoir computing device;

[0017] FIG. 7 is an illustration of a lookup table useful in explaining embodiments;

[0018] FIG. 8 is an illustration of example hardware description language for an implementation of a node;

[0019] FIG. 9 is an illustration of example hardware description language for an implementation of a delay line;

[0020] FIG. 10 is an illustration of example hardware description language describing an implementation of a reservoir;

[0021] FIG. 11 is an illustration of example hardware description language describing an implementation of a reservoir computer; and

[0022] FIG. 12 shows an exemplary computing environment in which example embodiments and aspects may be implemented.

DETAILED DESCRIPTION

[0023] This description provides examples not intended to limit the scope of the appended claims. The figures generally indicate the features of the examples, where it is understood and appreciated that like reference numerals are used to refer to like elements. Reference in the specification to "one embodiment" or "an embodiment" or "an example embodiment" means that a particular feature, structure, or characteristic described is included in at least one embodiment described herein and does not imply that the feature, structure, or characteristic is present in all embodiments described herein.

[0024] A reservoir computing device comprising an autonomous, time-delay, Boolean network configured on a field-programmable gate array (FPGA) is described. Such devices allow for complex networks consisting of thousands of nodes with an arbitrary network topology. Time-delays can be incorporated along network links, thereby allowing for extremely high-dimension reservoirs. The characteristic time scale of a network node is less than a nanosecond, allowing for information processing in the GHz regime. Furthermore, because the reservoir state is Boolean rather than real-valued, the calculation of an output from the reservoir state can be done rapidly with synchronous FPGA logic. Such a reservoir computing device may be used to forecast the dynamics of a chaotic system.

[0025] The logic elements in the reservoir are autonomous, meaning that the logic is not regulated by a global clock. When the logic is clocked, a software simulation of a reservoir computer is effectively obtained. The autonomous logic has a potential to go beyond what any standard computer simulation can do. Moreover, delay elements are incorporated between the network nodes in the reservoir to match the time scale of the reservoir with the time scale of the data injected into the reservoir. This is relevant for autonomous nodes.

[0026] As described further herein, a neural network technique known as reservoir computing is used. In this technique, a mapping from a time-dependent input signal u(t) to a desired output signal $v_d(t)$ is formed by (1) creating a randomly connected, recurrent network of nodes, (2) exciting the network with u(t) over some training period, and (3) training a readout layer that maps the network state $v_d(t)$. Reservoir computing demonstrates state-of-the-art performance at a number of time-dependent machine learning tasks, such as time-series prediction. Additionally, training can be done in seconds to minutes, compared to days with deep-learning techniques.

[0027] FIG. 1 is block diagram of an implementation of a reservoir computing device 100. The reservoir computing device 100 comprises an input layer 110, a reservoir 120, an output

layer 130, and feedback 140. The input layer 110 provides one or more input signals (e.g., u(t)) to the reservoir 120. The input signals can be weighted using values determined during training of the reservoir computing device 100. The input layer 110 may comprise a plurality of input channels that carry input signals.

[0028] The reservoir 120 may be a recurrent artificial neural network comprising a plurality of nodes 123. The reservoir 120 may contain interconnections that couple a pair of the nodes 123 together in the reservoir 120, such that one of the nodes 123 provides its output as an input to another of the nodes 123. Each of the nodes 123 may be weighted with a real-valued weight. The nodes 123 in the reservoir 120 may implement one or more logic gates, such as Boolean logic gates, to perform various operations on input signals from the input layer 110. The input layer 110 may be coupled to some or all of the nodes 123 (e.g., an input node subset of the nodes 123) depending on the implementation. Results from the nodes 123 may be provided from the reservoir 120 to the output layer 130. The output layer 130 may be coupled to some or all of the nodes 123 (e.g., an output node subset of the nodes 123). According to some aspects, the reservoir 120 may be implemented in integrated circuitry, such as an FPGA.

[0029] In an embodiment, the reservoir 120 is realized by an autonomous, time-delay, Boolean network configured on an FPGA. Together with the parallel nature of the network, this allows for up to ten times faster information processing than delay-coupled photonic devices.

[0030] The output layer 130 may receive output signals from the reservoir 120. The output layer 130 may comprise a plurality of output channels that carry output signals. Weights may be added to the output signals in the reservoir 120 before being provided to the output layer 130 (e.g., as $v_d(t)$). The weights may be determined during training of the reservoir computing device 100. Weights may also be applied to the input signals of the input layer 110 before being provided to the reservoir 120.

[0031] The feedback 140 may be comprised of feedback circuitry and/or feedback operations in which the output signal of the device 100 (i.e., the output of the output layer 130) is sent back to the input layer 110 to create feedback within the reservoir 120.

[0032] FIG. 2 is an illustration of an example reservoir computing device 200, and FIG. 3 is an operational flow of an implementation of a method 300 of reservoir computing. The device 200 comprises an input node 210 (or input layer), a reservoir 220 comprising a plurality of nodes 224, and an output node 230 (or output layer). Also shown are a plurality of links 225 between various ones of the input node 210, the nodes 224, and the output node 230. Given an input signal

$u(t)$ at the input node 210, and a desired output signal $v_d(t)$ at the output node 230, a reservoir computer constructs a mapping from $u(t)$ to $v_d(t)$ with the following steps.

[0033] At 310, create a randomly parameterized network of nodes and recurrent links called the reservoir with state $X(t)$ and dynamics described by $\dot{X}(t)=f[X(t),u(t)]$. At 320, excite the reservoir with an input signal $u(t)$ over some training period and observe the response of the reservoir. At 330, form a readout layer that transforms the reservoir state $X(t)$ to an output $v(t)$, such that $v(t)$ well approximates $v_d(t)$ during the training period.

[0034] No assumptions are made about the dynamics f. In general, it may include discontinuities, time-delays, or have components simply equal to $u(t)$ (i.e., the reservoir 220 may include a direct connection from the input 210 to the output 230).

[0035] Thus, in FIG. 2, a general reservoir computer learns to map an input onto a desired output. The network dynamics may contain propagation delays along the links (denoted by τ_{ij}) or through nodes (such as through the output layer, denoted by τ_{out}).

[0036] Reservoir computing demonstrates remarkable success at predicting a chaotic time-series, among other applications. The goal of this task is to predict the output of an unknown dynamical system after a training period. In the context of RC, this is accomplished by setting $v_d(t) = u(t)$, i.e., by training the reservoir computer to reproduce its inputs, and then allowing the newly-formed autonomous system to evolve in time beyond the end of the training period.

[0037] FIG. 4 is an illustration of another example reservoir computing device 400. This closed-loop system is illustrated in FIG. 4 and consists of the same components as in FIG. 2, but the input and output are the same signal. This technique can predict accurately the short-term behavior of a variety of systems, including the Mackey-Glass, Lorenz, and Kuramoto-Sivashinsky spatial-temporal systems using a software simulation of the reservoir. A reservoir computer trained in this manner can also learn the long-term behavior of complex systems, generating the true attractor of the target system and replicating its Lyapunov spectrum.

[0038] FIG. 5 is an operational flow of another implementation of a method 500 of reservoir computing. At 510, similar to 310, create a randomly parameterized network of nodes and recurrent links called the reservoir with state $X(t)$ and dynamics described by $\dot{X}(t)=f[X(t),u(t)]$. At 520, the reservoir computer is trained to reproduce its inputs. Thus, for the particular task of predicting a signal, the reservoir is trained so that the target output is equal to the input. At 530, after training is complete, feedback is provided so that the output is fed back as the input. Thus, after training, the output is fed back into the reservoir, resulting in an

autonomous dynamical system. If properly trained, the autonomous reservoir serves as a model for the dynamics that generated the input signal.

