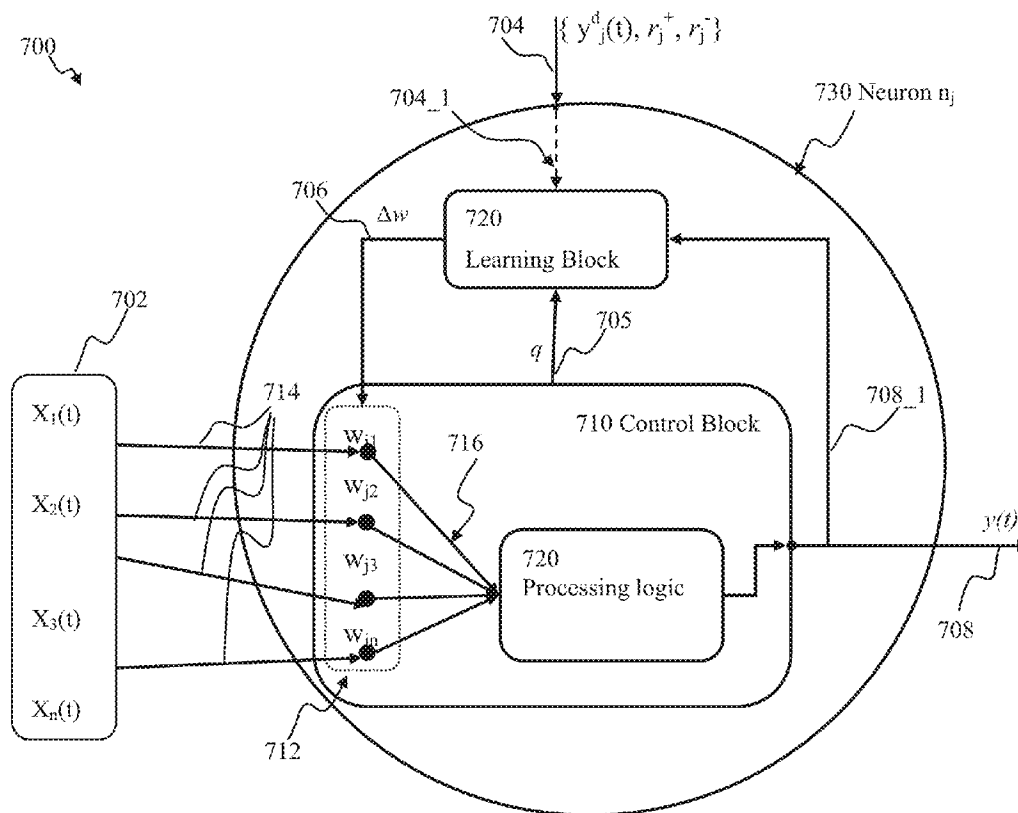




US 20130325768A1

(19) **United States**(12) **Patent Application Publication**
Sinyavskiy et al.(10) **Pub. No.: US 2013/0325768 A1**(43) **Pub. Date: Dec. 5, 2013**(54) **STOCHASTIC SPIKING NETWORK
LEARNING APPARATUS AND METHODS**(52) **U.S. Cl.**
USPC 706/16(75) Inventors: **Oleg Sinyavskiy**, San Diego, CA (US);
Olivier Coenen, San Diego, CA (US)(73) Assignee: **Brain Corporation**, San Diego, CA
(US)(21) Appl. No.: **13/487,533**(22) Filed: **Jun. 4, 2012****Publication Classification**(51) **Int. Cl.**
G06F 15/18 (2006.01)(57) **ABSTRACT**

Generalized learning rules may be implemented. A framework may be used to enable adaptive spiking neuron signal processing system to flexibly combine different learning rules (supervised, unsupervised, reinforcement learning) with different methods (online or batch learning). The generalized learning framework may employ time-averaged performance function as the learning measure thereby enabling modular architecture where learning tasks are separated from control tasks, so that changes in one of the modules do not necessitate changes within the other. Separation of learning tasks from the control tasks implementations may allow dynamic reconfiguration of the learning block in response to a task change or learning method change in real time. The generalized spiking neuron learning apparatus may be capable of implementing several learning rules concurrently based on the desired control application and without requiring users to explicitly identify the required learning rule composition for that task.



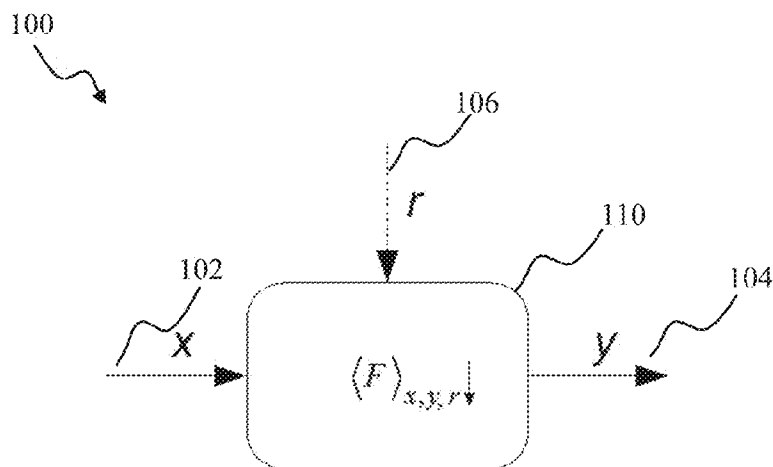


FIG. 1 (Prior Art)

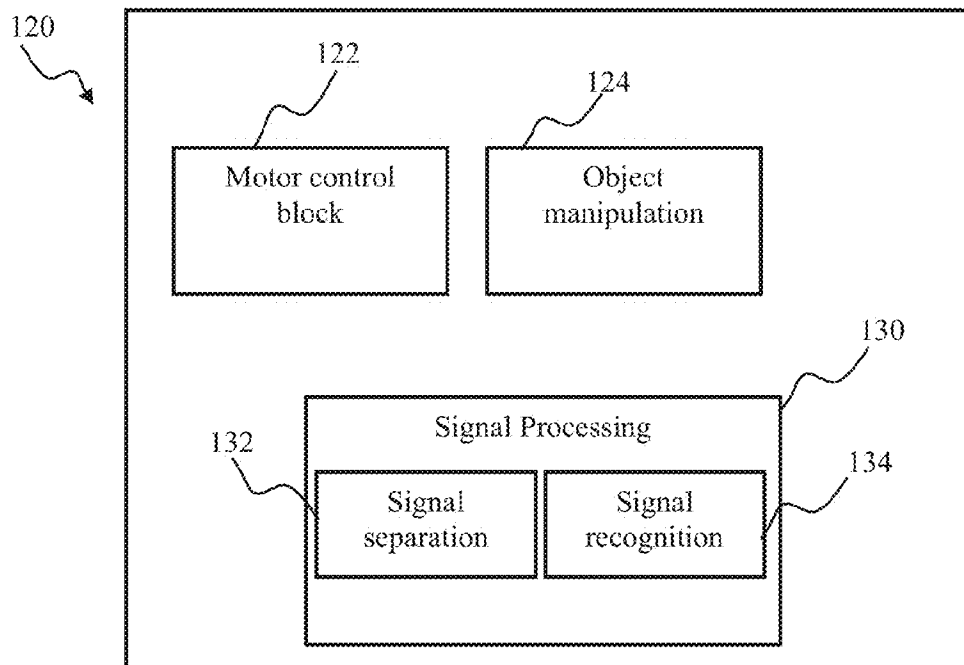


FIG. 1A (Prior Art)

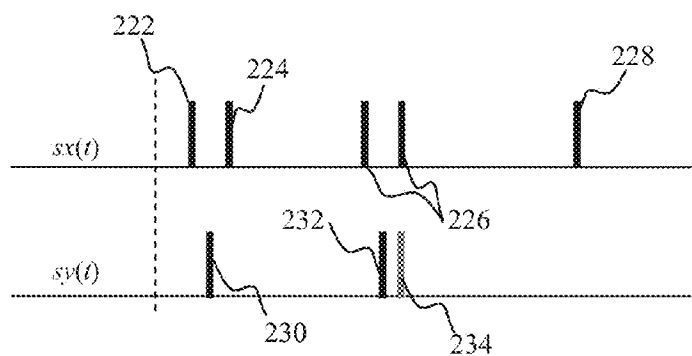


FIG. 2 (Prior Art)