[0039] FIG. 6 is block diagram of another implementation of a reservoir computing device 600. The reservoir computing device 600 can receive an input 605, such as input u(t) from a memory 620 or a computing device such as the computing device 1200 described with respect to FIG. 12. Depending on the implementation, the memory 620 may be comprised within, or in communication with, the reservoir computing device 600, comprised within the computing device 1200, or other suitable memory or storage device. In an embodiment, the device 600 may comprise an FPGA, with each component of the device 600 being implemented in the FPGA, although this is not intended to be limiting, as other implementations are contemplated, such as an Application Specific Integrated Circuit (ASIC), for example.

[0040] In an implementation, a controller 610 may store data to and/or retrieve data from the memory 620. The data may include the input 605, an output 655, and node data of a reservoir 630. Data associated with testing and training may also be provided to and from the controller 610 to and from a tester 640 and a trainer 650, respectively. The controller 620 may be configured to apply weighting to the input 605 and/or the output prior to being provided as the output 655. The weightings may be generated by a weighting module 660, provided to the controller 610, and applied to the various signals by the controller 610.

[0041] The reservoir 630 may process the input 605 and generate the output 655. In some embodiments, output from the reservoir 630 may be weighted by the controller 610. The controller 610 may then provide this weighted output of the reservoir 630 as the output 655.

[0042] Although training the network consists only of identifying optimal parameters in the readout layer, several factors may be considered in designing the reservoir. These factors are now described.

[0043] In general, both the reservoir and the source of the signal u(t) are dynamical systems with their own characteristic time scales. These time scales are similar for the reservoir to produce v(t). For software-based approaches to RC, these scales are matched by tuning the reservoir's temporal properties through accessible reservoir parameters, such as the response time τ_{node} (e.g., see FIG. 2) of reservoir nodes. However, with hardware-based approaches, the parameters controlling the time scale of reservoir dynamics are often more rigid. This may be compensated for by adjusting the time scale of the input signal and/or adding delays to the links within the reservoir.

[0044] It has been determined that an effective reservoir for RC is a system that possesses fading memory. That is, the reservoir state contains information about the input signal u(t), but the effect of small differences in u(t) dissipates over time. This is often referred to as the echo-state property. The autonomous reservoirs disclosed herein have the fading memory property. Furthermore, the characteristic time scale over which small differences dissipate can be tuned by adding delays to the links within the reservoir.

[0045] Each RC implementation couples u(t) to the reservoir in a technique-dependent way, such as spike-encoding in liquid state machines (LSMs) or by consideration of so-called "virtual nodes" in photonic reservoirs. The coupling in the FPGA-based approach described herein is complicated by the fact that nodes execute Boolean functions, whereas the input signal u(t) is a large-bit representation of a real number. Consider, as with most techniques for processing physical data, the limited precision and sampling rate of the input signal. The sampling rate is particularly relevant for a physical reservoir computer, as the reservoir nodes have their own, fixed characteristic time scale.

[0046] In software-based reservoir computing schemes, the readout layer performs its operation effectively instantaneously as far as the simulation is concerned. However, this is not possible when the reservoir is a continuously evolving physical system. There is a finite time required to calculate v(t), which can be interpreted as a propagation delay τ_{out} (e.g., see FIG. 2) through the readout layer and ultimately limits the rate at which predictions can be made in closed-loop operation. Consequently, v(t) is calculated from a measurement of $X(t - \tau_{out})$ for the predicted output to be ready to be fed back into the input at time t.

[0047] The hardware implementation of RC described herein provides a minimal output delay so that predictions can be made as rapidly as possible.

[0048] In an embodiment, an autonomous Boolean reservoir is used. More particularly, the reservoir is an autonomous, time-delay, Boolean reservoir realized on an FPGA. By forming the nodes of the reservoir from FPGA elements themselves, this approach exhibits faster computation than FPGA-accelerated neural networks which require explicit multiplication, addition, and non-linear transformation calculations at each time-step. This implementation also has the advantage of realizing the reservoir and the readout layer on the same platform without delays associated with transferring data between different hardware. Additionally, due to the Boolean-valued state of the reservoir, a linear readout layer, $v(t) = W_{out}X(t)$, is reduced to an addition of real numbers rather than a full matrix multiplication. This allows for much shorter total calculation time and thus faster real-time prediction than in opto-electronic RC.

[0049] It is noted that Boolean networks with time-delay can exhibit complex dynamics, including chaos. In fact, a single XOR node with delayed feedback can exhibit a fading memory condition and is suitable for RC on simple tasks such as binary pattern recognition. It has been proposed that individual FPGA nodes have dynamics that can be described by the Glass model given by

$$\gamma_i \dot{x}_i = -x_i + \Lambda_i(X_{i1}, X_{i2}, \ldots) \quad (1)$$

$$X_i = 1 \text{ if } x_i \geq q_i; \ 0 \text{ if } x_i < q_i \quad (2)$$

where x_i is the continuous variable describing the state of the node, γ_i describes the time scale of the node, q_i is a thresholding variable, and Λ_i is the Boolean function assigned to the node. The thresholded Boolean variable X_{ij} is the j-th input to the i-th node.

[0050] The Boolean reservoir is constructed by forming networks of nodes described by Eqs. (1) and (2) and the Boolean function

$$\Lambda_i = \Theta\left(\sum_j W^{ij} X_j + W_{in}^{ij} u_j\right) \quad (3)$$

where u_j are the bits of the input vector u, W is the reservoir-reservoir connection matrix, W_{in} is the input-reservoir connection matrix, and Θ is the Heaviside step function defined by

$$\Theta(x) = 1 \text{ if } x > 0; \ 0 \text{ if } x \leq 0 \quad (4).$$

[0051] The matrices W and W_{in} are chosen as follows. Each node receives the input from exactly k other randomly chosen nodes, thus determining k non-zero elements of each row of W. The non-zero elements of W are given a random value from a uniform distribution between −1 and 1. The maximum absolute eigenvalue (spectral radius) of the matrix W is calculated and used to scale W such that its spectral radius is ρ. A proportion σ of the nodes are chosen to receive input, thus determining the number of non-zero rows of W_{in}. The nonzero values of W_{in} are chosen carefully, but note that the scale of W_{in} does not need to be tuned, as it is apparent from Eq. (3) that only the relative scale of W and W_{in} determines Λ_i.

[0052] The three parameters defined above, k, ρ, and σ, are the three hyperparameters that characterize the topology of the reservoir. The parameter $\bar{\tau}$ characterizes delays introduced along links between nodes. Together, these four hyperparameters describe the reservoirs.

[0053] The presence of the $-x_i$ term in Eq. (1) represents the sluggish response of the node, i.e., its inability to change its state instantaneously. This results in an effective propagation delay of a signal through the node. To take advantage of this phenomenon, delays are created by connecting chains of pairs of inverter gates between nodes. These inverter gates have dynamics described by Eqs. (1) and (2) and

$$\Lambda_i(X) = 0 \text{ if } X=1; \ 1 \text{ if } X=0 \quad (5).$$

[0054] Note that the propagation delay through these nodes depends both on γ_i and q_i, both of which are heterogeneous throughout the chip due to small manufacturing differences. The mean propagation delay through the inverter gates is denoted by τ_{inv}, which is measured by recording the oscillation frequencies of variously sized loops of these gates.