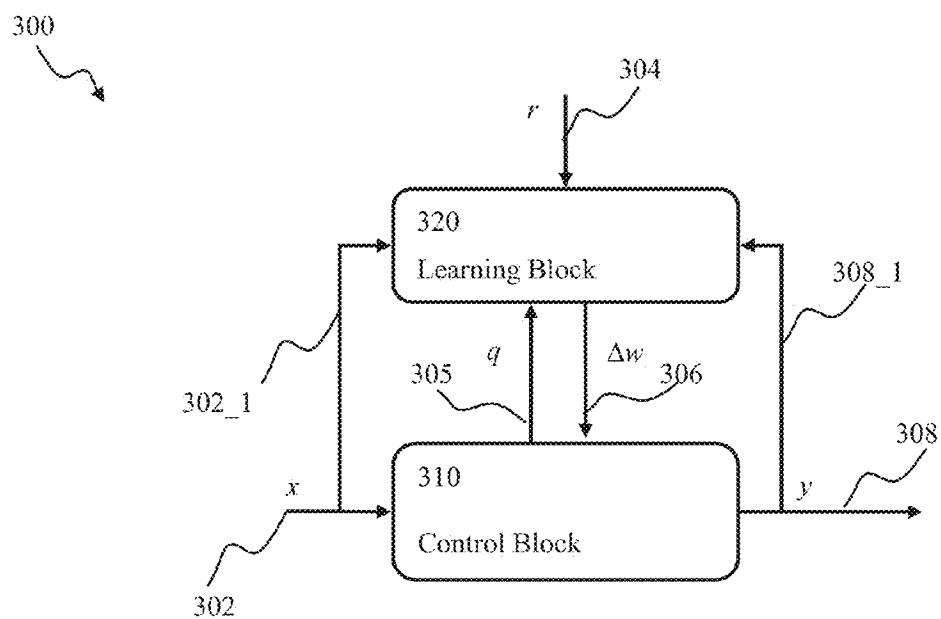


FIG. 3

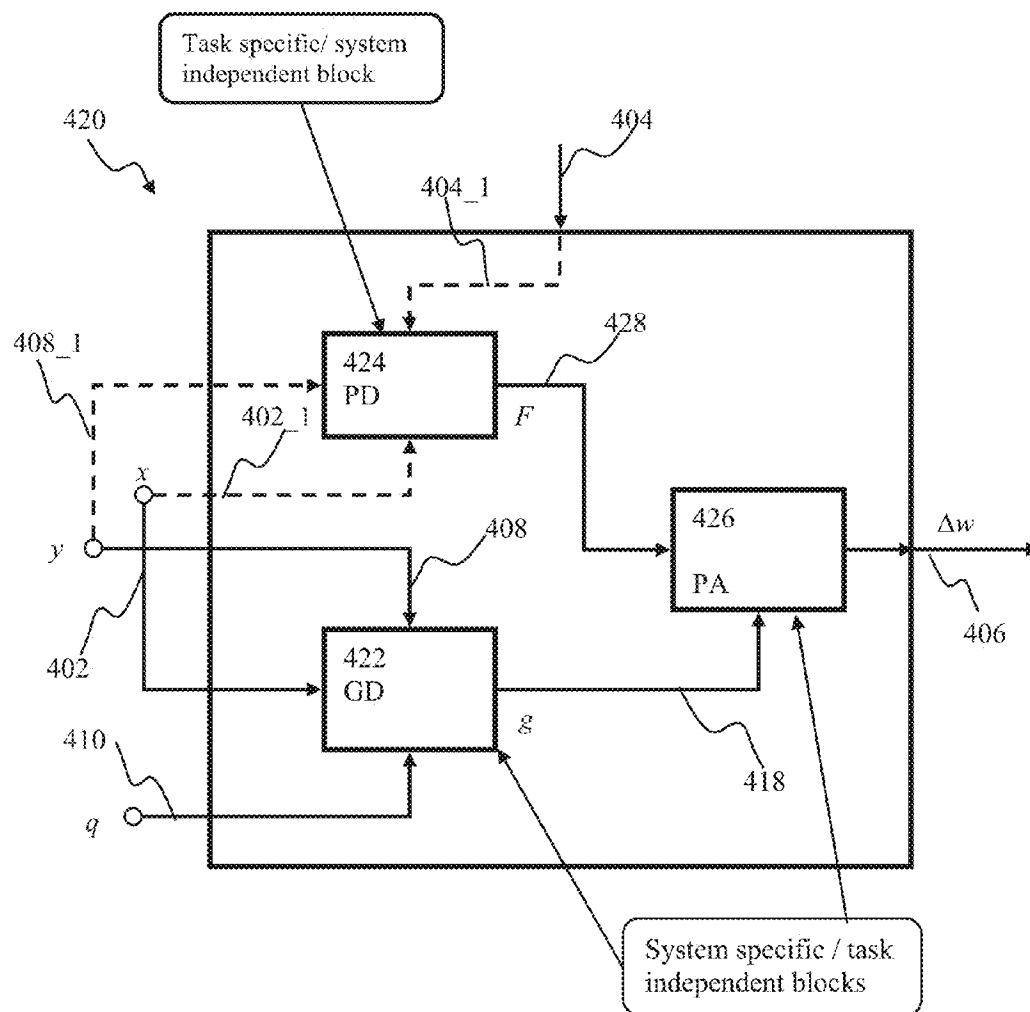


FIG. 4

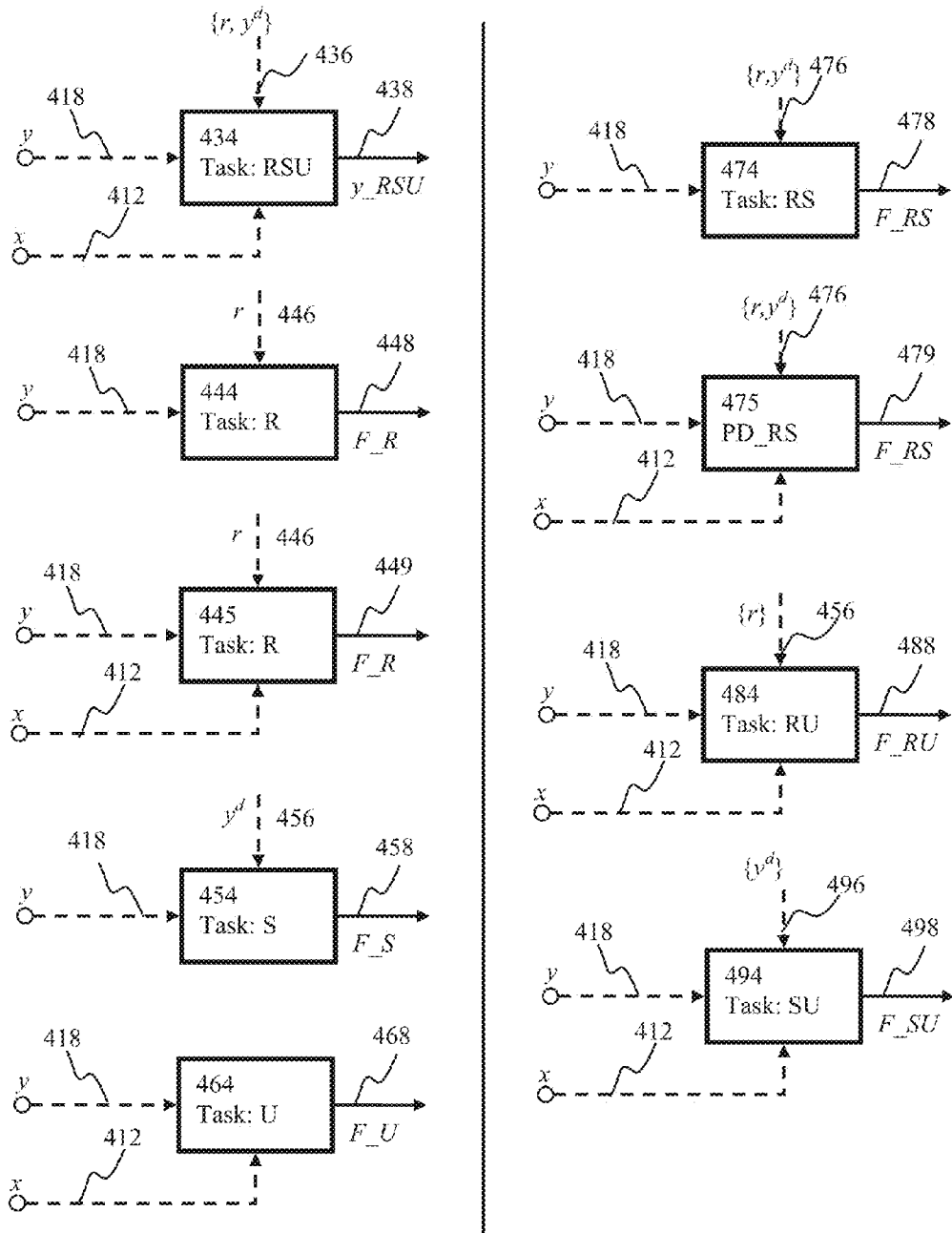


FIG. 4A

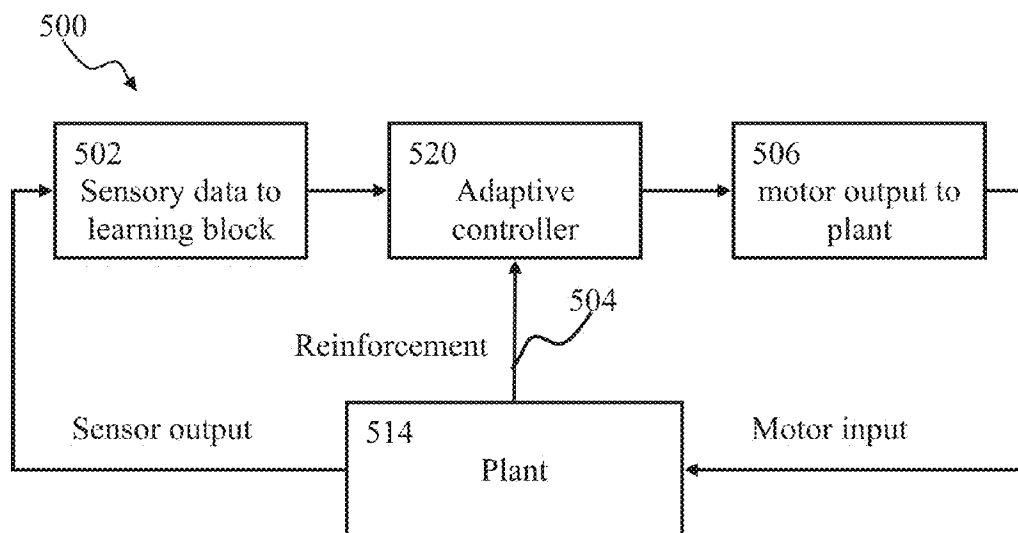


FIG. 5

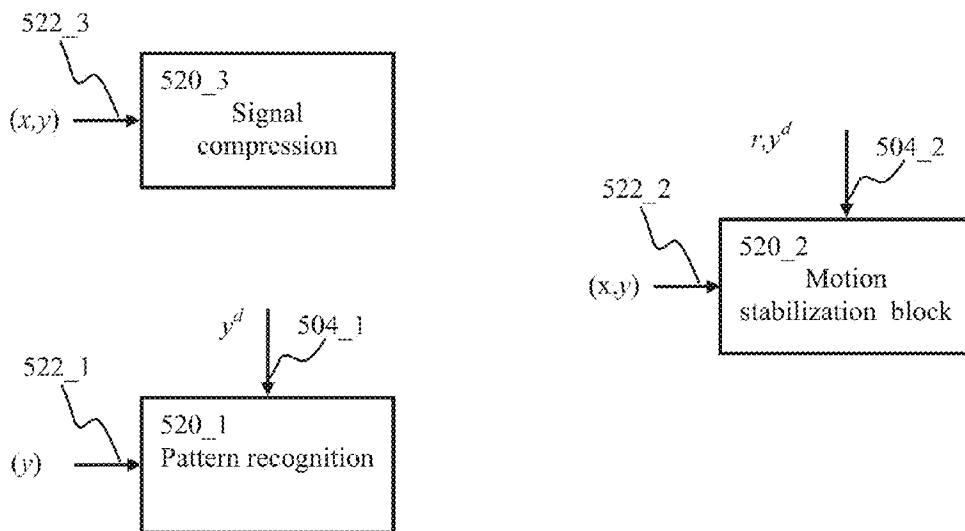


FIG. 5A

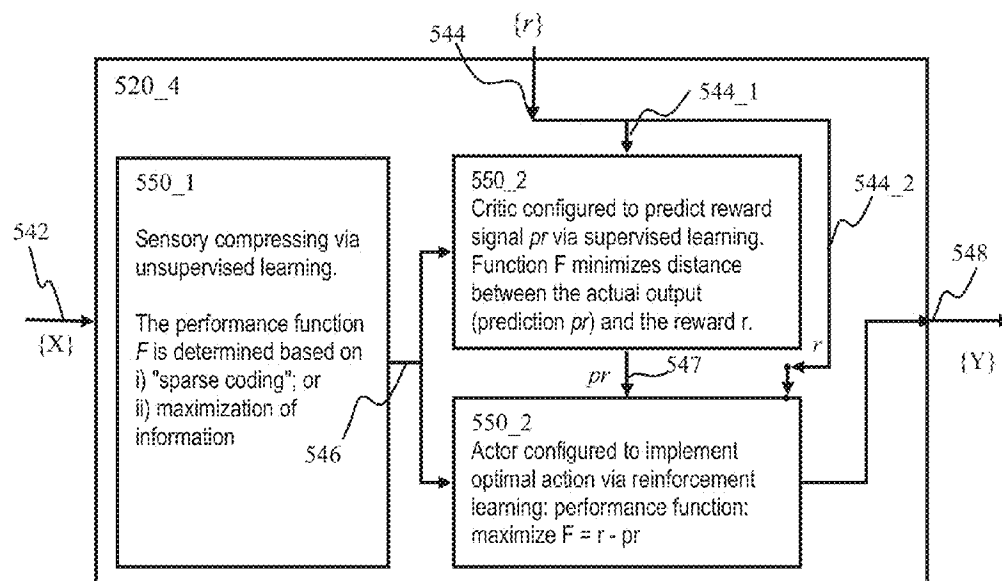


FIG. 5B

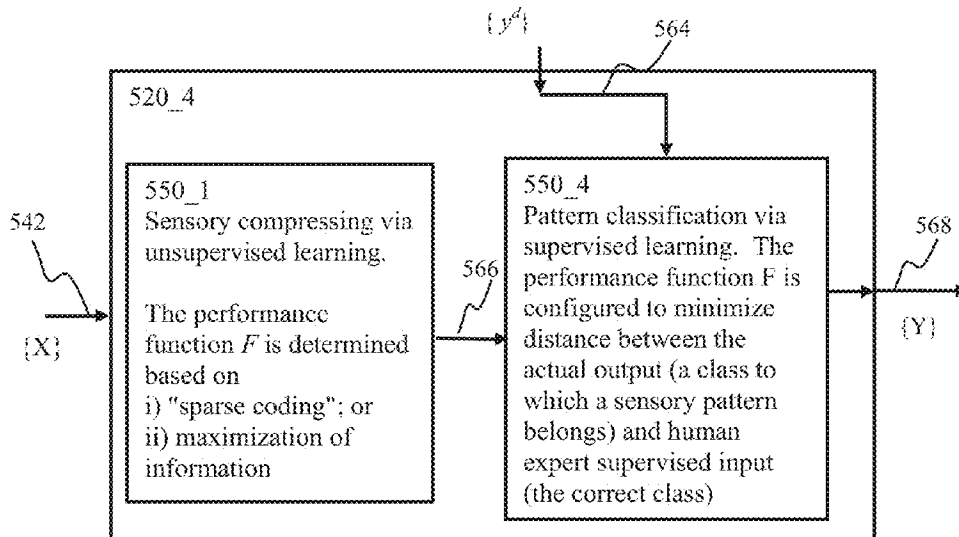


FIG. 5C

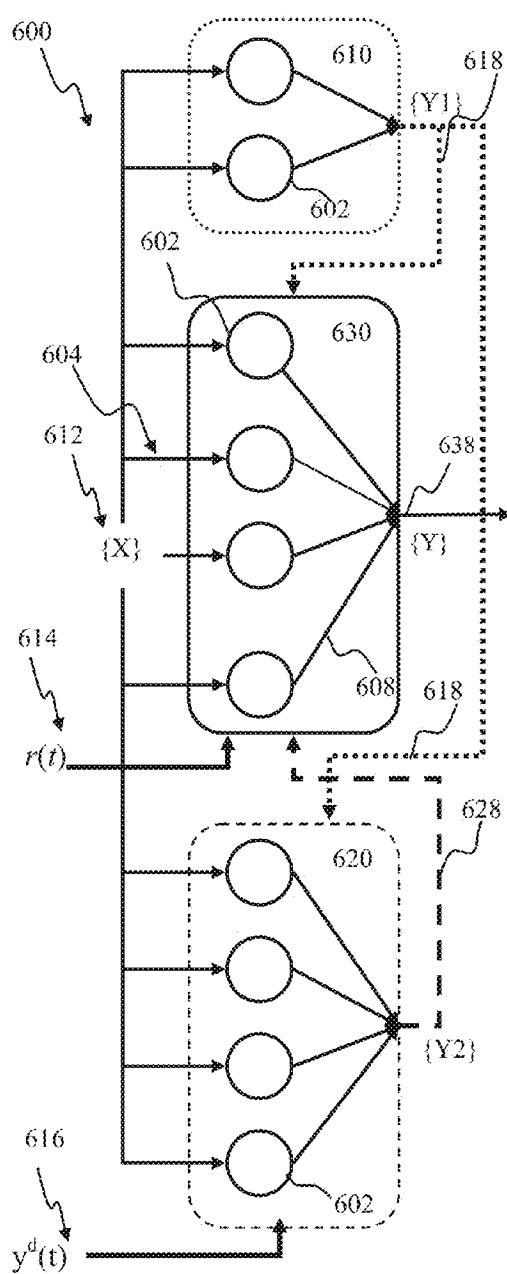


FIG. 6A

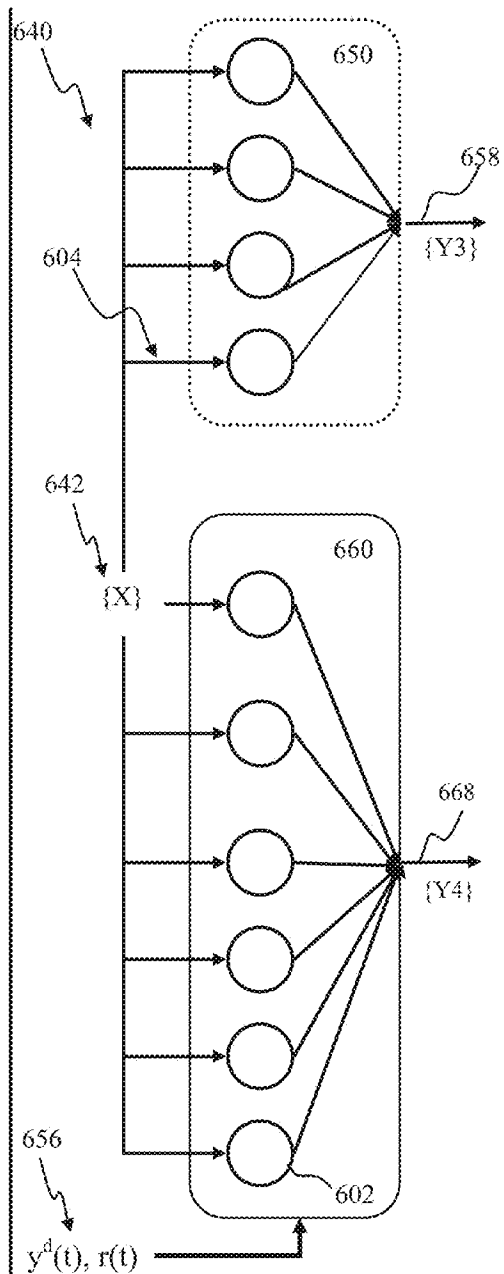


FIG. 6B

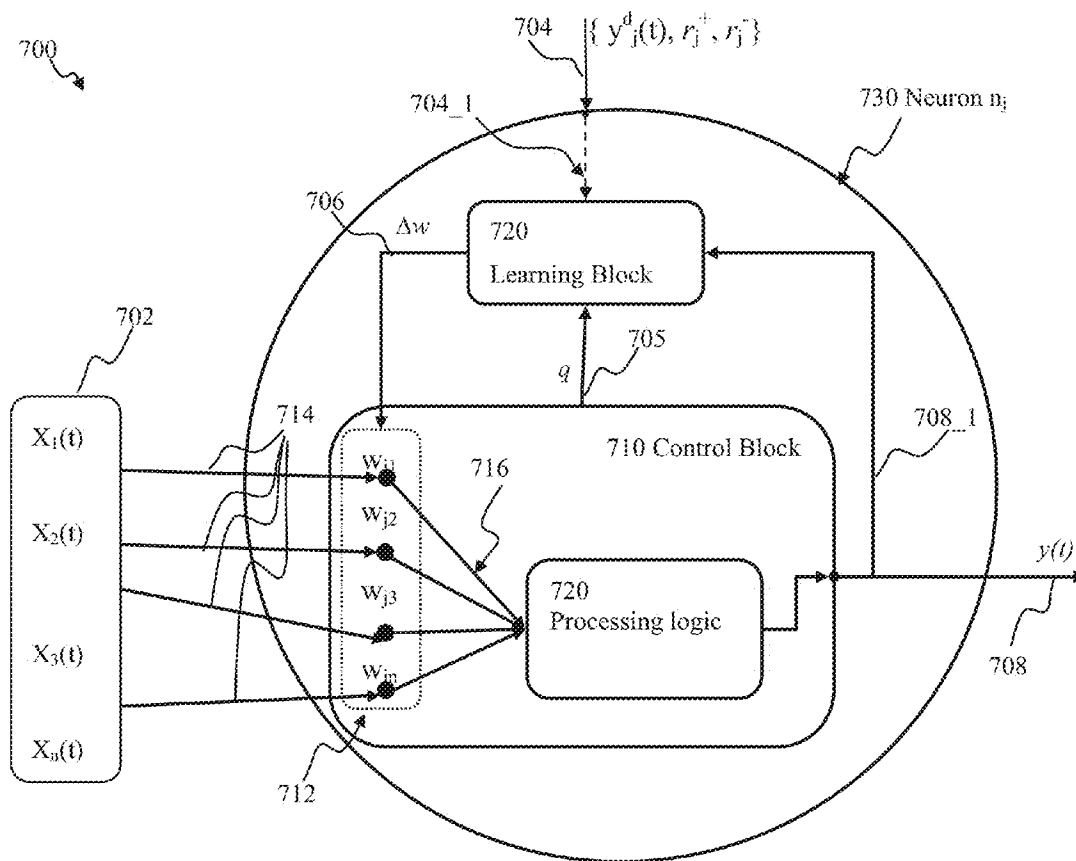


FIG. 7

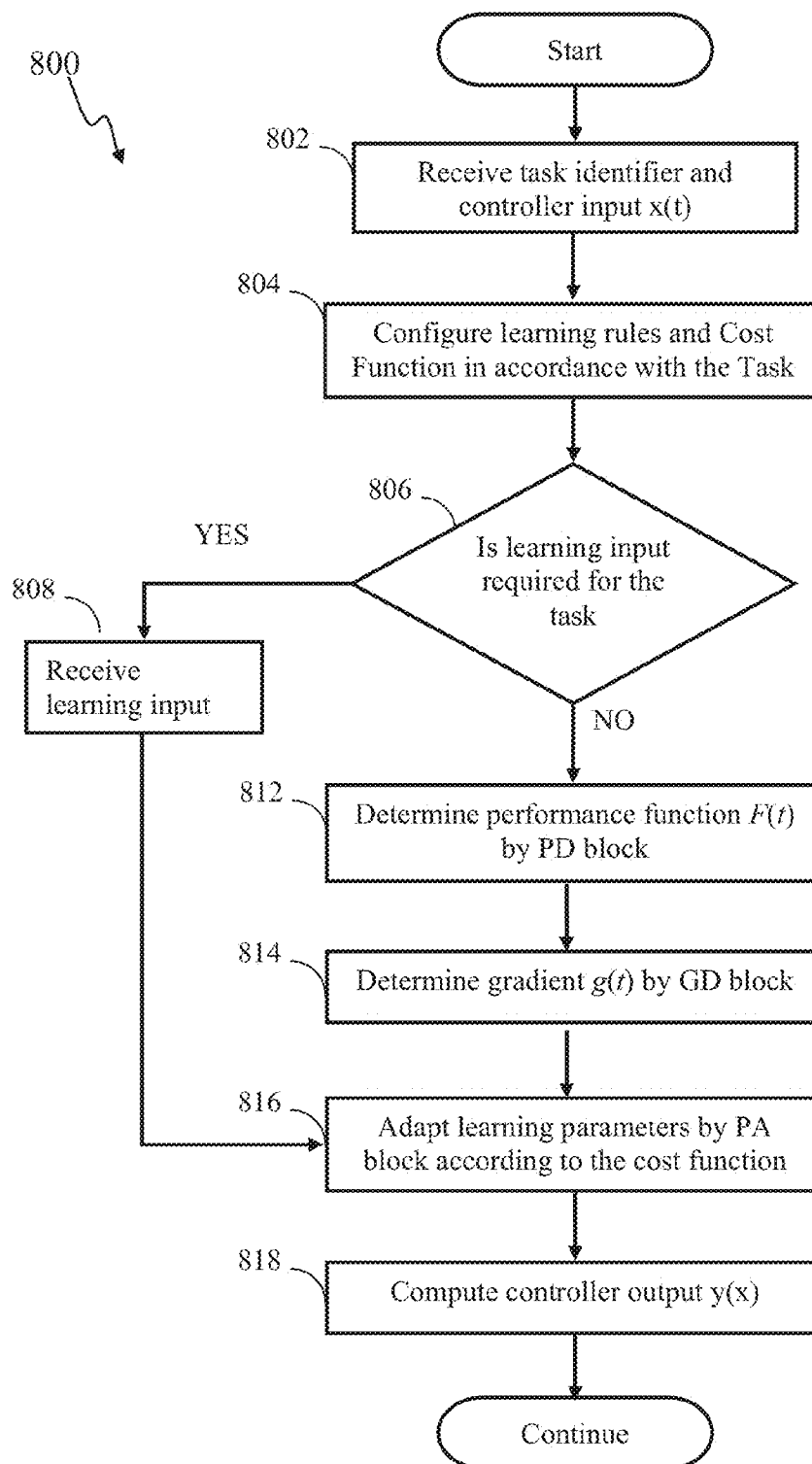


FIG. 8A

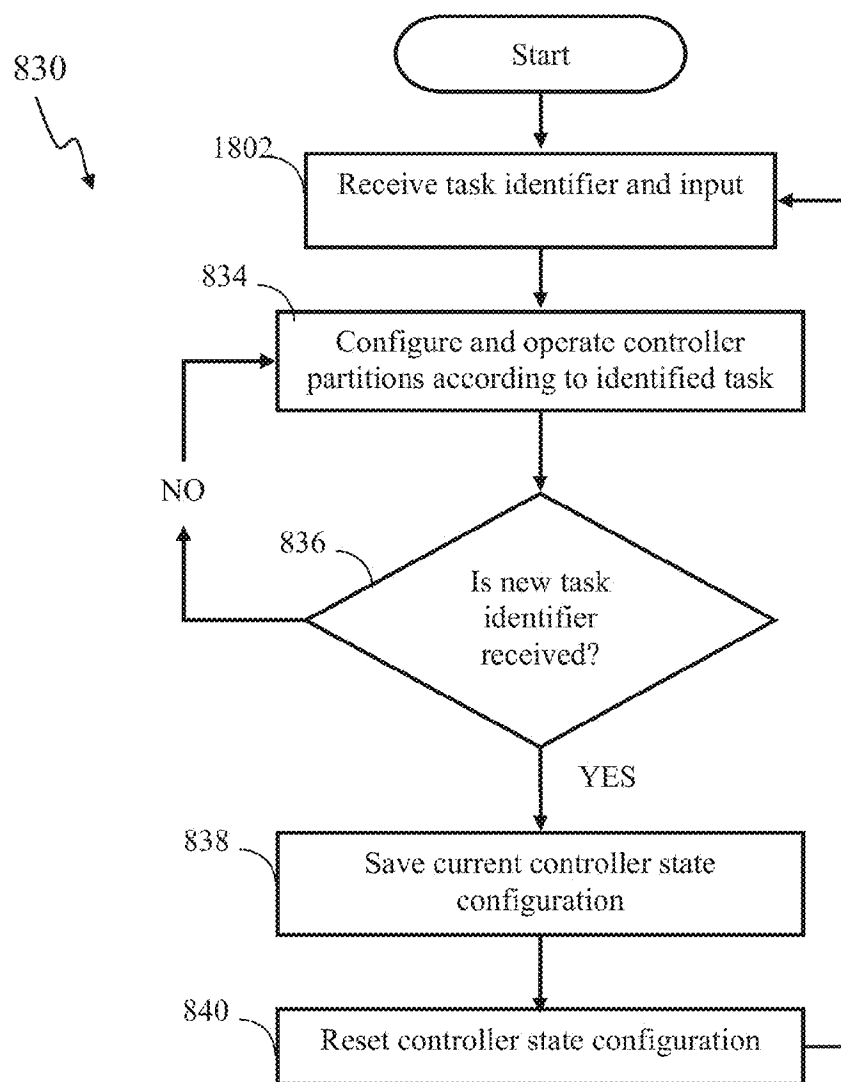


FIG. 8 B

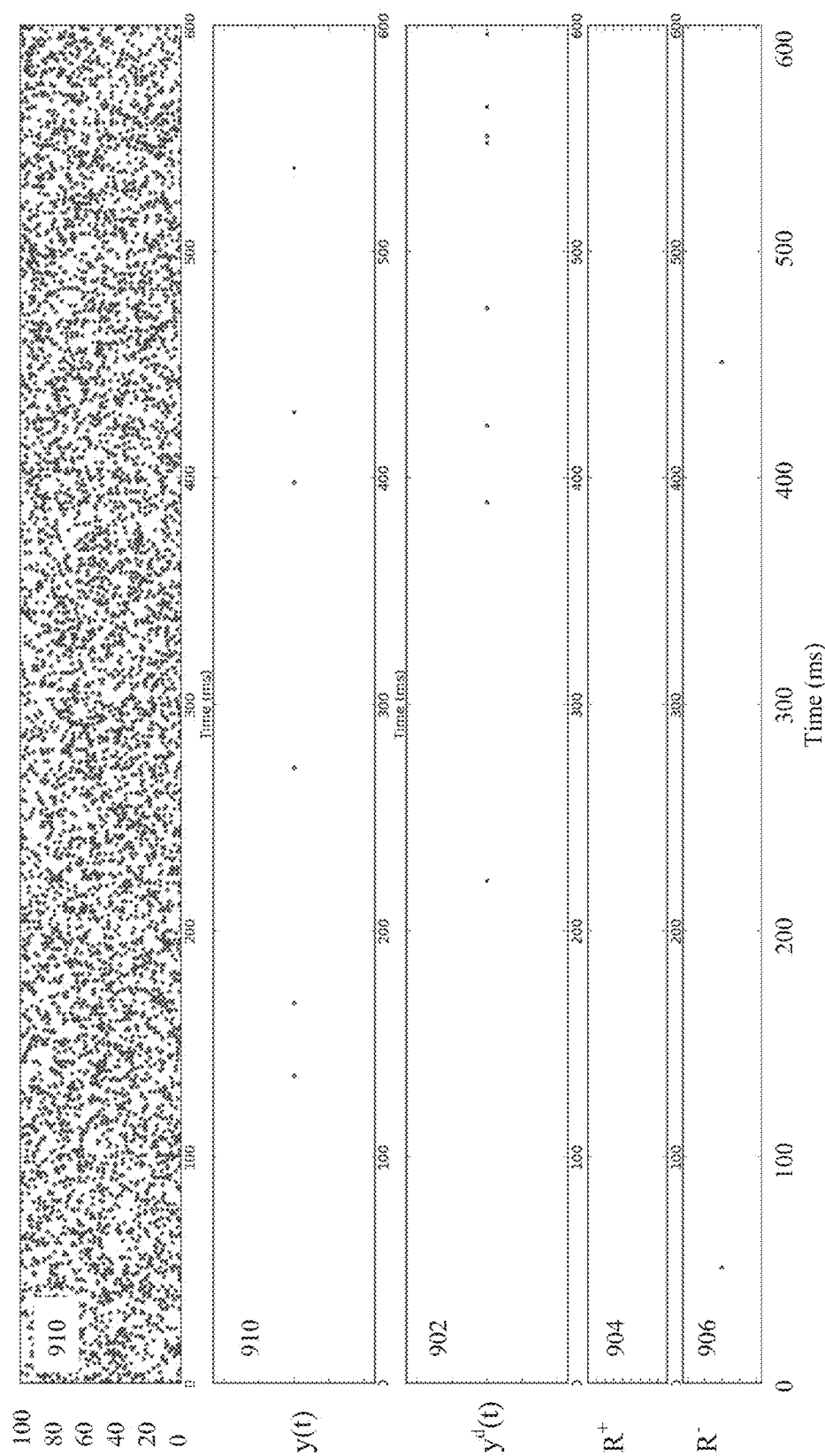


FIG. 9A

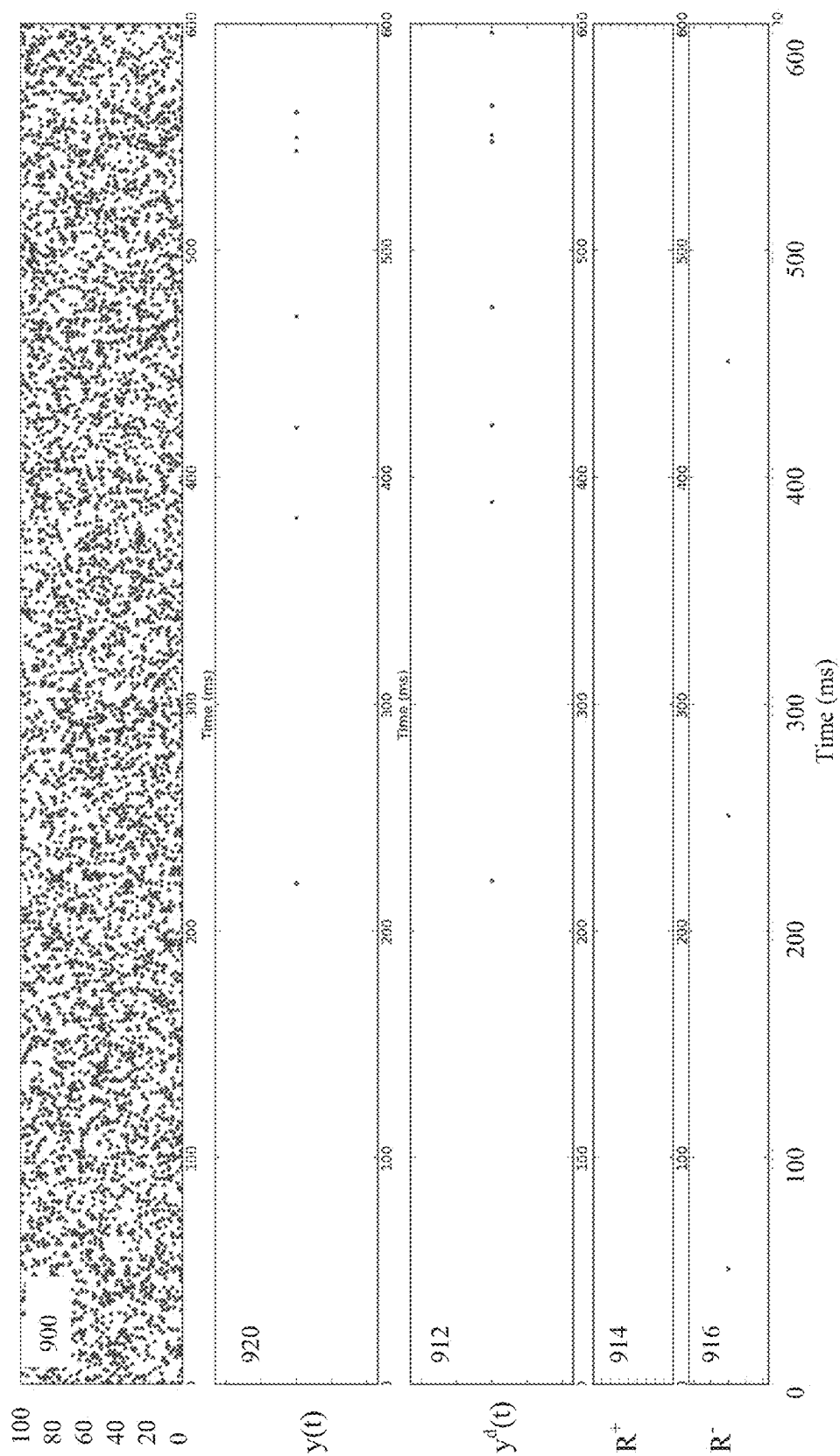
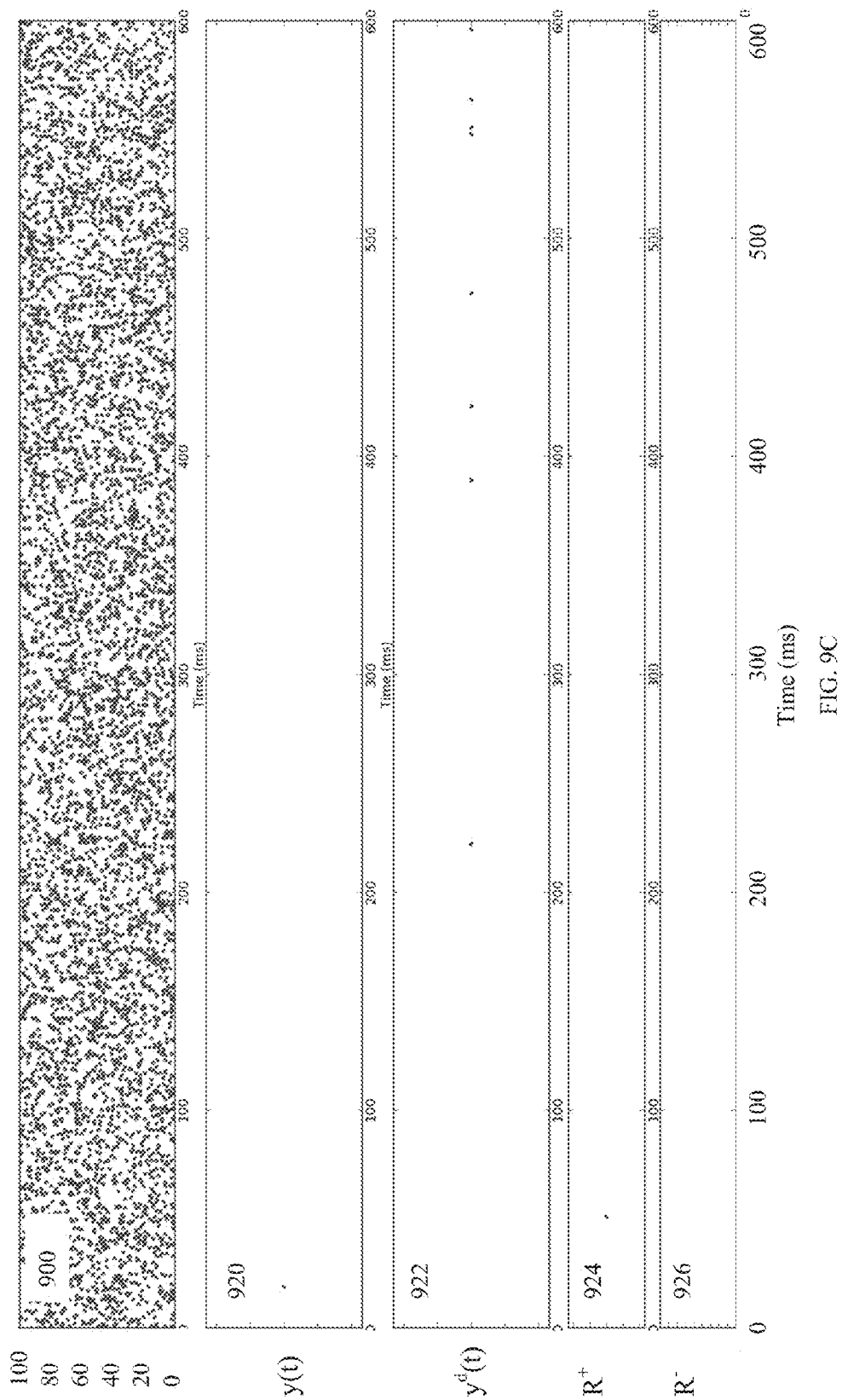
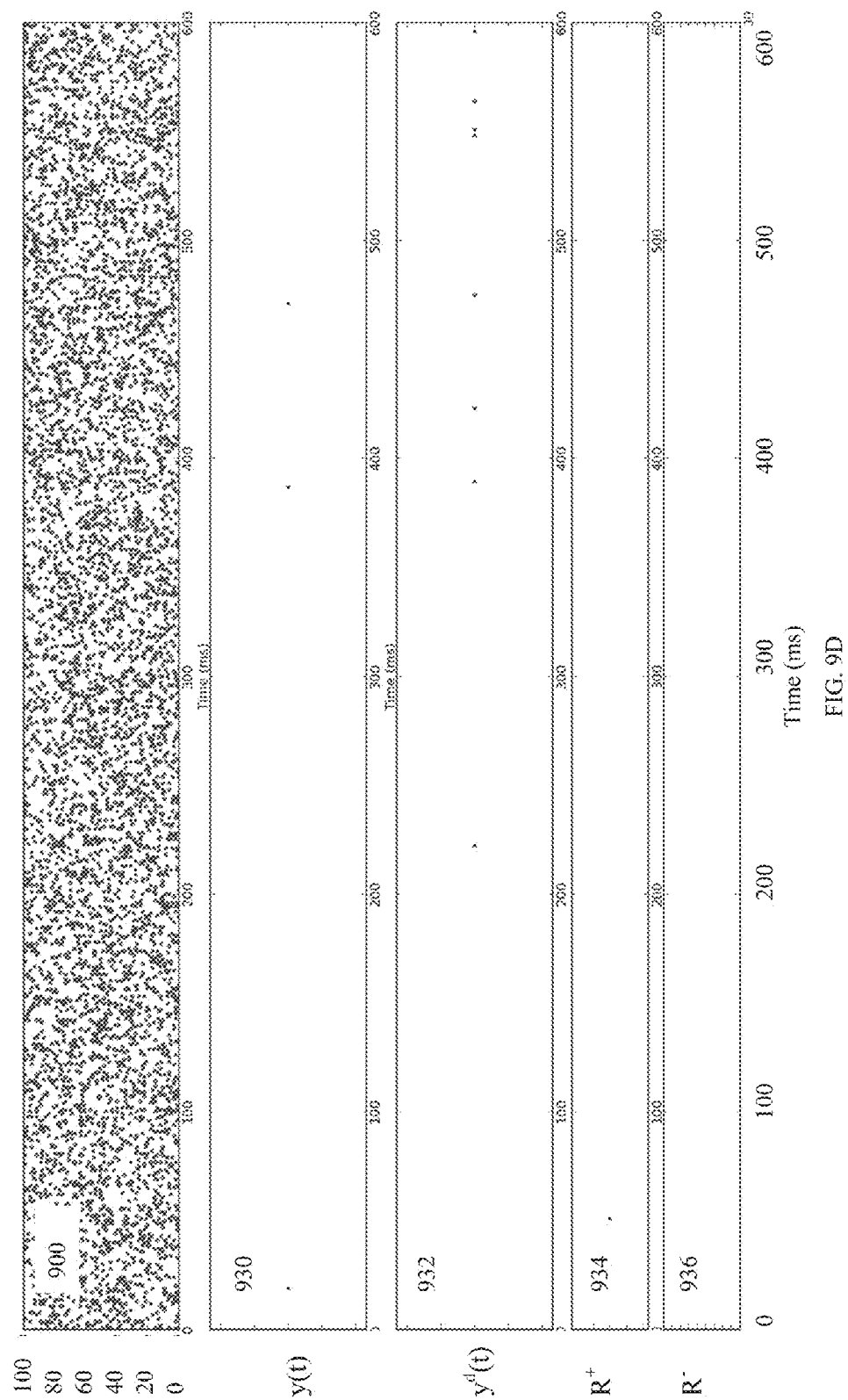
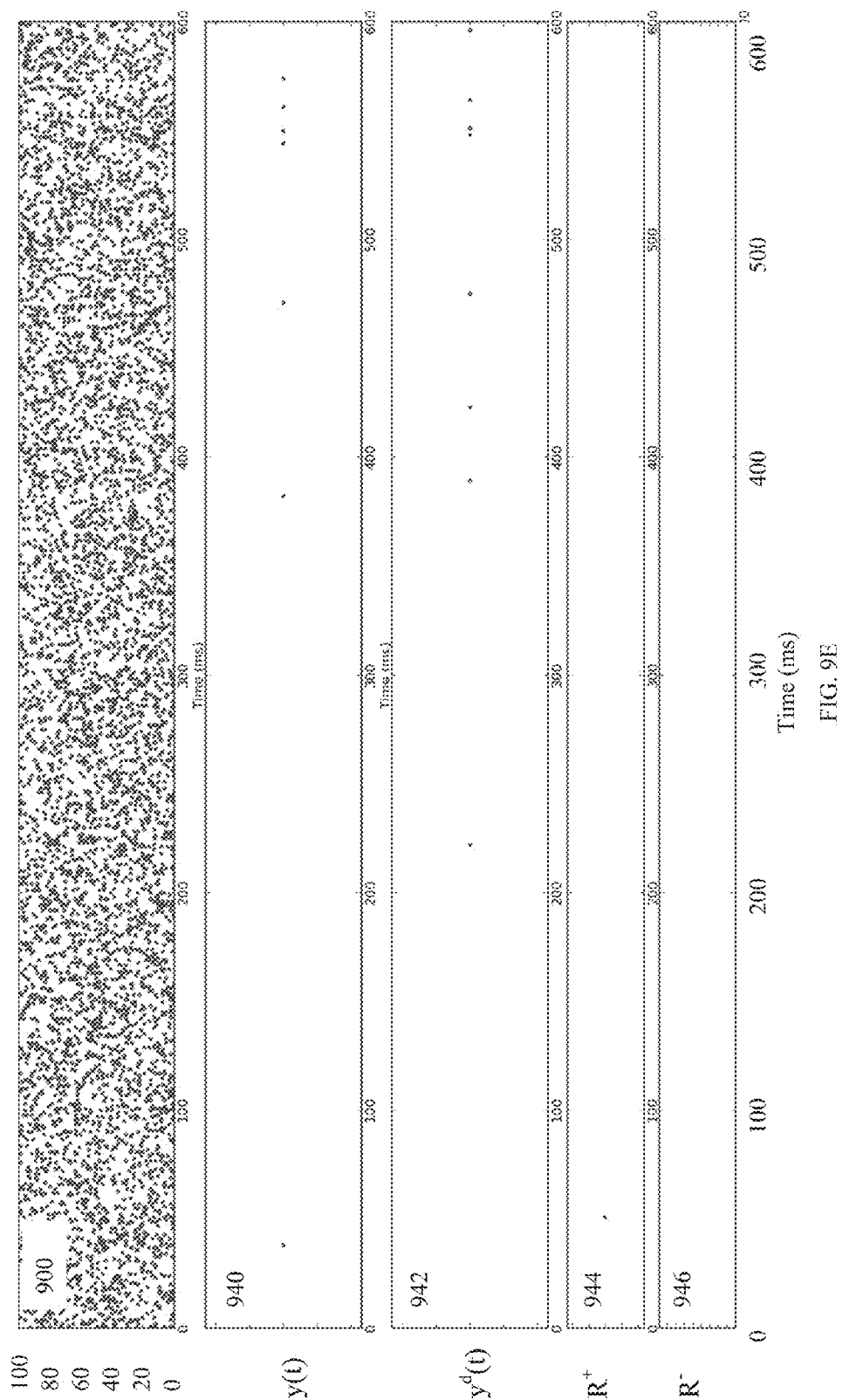