[0055] Exploit the propagation delays by inserting chains of pairs of inverter gates in between reservoir nodes, thus creating a time-delayed network. Set the mean delay $\bar{\tau}$ and randomly choose a delay time for each network link. This is similar to how the network topology is chosen by fixing certain hyperparameters and randomly choosing W and W_{in} subject to these parameters. The random delays are chosen from a uniform distribution between $\bar{\tau}/2$ and $3\bar{\tau}/2$ so that delays on the order of τ_{node} are avoided.

[0056] The addition of these delay chains is useful because the time scale of individual nodes is much faster than the speed at which synchronous FPGA logic can change the value of the input signal. Without any delays, it is impossible to match the time scales of the input signal with the reservoir state, resulting in poor RC performance. The time scales associated with the reservoir's fading memory are controlled by $\bar{\tau}$, thus demonstrating that the reservoir's time scales can be tuned with delay lines.

[0057] For the reservoir to learn about its input sequence, it will possess the fading memory property (although more may be required for replicating long-term behavior). Intuitively, this property implies that the reservoir state X(t) is a function of its input history, but is more strongly correlated with more recent inputs. More precisely, the fading memory property states that every reservoir state $X(t_0)$ is uniquely determined by a left-infinite input sequence $\{u(t) : t < t_0\}$.

[0058] The fading memory property is equivalent to the statement that, for any two reservoir states $X_1(t_0)$ and $X_2(t_0)$ and input signal $\{u(t) : t > t_0\}$,

$$\lim_{t \to \infty} \quad ||X_1(t) - X_2(t)||_2 = 0 \qquad (6).$$

[0059] Also of interest is the characteristic time scale over which this limit approaches zero, which may be understood as the Lyapunov exponent of the coupled reservoir-input system conditioned on the input.

[0060] Observe the fading memory property and measure the corresponding time scale with the following procedure. Prepare two input sequences $\{u_1(i\Delta t); -N \leq i \leq N\}$ and $\{u_2(i\Delta t); -N \leq i \leq N\}$, where Δt is the input sample and N is an integer such that $N\Delta t$ is sufficiently large. Each $u_1(i\Delta t)$ is drawn from a random, uniform distribution between -1 and 1. For $i \geq 0$, $u_2(i\Delta t) = u_1(i\Delta t)$. For $i < 0$, $u_2(i\Delta t)$ is drawn from a random, uniform distribution between -1 and 1. Drive

the reservoir with the first input sequence and observe the reservoir response $\{X_1(i\Delta t); -N \leq i \leq N\}$. After the reservoir is allowed to settle to its equilibrium state, drive it with the second input sequence and observe $\{X_2(i\Delta t); -N \leq i \leq N\}$. The reservoir is perturbed to effectively random reservoir states $X_1(0)$ and $X_2(0)$, because the input sequences are unequal for $i < 0$. For $i \geq 0$, the input sequences are equal, and the difference in Eq. (6) can be calculated.

[0061] For a given reservoir, this procedure is repeated 100 times with different input sequences. For each pair of sequences, the state difference is fit to $\exp(-t/\lambda)$, and the λ's are averaged over all 100 sequences. λ is referred to as the reservoir's decay time, and has a linear dependence on hyperparameter $\bar{\tau}$, with the relationship being approximately linear for fixed k, ρ, and σ. This is consistent with $\bar{\tau}$ being the dominate time scale of the reservoir rather than τ_{node}, which is a motivation for including delay lines in the reservoir construction.

[0062] The reservoir implementation is an autonomous system without a global clock, allowing for continuously evolving dynamics. However, the input layer is a synchronous FPGA design that sets the state of the input signal u(t). Prior to operation, a sequence of values for u(t) is stored in the FPGA memory blocks. During the training period, the input layer sequentially changes the state of the input signal according to the stored values.

[0063] For the prediction task, the stored values of u(t) are observations of some time-series from $t = -T_{train}$ to $t = 0$. This signal maybe defined on the entire real interval $[-T_{train}, 0]$, but only a finite sampling may be stored in the FPGA memory and presented as the input to the reservoir. The signal may also take real values, but only a finite resolution at each sampling interval may be stored. The actual input signal u(t) is thus discretized in two ways: u(t) is held constant along intervals of length t_{sample}, and u(t) is approximated by an n−bit representation of real numbers.

[0064] It is noted that t_{sample} may be no smaller than the minimum time in which the clocked FPGA logic can change the state of the input signal. However, it has been determined that t_{sample} is preferably greater than or equal to τ_{out}.

[0065] The Boolean functions described by Eqs. (3) and (4) are defined according to Boolean values u_j, which are the bits in the n-bit representation of the input signal. If the elements of W_{in} are drawn randomly from a single distribution, then the reservoir state is as much affected by the least significant bit of u(t) as it is the most significant. This leads to the reservoir state being distracted by small differences in the input signal and fails to produce a working reservoir computer.

[0066] For a scalar input u(t), correct this shortcoming by choosing the rows of W_{in} such that

$$\sum_j W_{in}^{i,j} u_j \approx \widetilde{W}_{in}^{i} u \qquad (7)$$

where $\widetilde{W}_{in}$ is an effective input matrix with non-zero values drawn randomly between 1 and −1. The relationship is approximate in the sense that u is a real-number and u_j is a binary representation of that number. For the two complement representations, this is done by choosing

$$W_{in}^{i,j} = -2^{(n-1)}\widetilde{W}_{in}^{i} \text{ if j = n; else } = +2^{(j-1)}\widetilde{W}_{in}^{i} \qquad (8).$$

[0067] A disadvantage of such an implementation is that every bit in the representation of u must go to every node in the reservoir. If a node has k recurrent connections, then it must execute a n+k to 1 Boolean function, as can be seen from Eq. (3). Boolean functions with more inputs take more FPGA resources to realize in hardware, and it takes more time for a compiler to simplify the function. It has been determined that an 8-bit representation of u is sufficient for the prediction task described here while maintaining achievable networks.

[0068] Similar to the input layer, the output layer is constructed from synchronous FPGA logic. Its function is to observe the reservoir state and, based on a learned output matrix W_{out}, produce the output v(t). This operation uses a time τ_{out} that is considered to be a propagation delay through the output layer and, as such, v(t) is calculated from $X(t - \tau_{out})$.

[0069] For the time-series prediction task, the desired reservoir output $v_d(t)$ is just u(t). The input signal is discretized both in time and in precision. Thus, v(t) is discretized in the same fashion. Note that because the reservoir state X(t) is Boolean valued, a linear transformation W_{out} of the reservoir state is equivalent to a partial sum of the weights W_{out}, where W_{out}^{i} is included in the sum only if $X_i(t) = 1$.

[0070] The inclusion of a direct connection greatly improves prediction performance. Though this involves a multiplication of 8-bit numbers, it only slightly increases τ_{out} because this multiplication can be done in parallel with the calculation of the addition of the Boolean reservoir state.

[0071] With the above considerations in mind, the output layer is constructed as follows: on the rising edge of a global clock with period t_{global}, the reservoir state is passed to a register in the output layer. The output layer calculates $W_{out}X$ with synchronous logic and in one clock cycle, where the weights W_{out} are stored in on-board memory blocks. The calculated output v(t) is passed to a register on the edge of the global clock. If t > 0, i.e., if the training period has ended, the input layer passes v(t) to the reservoir rather than the next stored value of u(t).

[0072] For v(t) to have the same discretized form as u(t), the global clock period t_{global} is set equal to the input period t_{sample}, which means the fastest the reservoir computer can produce predictions is once every $\max\{\tau_{out}, t_{sample}\}$. While t_{sample} is independent of the size of the reservoir and precision of the input, τ_{out} depends on both.

[0073] The reservoir computer, including the autonomous reservoir and the synchronous input and output layers, may be used to predict a chaotic time-series. To quantify the performance of the prediction algorithm, compute the normalized root-mean-square error (NRMSE) over one Lyapunov time T, where T is the inverse of the largest Lyapunov exponent. The $NRMSE_T$ is therefore defined as

$$NRMSE_T = \sqrt{\frac{\sum_{t=0}^{T}[u(t)-v(t)]^2}{T\sigma^2}} \quad (9)$$

where σ^2 is the variance of u(t).

[0074] To train the reservoir computer, the reservoir is initially driven with the stored values of u(t) and the reservoir response is recorded. This reservoir response is then transferred to a host PC. The output weights W_{out} are chosen to minimize

$$\sum_{t=-T_{train}}^{0}[u(t) - v(t)]^2 + r|W_{out}|^2 \quad (10)$$

where r is the ridge regression parameter and is included in Eq. (6) to discourage over-fitting to the training set. The value of r is chosen by leave-one-out cross validation on the training set.

[0075] An implementation of hardware description code for the reservoir nodes, delay lines, and a small reservoir is provided. The example code herein is written in Verilog (a hardware description language (HDL) used to configure and model electronic systems) and compiled using Altera's Quartus Prime software. Some parts of the hardware description language depend on the number of reservoir nodes N, the node in-degree k, and the number of bits n used to represent the input signal u(t). The hardware description language is provided herein for N = 3, k = 2, and n = 1, but generalizations are straightforward.

[0076] As discussed further herein, reservoir nodes implement a Boolean function $\Lambda_i : Z_2^{k+n} \rightarrow Z_2$ of the form given in Eq. (3). Each Boolean function can be defined by a Boolean string of length 2^{k+n} that specifies the lookup-table (LUT) corresponding of the Boolean function. For example, the "and" function maps $Z_2^2 \rightarrow Z_2$ and has the LUT defined in FIG. 7. FIG. 7 is an illustration of a lookup table 700 for the "and" function that is useful in explaining embodiments. The "and" function can be specified by the Boolean string that makes up the column 710. The Boolean string that defines the "and" function is 0001 as can be seen from the column 710 of the LUT 700.

[0077] FIG. 8 is an illustration of example hardware description language 800 for an implementation of a node. FIG. 8 shows Verilog hardware description language for a generic node that can implement any 3-input Boolean function, specified by a Boolean string of length 8. The hardware description language given in FIG. 8 generates a node with Boolean function based on any LUT of length $2^3 = 8$. The module **node** is declared in line 1 with inputs **node_in** and output **node_out**. The width of **node_in** is 3 bits as specified in line 3. The parameter **lut** is declared in line 2. Note that it is initialized to some value as required by Quartus, but this value is changed whenever a node is declared within the larger hardware description language that defines the complete reservoir.

[0078] The main part of the hardware description language is within an **always @(*)** block, which creates an inferred sensitivity list and is used to create arbitrary combinational logic. Line 7 specifies that values before the colon in the proceeding lines correspond to **node_in**. The statement following the colon determines which value is assigned to **node_out**. In effect, line 8 specifies that, whenever the value of **node_in** is a 3-bit string equal to 000, the value of **node_out** is whatever the value of **lut[7]** is. For example, if an instance of the module **node** is created with parameter **lut=8'b00000001**, then the node will execute the 3 input and function.

[0079] Delay lines are created as chains of pairs of inverter gates. Such a chain of length 2m is created with the hardware description language in FIG. 9. FIG. 9 is an illustration of example hardware description language 900 for an implementation of a delay line. Fig. 9 shows Verilog hardware description language for a delay line with 2m inverter gates. Similarly to the **node** module, the **delay_line** module is declared in line 1 with the input **delay_in** and output **delay_out**. It has a parameter m which specifies the number of pairs in the chain and can be changed when calling a specific instance of **delay_line**. A number of wires are declared in line 5 and will be used as the inverter gates. Note the directive **/*synthesis keep*/**, which instructs the compiler to not simplify the module by eliminating the inverter gates. This is used because otherwise the compiler would realize that **delay_line**'s function is trivial and remove all of the inverter gates.

[0080] Lines 7–8 specify the beginning and end of the delay chain as the **delay_in** and **delay_out**, respectively. Lines 10–16 use a **generate** block to create a loop that places inverter gates in between **delay_in** and **delay_out**, resulting in a delay chain of length 2m.

[0081] The **reservoir** module is the hardware description language that creates N instances of **node** and connects them Nk instances of **delay_line**. As an illustrative example, consider a 3-node reservoir with the following parameters

$$W = \begin{bmatrix} 0.1 & 0.3 & 0 \\ -0.2 & 0 & 0.1 \\ -0.3 & 0.2 & 0 \end{bmatrix} \quad \text{(A1)}$$

$$W_{in} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.2 \end{bmatrix} \quad \text{(A2)}$$

$$\tau = \begin{bmatrix} 10 & 15 & 0 \\ 6 & 0 & 7 \\ 12 & 10 & 0 \end{bmatrix} \quad \text{(A3)}$$

and only a 1-bit representation of u(t). When u(t) and x(t) are passed into the **node** module, index such that u(t) comes first, as seen from the **reservoir** module below.

[0082] With Eqs. (3) and (A1)-(A3), the LUTs for each node can be explicitly calculated as 01111111, 0100000000, and 01001101 for nodes 1–3, respectively. The matrix $\boldsymbol{\tau}$ specifies the delays in integer multiples of $2\tau_{inv}$. A network with this specification is realized by the module **reservoir** in FIG. 10 and the **node** and **delay_in** modules described herein.

[0083] FIG. 10 is an illustration of example hardware description language 1000 describing an implementation of a reservoir (i.e., Verilog hardware description language describing a simple reservoir). The connections and LUTs are determined from Eqs. (3) and (A1)-(A3). Lines 9–11 declare three nodes. Lines 13–18 declare delay lines that connect them.

[0084] Thus, like the other modules, **reservoir** requires a module declaration, parameter declarations, and input/output declarations. Here, declare a wire **x_tau** that is the delayed reservoir state. In lines 9–11, the nodes are declared with the appropriate parameters and connections and are named **node_0**, **node_1**, and **node_2**, respectively. The six delay lines are declared and named in lines 13-18.

[0085] Synchronous components that interact with the autonomous reservoir regulate the reservoir input signal, the operation mode (training or autonomous), the calculation of the output signal, and record the reservoir state.

[0086] A **sampler** module reads data from the reservoir and a **player** module writes data into the reservoir. The details of these modules are not discussed here as they depend on the device and the application of the reservoir computer. These modules are synchronized by a global clock **clk** such that **sampler** (**player**) reads (writes) data on the rising edge of **clk**.

[0087] FIG. 11 is an illustration of example hardware description language 1100 describing an implementation of a reservoir computer. FIG. 11 contains the **reservoir** module discussed above and various synchronous components. FIG. 11 shows sample Verilog hardware description language for a high level module **reservoir_computer** containing the reservoir and synchronous components. An instance of a **sampler** module is coupled to a global clock **clk** and

outputs an m-bit wide signal **u**, a 1 bit signal **mode** that determines the mode of operation for the reservoir, and a 2m(N+1)-bit wide signal **W_out** that determines the output weight matrix. An instance of a **player** module is also coupled to a global clock **clk** and inputs an N-bit wide signal **x** and a m-bit wide signal v. Depending on how these modules are implemented, they may also be coupled to other components, such as on-board memory or other FPGA clocks.