FIG. 9B







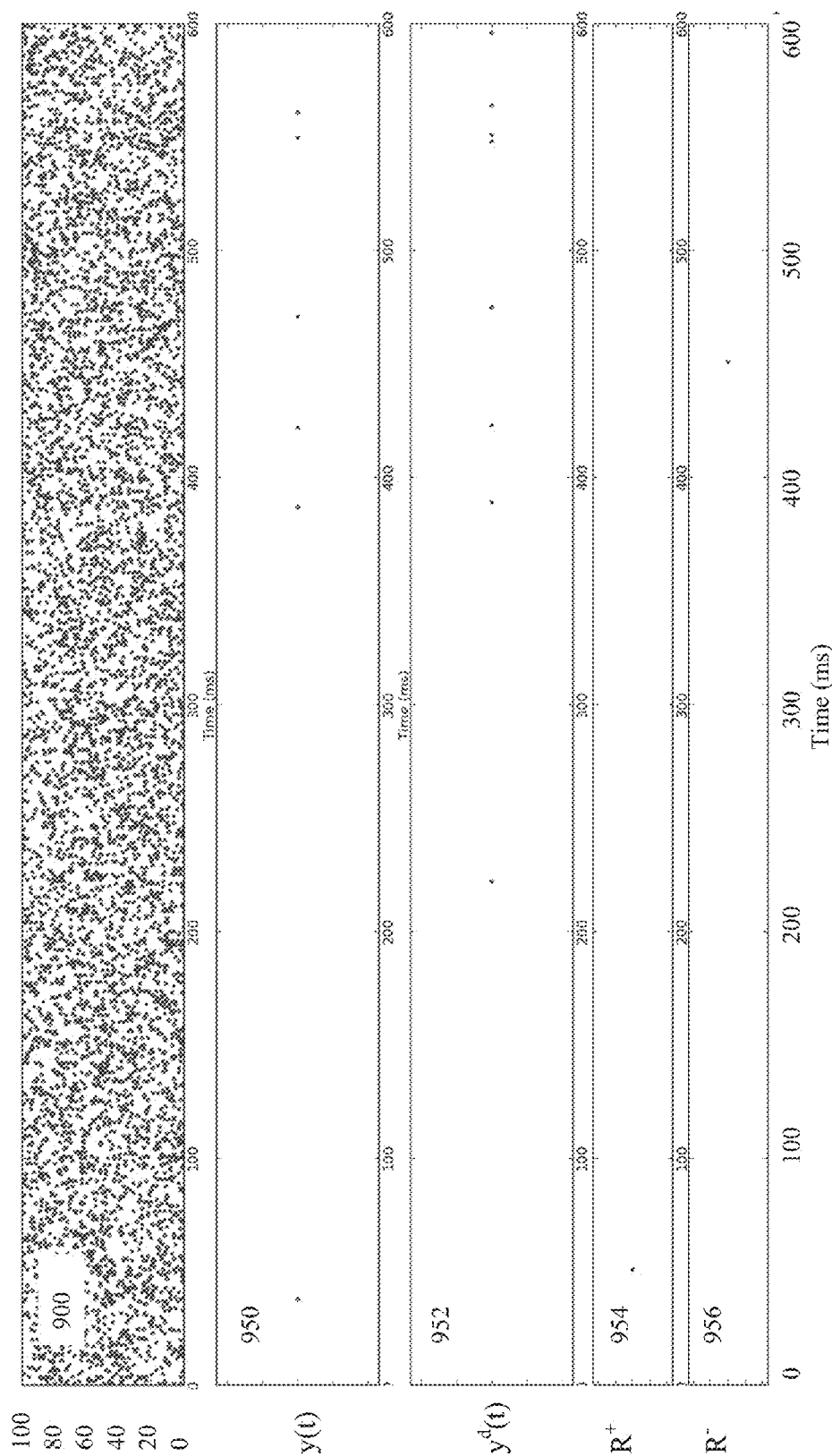
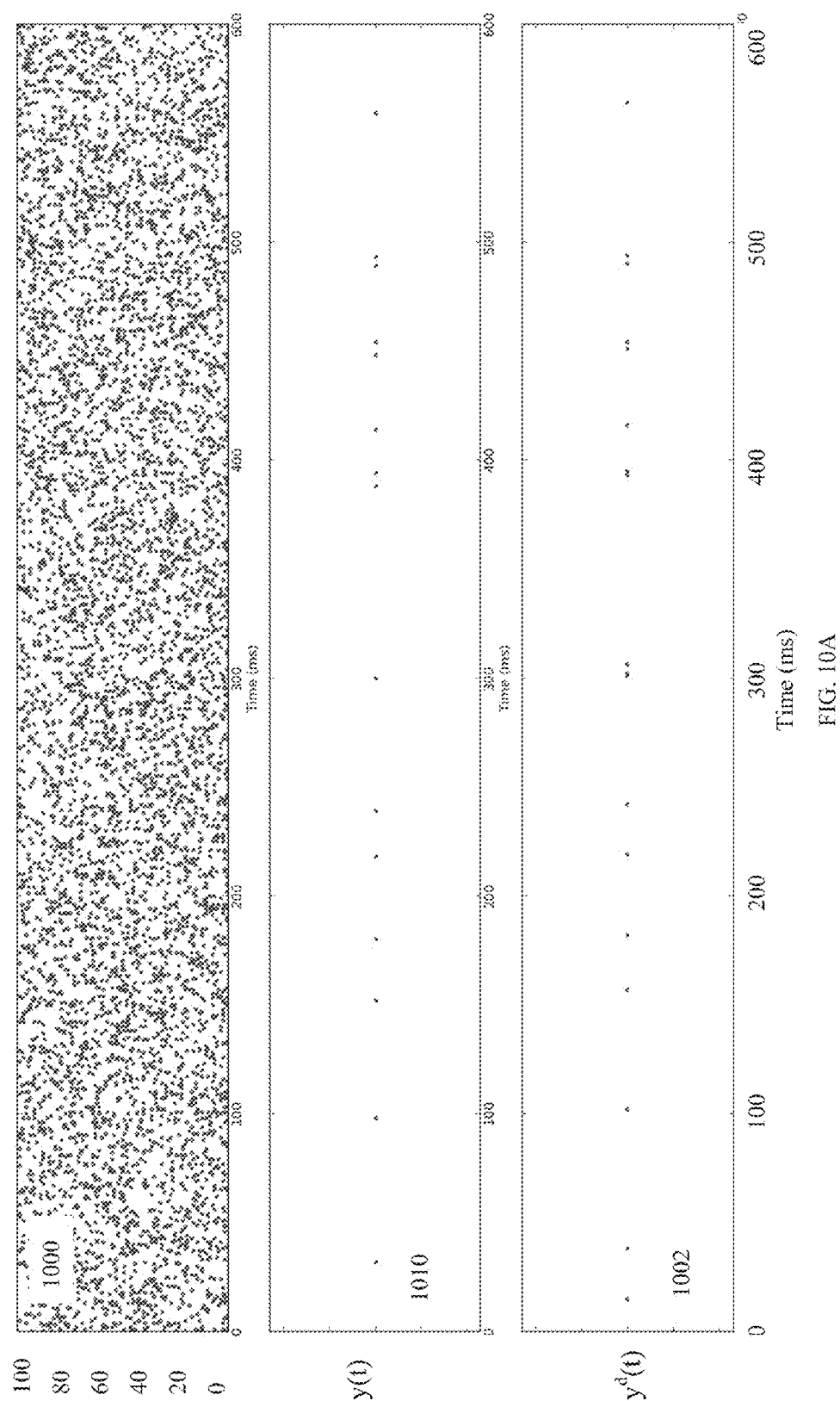
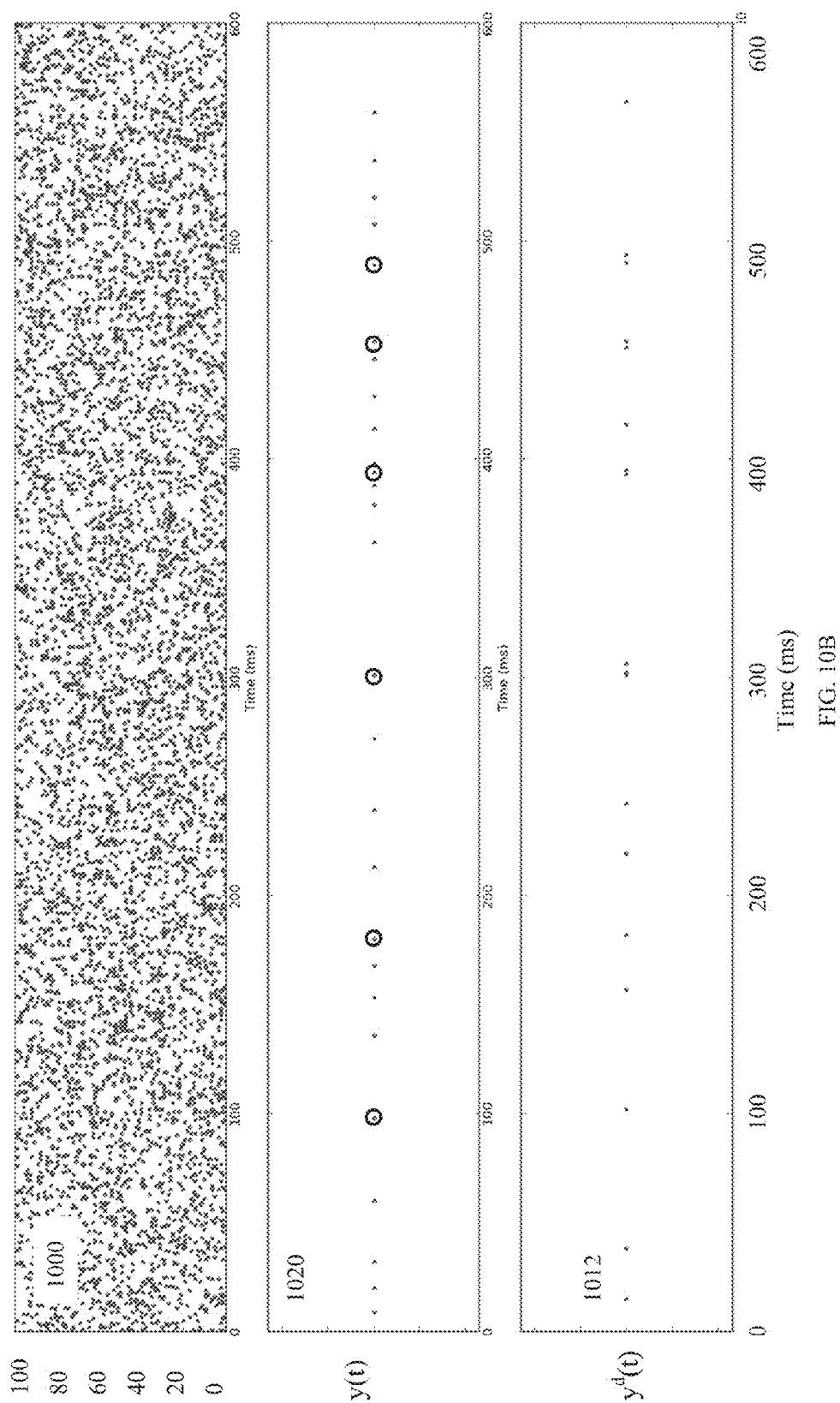


FIG. 9F





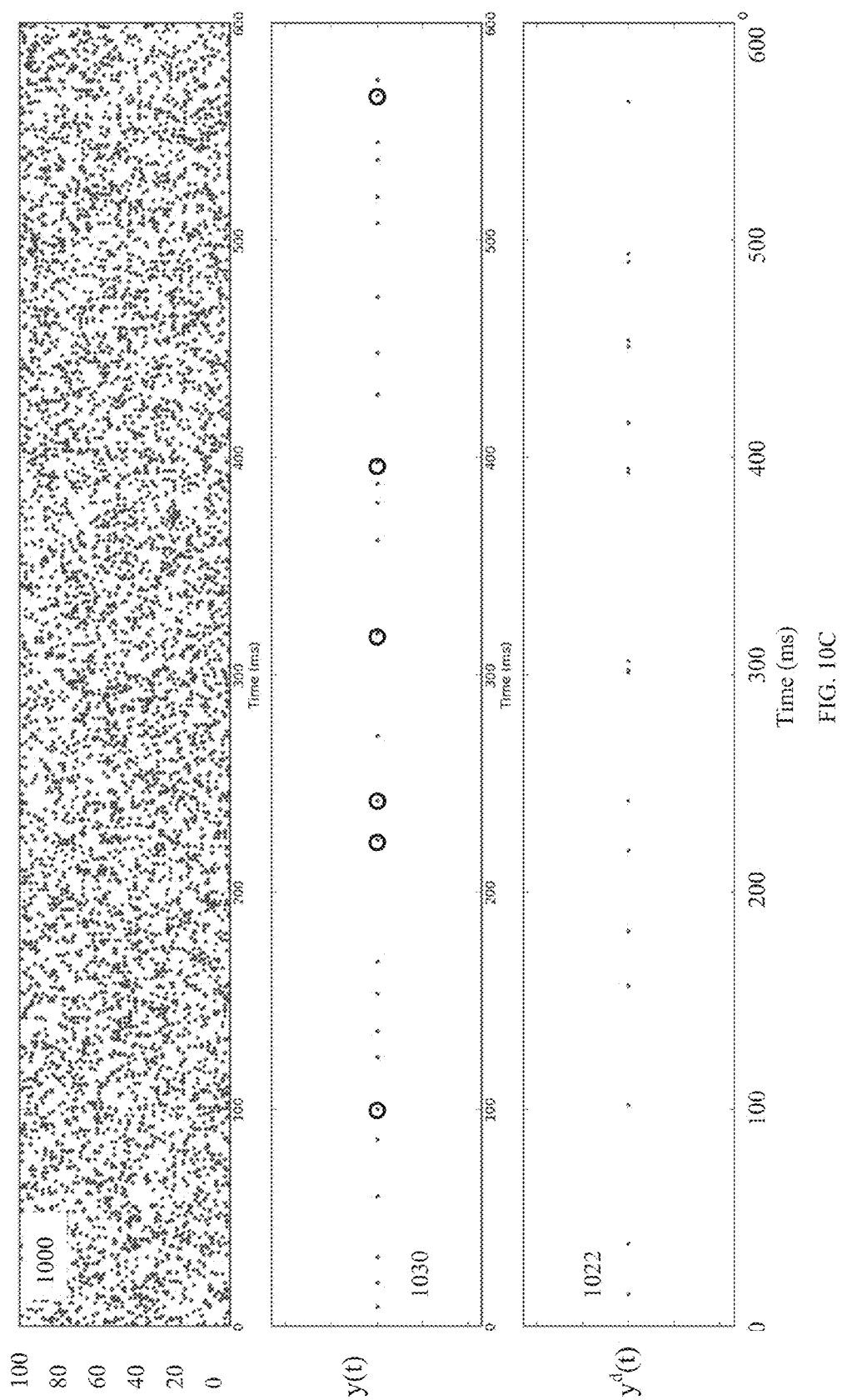


FIG. 10C

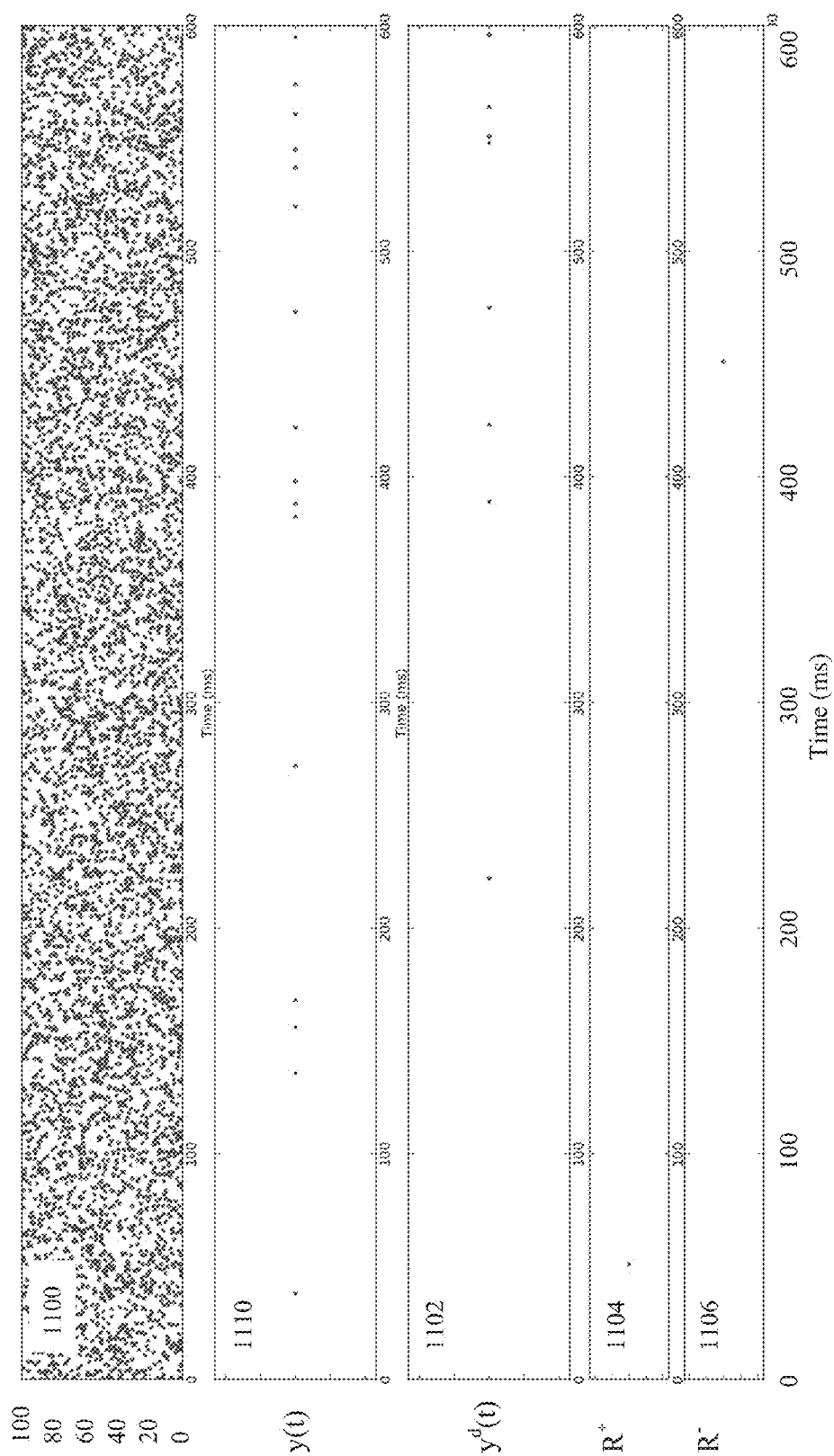


FIG. 11

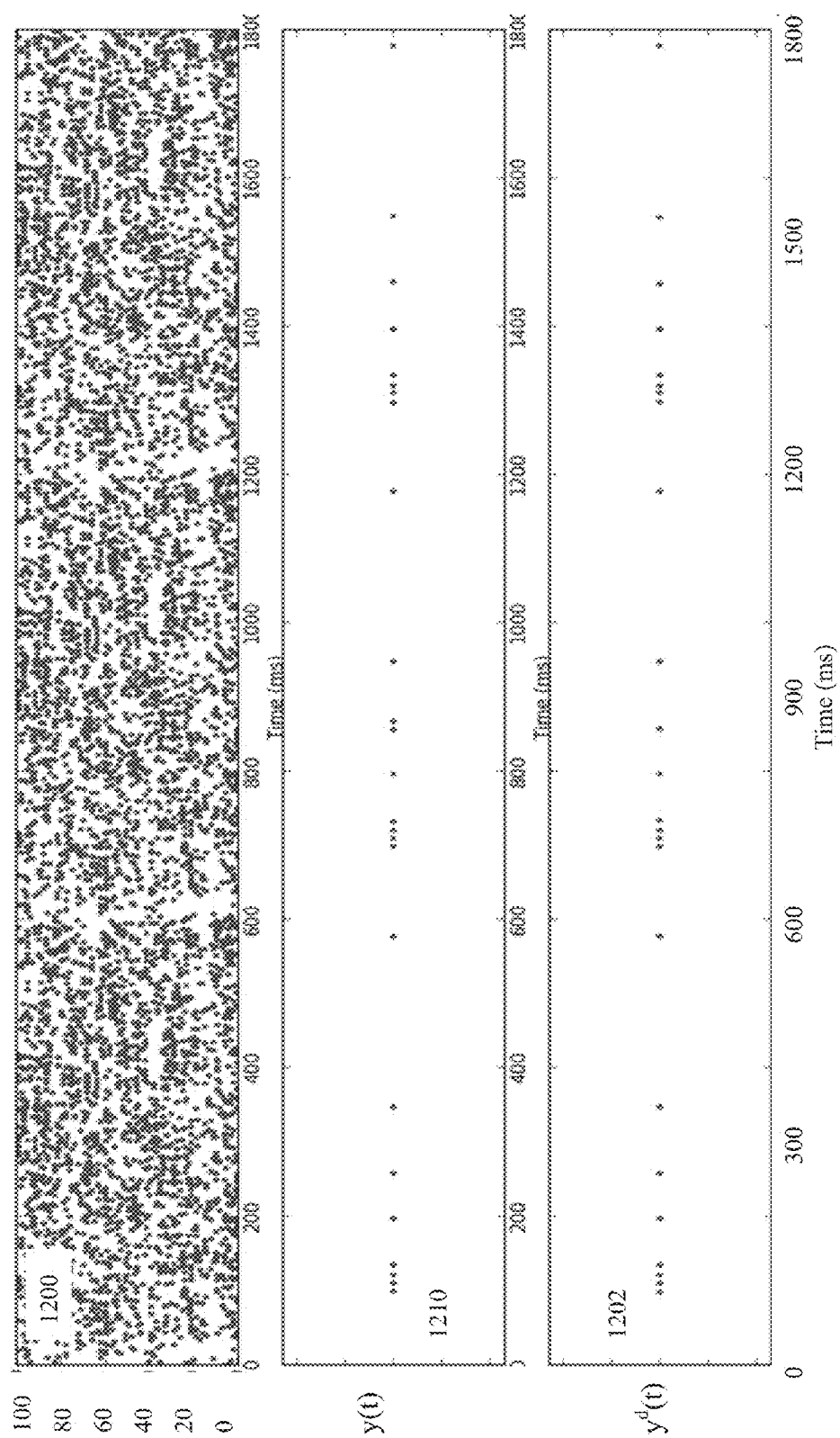


FIG. 12

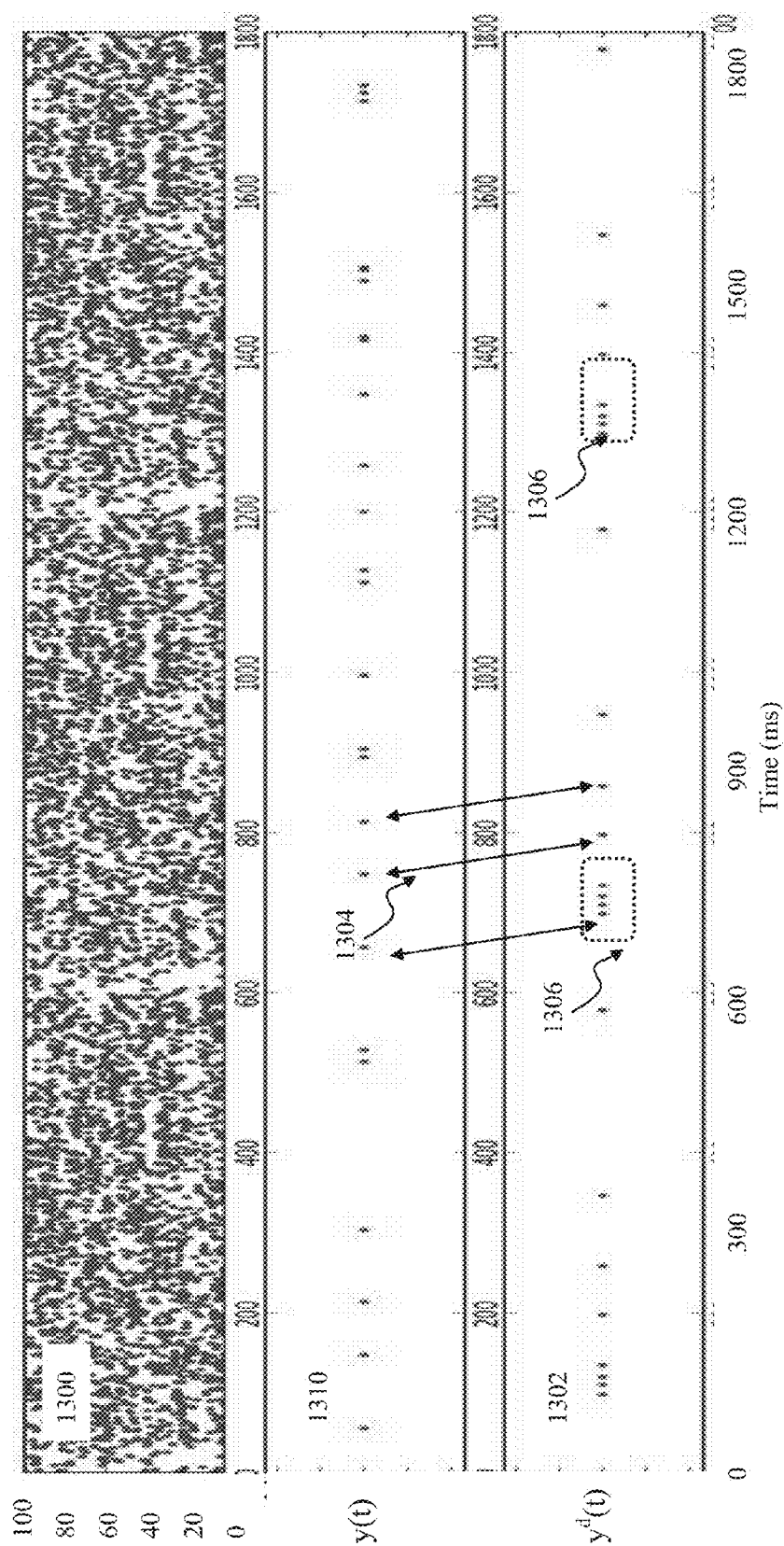


FIG. 13

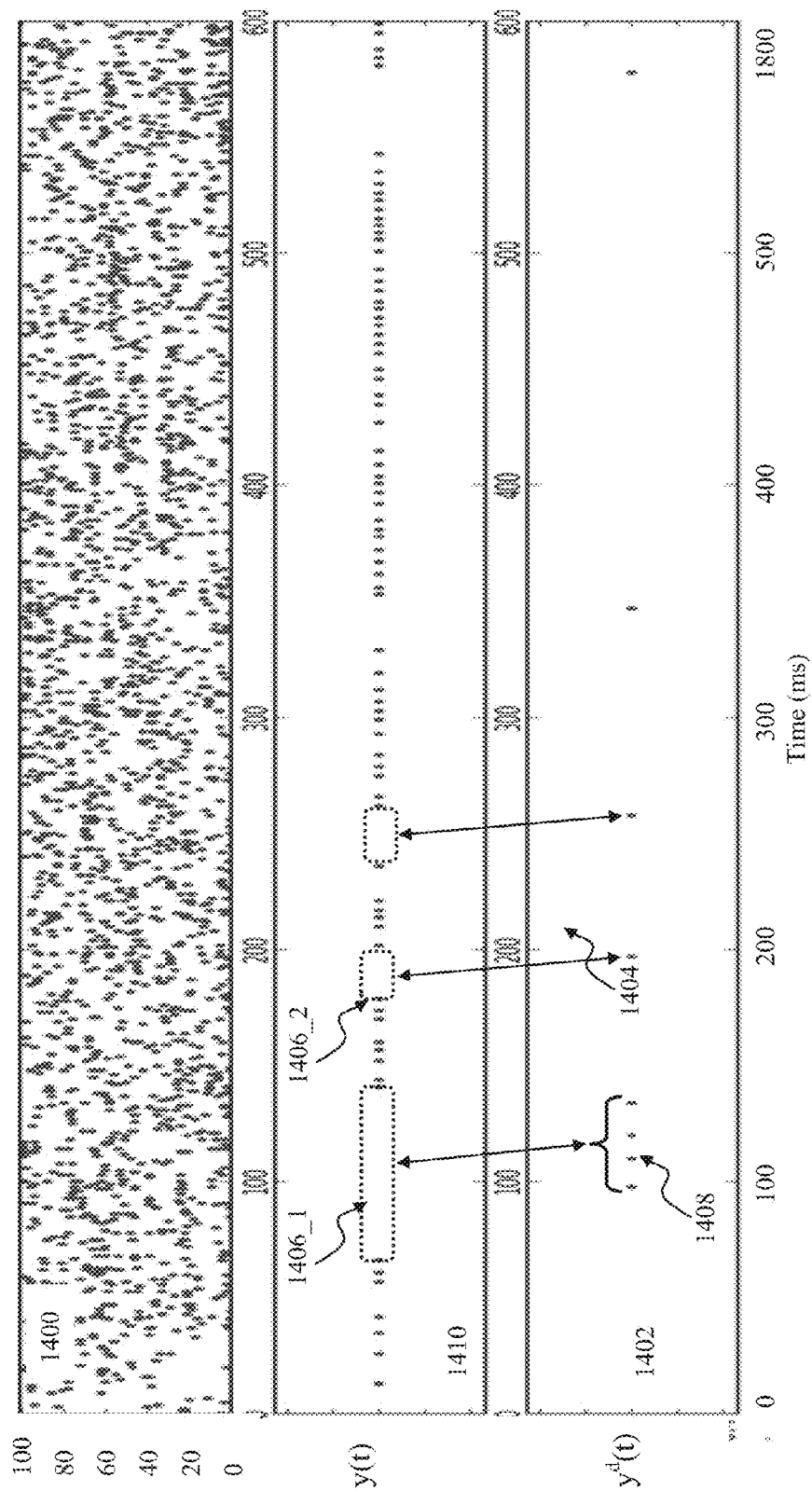


FIG. 14

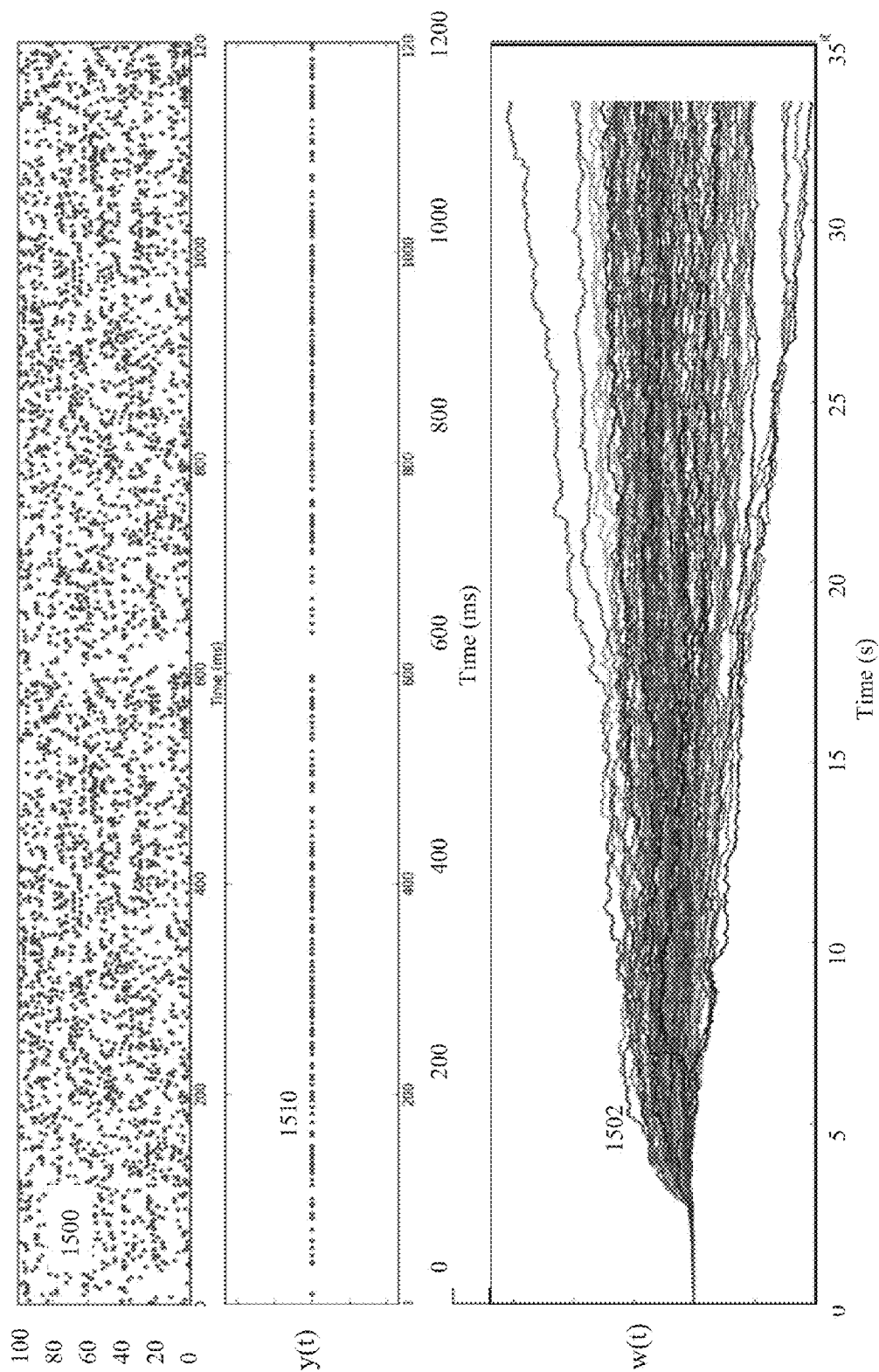


FIG. 15A

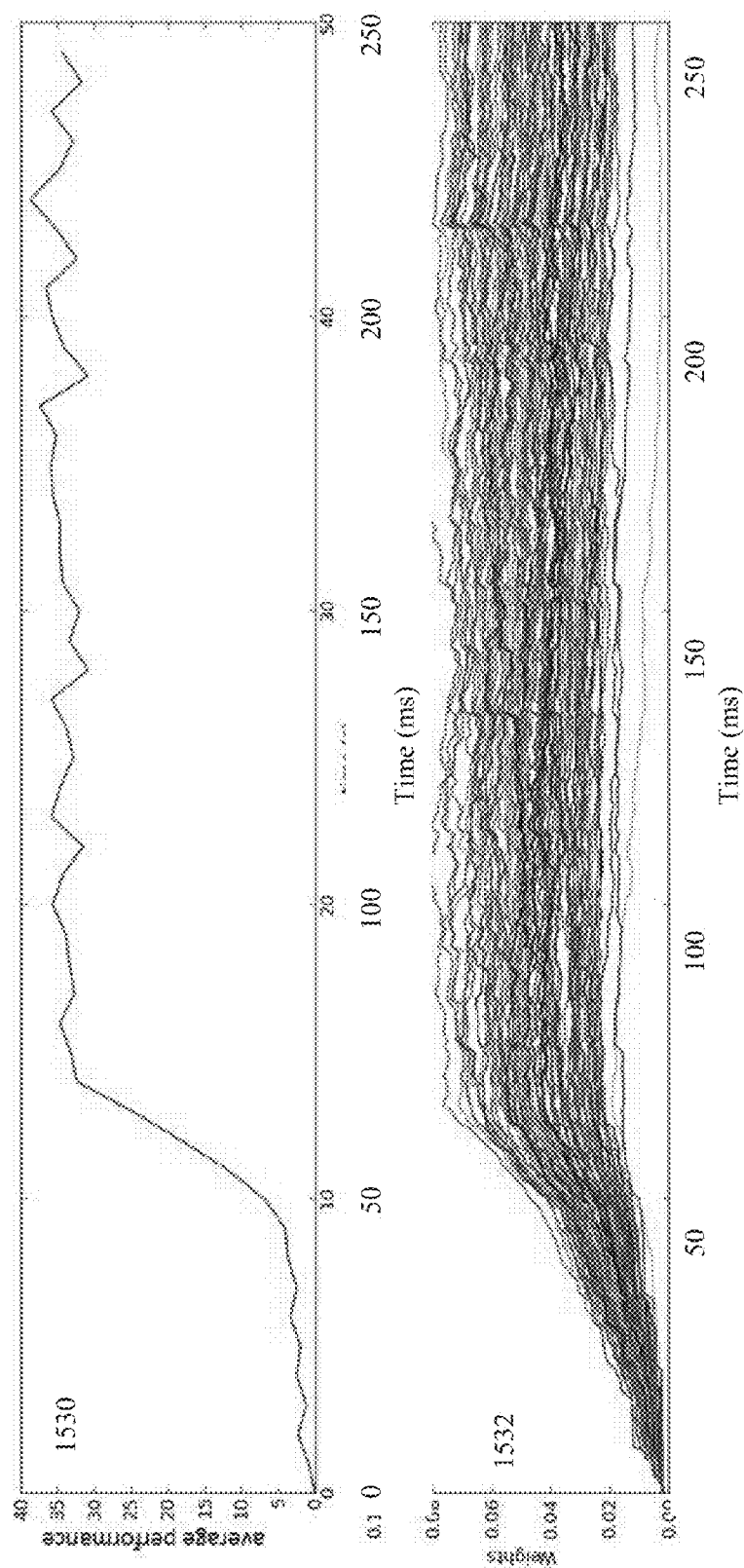


FIG. 15B

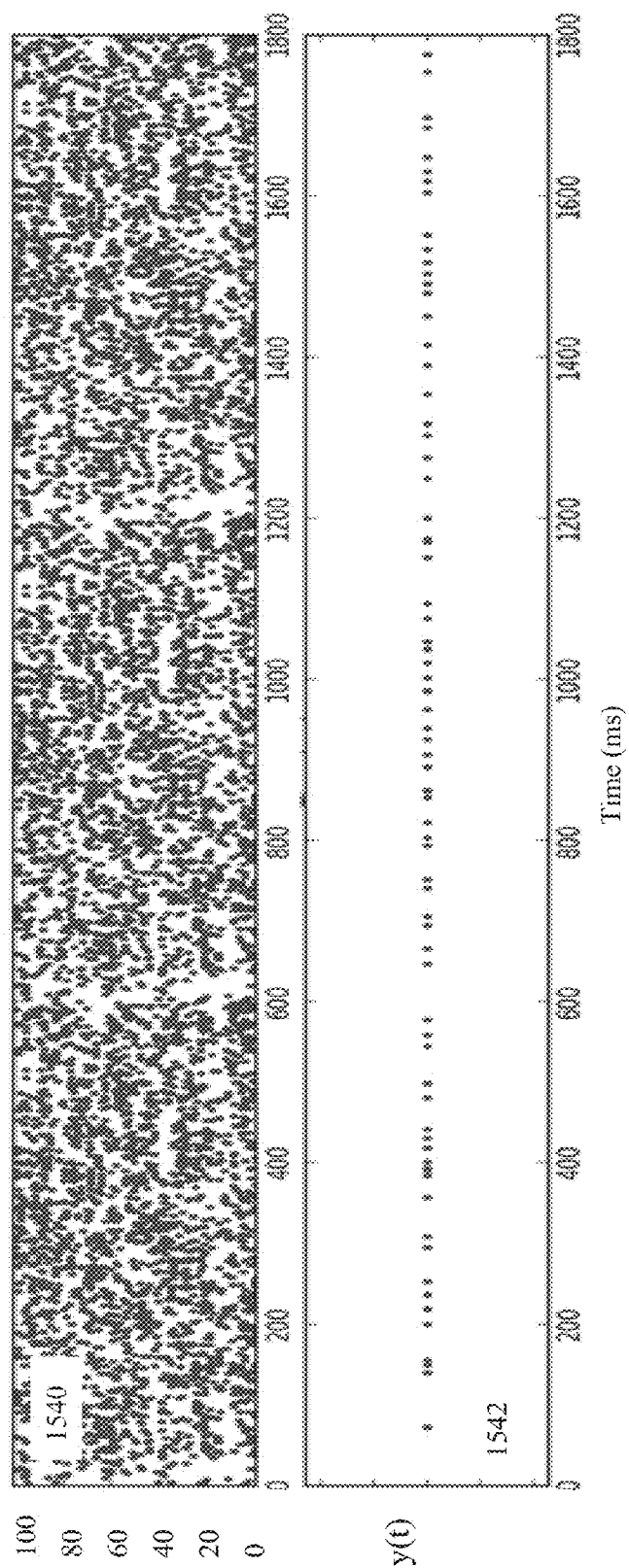


FIG. 15C

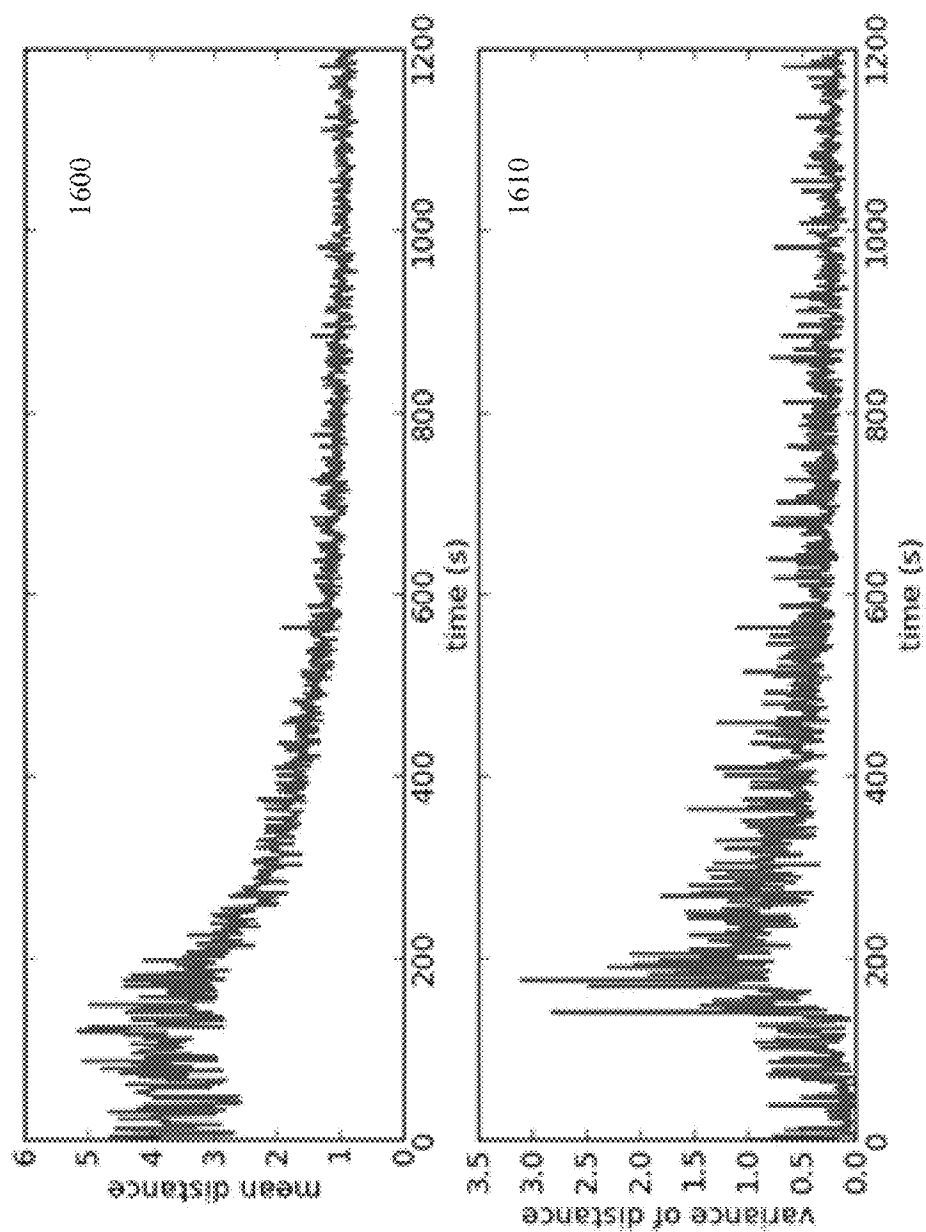


FIG. 16

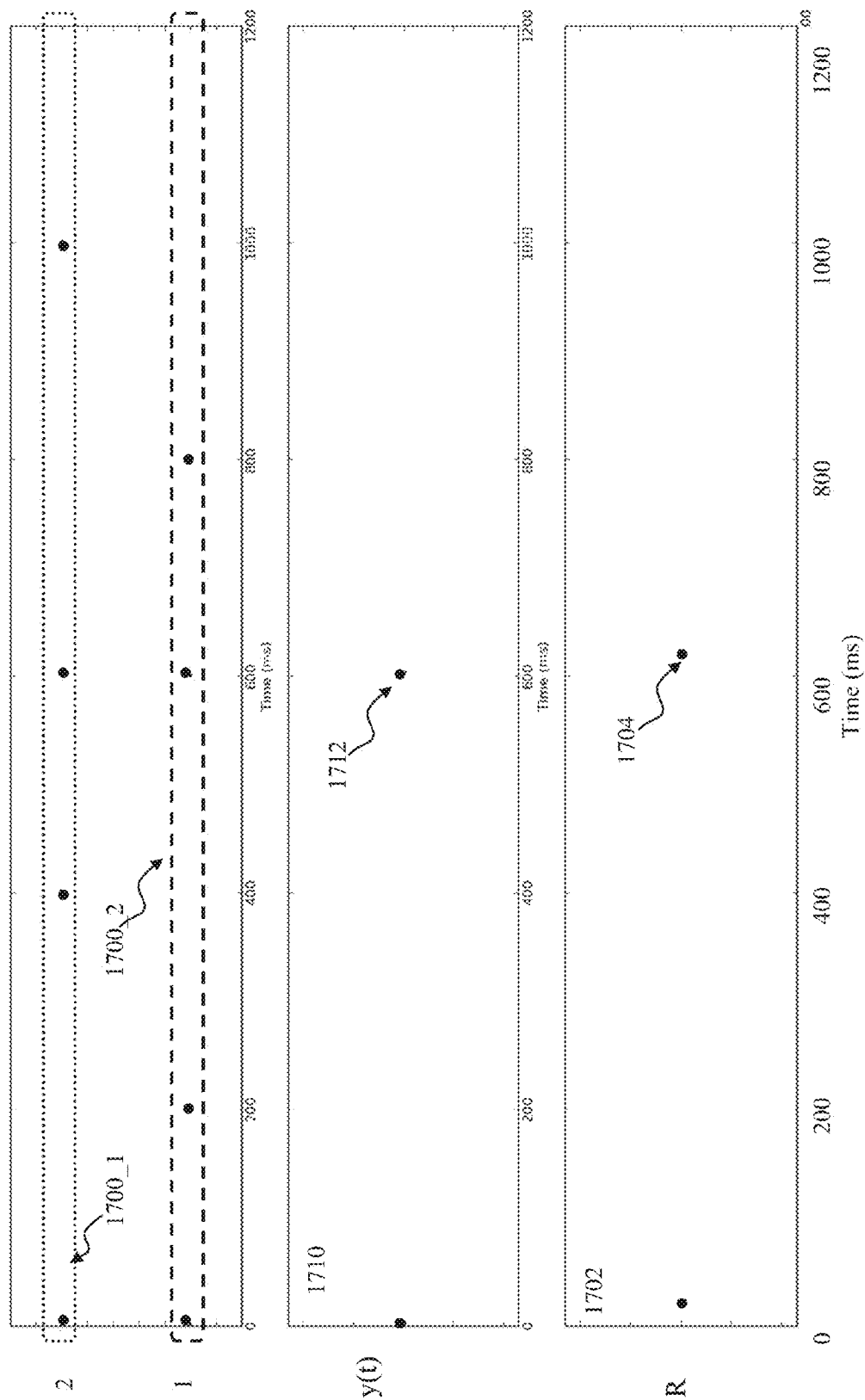


FIG. 17A

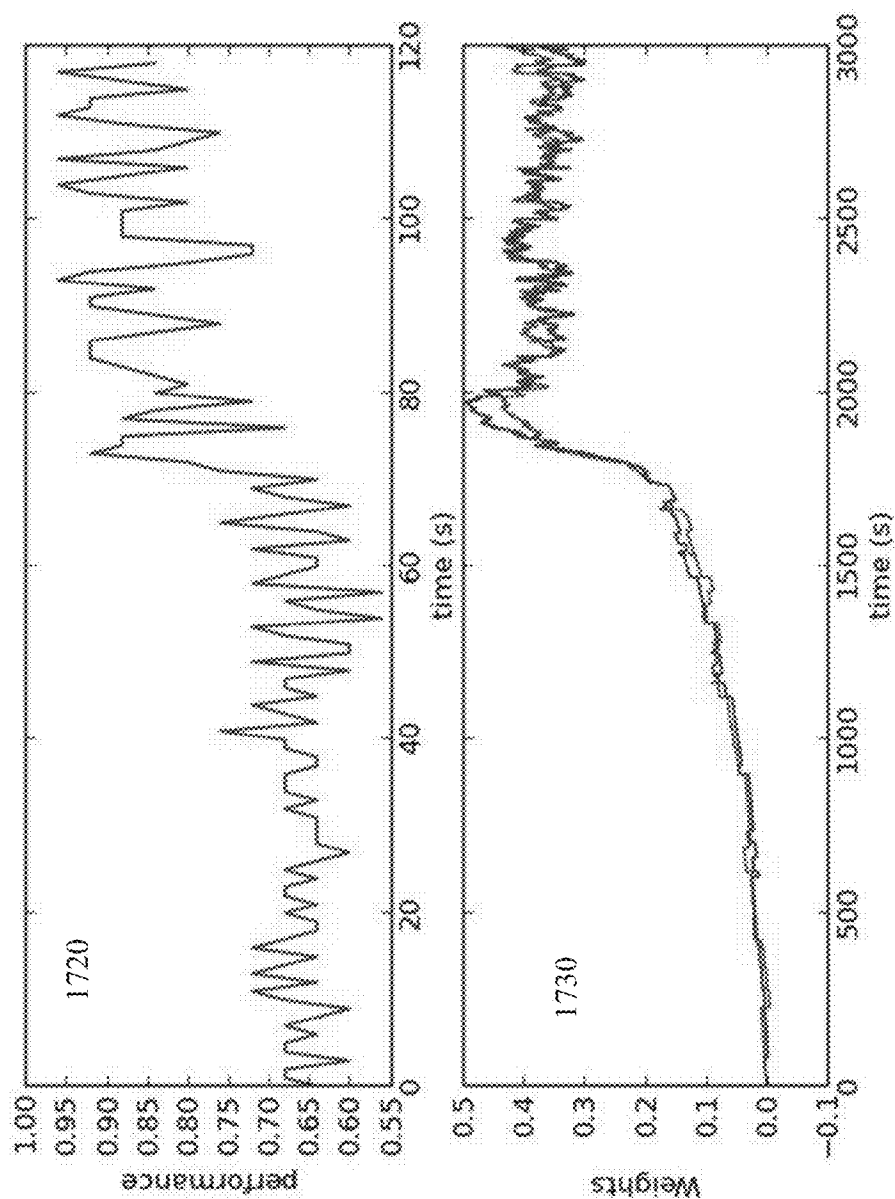


FIG. 17B

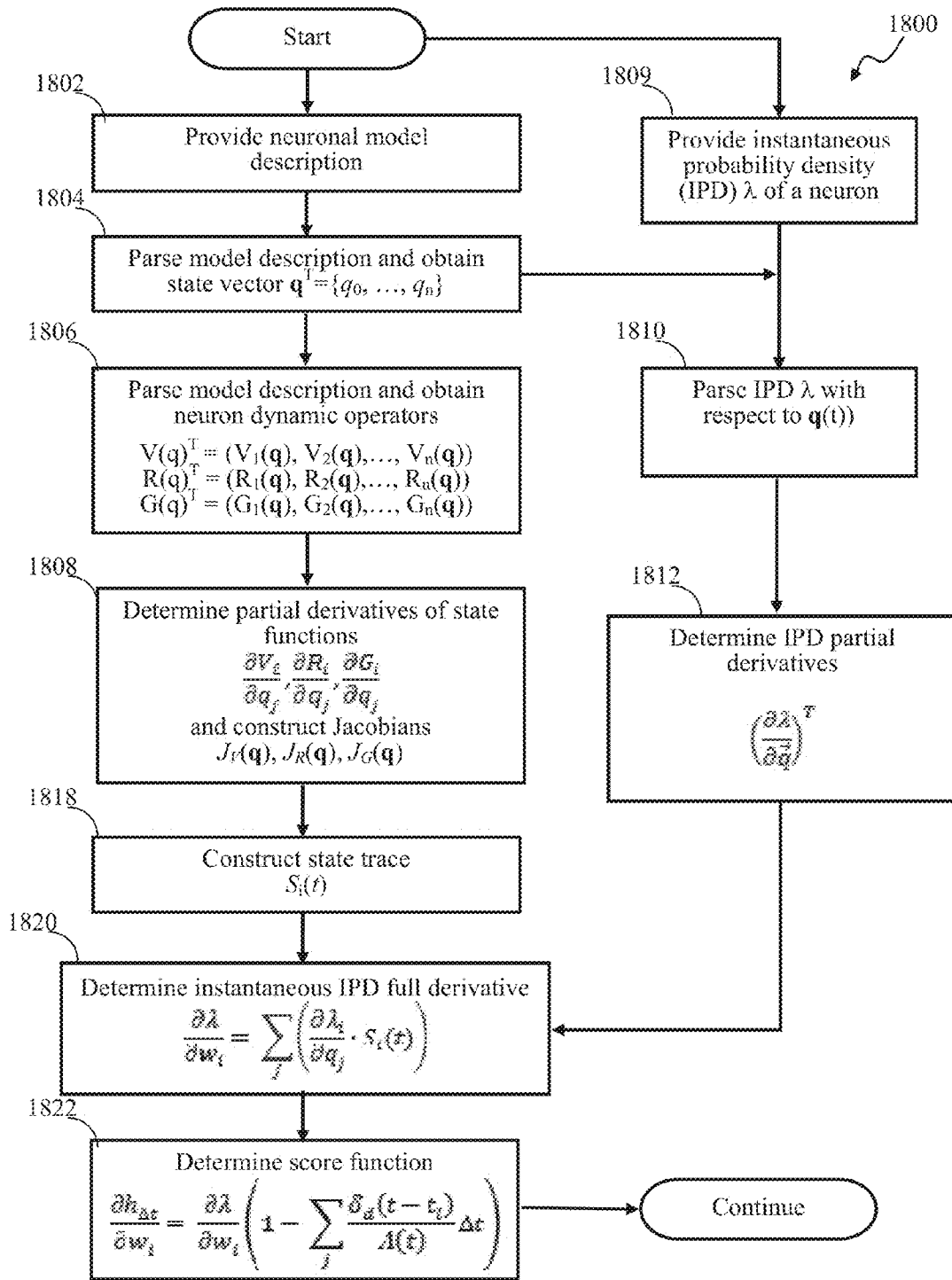


FIG. 18

```
01 parameters = {} # {'a':0.02,'b':0.2,'c':-65,'d':8}
02 neuron_equations = '''
03 dv/dt = 0.04*v**2 + 5*v + 140 - u + (c-v)*delta_out + i_ext_v
04 du/dt = a*(b*v-u) + d*delta_out
05 i_ext_v= w*eps(t - t_in)'''
06 stochastic_threshold = '''lambda = lambda0*exp(k*(v-theta))''' #1810
07 state_traces = statetracesmodel(neuron_equations, parameters) #1802,
    1804
08 state_traces.process()#1806, 1808
09 derived_state_trace = state_traces.get_eqs_str()#1818
10 threshold_model = stochasticthresholdmodel( stochastic_threshold,
    parameters)
11 threshold_model.calc_partial_derivatives(
    state_traces.get_state_vars())#1812
12 threshold_model.calc_full_derivative()#1820
13 IPD_derivative = threshold_model.get_eqs_str() #1822
14 score_function = get_score_function_equations() #1822
```

FIG. 19A

```
01 import sympy
02 import re

03 class StateVariableEquation():
04     def __init__(self, name, eq, params):
05         self.name = name
06         self.eq = sympy.simplify(eq)

07     for name, val in params.iteritems():
08         x = sympy.Symbol(name)
09         self.eq = self.eq.subs(x, val)

10     def extract_V_R_G(self):
11         I_ext = sympy.Symbol('I_ext_' + self.name)
12         delta_out = sympy.Symbol('delta_out')

13         d = sympy.collect(self.eq, [delta_out, I_ext], evaluate=False)
14         try:
15             self.R = d[delta_out]
16         except:
17             self.R = 0

18         try:
19             self.G = d[I_ext]
20         except:
21             self.G = 0

22         self.V = self.eq
23         self.V = self.V.subs(delta_out, 0)
24         self.V = self.V.subs(I_ext, 0)

25     def calc_jacobians(self, vars, I_ext):
26         self.J_V = {}
27         for v in vars:
28             var = sympy.Symbol(v)
29             self.J_V[v] = sympy.diff(self.V, var)

30         self.J_R = {}
31         for v in vars:
32             var = sympy.Symbol(v)
33             self.J_R[v] = sympy.diff(self.R, var)

34         self.J_G = {}
35         for v in vars:
36             var = sympy.Symbol(v)
37             self.J_G[v] = sympy.diff(self.G, var)

38         self.I_ext = sympy.simplify(I_ext)
39         w = sympy.Symbol('w')
40         self.dI_ext = sympy.diff(self.I_ext, w)
```

```
41 def calc_state_traces_eq(self):
42 self.state_evolution_eq = sympy.simplify(0)
43 for v, j_v in self.J_V.iteritems():
44 self.state_evolution_eq += sympy.Symbol('s_'+v)*j_v
45
46 for v, j_g in self.J_G.iteritems():
47 self.state_evolution_eq += sympy.Symbol('s_'+v)*j_g*self.I_ext
48
49 self.reset_eq = sympy.simplify(0)
50 for v, j_r in self.J_R.iteritems():
51 self.reset_eq += sympy.Symbol('s_'+v)*j_r
52
53 self.input_eq = self.G*self.dI_ext
54
55 self.full_eq = self.state_evolution_eq +
56 self.reset_eq*sympy.Symbol('delta_out') + self.input_eq
57
58 def get_diff_eq_str(self):
59 return 'ds_%s/dt = %s' % (self.name, self.full_eq)
60
61 class StateTracesModel():
62 def __init__(self, neuron_eqs, parameters):
63 separated_equations = re.sub('\\\\s*?\\n', ' ', neuron_eqs)
64 diffeq_pattern = re.compile('\\s*d(\\w+)\\s*/\\s*dt\\s*=\\s*(.*)\\s*')
65 empty_pattern = re.compile('\\s*$')
66 ext_I_pattern = re.compile('\\s*I_ext_(\\w+)\\s*=\\s*(.*)\\s*')
67
68 self.sym_eqs = {}
69 self.I_ext_eqs = {}
70
71 self.parameters = parameters
72
73 self.variables_in_written_order = []
74
75 for line in separated_equations.splitlines():
76 if empty_pattern.match(line):
77 continue
78 result = None
79 result = diffeq_pattern.match(line)
80 if result:
81 name, eq = result.groups()
82 eq = StateVariableEquation(name, eq, parameters)
83 self.sym_eqs[name] = eq
84 self.variables_in_written_order.append(name)
85 else:
86 result = ext_I_pattern.match(line)
87 if result:
88 name, eq = result.groups()
89 self.I_ext_eqs[name] = eq
90
91 if not result:
92 raise Exception('Equation was not recognized!')
```

```
81 def process(self):
82     for eq in self.sym_eqs.itervalues():
83         eq.extract_V_R_G()

84     for name, eq in self.sym_eqs.iteritems():
85         try:
86             I_ext = self.I_ext_eqs[name]
87         except:
88             I_ext = 0
89         eq.calc_jacobians(self.sym_eqs.keys(), I_ext)

90     for eq in self.sym_eqs.itervalues():
91         eq.calc_state_traces_eq()

92     def get_eqs_str(self):
93         S_equations = ''
94         for var in self.variables_in_written_order:
95             S_equations += self.sym_eqs[var].get_diff_eq_str() + '\n'

96     return S_equations[0:-1]

97     def get_state_vars(self):
98         return self.variables_in_written_order

99     class StochasticThresholdModel():
100         def __init__(self, stochastic_thresh_eqs, params):
101             separated_equations = re.sub('\\\\s*?\n', ' ',
102                                           stochastic_thresh_eqs)

102         lambda_pattern = re.compile('\s*lambda\s*=\s*(.*)\s*')
103         empty_pattern = re.compile('\s*$')

104         self.lambda_eq = ''

105         for line in separated_equations.splitlines():
106             if empty_pattern.match(line):
107                 continue
108             result = lambda_pattern.match(line)
109             if not result:
110                 raise Exception('Equation was not recognized!')
111             self.lambda_eq = result.groups()[0]
112             break

113         self.lambda_eq = sympy.sympify(self.lambda_eq)

114         for name, val in params.iteritems():
115             x = sympy.Symbol(name)
116             self.lambda_eq = self.lambda_eq.subs(x, val)

117         def calc_partial_derivatives(self, state_vars):
118             self.dlambda = {}
```

```
119     for v in state_vars:
120         var = sympy.Symbol(v)
121         self.dlambda[v] = sympy.diff(self.lambda_eq, var)

122     def calc_full_derivative(self):
123         self.result_eq = sympy.simplify(0)
124         for v, eq in self.dlambda.iteritems():
125             var = sympy.Symbol('s_'+v)
126             self.result_eq += var*eq

127     def get_eqs_str(self):
128         return 'dlambda = ' + str(self.result_eq)

129     def get_score_function_equations():
130         str = 'dh = dlambda*(1 - dt*delta_out/Lambda_big)'
131         return str
```

FIG. 19B