[0088] In line 17, the state of **mode** determines whether **u** or **v** drives the reservoir. This bit is set to 1 during training and set to 0 after training to allow the reservoir to evolve autonomously. **clk** registers **x** and **v** so that **output_layer** sees a value of **x** that is constant throughout one period t_{sample} and outputs a value **v** that is constant over that same interval. The module **output_layer** performs the operation $\mathbf{W}_{out}(\mathbf{x}, \mathbf{u})$, as described above. **W_out** is a flattened array of the N+1 output weights represented by 2m bits, with the extra bits being used to avoid errors in the intermediate addition calculations.

[0089] Thus, embodiments include a reservoir computer where the recurrent neural network is an autonomous, time-delay, Boolean network. This network can be readily realized on a field-programmable gate array. This choice of neural network has the advantages that (1) the network state is Boolean and therefore the map from the network state to the desired signal can be done extremely rapidly and (2) the readout layer can be done on the same hardware as the network, eliminating delays associated with the transfer of data between hardware. These advantages allow for the prediction of a time-series at extremely high speed, such as predictions at a rate of 160 MHz on an Intel/Altera Arria® 10 FPGA. Although the Intel/Altera Arria® 10 FPGA and aspects of its associated hardware description language are described with respect to some implementations, this is not intended to be limiting, as FPGAs and other devices, as well as associated hardware description languages, from many companies and vendors are contemplated and may be used depending on the particular implementation.

[0090] Implementations have industry application where it is desired to know the future state of an unknown or a partially unknown system that evolves on a nanosecond to microsecond time scale. Examples of such systems include analyzing radio frequency signals, realizing cryptographic radio-frequency coding and decoding systems, and behavior of high-frequency nano- or micro-scale devices.

[0091] FIG. 12 shows an exemplary computing environment in which example embodiments and aspects may be implemented. The computing device environment is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality.

[0092] Numerous other general purpose or special purpose computing devices environments or configurations may be used. Examples of well known computing devices, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, network personal computers (PCs), minicomputers, mainframe computers, embedded systems, distributed computing environments that include any of the above systems or devices, and the like.

[0093] Computer-executable instructions, such as program modules, being executed by a computer may be used. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Distributed computing environments may be used where tasks are performed by remote processing devices that are linked through a communications network or other data transmission medium. In a distributed computing environment, program modules and other data may be located in both local and remote computer storage media including memory storage devices.

[0094] With reference to FIG. 12, an exemplary system for implementing aspects described herein includes a computing device, such as computing device 1200. In its most basic configuration, computing device 1200 typically includes at least one processing unit 1202 and memory 1204. Depending on the exact configuration and type of computing device, memory 1204 may be volatile (such as random access memory (RAM)), non-volatile (such as read-only memory (ROM), flash memory, etc.), or some combination of the two. This most basic configuration is illustrated in FIG. 12 by dashed line 1206.

[0095] Computing device 1200 may have additional features/functionality. For example, computing device 1200 may include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in FIG. 12 by removable storage 1208 and non-removable storage 1210.

[0096] Computing device 1200 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by the device 1200 and includes both volatile and non-volatile media, removable and non-removable media.

[0097] Computer storage media include volatile and non-volatile, and removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 1204, removable storage 1208, and non-removable storage 1210 are all examples of computer storage

media. Computer storage media include, but are not limited to, RAM, ROM, electrically erasable program read-only memory (EEPROM), flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device 1200. Any such computer storage media may be part of computing device 1200.

[0098] Computing device 1200 may contain communication connection(s) 1212 that allow the device to communicate with other devices. Computing device 1200 may also have input device(s) 1214 such as a keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 1216 such as a display, speakers, printer, etc. may also be included. All these devices are well known in the art and need not be discussed at length here.

[0099] In an implementation, a reservoir computing device comprises an input layer configured to receive at least one input signal; a reservoir comprising a plurality of nodes, wherein at least one node of the plurality of nodes is coupled to the input layer to receive the at least one input signal, wherein the reservoir comprises an autonomous time-delay Boolean network; and an output layer configured to output at least one output signal, wherein at least another node of the plurality of nodes is coupled to the output layer, wherein the input layer, the reservoir, and the output layer are comprised in integrated circuitry.

[00100] Implementations may include some or all of the following features. The integrated circuitry comprises a field-programmable gate array (FPGA). The reservoir computing device further comprises feedback from the output layer to the input layer, wherein an output signal of the output layer is an input signal to the input layer. The input layer, the reservoir, and the output layer are configured to predict a behavior of a chaotic dynamical system. The reservoir comprises a randomly parameterized network of nodes and recurrent links. The autonomous time-delay Boolean network comprises a plurality of autonomous logic elements. The reservoir is a recurrent artificial neural network. The reservoir comprises a plurality of interconnections, wherein each interconnection of the plurality of interconnections couples a pair of the plurality of the nodes. The at least one input signal is weighted. The at least one output signal is weighted. Each of the plurality of nodes is weighted. The plurality of nodes implements at least one Boolean logic gate to perform at least one operation on the at least one input signal.

[00101] In an implementation, a reservoir computing device comprises a reservoir comprising an autonomous time-delay Boolean network of nodes; and a controller configured to provide input data to the reservoir and receive output data from the reservoir, wherein the reservoir

and the controller are comprised in integrated circuitry and configured to provide time-series prediction.

[00102] Implementations may include some or all of the following features. The integrated circuitry comprises a field-programmable gate array (FPGA). The reservoir computing device further comprises a memory that stores the input data, the output data, and node data of the autonomous time-delay Boolean network of nodes. The reservoir computing device further comprises a weighting module that applies weight to at least one of the input data, the output data, or the node data. The reservoir, the controller, the memory, and the weighting module are configured to predict a behavior of a chaotic dynamical system. The reservoir comprises a randomly parameterized network of nodes and recurrent links. The autonomous time-delay Boolean network comprises a plurality of autonomous logic elements. The reservoir comprises a plurality of interconnections, wherein each interconnection of the plurality of interconnections couples a pair of the nodes.

[00103] It should be understood that the various techniques described herein may be implemented in connection with hardware components or software components or, where appropriate, with a combination of both. Illustrative types of hardware components that can be used include Field-Programmable Gate Arrays (FPGAs), Application-specific Integrated Circuits (ASICs), Application-specific Standard Products (ASSPs), System-on-a-chip systems (SOCs), Complex Programmable Logic Devices (CPLDs), etc. The methods and apparatus of the presently disclosed subject matter, or certain aspects or portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium where, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the presently disclosed subject matter.

[00104] Although exemplary implementations may refer to utilizing aspects of the presently disclosed subject matter in the context of one or more stand-alone computer systems, the subject matter is not so limited, but rather may be implemented in connection with any computing environment, such as a network or distributed computing environment. Still further, aspects of the presently disclosed subject matter may be implemented in or across a plurality of processing chips or devices, and storage may similarly be effected across a plurality of devices. Such devices might include personal computers, network servers, and handheld devices, for example.

[00105] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims.

What is claimed:

1. A reservoir computing device, comprising:

an input layer configured to receive at least one input signal;

a reservoir comprising a plurality of nodes, wherein at least one node of the plurality of nodes is coupled to the input layer to receive the at least one input signal, wherein the reservoir comprises an autonomous time-delay Boolean network; and

an output layer configured to output at least one output signal, wherein at least another node of the plurality of nodes is coupled to the output layer,

wherein the input layer, the reservoir, and the output layer are comprised in integrated circuitry.

2. The reservoir computing device of claim 1, wherein the integrated circuitry comprises a field-programmable gate array (FPGA).

3. The reservoir computing device of claim 1, further comprising feedback from the output layer to the input layer, wherein an output signal of the output layer is an input signal to the input layer.

4. The reservoir computing device of claim 3, wherein the input layer, the reservoir, and the output layer are configured to predict a behavior of a chaotic dynamical system.

5. The reservoir computing device of claim 1, wherein the reservoir comprises a randomly parameterized network of nodes and recurrent links.

6. The reservoir computing device of claim 1, wherein the autonomous time-delay Boolean network comprises a plurality of autonomous logic elements.

7. The reservoir computing device of claim 1, wherein the reservoir is a recurrent artificial neural network.

8. The reservoir computing device of claim 1, wherein the reservoir comprises a plurality of interconnections, wherein each interconnection of the plurality of interconnections couples a pair of the plurality of the nodes.

9. The reservoir computing device of claim 1, wherein the at least one input signal is weighted.

10. The reservoir computing device of claim 1, wherein the at least one output signal is weighted.

11. The reservoir computing device of claim 1, wherein each of the plurality of nodes is weighted.

12. The reservoir computing device of claim 1, wherein the plurality of nodes implements at least one Boolean logic gate to perform at least one operation on the at least one input signal.

13. A reservoir computing device, comprising:

a reservoir comprising an autonomous time-delay Boolean network of nodes; and

a controller configured to provide input data to the reservoir and receive output data from the reservoir, wherein the reservoir and the controller are comprised in integrated circuitry and configured to provide time-series prediction.

14. The reservoir computing device of claim 13, wherein the integrated circuitry comprises a field-programmable gate array (FPGA).

15. The reservoir computing device of claim 13, further comprising a memory that stores the input data, the output data, and node data of the autonomous time-delay Boolean network of nodes.

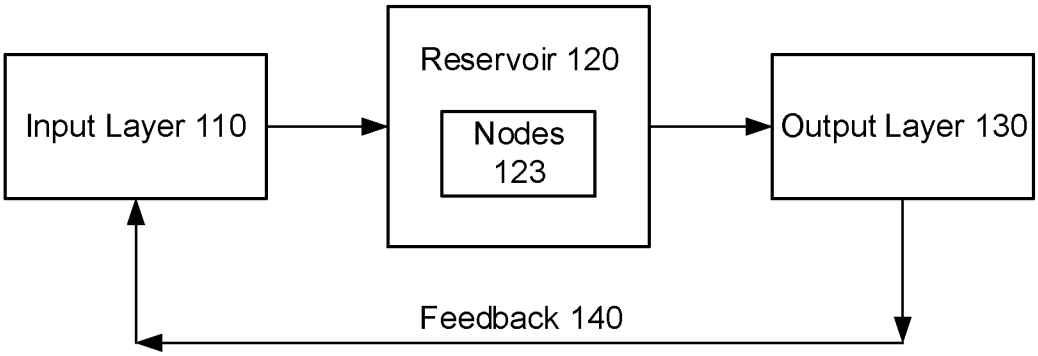
16. The reservoir computing device of claim 15, further comprising a weighting module that applies weight to at least one of the input data, the output data, or the node data.

17. The reservoir computing device of claim 16, wherein the reservoir, the controller, the memory, and the weighting module are configured to predict a behavior of a chaotic dynamical system.

18. The reservoir computing device of claim 13, wherein the reservoir comprises a randomly parameterized network of nodes and recurrent links.

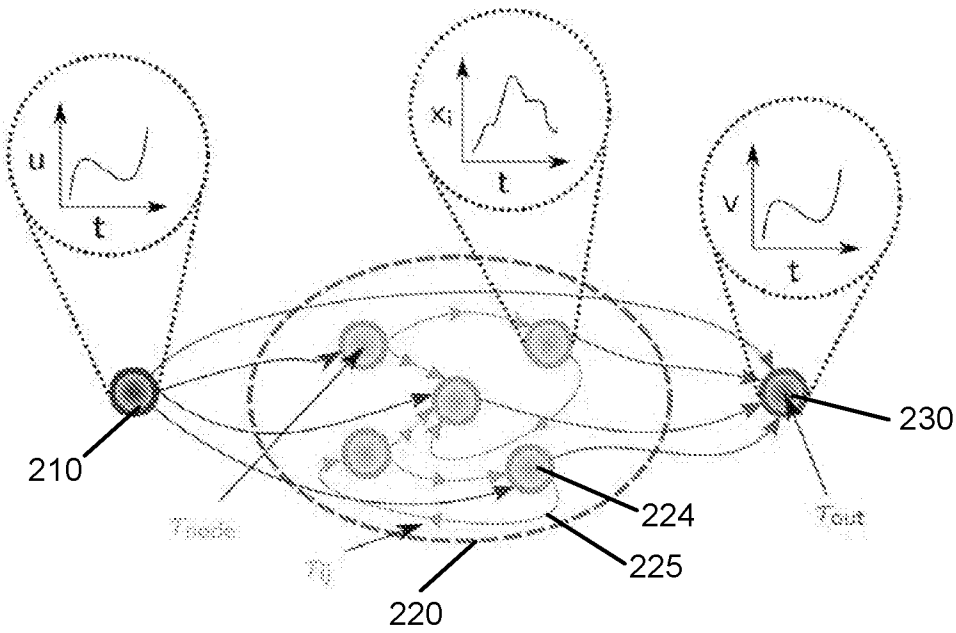
19. The reservoir computing device of claim 13, wherein the autonomous time-delay Boolean network comprises a plurality of autonomous logic elements.

20. The reservoir computing device of claim 13, wherein the reservoir comprises a plurality of interconnections, wherein each interconnection of the plurality of interconnections couples a pair of the nodes.