STOCHASTIC SPIKING NETWORK LEARNING APPARATUS AND METHODS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is related to a co-owned and co-pending U.S. patent application Ser. No. 13/XXX,XXX entitled “STOCHASTIC APPARATUS AND METHODS FOR IMPLEMENTING GENERALIZED LEARNING RULES”, [attorney docket 021672-0405921, client reference BC201202A] filed contemporaneously herewith, co-owned U.S. patent application Ser. No. 13/XXX,XXX entitled “IMPROVED LEARNING STOCHASTIC APPARATUS AND METHODS”, [attorney docket 021672-0407763, client reference BC201208A], filed contemporaneously herewith, and co-owned U.S. patent application Ser. No. 13/XXX,XXX entitled “DYNAMICALLY RECONFIGURABLE STOCHASTIC SPIKING NETWORK APPARATUS AND METHODS”, [attorney docket 021672-0407729, client reference BC201211A], filed contemporaneously herewith, each of the foregoing incorporated herein by reference in its entirety.

COPYRIGHT

[0002] A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND

[0003] 1. Field of the Disclosure

[0004] The present disclosure relates to implementing generalized learning rules in stochastic spiking neuron systems.

[0005] 2. Description of Related Art

[0006] Adaptive signal processing systems are well known in the arts of computerized control and information processing. One typical configuration of an adaptive system of prior art is shown in FIG. 1. The system **100** may be capable of changing or “learning” its internal parameters based on the input **102**, output **104** signals, and/or an external influence **106**. The system **100** may be commonly described using a function **110** that depends (including probabilistic dependence) on the history of inputs and outputs of the system and/or on some external signal r that is related to the inputs and outputs. The function $F(x,y,r)$ may be referred to as a “performance function”. The purpose of adaptation (or learning) may be to optimize the input-output transformation according to some criteria, where learning is described as minimization of an average value of the performance function F .

[0007] Although there are numerous models of adaptive systems, these typically implement a specific set of learning rules (e.g., supervised, unsupervised, reinforcement). Supervised learning may be the machine learning task of inferring a function from supervised (labeled) training data. Reinforcement learning may refer to an area of machine learning concerned with how an agent ought to take actions in an environment so as to maximize some notion of reward (e.g., immediate or cumulative). Unsupervised learning may refer to the problem of trying to find hidden structure in unlabeled

data. Because the examples given to the learner are unlabeled, there is no external signal to evaluate a potential solution.

[0008] When the task changes, the learning rules (typically effected by adjusting the control parameters $w = \{w_1, w_2, \dots, w_n\}$) may need to be modified to suit the new task. Hereinafter, the boldface variables and symbols with arrow superscripts denote vector quantities, unless specified otherwise. Complex control applications, such as for example, autonomous robot navigation, robotic object manipulation, and/or other applications may require simultaneous implementation of a broad range of learning tasks. Such tasks may include visual recognition of surroundings, motion control, object (face) recognition, object manipulation, and/or other tasks. In order to handle these tasks simultaneously, existing implementations may rely on a partitioning approach, where individual tasks are implemented using separate controllers, each implementing its own learning rule (e.g., supervised, unsupervised, reinforcement).

[0009] One conventional implementation of a multi-task learning controller is illustrated in FIG. 1A. The apparatus **120** comprises several blocks **120**, **124**, **130**, each implementing a set of learning rules tailored for the particular task (e.g., motor control, visual recognition, object classification and manipulation, respectively). Some of the blocks (e.g., the signal processing block **130** in FIG. 1A) may further comprise sub-blocks (e.g., the blocks **132**, **134**) targeted at different learning tasks. Implementation of the apparatus **120** may have several shortcomings stemming from each block having a task specific implementation of learning rules. By way of example, a recognition task may be implemented using supervised learning while object manipulator tasks may comprise reinforcement learning. Furthermore, a single task may require use of more than one rule (e.g., signal processing task for block **130** in FIG. 1A) thereby necessitating use of two separate sub-blocks (e.g., blocks **132**, **134**) each implementing different learning rule (e.g., unsupervised learning and supervised learning, respectively).

[0010] Artificial neural networks may be used to solve some of the described problems. An artificial neural network (ANN) may include a mathematical and/or computational model inspired by the structure and/or functional aspects of biological neural networks. A neural network comprises a group of artificial neurons (units) that are interconnected by synaptic connections. Typically, an ANN is an adaptive system that is configured to change its structure (e.g., the connection configuration and/or neuronal states) based on external or internal information that flows through the network during the learning phase.

[0011] A spiking neuronal network (SNN) may be a special class of ANN, where neurons communicate by sequences of spikes. SNN may offer improved performance over conventional technologies in areas which include machine vision, pattern detection and pattern recognition, signal filtering, data segmentation, data compression, data mining, system identification and control, optimization and scheduling, and/or complex mapping. Spike generation mechanism may be a discontinuous process (e.g., as illustrated by the input spikes $s_x(t)$ **220**, **222**, **224**, **226**, **228**, and output spikes $s_y(t)$ **230**, **232**, **234** in FIG. 2) and a classical derivative of function $F(s(t))$ with respect to spike trains $s_x(t)$, $s_y(t)$ is not defined.

[0012] Even when a neural network is used as the computational engine for these learning tasks, individual tasks may be performed by a separate network partition that implements a task-specific set of learning rules (e.g., adaptive control,

classification, recognition, prediction rules, and/or other rules). Unused portions of individual partitions (e.g., motor control when the robotic device is stationary) may remain unavailable to other partitions of the network that may require increased processing resources (e.g., when the stationary robot is performing face recognition tasks). Furthermore, when the learning tasks change during system operation, such partitioning may prevent dynamic retargeting (e.g., of the motor control task to visual recognition task) of the network partitions. Such solutions may lead to expensive and/or over-designed networks, in particular when individual portions are designed using the “worst possible case scenario” approach. Similarly, partitions designed using a limited resource pool configured to handle an average task load may be unable to handle infrequently occurring high computational loads that are beyond a performance capability of the particular partition, even when other portions of the networks have spare capacity.

[0013] By way of illustration, consider a mobile robot controlled by a neural network, where the task of the robot is to move in an unknown environment and collect certain resources by the way of trial and error. This can be formulated as reinforcement learning tasks, where the network is supposed to maximize the reward signals (e.g., amount of the collected resource). While in general the environment is unknown, there may be possible situations when the human operator can show to the network desired control signal (e.g., for avoiding obstacles) during the ongoing reinforcement learning. This may be formulated as a supervised learning task. Some existing learning rules for the supervised learning may rely on the gradient of the performance function. The gradient for reinforcement learning part may be implemented through the use of the adaptive critic; the gradient for supervised learning may be implemented by taking a difference between the supervisor signal and the actual output of the controller. Introduction of the critic may be unnecessary for solving reinforcement learning tasks, because direct gradient-based reinforcement learning may be used instead. Additional analytic derivation of the learning rules may be needed when the loss function between supervised and actual output signal is redefined.

[0014] While different types of learning may be formalized as a minimization of the performance function F , an optimal minimization solution often cannot be found analytically, particularly when relationships between the system's behavior and the performance function are complex. By way of example, nonlinear regression applications generally may not have analytical solutions. Likewise, in motor control applications, it may not be feasible to analytically determine the reward arising from external environment of the robot, as the reward typically may be dependent on the current motor control command and state of the environment.

[0015] Moreover, analytic determination of a performance function F derivative may require additional operations (often performed manually) for individual new formulated tasks that are not suitable for dynamic switching and reconfiguration of the tasks described before.

[0016] Some of the existing approaches of taking a derivative of a performance function without analytic calculations may include a “brute force” finite difference estimator of the gradient. However, these estimators may be impractical for use with large spiking networks comprising many (typically in excess of hundreds) parameters.

[0017] Derivative-free methods, specifically Score Function (SF), also known as Likelihood Ratio (LR) method, exist. In order to determine a direction of the steepest descent, these methods may sample the value of $F(x,y)$ in different points of parameter space according to some probability distribution. Instead of calculating the derivative of the performance function $F(x,y)$, the SR and LR methods utilize a derivative of the sampling probability distribution. This process can be considered as an exploration of the parameter space.

[0018] Although some adaptive controller implementations may describe reward-modulated unsupervised learning algorithms, these implementations of unsupervised learning algorithms may be multiplicatively modulated by reinforcement learning signal and, therefore, may require the presence of reinforcement signal for proper operation.

[0019] Many presently available implementations of stochastic adaptive apparatuses may be incapable of learning to perform unsupervised tasks while being influenced by additive reinforcement (and vice versa). Many presently available adaptive implementations may be task-specific and implement one particular learning rule (e.g., classifier unsupervised learning), and such devices invariably require retargeting (e.g., reprogramming) in order to implement different learning rules. Furthermore, presently available methodologies may not be capable of implementing generalized learning, where a combination of different learning rules (e.g., reinforcement, supervised and supervised) are used simultaneously for the same application (e.g., platform motion stabilization), thereby enabling, for example, faster learning convergence, better response to sudden changes, and/or improved overall stability, particularly in the presence of noise.

Stochastic Spiking Neuron Models

[0020] Where certain elements of these implementations can be partially or fully implemented using known components, only those portions of such known components that are necessary for an understanding of the present disclosure will be described, and detailed descriptions of other portions of such known components will be omitted so as not to obscure the disclosure.

[0021] Learning rules used with spiking neuron networks may be typically expressed in terms of original spike trains instead of their secondary features (e.g., the rate or the latency from the last spike). The result is that a spiking neuron operates on spike train space, transforming a vector of spike trains (input spike trains) into single element of that space (output train). Dealing with spike trains directly may be a challenging task. Not every spike train can be transformed to another spike train in a continuous manner. One common approach is to describe the task in terms of optimization of some function and then use gradient approaches in the parameter space of the spiking neuron. However gradient methods on discontinuous spaces such as spike trains space are not well developed. One approach may involve smoothing the spike trains first. Here output spike trains are smoothed with introduction of probabilistic measure on a spike trains space. Describing the spike pattern from a probabilistic point of view may lead to fruitful connections with the huge amount of topics within information theory, machine learning, Bayesian inference, statistical data analysis etc. This approach makes spiking neurons a good candidate to use SF/LR learning methods.

[0022] One technique frequently used when constructing learning rules in a spiking network, comprises application of

a random exploration process to a spike generation mechanism of a spiking neuron. This is often implemented by introducing a noisy threshold: probability of a spike generation may depend on the difference between neuron's membrane voltage and a threshold value. The usage of probabilistic spiking neuron models, in order to obtain gradient of the log-likelihood of a spike train with respect to neuron's weights, may comprise an extension of Hebbian learning framework to spiking neurons. The use of the log-likelihood gradient of a spike train may be extended to supervised learning. In some approaches, information theory framework may be applied to spiking neurons, as for example, when deriving optimal learning rules for unsupervised learning tasks via informational entropy minimization.

[0023] An application of the OLPOMDM algorithm to the solution of the reinforcement learning problems with simplified spiking neurons has been done. Extending of this algorithm to more plausible neuron model has been done. However no generalizations of the OLPOMDM algorithm have been done in order to use it unsupervised and supervised learning in spiking neurons. An application of reinforcement learning ideas to supervised learning has been described, however only heuristic algorithms without convergence guarantees have been used.

[0024] For a neuron, the probability of an output spike train, y , to have spikes at times t_f with no spikes at the other times on a time interval $[0, T]$, given the input spikes, x , may be given by the conditional probability density function $p(y|x)$ as:

$$p(y|x) = \prod_{t_f} \lambda(t_f) e^{-\int_0^T \lambda(\tau) d\tau} \quad (\text{Eqn. 1})$$

where $\lambda(t)$ represents an instantaneous probability density ("hazard") of firing.

[0025] The instantaneous probability density of the neuron can depend on neuron's state $q(t)$: $\lambda(t) = \lambda(q(t))$. For example, it can be defined according to its membrane voltage $u(t)$ for continuous time chosen as an exponential stochastic threshold:

$$\lambda(t) = \lambda_0 e^{\kappa(u(t) - \theta)} \quad (\text{Eqn. 2})$$

where $u(t)$ is the membrane voltage of the neuron, θ is the voltage threshold for generating a spike, κ is the probabilistic parameter, and λ_0 is the basic (spontaneous) firing rate of the neuron.

[0026] Some approaches utilize sigmoidal stochastic threshold, expressed as:

$$\lambda(t) = \frac{\lambda_0}{1 - e^{-\kappa(u(t) - \theta)}} \quad (\text{Eqn. 3})$$

or an exponential-linear stochastic threshold:

$$\lambda(t) = \lambda_0 \ln(1 + e^{\kappa(u(t) - \theta)}) \quad (\text{Eqn. 4})$$

where λ_0 , κ , θ are parameters with a similar meaning to the parameters in the exponential threshold model Eqn. 2.

[0027] Models of the stochastic threshold exist comprising refractory mechanisms that modulate the instantaneous probability of firing after the last output spike $\lambda(t) = \hat{\lambda}(t) R(t, t_{last}^{out})$, where $\hat{\lambda}(t)$ is the original stochastic threshold function (such as exponential or other), and $R(t, t_{last}^{out} - t)$ is the dynamic refractory coefficient that depends on the time since the last output spike t_{last}^{out} .

[0028] For discrete time steps, an approximation for the probability $\Lambda(u(t)) \in (0, 1)$ of firing in the current time step may be given by:

$$\Lambda(u(t)) = 1 - e^{-\lambda(u(t)) \Delta t} \quad (\text{Eqn. 5})$$

where Δt is time step length.

[0029] In one dimensional deterministic spiking models, such as Integrate-and-Fire (IF), Quadratic Integrate-and-Fire (QIF) and others, membrane voltage $u(t)$ is the only one state variable ($q(t) = u(t)$) that is "responsible" for spike generation through deterministic threshold mechanism. There also exist plenty of more complex multidimensional spiking models. For example, a simple spiking model may comprise two state variables where only one of them is compared with a threshold value. However, even detailed neuron models may be parameterized using a single variable (e.g., an equivalent of "membrane voltage" of biological neuron) and use it with a suitable threshold in order to determine the presence of spike. Such models are often extended to describe stochastic neurons by replacing deterministic threshold with a stochastic threshold.

[0030] Generalized dynamics equations for spiking neurons models are often expressed as a superposition of input, interaction between the input current and the neuronal state variables, and neuron reset after the spike as follows:

$$\frac{d\vec{q}}{dt} = V(\vec{q}) + \sum_{i \in \text{out}} R(\vec{q}) \delta(t - t^{out}) + G(\vec{q}) I^{ext} \quad (\text{Eqn. 6})$$

where: $\vec{q}$ is a vector of internal state variables (e.g., comprising membrane voltage); I^{ext} is external input to the neuron; V —is the function that defines evolution of the state variables; G describes the interaction between the input current and the state variables (for example, to model synaptic depletion); and R describes resetting the state variables after the output spikes at t^{out} .

[0031] For example, for IF model the state vector and the state model may be expressed as:

$$\vec{q} = u(t); V(\vec{q}) = -Cu; R(\vec{q}) = u^{res} - u; G(\vec{q}) = 1, \quad (\text{Eqn. 7})$$

where C is a membrane constant, and u_{res} is the value to which voltage is set after output spike (reset value). Accordingly, Eqn. 6 becomes:

$$\frac{du}{dt} = -Cu + \sum_{i \in \text{out}} (u_{refr} - u) \delta(t - t^{out}) + I^{ext} \quad (\text{Eqn. 8})$$

[0032] For some simple neuron models, Eqn. 6 may be expressed as:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + \sum_{i \in \text{out}} (c - v) \delta(t - t^{out}) + I^{ext} \quad (\text{Eqn. 9})$$

$$\frac{du}{dt} = a(bv - u) + d \sum_{i \in \text{out}} \delta(t - t^{out})$$

where

$$\vec{q} = \begin{pmatrix} v(t) \\ u(t) \end{pmatrix}; V(\vec{q}) = \begin{pmatrix} 0.04v^2(t) - 5v(t) + 140 - u(t) \\ a(bv(t) - u(t)) \end{pmatrix}; \quad (\text{Eqn. 10})$$

$$R(\vec{q}) = \begin{pmatrix} c - v(t) \\ d \end{pmatrix}; G(\vec{q}) = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

and a , b , c , d are parameters of the model.

[0033] Many presently available implementations of stochastic adaptive apparatuses may be incapable of learning to perform unsupervised tasks while being influenced by additive reinforcement (and vice versa). Many presently available adaptive implementations may be task-specific and implement one particular learning rule (e.g., classifier unsupervised learning), and such devices invariably require retargeting (e.g., reprogrammed) in order to implement different learning rules.

[0034] Accordingly, there is a salient need for machine learning apparatus and methods to implement generalized stochastic learning in spiking networks that is configured to handle simultaneously any learning rule combination (e.g., reinforcement, supervised, unsupervised, online, batch) and is capable of, inter alia, dynamic reconfiguration using the same set of network resources.

SUMMARY

[0035] The present disclosure satisfies the foregoing needs by providing, inter alia, apparatus and methods for implementing generalized probabilistic learning configured to handle simultaneously various learning rule combinations.

[0036] One aspect of the disclosure relates to one or more systems and/or computer-implemented methods for effectuating a spiking network stochastic signal processing system configured to implement task-specific learning. In one implementation, the system may comprise a controller apparatus configured to generate output control signal y based at least in part on input signal x , the controller apparatus characterized by a controller state parameter S , and a control parameter w ; and a learning apparatus configured to: generate an adjustment signal dw based at least in part on the input signal x , the controller state parameter S , and the output signal y ; and provide the adjustment signal dw to the controller apparatus, thereby effecting the learning where the control parameter may be configured in accordance with the task; and the adjustment signal dw may be configured to modify the control parameter based at least in part on the input signal x and the output signal y .

[0037] In some implementations, the output control signal y may comprise a spike train configured based at least in part the adjustment signal dw ; and the learning apparatus may comprise a task-specific block, configured independent from the controller state parameter, the task-specific block configured to implement the task-specific learning; and a controller-specific block, configured independent from the task-specific learning; and the task-specific learning may be characterized by a performance function, the performance function configured to effect at least unsupervised learning rule.

[0038] In some implementations, the system may further comprise a teaching interface operably coupled to the learning apparatus and configured to provide a teaching signal; the teaching signal may comprise a desired controller output signal; and the performance function may be further configured to effect a supervised learning rule, based at least in part on the desired controller output signal; and the teaching signal may further comprise a reinforcement spike train associated with current performance of the controller apparatus relative to desired performance; and the performance function may be further configured to effect a reinforcement learning rule, based at least in part on the reinforcement spike train.

[0039] In some implementations, the current performance may be based at least in part on adjustment of the control parameter from a prior state w_0 to current state w_c ; the reinforcement

may be positive when the current performance may be closer to desired performance of the controller; and the reinforcement may be negative when the current performance may be farther from the desired performance; and the task-specific learning may comprise a hybrid learning rule comprising a combination of the reinforcement, the supervised and the unsupervised learning rules simultaneous with one another.

[0040] In some implementations, the adjustment signal dw may be determined as a product of controller performance function F with a gradient of per-stimulus entropy parameter h , the gradient may be determined with respect to the control parameter w ; and per-stimulus entropy parameter h may be configured to characterize dependence of the output signal y on (i) the input signal x ; and (ii) the control parameter w ; and the per-stimulus entropy parameter may be determined based on a natural logarithm of $p(y|x,w)$, where p denotes conditional probability of signal y given signal x with respect to the control parameter w .

[0041] Another aspect of the disclosure relates to one or more apparatuses for effectuating learning in a stochastic spiking network. In one implementation, a computer readable apparatus may comprise a storage medium, the storage medium comprising a plurality of instructions to adjust a learning parameter associated with a computerized spiking neuron configured to produce output spike signal y consistent with (i) an input spike signal x , and (ii) a learning task, the instructions configured to, when executed construct time derivative representation of a trace S of a neuron state, based at least in part on the input spike signal x and a state parameter q ; obtain a realization of the trace S , based at least in part in integrating the time derivative representation; and determine adjustment dw of the parameter w , based at least in part on the trace S ; and the adjustment dw may be configured to transition the neuron state towards a target state, the target state associated with the neuron generating the output spike signal y .

[0042] In some implementations, the integrating representation may be effected via symbolic integration operation, the state parameter q may be configured to characterize time evolution of the neuron state; the realization of the trace S may comprise an analytic solution of the time derivative representation; and the construct of the time derivative representation enables to attain the integration via symbolic integration operation.

[0043] In some implementations, the state parameter q may be configured to characterize time evolution of the neuron state in accordance with a state evolution process characterized by: a response mode and a transition mode, the response mode may be associated with generating a neuronal response P ; state transition term V describing changes of neuronal state in the transition mode; state transition term R describing changes of state set in the response mode; and state transition term G describing changes of state set due to the input x ; the state parameter q may be configured to characterize neuron membrane voltage; and the input may comprise analog signal and the state transition term G may be configured to describe changes of the voltage due to the analog signal.

[0044] In some implementations, the state parameter q may comprise neuron excitability and, the time derivative representation may comprise a sum of V, R, G each multiplicatively combined with the trace S , state transition term V may comprise the trace S multiplicatively combined with a Jacobian matrix J_v configured in accordance with the transition mode

of the evolution process; state transition term R may comprise the trace S multiplicatively combined with a Jacobian matrix Jr configured in accordance with the response mode of the evolution process; and state transition term G may comprise the trace S multiplicatively combined with a Jacobian matrix Jg configured in accordance with the input x.

[0045] In some implementations, the input may comprise feed-forward input via an interface; and the learning parameter may comprise efficacy associated with the interface, the interface may comprise synaptic connection and the learning parameter may comprise connection weight, the state parameter q may be configured to describe time evolution of the neuron state in accordance with a state evolution process, characterized by evolution process may be characterized by an instantaneous probability density distribution IPD of generating neuronal response P.

[0046] In one or more implementations, the instructions are further configured to, when executed, determine derivative $d\lambda/dw$ of the IPD, with respect to the learning parameter w, based at least in part on the trace S; and obtain an instantaneous score function value g, based at least in part on the derivative $d\lambda/dw$; and the determine the adjustment dw may be based at least in part on the instantaneous score function value g, where the determination of the realization of the trace S, and the determination of the derivative $d\lambda/dw$, and the obtaining of the instantaneous score function value g cooperate to produce the adjustment dw such that a next instance of the neuron state, associated with an adjusted value w2, configured based on the current value w1 and the adjustment dw, may be closer to the target state.

[0047] In one or more implementations, computerized apparatus may be configured to process input spike train x using hybrid learning rule, the apparatus comprising stochastic learning block configured to produce learning signal based at least in part on the input spike x and training signal r, the hybrid learning rule may be configured to simultaneously effect reinforcement learning rule and unsupervised learning rule.

[0048] In some implementations, the stochastic learning block may be operable according to a stochastic process characterized by a current state and a desired state, the process being described by at least a state variable configured to transition the learning block from current state to the desired state, the training signal r may comprise a reinforcement spiking indicator associated with current performance relative to desired performance of the apparatus, the current performance corresponding to the current state and the desired performance corresponding to the desired state, the current performance may be effected, at least partly, by transition from a prior state to the current state, the reinforcement learning may be configured based at least in part on the reinforcement spiking indicator so that it provides: positive reinforcement when a distance measure between the current state and the desired state may be smaller compared to the distance measure between the prior state and the desired state; and negative reinforcement when the distance measure between the current state and the desired state may be greater compared to the distance measure between the prior state and the desired state.

[0049] In some implementations, the training signal r further may comprise desired output spike train yd, current performance may be effected, at least partly, by transition from prior state to the current state; and the reinforcement learning may be configured based at least in part on the reinforcement

spiking indicator so that: positive reinforcement when the current performance may be closer to the desired performance, and the reinforcement may be negative when the current performance may be farther from the desired performance.

[0050] In some implementations, the stochastic learning block may be operable according to stochastic process characterized by current state and desired state, the process being described by at least state variable configured to transition the learning block from current state to the desired state; the hybrid learning rule may be characterized by a hybrid performance function F comprising a simultaneous combination of reinforcement learning performance function Fre and supervised learning performance function Fsu; and the simultaneous combination may be effectuated by at least in part on a value of the hybrid performance function F determined at a time step t, the value comprising reinforcement performance function Fre value and supervised learning performance function Fsu value.

[0051] These and other objects, features, and characteristics of the present disclosure, as well as the methods of operation and functions of the related elements of structure and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification, wherein like reference numerals designate corresponding parts in the various figures. It is to be expressly understood, however, that the drawings are for the purpose of illustration and description only and are not intended as a definition of the limits of the disclosure. As used in the specification and in the claims, the singular form of "a", "an", and "the" include plural referents unless the context clearly dictates otherwise.

BRIEF DESCRIPTION OF THE DRAWINGS

[0052] FIG. 1 is a block diagram illustrating a typical architecture of an adaptive system according to prior art.

[0053] FIG. 1A is a block diagram illustrating multi-task learning controller apparatus according to prior art.

[0054] FIG. 2 is a graphical illustration of typical input and output spike trains according to prior art.

[0055] FIG. 3 is a block diagram illustrating generalized learning apparatus, in accordance with one or more implementations.

[0056] FIG. 4 is a block diagram illustrating learning block apparatus of FIG. 3, in accordance with one or more implementations.

[0057] FIG. 4A is a block diagram illustrating exemplary implementations of performance determination block of the learning block apparatus of FIG. 4, in accordance with the disclosure.

[0058] FIG. 5 is a block diagram illustrating generalized learning apparatus, in accordance with one or more implementations.

[0059] FIG. 5A is a block diagram illustrating generalized learning block configured for implementing different learning rules, in accordance with one or more implementations.

[0060] FIG. 5B is a block diagram illustrating generalized learning block configured for implementing different learning rules, in accordance with one or more implementations.

[0061] FIG. 6A is a block diagram illustrating a spiking neural network, comprising three dynamically configured

partitions, configured to effectuate generalized learning block of FIG. 4, in accordance with one or more implementations.

[0062] FIG. 6B is a block diagram illustrating a spiking neural network, comprising two dynamically configured partitions, adapted to effectuate generalized learning, in accordance with one or more implementations.

[0063] FIG. 7 is a block diagram illustrating spiking neural network configured to effectuate multiple learning rules, in accordance with one or more implementations.

[0064] FIG. 8A is a logical flow diagram illustrating generalized learning method for use with the apparatus of FIG. 5A, in accordance with one or more implementations.

[0065] FIG. 8B is a logical flow diagram illustrating dynamic reconfiguration method for use with the apparatus of FIG. 5A, in accordance with one or more implementations.

[0066] FIG. 9A is a plot presenting simulations data illustrating operation of the neural network of FIG. 7 prior to learning, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network before learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0067] FIG. 9B is a plot presenting simulations data illustrating supervised learning operation of the neural network of FIG. 7, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network before learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0068] FIG. 9C is a plot presenting simulations data illustrating reinforcement learning operation of the neural network of FIG. 7, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0069] FIG. 9D is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising reinforcement learning aided with small portion of supervisor spikes, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0070] FIG. 9E is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising an equal mix of reinforcement and supervised learning signals, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0071] FIG. 9F is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising supervised learning augmented with a 50% fraction of reinforcement spikes, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after

learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0072] FIG. 10A is a plot presenting simulations data illustrating supervised learning operation of the neural network of FIG. 7, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network before learning; (iii) supervisor spike pattern.

[0073] FIG. 10B is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising supervised learning augmented by a small amount of unsupervised learning, modeled as 15% fraction of randomly distributed (Poisson) spikes, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern.

[0074] FIG. 10C is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising supervised learning augmented by a substantial amount of unsupervised learning, modeled as 80% fraction of Poisson spikes, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern.

[0075] FIG. 11 is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising supervised learning and reinforcement learning, augmented by a small amount of unsupervised learning, modeled as 15% fraction of Poisson spikes, in accordance with one or more implementations, where data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern; (iv) positive reinforcement spike pattern; and (v) negative reinforcement spike pattern.

[0076] FIG. 12 is a plot presenting simulations data illustrating supervised learning operation of the spiking neural network of FIG. 7. Data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern.

[0077] FIG. 13 is a plot presenting simulations data illustrating predictive supervised learning operation of the spiking neural network of FIG. 7. Data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern.

[0078] FIG. 14 is a plot presenting simulations data illustrating reciprocal supervised learning operation of the spiking neural network of FIG. 7. Data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) supervisor spike pattern.

[0079] FIG. 15A is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising unsupervised learning. Data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network after learning; (iii) evolution of weights during learning.

[0080] FIG. 15B is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising unsupervised learning via Kullback-Liebler divergence minimization. Data in the top panels represents the average performance, while data in the bottom panel shows evolution of weights during learning.

[0081] FIG. 15C is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising

unsupervised learning via Kullback-Liebler divergence minimization. Data in the top panels represents the input spike pattern; while data in the bottom panel shows network output after learning.

[0082] FIG. 16 is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising reinforcement learning. Data in the top panel illustrate mean distance $\langle d \rangle$ between the actual position of an AUV $y(t)$ and the desired position of the AUV $y^d(t)$; while data in the bottom panel present variance of the distance d .

[0083] FIG. 17A is a plot presenting simulations data illustrating operation of the neural network of FIG. 7, comprising reinforcement learning. Data in the panels from top to bottom comprise: (i) input spike pattern; (ii) output activity of the network at time interval corresponding to the epoch 250 of FIG. 17B, (iii) reward spike pattern.

[0084] FIG. 17B is a plot presenting simulations data illustrating averaged performance (top) and evolution of weights (bottom) of the spiking neural network of FIG. 7, comprising reinforcement learning configured in accordance with one or more implementations.

[0085] FIG. 18 is a logical flow diagram illustrating automatic computation of eligibility traces in a spiking neural network, in accordance with one or more implementations.

[0086] FIG. 19 is a program listing illustrating textual description of spiking neuron dynamics and stochastic properties configured for processing by Matlab® symbolic computation engine in order to automatically generate score function, in accordance with one or more implementations.

[0087] All Figures disclosed herein are © Copyright 2012 Brain Corporation. All rights reserved.

DETAILED DESCRIPTION

[0088] Exemplary implementations of the present disclosure will now be described in detail with reference to the drawings, which are provided as illustrative examples so as to enable those skilled in the art to practice the disclosure. Notably, the figures and examples below are not meant to limit the scope of the present disclosure to a single implementation, but other implementations are possible by way of interchange of or combination with some or all of the described or illustrated elements. Wherever convenient, the same reference numbers will be used throughout the drawings to refer to same or similar parts.

[0089] Where certain elements of these implementations can be partially or fully implemented using known components, only those portions of such known components that are necessary for an understanding of the present disclosure will be described, and detailed descriptions of other portions of such known components will be omitted so as not to obscure the disclosure.