Input Layer 110
Reservoir 120
Nodes 123
Output Layer 130
Feedback 140

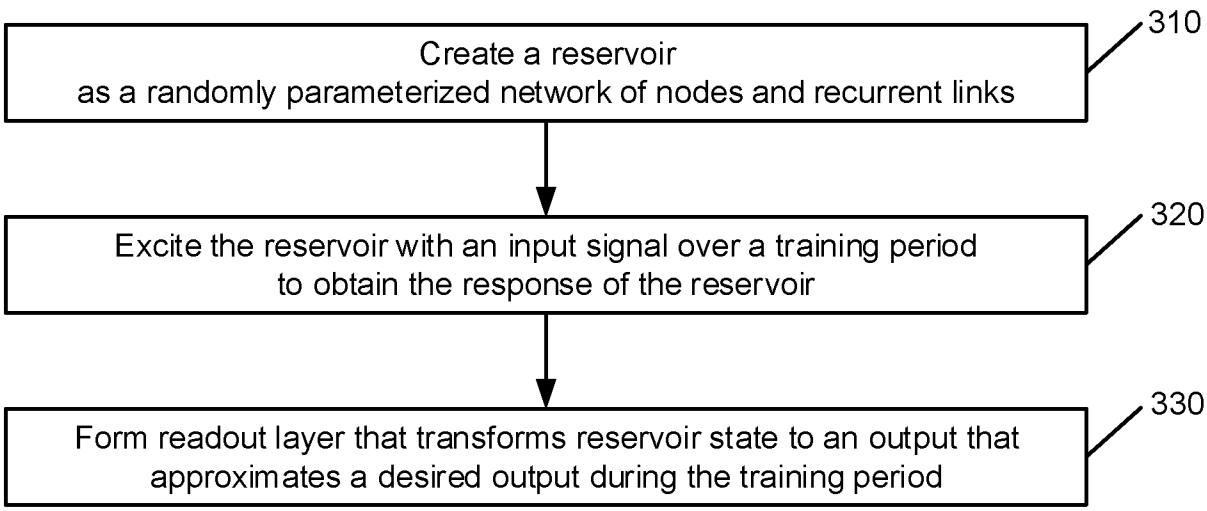
100

FIG. 1



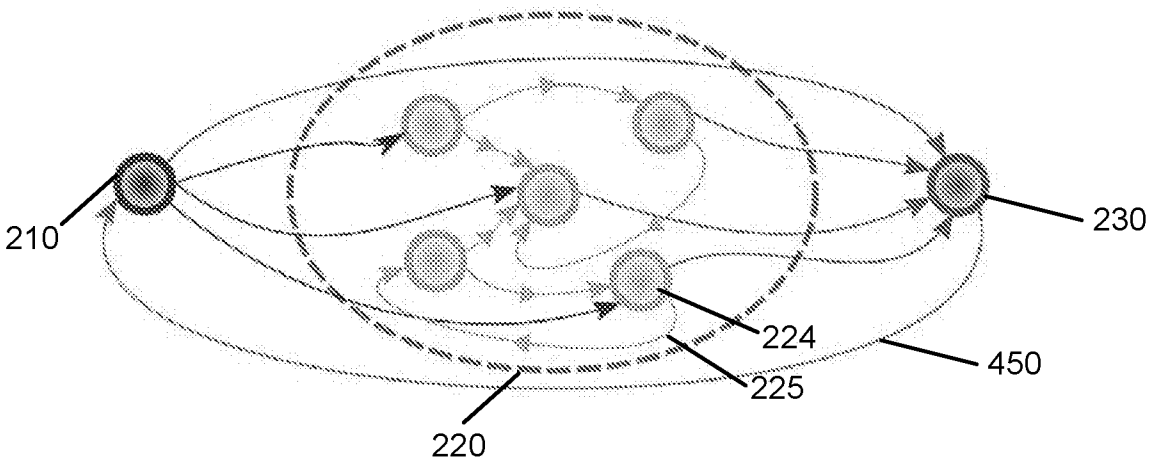
200

FIG. 2



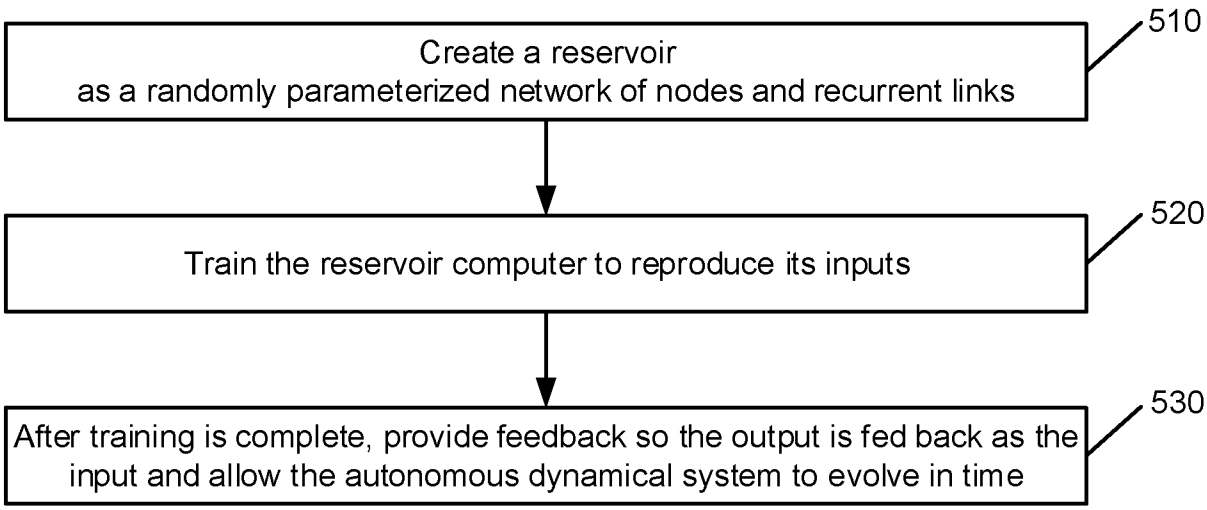
300

FIG. 3



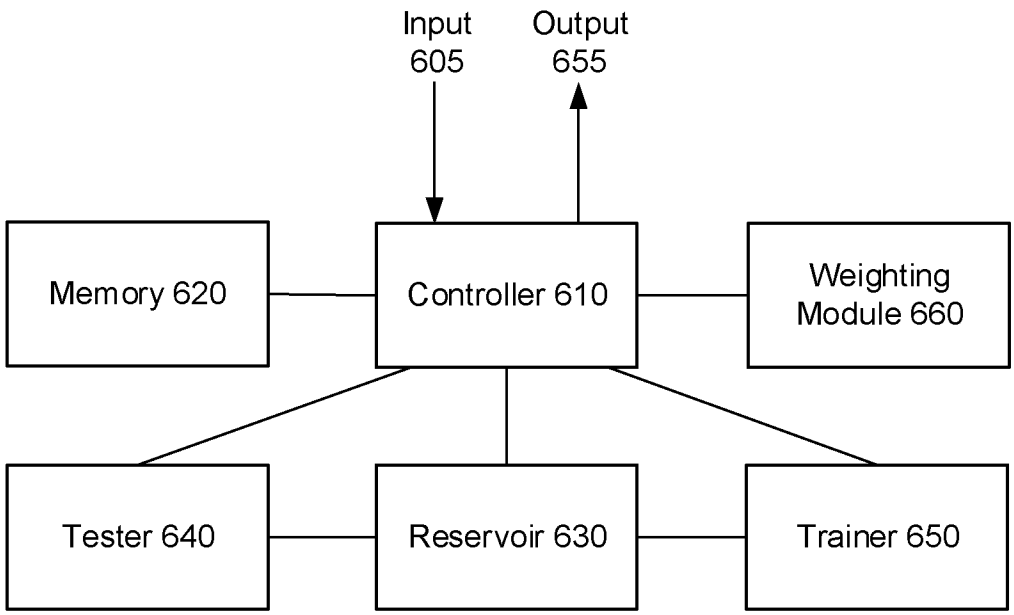
400

FIG. 4



500

FIG. 5


Input
605
Output
655
Memory 620
Controller 610
Weighting
Module 660
Tester 640
Reservoir 630
Trainer 650

600

FIG. 6

710

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

700

FIG. 7

```
 1 module node(node_in, node_out);
 2 parameter lut = 8'b00000001;
 3 input[2:0] node_in;
 4 output reg node_out;
 5
 6 always @(*) begin
 7   case (node_in)
 8     3'b000 : node_out = lut[7];
 9     3'b001 : node_out = lut[6];
10     3'b010 : node_out = lut[5];
11     3'b011 : node_out = lut[4];
12     3'b100 : node_out = lut[3];
13     3'b101 : node_out = lut[2];
14     3'b110 : node_out = lut[1];
15     3'b111 : node_out = lut[0];
16   endcase
17 end
18 endmodule
```