[0090] In the present specification, an implementation showing a singular component should not be considered limiting; rather, the disclosure is intended to encompass other implementations including a plurality of the same component, and vice-versa, unless explicitly stated otherwise herein.

[0091] Further, the present disclosure encompasses present and future known equivalents to the components referred to herein by way of illustration.

[0092] As used herein, the term “bus” is meant generally to denote all types of interconnection or communication architecture that is used to access the synaptic and neuron memory. The “bus” may be optical, wireless, infrared, and/or another

type of communication medium. The exact topology of the bus could be for example standard “bus”, hierarchical bus, network-on-chip, address-event-representation (AER) connection, and/or other type of communication topology used for accessing, e.g., different memories in pulse-based system.

[0093] As used herein, the terms “computer”, “computing device”, and “computerized device” may include one or more of personal computers (PCs) and/or minicomputers (e.g., desktop, laptop, and/or other PCs), mainframe computers, workstations, servers, personal digital assistants (PDAs), handheld computers, embedded computers, programmable logic devices, personal communicators, tablet computers, portable navigation aids, J2ME equipped devices, cellular telephones, smart phones, personal integrated communication and/or entertainment devices, and/or any other device capable of executing a set of instructions and processing an incoming data signal.

[0094] As used herein, the term “computer program” or “software” may include any sequence of human and/or machine cognizable steps which perform a function. Such program may be rendered in a programming language and/or environment including one or more of C/C++, C#, Fortran, COBOL, MATLAB™, PASCAL, Python, assembly language, markup languages (e.g., HTML, SGML, XML, VoXML), object-oriented environments (e.g., Common Object Request Broker Architecture (CORBA)), Java™ (e.g., J2ME, Java Beans), Binary Runtime Environment (e.g., BREW), and/or other programming languages and/or environments.

[0095] As used herein, the terms “connection”, “link”, “transmission channel”, “delay line”, “wireless” may include a causal link between any two or more entities (whether physical or logical/virtual), which may enable information exchange between the entities.

[0096] As used herein, the term “memory” may include an integrated circuit and/or other storage device adapted for storing digital data. By way of non-limiting example, memory may include one or more of ROM, PROM, EEPROM, DRAM, Mobile DRAM, SDRAM, DDR/2 SDRAM, EDO/FPMS, RLDRAM, SRAM, “flash” memory (e.g., NAND/NOR), memristor memory, PSRAM, and/or other types of memory.

[0097] As used herein, the terms “integrated circuit”, “chip”, and “IC” are meant to refer to an electronic circuit manufactured by the patterned diffusion of trace elements into the surface of a thin substrate of semiconductor material. By way of non-limiting example, integrated circuits may include field programmable gate arrays (e.g., FPGAs), a programmable logic device (PLD), reconfigurable computer fabrics (RCFs), application-specific integrated circuits (ASICs), and/or other types of integrated circuits.

[0098] As used herein, the terms “microprocessor” and “digital processor” are meant generally to include digital processing devices. By way of non-limiting example, digital processing devices may include one or more of digital signal processors (DSPs), reduced instruction set computers (RISC), general-purpose (CISC) processors, microprocessors, gate arrays (e.g., field programmable gate arrays (FPGAs)), PLDs, reconfigurable computer fabrics (RCFs), array processors, secure microprocessors, application-specific integrated circuits (ASICs), and/or other digital processing devices. Such digital processors may be contained on a single unitary IC die, or distributed across multiple components.

[0099] As used herein, the term “network interface” refers to any signal, data, and/or software interface with a component, network, and/or process. By way of non-limiting example, a network interface may include one or more of FireWire (e.g., FW400, FW800, etc.), USB (e.g., USB2), Ethernet (e.g., 10/100, 10/100/1000 (Gigabit Ethernet), 10-Gig-E, etc.), MoCA, Coaxsys (e.g., TVnet™), radio frequency tuner (e.g., in-band or OOB, cable modem, etc.), Wi-Fi (802.11), WiMAX (802.16), PAN (e.g., 802.15), cellular (e.g., 3G, LTE/LTE-A/TD-LTE, GSM, etc.), IrDA families, and/or other network interfaces.

[0100] As used herein, the terms “node”, “neuron”, and “neuronal node” are meant to refer, without limitation, to a network unit (e.g., a spiking neuron and a set of synapses configured to provide input signals to the neuron) having parameters that are subject to adaptation in accordance with a model.

[0101] As used herein, the terms “state” and “node state” is meant generally to denote a full (or partial) set of dynamic variables used to describe node state.

[0102] As used herein, the term “synaptic channel”, “connection”, “link”, “transmission channel”, “delay line”, and “communications channel” include a link between any two or more entities (whether physical (wired or wireless), or logical/virtual) which enables information exchange between the entities, and may be characterized by a one or more variables affecting the information exchange.

[0103] As used herein, the term “Wi-Fi” includes one or more of IEEE-Std. 802.11, variants of IEEE-Std. 802.11, standards related to IEEE-Std. 802.11 (e.g., 802.11a/b/g/n/s/v), and/or other wireless standards.

[0104] As used herein, the term “wireless” means any wireless signal, data, communication, and/or other wireless interface. By way of non-limiting example, a wireless interface may include one or more of Wi-Fi, Bluetooth, 3G (3GPP/3GPP2), HSDPA/HSUPA, TDMA, CDMA (e.g., IS-95A, WCDMA, etc.), FHSS, DSSS, GSM, PAN/802.15, WiMAX (802.16), 802.20, narrowband/FDMA, OFDM, PCS/DCS, LTE/LTE-A/TD-LTE, analog cellular, CDPD, satellite systems, millimeter wave or microwave systems, acoustic, infrared (i.e., IrDA), and/or other wireless interfaces.

Overview

[0105] The present disclosure provides, among other things, a computerized apparatus and methods for implementing generalized learning rules given multiple cost measures. In one implementation of the disclosure, adaptive spiking neuron network signal processing system may flexibly combine different learning rules (e.g., supervised, unsupervised, reinforcement learning, and/or other learning rules) with different methods (e.g., online, batch, and/or other learning methods). The generalized learning apparatus of the disclosure may employ, in some implementations, modular architecture where learning tasks are separated from control tasks, so that changes in one of the blocks do not necessitate changes within the other block. By separating implementation of learning tasks from the control tasks, the framework may further allow dynamic reconfiguration of the learning block in response to a task change or learning method change in real time. The generalized learning apparatus may be capable of implementing several learning rules concurrently based on the desired control task and without requiring users to explicitly identify the required learning rule composition for that application.

[0106] The generalized learning framework described herein advantageously provides for learning implementations that do not affect regular operation of the signal system (e.g., processing of data). Hence, a need for a separate learning stage may be obviated so that learning may be turned off and on again when appropriate.

[0107] One or more generalized learning methodologies described herein may enable different parts of the same network to implement different adaptive tasks. The end user of the adaptive device may be enabled to partition network into different parts, connect these parts appropriately, and assign cost functions to each task (e.g., selecting them from pre-defined set of rules or implementing a custom rule). A user may not be required to understand detailed implementation of the adaptive system (e.g., plasticity rules, neuronal dynamics, etc.) nor is he required to be able to derive the performance function and determine its gradient for each learning task. Instead, a user may be able to operate generalized learning apparatus of the disclosure by assigning task functions and connectivity map to each partition.

Generalized Learning Apparatus

[0108] Detailed descriptions of various implementations of apparatuses and methods of the disclosure are now provided. Although certain aspects of the disclosure may be understood in the context of robotic adaptive control system comprising a spiking neural network, the disclosure is not so limited. Implementations of the disclosure may also be used for implementing a variety of learning systems, such as, for example, signal prediction (e.g., supervised learning), finance applications, data clustering (e.g., unsupervised learning), inventory control, data mining, and/or other applications that do not require performance function derivative computations.

[0109] Implementations of the disclosure may be, for example, deployed in a hardware and/or software implementation of a neuromorphic computer system. In some implementations, a robotic system may include a processor embodied in an application specific integrated circuit, which can be adapted or configured for use in an embedded application (e.g., a prosthetic device).

[0110] FIG. 3 illustrates one exemplary learning apparatus useful to the disclosure. The apparatus 300 shown in FIG. 3 comprises the control block 310, which may include a spiking neural network configured to control a robotic arm and may be parameterized by the weights of connections between artificial neurons, and learning block 320, which may implement learning and/or calculating the changes in the connection weights. The control block 310 may receive an input signal x , and may generate an output signal y . The output signal y may include motor control commands configured to move a robotic arm along a desired trajectory. The control block 310 may be characterized by a system model comprising system internal state variables S . An internal state variable S may include a membrane voltage of the neuron, conductance of the membrane, and/or other variables. The control block 310 may be characterized by learning parameters w , which may include synaptic weights of the connections, firing threshold, resting potential of the neuron, and/or other parameters. In one or more implementations, the parameters w may comprise probabilities of signal transmission between the units (e.g., neurons) of the network.

[0111] The input signal $x(t)$ may comprise data used for solving a particular control task. In one or more implemen-

tations, such as those involving a robotic arm or autonomous robot, the signal $x(t)$ may comprise a stream of raw sensor data (e.g., proximity, inertial, terrain imaging, and/or other raw sensor data) and/or preprocessed data (e.g., velocity, extracted from accelerometers, distance to obstacle, positions, and/or other preprocessed data). In some implementations, such as those involving object recognition, the signal $x(t)$ may comprise an array of pixel values (e.g., RGB, CMYK, HSV, HSL, grayscale, and/or other pixel values) in the input image, or preprocessed data (e.g., levels of activations of Gabor filters for face recognition, contours, and/or other preprocessed data). In one or more implementations, the input signal $x(t)$ may comprise desired motion trajectory, for example, in order to predict future state of the robot on the basis of current state and desired motion.

[0112] The control block **310** of FIG. 3 may comprise a probabilistic dynamic system, which may be characterized by an analytical input-output ($x \rightarrow y$) probabilistic relationship having a conditional probability distribution associated therewith:

$$P = p(y|x, w) \quad (\text{Eqn. 11})$$

In Eqn. 11, the parameter w may denote various system parameters including connection efficacy, firing threshold, resting potential of the neuron, and/or other parameters. The analytical relationship of Eqn. 1 may be selected such that the gradient of $\ln [p(y|x, w)]$ with respect to the system parameter w exists and can be calculated. The framework shown in FIG. 3 may be configured to estimate rules for changing the system parameters (e.g., learning rules) so that the performance function $F(x, y, r)$ is minimized for the current set of inputs and outputs and system dynamics S .

[0113] In some implementations, the control performance function may be configured to reflect the properties of inputs and outputs (x, y). The values $F(x, y, r)$ may be calculated directly by the learning block **320** without relying on external signal r when providing solution of unsupervised learning tasks.

[0114] In some implementations, the value of the function F may be calculated based on a difference between the output y of the control block **310** and a reference signal y^d characterizing the desired control block output. This configuration may provide solutions for supervised learning tasks, as described in detail below.

[0115] In some implementations, the value of the performance function F may be determined based on the external signal r . This configuration may provide solutions for reinforcement learning tasks, where r represents reward and punishment signals from the environment.

Learning Block

[0116] The learning block **320** may implement learning framework according to the implementation of FIG. 3 that enables generalized learning methods without relying on calculations of the performance function F derivative in order to solve unsupervised, supervised, reinforcement, and/or other learning tasks. The block **320** may receive the input x and output y signals (denoted by the arrow **302_1**, **308_1**, respectively, in FIG. 3), as well as the state information **305**. In some implementations, such as those involving supervised and reinforcement learning, external teaching signal r may be provided to the block **320** as indicated by the arrow **304** in FIG. 3. The teaching signal may comprise, in some imple-

mentations, the desired motion trajectory, and/or reward and punishment signals from the external environment.

[0117] In one or more implementations the learning block **320** may optimize performance of the control system (e.g., the system **300** of FIG. 3) that is characterized by minimization of the average value of the performance function $F(x, y, r)$ as described in detail in co-owned and co-pending U.S. patent application Ser. No. 13/XXX,XXX entitled "STOCHASTIC APPARATUS AND METHODS FOR IMPLEMENTING GENERALIZED LEARNING RULES", incorporated supra. The above-referenced application describes, in one or more implementations, minimizing the average performance $\langle F \rangle_{x, y, r}$, using, for example, gradient descend algorithms where

$$\frac{\partial}{\partial w_i} \langle F(x, y, r) \rangle_{x, y, r} = \quad (\text{Eqn. 12})$$

$$\left\langle \left\langle F(x, y, r) \frac{\partial}{\partial w_i} \ln(p(y|x, w)) \right\rangle_{x, y} \right\rangle_r$$

where:

$$-\ln(p(y|x, w)) = h(y|x, w) \quad (\text{Eqn. 13})$$

is the per-stimulus entropy of the system response (or 'surprisal'). The probability of the external signal $p(r|x, y)$ may be characteristic of the external environment and may not change due to adaptation. That property may allow omission of averaging over external signals r in subsequent consideration of learning rules.

[0118] As illustrated in FIG. 3, the learning block may have access to the system's inputs and outputs, and/or system internal state S . In some implementations, the learning block may be provided with additional inputs **304** (e.g., reinforcement signals, desired output, and/or current costs of control movements, etc.) that are related to the current task of the control block.

[0119] The learning block may estimate changes of the system parameters w that minimize the performance function F , and may provide the parameter adjustment information Δw to the control block **310**, as indicated by the arrow **306** in FIG. 3. In some implementations, the learning block may be configured to modify the learning parameters w of the controller block. In one or more implementations (not shown), the learning block may be configured to communicate parameters w (as depicted by the arrow **306** in FIG. 3) for further use by the controller block **310**, or to another entity (not shown).

[0120] By separating learning related tasks into a separate block (e.g., the block **320** in FIG. 3) from control tasks, the architecture shown in FIG. 3 may provide flexibility of applying different (or modifying) learning algorithms without requiring modifications in the control block model. In other words, the methodology illustrated in FIG. 3 may enable implementation of the learning process in such a way that regular functionality of the control aspects of the system **300** is not affected. For example, learning may be turned off and on again as required with the control block functionality being unaffected.

[0121] The detailed structure of the learning block **420** is shown and described with respect to FIG. 4. The learning block **420** may comprise one or more of gradient determination (GD) block **422**, performance determination (PD) block **424** and parameter adaptation block (PA) **426**, and/or other components. The implementation shown in FIG. 4 may

decompose the learning process of the block 420 into two parts. A task-dependent/system independent part (i.e., the block 420) may implement a performance determination aspect of learning that is dependent only on the specified learning task (e.g., supervised). Implementation of the PD block 424 may not depend on particulars of the control block (e.g., block 310 in FIG. 3) such as, for example, neural network composition, neuron operating dynamics, and/or other particulars). The second part of the learning block 420, comprised of the blocks 422 and 426 in FIG. 4, may implement task-independent/system dependent aspects of the learning block operation. The implementation of the GD block 422 and PA block 426 may be the same for individual learning rules (e.g., supervised and/or unsupervised). The GD block implementation may further comprises particulars of gradient determination and parameter adaptation that are specific to the controller system 310 architecture (e.g., neural network composition, neuron operating dynamics, and/or plasticity rules). The architecture shown in FIG. 4 may allow users to modify task-specific and/or system-specific portions independently from one another, thereby enabling flexible control of the system performance. An advantage of the framework may be that the learning can be implemented in a way that does not affect the normal protocol of the functioning of the system (except of changing the parameters w). For example, there may be no need in a separate learning stage and learning may be turned off and on again when appropriate.

Gradient Determination Block

[0122] The GD block may be configured to determine the score function g by, inter alia, computing derivatives of the logarithm of the conditional probability with respect to the parameters that are subjected to change during learning based on the current inputs x , outputs y , and state variables S , denoted by the arrows 402, 408, 410, respectively, in FIG. 4. The GD block may produce an estimate of the score function g , denoted by the arrow 418 in FIG. 4 that is independent of the particular learning task, (e.g., reinforcement, unsupervised, and/or supervised learning). In some implementations, where the learning model comprises multiple parameters w_i , the score function g may be represented as a vector g , comprising scores g_i associated with individual parameter components w_i .

[0123] Implementation of this block may be non-trivial for the complex adaptive systems, such as spiking neural networks. However, using the framework described herein, this implementation may need to be changed only once and then used without changing for different learning tasks, as described in detail below.

[0124] In order to apply SF/LR methods for spiking neurons, a score function

$$g_i = \frac{\partial h(y|x)}{\partial w_i}$$

may be calculated for individual spiking neurons parameters to be changed. If spiking patterns are viewed on finite interval length T as an input x and output y of the neuron, then the score function may take the following form:

$$g_i = \frac{\partial h(y_T|x_T)}{\partial w_i} = - \sum_{t_i \in y_T} \frac{1}{\lambda(t_i)} \frac{\partial \lambda(t_i)}{\partial w_i} + \int_T \frac{\partial \lambda(s)}{\partial w_i} ds. \quad (\text{Eqn. 14})$$

where time moments t_i belong to neuron's output pattern y_T (neuron generates spike at these time moments).

[0125] If an output of the neuron at each time moment is considered (e.g., whether there is an output spike or not), then an instantaneous value of the score function may be calculated that is a time derivative of the interval score function:

$$g_i = \frac{\partial h(y(t)|x)}{\partial w_i} = \frac{\partial \lambda(t)}{\partial w_i} \left(1 - \sum_{t_i} \frac{\delta(t - t_i)}{\lambda(t)} \right) \quad (\text{Eqn. 15})$$

where t_i is times of output spikes and $\delta(t)$ is a delta function.

[0126] For discrete time the score function for spiking pattern on interval T may be calculated as:

$$g_i = \frac{\partial h(y_T|x_T)}{\partial w_i} = - \sum_{t_i \in y_T} \frac{1 - \Lambda(t_i)}{\Lambda(t_i)} \frac{\partial \lambda(t_i)}{\partial w_i} \Delta t + \sum_{t_i \notin y_T} \frac{\partial \lambda(t_i)}{\partial w_i} \cdot \Delta t \quad (\text{Eqn. 16})$$

where $t_i \in y_T$ denotes time steps when neuron generated a spike.

[0127] Instantaneous value of the score function in discrete time may equal:

$$g_i = \frac{\partial h_{\Delta t}}{\partial w_i} = \frac{\partial \lambda}{\partial w_i} \left(1 - \sum_{t_i} \frac{1 - \Lambda(t_i)}{\Lambda(t_i)} \delta_d(t - t_i) \right) \quad (\text{Eqn. 17})$$

where t_i is the times of output spikes and $\delta_d(t)$ is the Kronecker delta.

[0128] In order to calculate the score function,

$$\frac{\partial \lambda(t)}{\partial w_i}$$

may be calculated, which is a derivative of the instantaneous probability density with respect to a learning parameter w_i of the i -th neuron. Without loss of generality, two cases of learning are considered below: input weights learning (synaptic plasticity) and stochastic threshold tuning (intrinsic plasticity). A derivative of other less common parameters of the neuron model (e.g., membrane, synaptic dynamic, and/or other constants) may be calculated.

[0129] The neuron may receive n input spiking channels. External current to the neuron I^{ext} in the neuron's dynamic equation may be modeled Eqn. 6 as a sum of filtered and weighted input spikes from all input channels:

$$I^{ext} = \sum_i^n \sum_{t_j^i \in x^i} w_i \epsilon(t - t_j^i) \quad (\text{Eqn. 18})$$

where: i is the index of the input channel; x^i is the stream of input spikes on the i -th channel; t_j^i is the times of input spikes in the i -th channel; w_i is the weight of the i -th channel; and $\epsilon(t)$ is a generic function that models post-synaptic currents from

input spikes. In some implementations, the post-synaptic current function may be configured as: $\epsilon(t) = \delta(t)$, $\epsilon(t) = e^{-t/\tau_s} H(t)$, where $\delta(t)$ is a delta function, $H(t)$ is a Heaviside function, and τ_s is a synaptic time constant.

[0130] A derivative of instantaneous probability density with respect to the i-th channel's weight may be taken using chain rule:

$$\frac{\partial \lambda}{\partial w_i} = \sum_j \left(\frac{\partial \lambda_i}{\partial q_j} \cdot \nabla_{w_i} q_j \right) \text{ where } \frac{\partial \lambda}{\partial q} \quad (\text{Eqn. 19})$$

is a vector of derivatives of instantaneous probability density with respect to the state variable; and

$$S_i(t) = \nabla_{w_i} \vec{q} \quad (\text{Eqn. 20})$$

is the gradient of the neuron internal state with respect to the ith weight (also referred to as the i-th state eligibility trace). In order to determine the state eligibility trace of Eqn. 20 for generalized neuronal model, such as, for example, described by equations Eqn. 6 and Eqn. 18, derivative with respect to the learning weight w_i may be determined as:

$$\frac{\partial}{\partial w_i} \left(\frac{d\vec{q}}{dt} \right) = \frac{\partial}{\partial w_i} (V(\vec{q})) + \frac{\partial}{\partial w_i} \left(\sum_{out} R(\vec{q}) \delta(t - t^{out}) \right) + \frac{\partial}{\partial w_i} (G(\vec{q}) I^{ext}) \quad (\text{Eqn. 21})$$

[0131] The order in which the derivatives in the left side of the equations are taken may be changed, and then the chain rule may be used to obtain the following equations (arguments of evolution functions are omitted):

$$\frac{dS_i(t)}{dt} = (J_V(\vec{q}) + J_G(\vec{q}) \cdot I^{ext}) \cdot S_i + \sum_{out} J_R(\vec{q}) \cdot S_i \cdot \delta(t - t^{out}) + G(\vec{q}) \sum_{t_j \in x^i} \epsilon(t - t_j), \quad (\text{Eqn. 22})$$

Where J_F , J_R , J_G are Jacobian matrices of the respective evolution functions V , R , G .

[0132] As an example, evaluating Jacobean matrices IF neuron may produce:

$$J_V = -C; J_R = -1; G(\vec{q}) = 1; J_G = 0, \quad (\text{Eqn. 23})$$

so Eqn. 22 for the i-th state eligibility trace may take the following form:

$$\frac{d}{dt} u_{w_i} = -C u_{w_i} - \sum_{out} u_{w_i} \cdot \delta(t - t^{out}) + \sum_{t_j \in x^i} \epsilon(t - t_j) \quad (\text{Eqn. 24})$$

where u_{w_i} denotes derivative of the state variable (e.g., voltage) with respect to the i-th weight.

[0133] A solution of Eqn. 24 may represent post-synaptic potential for the i-th unit and may be determined as a sum of all received input spikes at the unit (e.g., a neuron), where the unit is reset to zero after each output spike:

$$u_{w_i} = \sum_{t_j \in x^i} \int_{-\infty}^t e^{-(t-\tau)C} \epsilon(\tau - t_j) = \sum_{t_j \in x^i} \alpha(t - t_j) \quad (\text{Eqn. 25})$$

where $\alpha(t)$ is the post-synaptic potential (PSP) from the jth input spike.

[0134] Applying the framework of Eqn. 22-Eqn. 25 to a previously described neuronal (hereinafter IZ neuronal), the Jacobian matrices of the respective evolution functions F , R , G may be expressed as:

$$J_V = \begin{pmatrix} 0.08v(t) + 5 & -1 \\ ab & a \end{pmatrix}; \quad (\text{Eqn. 26})$$

$$J_R = \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}; \quad G(\vec{q}) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad J_G = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The IZ neuronal model may further be characterized using two first-order nonlinear differential equations describing time evolution of synaptic weights associated with each input interface (e.g., pre-synaptic connection) of a neuron, in the following form:

$$\frac{d}{dt} v_{w_i} = (0.08v + 5)v_{w_i} - u_{w_i} - \sum_{out} u_{w_i} \cdot \delta(t - t^{out}) + \sum_{t_j \in x^i} \epsilon(t - t_j) \quad (\text{Eqn. 27})$$

$$\frac{d}{dt} u_{w_i} = abv_{w_i} - au_{w_i}$$

When using the exponential stochastic threshold configured as:

$$\lambda = \lambda_0 e^{K(\langle \lambda \rangle - \Theta)}, \quad (\text{Eqn. 28})$$

then the derivative of the IPD for IZ neuronal neuron becomes:

$$\frac{\partial \lambda}{\partial w_i} = v_{w_i} K \lambda(t). \quad (\text{Eqn. 29})$$

[0135] If we use the exponential stochastic threshold Eqn. 2, the final expression for the derivative of instantaneous probability

$$\frac{\partial \lambda(t)}{\partial w}$$

for IF neuron becomes:

$$\frac{\partial \lambda}{\partial w_i} = \frac{\partial \lambda}{\partial u} \frac{\partial u}{\partial w_i} = K \lambda(t) \sum_{t_j \in x^i} \alpha(t - t_j) \quad (\text{Eqn. 30})$$

Combining Eqn. 30 with Eqn. 15 and Eqn. 17 we obtain score function values for the stochastic Integrate-and-Fire neuron in continuous time-space as:

$$g_i = \frac{\partial h(y(t) | x)}{\partial w_i} = K \sum_{t_j \in x^i} \alpha(t - t_j) \left(\lambda(t) - \sum_{out \in y} \delta(t - t^{out}) \right) \quad (\text{Eqn. 31})$$

and in discrete time:

$$g_i = \frac{\partial h_{\Delta t}(y(t) | x)}{\partial w_i} = K \lambda(t) \sum_{t_j \in x^i} \alpha(t - t_j) \left(1 - \sum_{out \in y} \frac{\delta_d(t - t^{out})}{\Lambda(t)} \right) \Delta t \quad (\text{Eqn. 32})$$

[0136] In one or more implementations, the gradient determination block may be configured to determine the score function g based on particular inputs into the neuron(s), neuron outputs, and internal neuron state, according, for example with Eqn. 15. Furthermore, in some implementations, using the methodology described herein and providing description

of neurons dynamics and stochastic properties in textual form, as shown and described in detail with respect to FIG. 19 below, advantageously allows the use of analytical mathematics computer aided design (CAD) tools in order to automatically obtain score function, such as for example Eqn. 32.