800

FIG. 8

```
1  module delay_line(delay_in, delay_out);
2  parameter m = 15;
3  input delay_in;
4  output delay_out;
5  wire [2*m-1:0] delay /*synthesis keep */;
6
7  assign delay[0] = ~delay_in;
8  assign delay_out = delay[2*m-1];
9
10 genvar i;
11 generate
12   for (i=0;i<2*m-1;i=i+1)
13   begin : generate_delay
14     assign delay [i+1] = ~delay[i];
15   end
16 endgenerate
17 endmodule
```

900

FIG. 9

```
 1 module reservoir (u, x);
 2 parameter N = 3;
 3 parameter k = 2;
 4 parameter m = 1;
 5 input [m-1:0] u;
 6 output [N-1:0] x;
 7 wire [N*k-1:0] x_tau;
 8
 9 node #(8'b01111111) node_0 ({u, x_tau[0], x_tau
       [1]}, x[0]);
10 node #(8'b01000000) node_1 ({u, x_tau[2], x_tau
       [3]}, x[1]);
11 node #(8'b01011101) node_2 ({u, x_tau[4], x_tau
       [5]}, x[2]);
12
13 delay_line #(10) delay_0 (x[0], x_tau[0]);
14 delay_line #(15) delay_1 (x[1], x_tau[1]);
15 delay_line #(6) delay_2 (x[0], x_tau[2]);
16 delay_line #(7) delay_3 (x[2], x_tau[3]);
17 delay_line #(10) delay_4 (x[0], x_tau[4]);
18 delay_line #(12) delay_5 (x[1], x_tau[5]);
19 endmodule
```

1000

FIG. 10

```
 1 module reservoir_computer(clk);
 2 input clk;
 3
 4 wire mode;
 5 wire [m-1:0] u;
 6 wire [2*m*(N+1)-1:0] W_out;
 7 wire [N-1:0] x;
 8 wire [m-1:0] v;
 9
10 reg [N-1:0] x_reg;
11 reg [m-1:0] v_reg;
12 wire [m-1:0] u_v;
13
14 sampler sampler_1 (clk, mode, u, W_out);
15 player player_1 (clk, x, v);
16
17 assign u_v = mode ? u : v;
18
19 reservoir reservoir_1 (u_v, x);
20
21 output_layer output_layer 1 (W_out, u_v, x_reg,
       v);
22
23 always @(posedge clk) begin
24   x_reg <= x;
25   v_reg <= v;
26 end
27 endmodule
```

1100

FIG. 11

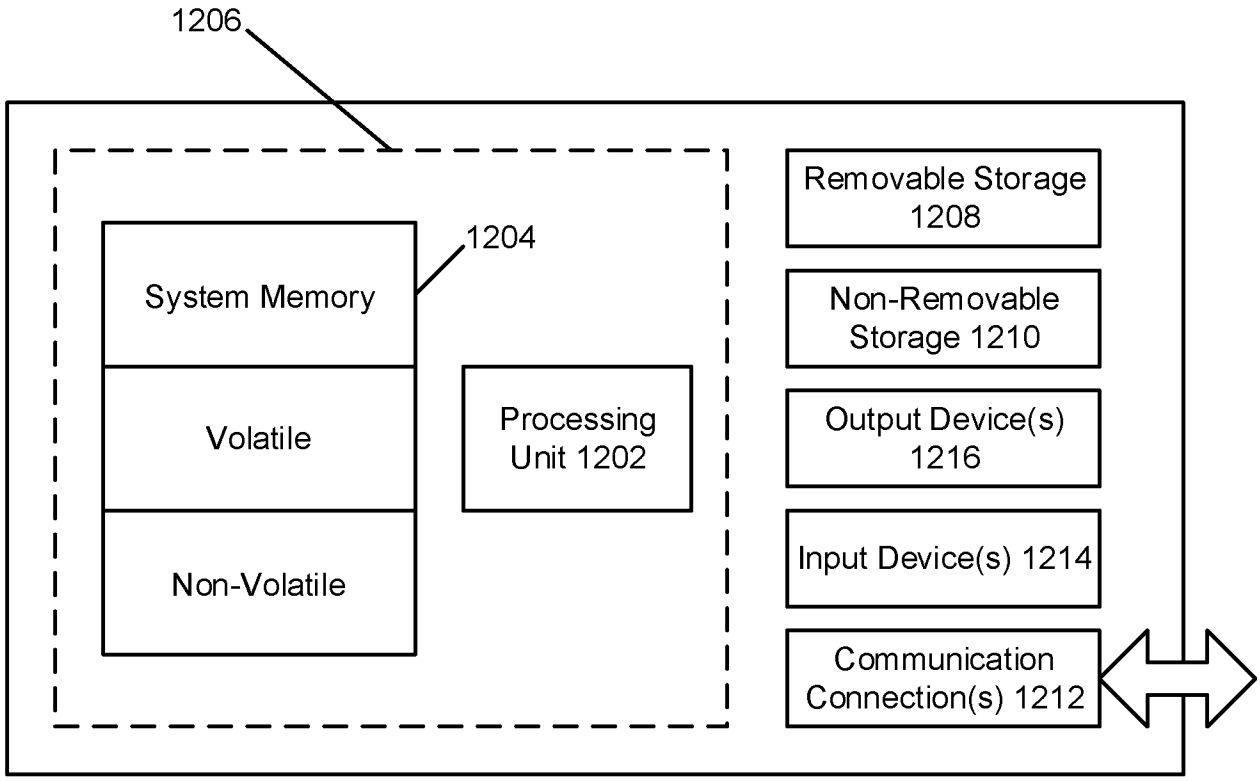

1206
1204
System Memory
Volatile
Non-Volatile
Processing Unit 1202
Removable Storage 1208
Non-Removable Storage 1210
Output Device(s) 1216
Input Device(s) 1214
Communication Connection(s) 1212
1200

FIG. 12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2019/024296

A. CLASSIFICATION OF SUBJECT MATTER

IPC(8) - B25J 9/16; G06N 3/04 (2019.01)

CPC - G06N 3/063; G06N 3/084; G06F 15/16; G06N 3/02; G06N 3/06; G06Q 10/10; G06T 1/20 (2019.05)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

See Search History document

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

USPC - 706/12; 706/20 (keyword delimited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

See Search History document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	ALOMAR et al. "FPGA-Based Stochastic Echo State Networks for Time-Series Forecasting," Computational Intelligence and Neuroscience Volume 2016, Article ID 3917892, http://dx.doi.org/10.1155/2016/3917892, [retrieved on 2019-05-22]. Retrieved from the Internet: <URL:https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4735989/> pp 1-14	1-20
Y	LOHMAN et al. "Transient Dynamics and their Control in time-delay autonomous Boolean ring Networks," PACS numbers: 05.45.-a, 87.18.Cf, 87.19.lr, Ohio State University, 18 January 2017, [retrieved on 2019-05-22]. Retrieved from the Internet: <URL: https://pdfs.semanticscholar.org/9d63/4b4cf2c58b0de9ad9f6a960128a1e98d98de.pdf> pp 1-12	1-20
A	WO 2014/203039 A1 (ASELSAN ELEKTRONIK SANAYI VE TICARET ANONIM SIRKETI) 24 December 2014 (24.12.2014) entire document	1-20
A	US 9,489,623 B1 (BRAIN CORPORATION) 08 November 2016 (08.11.2016) entire document	1-20

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
23 May 2019	19 JUN 2019
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, VA 22313-1450 Facsimile No. 571-273-8300	Authorized officer Blaine R. Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774