Performance Determination Block

[0137] The PD block may be configured to determine the performance function F based on the current inputs x , outputs y , and/or training signal r , denoted by the arrow 404 in FIG. 4. In some implementations, the external signal r may comprise the reinforcement signal in the reinforcement learning task. In some implementations, the external signal r may comprise reference signal in the supervised learning task. In some implementations, the external signal r comprises the desired output, current costs of control movements, and/or other information related to the current task of the control block (e.g., block 310 in FIG. 3). Depending on the specific learning task (e.g., reinforcement, unsupervised, or supervised) some of the parameters x , y , r may not be required by the PD block illustrated by the dashed arrows 402_1, 408_1, 404_1, respectively, in FIG. 4A. The learning apparatus configuration depicted in FIG. 4 may decouple the PD block from the controller state model so that the output of the PD block depends on the learning task and is independent of the current internal state of the control block.

Generalized Performance Determination

[0138] In some implementations the PD block may transmit the external signal r to the learning block (as illustrated by the arrow 404_1) so that:

$$F(t)=r(t), \quad (\text{Eqn. 33})$$

where signal r provides reward and/or punishment signals from the external environment. By way of illustration, a mobile robot, controlled by spiking neural network, may be configured to collect resources (e.g., clean up trash) while avoiding obstacles (e.g., furniture, walls). In this example, the signal r may comprise a positive indication (e.g., representing a reward) at the moment when the robot acquires the resource (e.g., picks up a piece of rubbish) and a negative indication (e.g., representing a punishment) when the robot collides with an obstacle (e.g., wall). Upon receiving the reinforcement signal r , the spiking neural network of the robot controller may change its parameters (e.g., neuron connection weights) in order to maximize the function F (e.g., maximize the reward and minimize the punishment).

[0139] In some implementations, the PD block may determine the performance function by comparing current system output with the desired output using a predetermined measure (e.g., a distance d):

$$F(t)=d(y(t),y^d(t)), \quad (\text{Eqn. 34})$$

where y is the output of the control block (e.g., the block 310 in FIG. 3) and $r=y^d$ is the external reference signal indicating the desired output that is expected from the control block. In some implementations, the external reference signal r may depend on the input x into the control block. In some implementations, the control apparatus (e.g., the apparatus 300 of FIG. 3) may comprise a spiking neural network configured for pattern classification. A human expert may present to the network an exemplary sensory pattern x and the desired output y^d that describes the input pattern x class. The network may change (e.g., adapt) its parameters w to achieve the desired response on the presented pairs of input x and desired

response y^d . After learning, the network may classify new input stimuli based on one or more past experiences.

[0140] In some implementations, such as when characterizing a control block utilizing analog output signals, the distance function may be determined using the squared error estimate as follows:

$$F(t)=(y(t)-y^d(t))^2. \quad (\text{Eqn. 35})$$

[0141] In some implementations, such as those applicable to control blocks using spiking output signals, the distance measure may be determined using the squared error of the convolved signals y , y^d as follows:

$$F=[(y*\alpha)-(y^d*\beta)]^2, \quad (\text{Eqn. 36})$$

where α , β are finite impulse response kernels. In some implementations, the distance measure may utilize the mutual information between the output signal and the reference signal.

[0142] In some implementations, the PD may determine the performance function by comparing one or more particular characteristic of the output signal with the desired value of this characteristic:

$$F=[f(y)-f^d(y)]^2, \quad (\text{Eqn. 37})$$

where f is a function configured to extract the characteristic (or characteristics) of interest from the output signal y . By way of example, useful with spiking output signals, the characteristic may correspond to a firing rate of spikes and the function $f(y)$ may determine the mean firing from the output. In some implementations, the desired characteristic value may be provided through the external signal as

$$r=f^d(y). \quad (\text{Eqn. 38})$$

In some implementations, the $f^d(y)$ may be calculated internally by the PD block.

[0143] In some implementations, the PD block may determine the performance function by calculating the instantaneous mutual information i between inputs and outputs of the control block as follows:

$$F=i(x,y)=-\ln(p(y))+\ln(p(y|x)), \quad (\text{Eqn. 39})$$

where $p(y)$ is an unconditioned probability of the current output. It is noteworthy that the average value of the instantaneous mutual information may equal the mutual information $I(x,y)$. This performance function may be used to implement ICA (unsupervised learning).

[0144] In some implementations, the PD block may determine the performance function by calculating the unconditional instantaneous entropy h of the output of the control block as follows:

$$F=h(x,y)=-\ln(p(y)). \quad (\text{Eqn. 40})$$

where $p(y)$ is an unconditioned probability of the current output. It is noteworthy that the average value of the instantaneous unconditional entropy may equal the unconditional $H(x,y)$. This performance function may be used to reduce variability in the output of the system for adaptive filtering.

[0145] In some implementations, the PD block may determine the performance function by calculating the instantaneous Kullback-Leibler divergence d_{KL} between the output probability distribution $p(y|x)$ of the control block and some desired probability distribution $\theta(y|x)$ as follows:

$$F=d_{KL}(x,y)=\ln(p(y|x))-\ln(\theta(y|x)). \quad (\text{Eqn. 41})$$

It is noteworthy that the average value of the instantaneous Kullback-Leibler divergence may equal the $d_{KL}(p, \theta)$. The

performance function of Eqn. 41 may be applied in unsupervised learning tasks in order to restrict a possible output of the system. For example, if $\theta(y)$ is a Poisson distribution of spikes with some firing rate R , then minimization of this performance function may force the neuron to have the same firing rate R .

[0146] In some implementations, the PD block may determine the performance function for the sparse coding. The sparse coding task may be an unsupervised learning task where the adaptive system may discover hidden components in the data that describes data the best with a constraint that the structure of the hidden components should be sparse:

$$F = \|x - A(y, w)\|^2 + \|y\|^2, \quad (\text{Eqn. 42})$$

where the first term quantifies how close the data x can be described by the current output y , where $A(y, w)$ is a function that describes how to decode an original data from the output. The second term may calculate a norm of the output and may imply restrictions on the output sparseness.

[0147] A learning framework of the present innovation may enable generation of learning rules for a system, which may be configured to solve several completely different task-types simultaneously. For example, the system may learn to control an actuator while trying to extract independent components from movement trajectories of this actuator. The combination of tasks may be done as a linear combination of the performance functions for each particular problem:

$$F = C(F_1, F_2, \dots, F_n), \quad (\text{Eqn. 43})$$

where: $F_1, F_2, \dots, F_n$ are performance function values for different tasks, and C is a combination function.

[0148] In some implementations, the combined cost function C may comprise a weighted linear combination of individual cost functions corresponding to individual learning tasks:

$$C(F_1, F_2, \dots, F_n) = \sum_k a_k F_k, \quad (\text{Eqn. 44})$$

where a_k are combination weights.

[0149] It is recognized by those skilled in the arts that linear cost function combination described by 44 illustrates one particular implementation of the disclosure and other implementations (e.g., a nonlinear combination) may be used as well.

[0150] In some implementations, the PD block may be configured to calculate the baseline of the performance function values (e.g., as a running average) and subtract it from the instantaneous value of the performance function in order to increase learning speed of learning. The output of the PD block may comprise a difference between the current value $F(t)^{cur}$ of the performance function and its time average $\langle F \rangle$:

$$F(t) = F(t)^{cur} - \langle F \rangle. \quad (\text{Eqn. 45})$$

In some implementations, the time average of the performance function may comprise an interval average, where learning occurs over a predetermined interval. A current value of the cost function may be determined at individual steps within the interval and may be averaged over all steps. In some implementations, the time average of the performance function may comprise a running average, where the current value of the cost function may be low-pass filtered according to:

$$\frac{dF(t)}{dt} = -\tau F(t) + F(t)^{cur}, \quad (\text{Eqn. 46})$$

thereby producing a running average output.

[0151] Referring now to FIG. 4A different implementations of the performance determination block (e.g., the block 424 of FIG. 4) are shown. The PD block implementation denoted 434, may be configured to simultaneously implement reinforcement, supervised and unsupervised (RSU) learning rules; and/or receive the input signal $x(t)$ 412, the output signal $y(t)$ 418, and/or the learning signal 436. The learning signal 436 may comprise the reinforcement component $r(t)$ and the desired output (teaching) component $y^d(t)$. In one or more implementations, the output performance function F_{RSU} 438 of the RSUPD block may be determined in accordance with Eqn. 69 described below.

[0152] The PD blocks 444, 445, may implement the reinforcement (R) learning rule. The output 448 of the block 444 may be determined based on the output signal $y(t)$ 418 and the reinforcement signal $r(t)$ 446. In one or more implementations, the output 448 of the RSUPD block may be determined in accordance with Eqn. 38. The performance function output 449 of the block 445 may be determined based on the input signal $x(t)$, the output signal $y(t)$, and/or the reinforcement signal $r(t)$.

[0153] The PD block implementation denoted 454, may be configured to implement supervised (S) learning rules to generate performance function F_S 458 that is dependent on the output signal $y(t)$ value 418 and the teaching signal $y^d(t)$ 456. In one or more implementations, the output 458 of the PD 454 block may be determined in accordance with Eqn. 34-Eqn. 37.

[0154] The output performance function 468 of the PD block 464 implementing unsupervised learning may be a function of the input $x(t)$ 412 and the output $y(t)$ 418. In one or more implementations, the output 468 may be determined in accordance with Eqn. 39-Eqn. 42.

[0155] The PD block implementation denoted 474 may be configured to simultaneously implement reinforcement and supervised (RS) learning rules. The PD block 474 may not require the input signal $x(t)$, and may receive the output signal $y(t)$ 418 and the teaching signals $r(t)$, $y^d(t)$ 476. In one or more implementations, the output performance function F_{RS} 478 of the PD block 474 may be determined in accordance with Eqn. 43, where the combination coefficient for the unsupervised learning is set to zero. By way of example, in some implementations reinforcement learning task may be to acquire resources by the mobile robot, where the reinforcement component $r(t)$ provides information about acquired resources (reward signal) from the external environment, while at the same time a human expert shows the robot what should be desired output signal $y^d(t)$ to optimally avoid obstacles. By setting a higher coefficient to the supervised part of the performance function, the robot may be trained to try to acquire the resources if it does not contradict with human expert signal for avoiding obstacles.

[0156] The PD block implementation denoted 475 may be configured to simultaneously implement reinforcement and supervised (RS) learning rules. The PD block 475 output may be determined based the output signal 418, the learning signals 476, comprising the reinforcement component $r(t)$ and the desired output (teaching) component $y^d(t)$ and on the input signal 412, that determines the context for switching between supervised and reinforcement task functions. By way of example, in some implementations, reinforcement learning task may be used to acquire resources by the mobile robot, where the reinforcement component $r(t)$ provides information about acquired resources (reward signal) from

the external environment, while at the same time a human expert shows the robot what should be desired output signal $y^d(t)$ to optimally avoid obstacles. By recognizing obstacles, avoidance context on the basis of some clues in the input signal, the performance signal may be switched between supervised and reinforcement. That may allow the robot to be trained to try to acquire the resources if it does not contradict with human expert signal for avoiding obstacles. In one or more implementations, the output performance function 479 of the PD 475 block may be determined in accordance with Eqn. 43, where the combination coefficient for the unsupervised learning is set to zero.

[0157] The PD block implementation denoted 484 may be configured to simultaneously implement reinforcement, and unsupervised (RU) learning rules. The output 488 of the block 484 may be determined based on the input and output signals 412, 418, in one or more implementations, in accordance with Eqn. 43. By way of example, in some implementations of sparse coding (unsupervised learning), the task of the adaptive system on the robot may be not only to extract sparse hidden components from the input signal, but to pay more attention to the components that are behaviorally important for the robot (that provides more reinforcement after they can be used).

[0158] The PD block implementation denoted 494, which may be configured to simultaneously implement supervised and unsupervised (SU) learning rules, may receive the input signal $x(t)$ 412, the output signal $y(t)$ 418, and/or the teaching signal $y^d(t)$ 436. In one or more implementations, the output performance function F_{SU} 438 of the SU PD block may be determined in accordance with Eqn. 68 described below.

[0159] By the way of example, the stochastic learning system (that is associated with the PD block implementation 494) may be configured to learn to implement unsupervised data categorization (e.g., using sparse coding performance function), while simultaneously receiving external signal that is related to the correct category of particular input signals. In one or more implementations such reward signal may be provided by a human expert.

Performance Determination for Spiking Neurons

[0160] In one or more implementations of reinforcement learning, the PD block (e.g., the block 424 of FIG. 4) may generate the performance signal based on analog and/or spiking reward signal r (e.g., the signal 404 of FIG. 4). In one implementation, the performance signal F (e.g., the signal 428 of FIG. 4) may comprise the reward signal $r(t)$, transmitted to the PA block (e.g., the block 426 of FIG. 4) by the PD block.

[0161] In one or more implementations related to analog reward signal, in order to reduce computational load on the PA block related to application of weight changes, the PD block may transform the analog reward $r(t)$ into spike form.

[0162] In one or more implementations of supervised learning, the current performance F may be determined based on the output of the neuron and the external reference signal (e.g., the desired output $y^d(t)$). For example, a distance measure may be calculated using a low-pass filtered version of the desired $y^d(t)$ and actual $y(t)$ outputs. In some implementations, a running distance between the filtered spike trains may be determined according to

$$F(x(t), y(t)) = \quad (\text{Eqn. 47})$$

$$\left(\int_{-\infty}^{\infty} y(s)a(\tau-s)d\tau - \int_{-\infty}^{\infty} y^d(s)b(\tau-s)d\tau \right)^2 \text{ where: } y(t) = \sum_i \delta(t - t_i^{\text{out}}), y^d(t) = \sum_j \delta(t - t_j^d),$$

with $y(t)$ and $y^d(t)$ being the actual and desired output spike trains; $\delta(t)$ is the Dirac delta function; t_i^{out}, t_j^d are the output and desired spike times, respectively; and $a(t), b(t)$ are positive finite-response kernels. In some implementations, the kernel $a(t)$ may comprise an exponential trace: $a(t) = e^{-t/\tau_a}$.

[0163] In some implementations of supervised learning, spiking neuronal network may be configured to learn to minimize a Kullback-Leibler distance between the actual and desired output:

$$F(x(t), y(t)) = D_{KL}(y(t) \| r(t)). \quad (\text{Eqn. 48})$$

[0164] In some implementations, if $r(t)$ is a Poisson spike train with a fixed firing rate, the D_{KL} learning may enable stabilization of the neuronal firing rate.

[0165] In some implementations of supervised learning, for example in “information bottleneck”, part of the performance optimization may comprise maximization of the mutual information between the actual output $y(t)$ and some reference signal $r(t)$. For a given input and output, the performance function may be expressed as:

$$F(x(t), y(t)) = I(y(t), r(t)). \quad (\text{Eqn. 49})$$

[0166] In one or more implementations of unsupervised learning, the cost function may be obtained by a minimization of the conditional informational entropy of the output spiking pattern:

$$F(x, y) = H(y|x) \quad (\text{Eqn. 50})$$

so as to provide a more stable neuron output y for a given input x .

Parameter Changing Block

[0167] The parameter changing PA block (the block 426 in FIG. 4) may determine changes of the control block parameters Δw_i according to a predetermined learning algorithm, based on the performance function F and the gradient g it receives from the PD block 424 and the GD block 422, as indicated by the arrows marked 428, 430, respectively, in FIG. 4. Particular implementation of the learning algorithm within the block 426 may depend on the type of the learning task (e.g., online or batch learning) used by the learning block 320 of FIG. 3.

[0168] Several exemplary implementations of PA learning algorithms applicable with spiking control signals are described below. In some implementations, the PA learning algorithms may comprise a multiplicative online learning rule, where control parameter changes are determined as follows:

$$\Delta w_i(t) = \gamma F(t) \dot{x}_i(t), \quad (\text{Eqn. 51})$$

where γ is the learning rate configured to determine speed of learning adaptation. The learning method implementation according to (Eqn. 51) may be advantageous in applications where the performance function $F(t)$ depends on the current values of the inputs x , outputs y , and/or signal r .

[0169] In some implementations, the control parameter adjustment Δw may be determined using an accumulation of the score function gradient and the performance function values, and applying the changes at a predetermined time instance (corresponding to, e.g., the end of the learning epoch):

$$\Delta w(t) = \frac{\gamma}{N^2} \cdot \sum_{i=0}^{N-1} F(t - i\Delta t) \cdot \sum_{i=0}^{N-1} g(t - i\Delta t), \quad (\text{Eqn. 52})$$

where: T is a finite interval over which the summation occurs; N is the number of steps; and Δt is the time step determined as T/N.

The summation interval T in Eqn. 52 may be configured based on the specific requirements of the control application. By way of illustration, in a control application where a robotic arm is configured to reaching for an object, the interval may correspond to a time from the start position of the arm to the reaching point and, in some implementations, may be about 1 s-50 s. In a speech recognition application, the time interval T may match the time required to pronounce the word being recognized (typically less than 1 s-2 s). In some implementations of spiking neuronal networks, Δt may be configured in range between 1 ms and 20 ms, corresponding to 50 steps (N=50) in one second interval.

[0170] The method of Eqn. 52 may be computationally expensive and may not provide timely updates. Hence, it may be referred to as the non-local in time due to the summation over the interval T. However, it may lead to unbiased estimation of the gradient of the performance function.

[0171] In some implementations, the control parameter adjustment Δw_i may be determined by calculating the traces of the score function $e_i(t)$ for individual parameters w_i . In some implementations, the traces may be computed using a convolution with an exponential kernel β as follows:

$$\vec{e}(t+\Delta t) = \beta \vec{e}(t) + \vec{g}(t), \quad (\text{Eqn. 53})$$

where β is the decay coefficient. In some implementations, the traces may be determined using differential equations:

$$\frac{d}{dt} \vec{e}(t) = -\tau \vec{e}(t) + \vec{g}(t). \quad (\text{Eqn. 54})$$

The control parameter w may then be adjusted as:

$$\vec{\Delta w}(t) = \gamma F(t) \vec{e}(t), \quad (\text{Eqn. 55})$$

where γ is the learning rate. The method of Eqn. 53-Eqn. 55 may be appropriate when a performance function depends on current and past values of the inputs and outputs and may be referred to as the OLPOMDP algorithm. While it may be local in time and computationally simple, it may lead to biased estimate of the performance function. By way of illustration, the methodology described by Eqn. 53-Eqn. 55 may be used, in some implementations, in a rescue robotic device configured to locate resources (e.g., survivors, or unexploded ordnance) in a building. The input x may correspond to the robot current position in the building. The reward r (e.g., the successful location events) may depend on the history of inputs and on the history of actions taken by the agent (e.g., left/right turns, up/down movement, etc.).

[0172] In some implementations, the control parameter adjustment Δw determined using methodologies of the Eqns. 16, 17, 19 may be further modified using, in one variant, gradient with momentum according to:

$$\Delta \dot{w}(t) \Rightarrow \mu \Delta \dot{w}(t-\Delta t) + \Delta \dot{w}(t), \quad (\text{Eqn. 56})$$

where μ is the momentum coefficient. In some implementations, the sign of gradient may be used to perform learning adjustments as follows:

$$\Delta w_i(t) \Rightarrow \frac{\Delta w_i(t)}{|\Delta w_i(t)|}. \quad (\text{Eqn. 57})$$

In some implementations, gradient descend methodology may be used for learning coefficient adaptation.

[0173] In some implementations, the gradient signal g , determined by the PD block 422 of FIG. 4, may be subsequently modified according to another gradient algorithm, as described in detail below. In some implementations, these modifications may comprise determining natural gradient, as follows:

$$\Delta w = \left\langle g \cdot \frac{r}{g} \right\rangle_{x,y}^{-1} \cdot \left\langle g \cdot F \right\rangle_{x,y}, \quad (\text{Eqn. 58})$$

where $\left(\vec{g} \vec{g}^T \right)_{x,y}$ is the Fisher information metric matrix. Applying the following transformation to Eqn. 21:

$$\left\langle g \left(\frac{r}{g} \Delta w - F \right) \right\rangle_{x,y} = 0, \quad (\text{Eqn. 59})$$

[0174] natural gradient from linear regression task may be obtained as follows:

$$G \Delta w = \vec{F} \quad (\text{Eqn. 60})$$

[0175] where $G = [\vec{g}_0^T, \dots, \vec{g}_n^T]$ is a matrix comprising n samples of the score function g , $\vec{F} = [F_0, \dots, F_n]$ is a vector of performance function samples, and n is a number of samples that should be equal or greater of the number of the parameters w_i . While the methodology of Eqn. 58-Eqn. 60 may be computationally expensive, it may help dealing with 'plateau'-like landscapes of the performance function.

Signal Processing Apparatus

[0176] In one or more implementations, the generalized learning framework described supra may enable implementing signal processing blocks with tunable parameters w . Using the learning block framework that provides analytical description of individual types of signal processing block may enable it to automatically calculate the appropriate score function

$$\frac{\partial h(x|y)}{\partial w_i}$$

for individual parameters of the block. Using the learning architecture described in FIG. 3, a generalized implementa-

tion of the learning block may enable automatic changes of learning parameters w by individual blocks based on high level information about the subtask for each block. A signal processing system comprising one or more of such generalized learning blocks may be capable of solving different learning tasks useful in a variety of applications without substantial intervention of the user. In some implementations, such generalized learning blocks may be configured to implement generalized learning framework described above with respect to FIGS. 3-4A and delivered to users. In developing complex signal processing systems, the user may connect different blocks, and/or specify a performance function and/or a learning algorithm for individual blocks. This may be done, for example, with the special graphical user interface (GUI), which may allow blocks to be connected using a mouse or other input peripheral by clicking on individual blocks and using defaults or choosing the performance function and a learning algorithm from a predefined list. Users may not need to re-create a learning adaptation framework and may rely on the adaptive properties of the generalized learning blocks that adapt to the particular learning task. When the user desires to add a new type of block into the system, he may need to describe it in a way suitable to automatically calculate a score functions for individual parameters.

[0177] FIG. 5 illustrates one exemplary implementation of a robotic apparatus 500 comprising adaptive controller apparatus 512. In some implementations, the adaptive controller 520 may be configured similar to the apparatus 300 of FIG. 3 and may comprise generalized learning block (e.g., the block 420), configured, for example according to the framework described above with respect to FIG. 4, supra, is shown and described. The robotic apparatus 500 may comprise the plant 514, corresponding, for example, to a sensor block and a motor block (not shown). The plant 514 may provide sensory input 502, which may include a stream of raw sensor data (e.g., proximity, inertial, terrain imaging, and/or other raw sensor data) and/or preprocessed data (e.g., velocity, extracted from accelerometers, distance to obstacle, positions, and/or other preprocessed data) to the controller apparatus 520. The learning block of the controller 520 may be configured to implement reinforcement learning, according to, in some implementations Eqn. 38, based on the sensor input 502 and reinforcement signal 504 (e.g., obstacle collision signal from robot bumpers, distance from robotic arm endpoint to the desired position), and may provide motor commands 506 to the plant. The learning block of the adaptive controller apparatus (e.g., the apparatus 520 of FIG. 5) may perform learning parameter (e.g., weight) adaptation using reinforcement learning approach without having any prior information about the model of the controlled plant (e.g., the plant 514 of FIG. 5). The reinforcement signal $r(t)$ may inform the adaptive controller that the previous behavior led to “desired” or “undesired” results, corresponding to positive and negative reinforcements, respectively. While the plant 514 must be controllable (e.g., via the motor commands in FIG. 5) and the control system may be required to have access to appropriate sensory information (e.g., the data 502 in FIG. 5), detailed knowledge of motor actuator dynamics or of structure and significance of sensory signals may not be required to be known by the controller apparatus 520.

[0178] It will be appreciated by those skilled in the arts that the reinforcement learning configuration of the generalized learning controller apparatus 520 of FIG. 5 is used to illustrate

one exemplary implementation of the disclosure and myriad other configurations may be used with the generalized learning framework described herein. By way of example, the adaptive controller 520 of FIG. 5 may be configured for: (i) unsupervised learning for performing target recognition, as illustrated by the adaptive controller 520_3 of FIG. 5A, receiving sensory input and output signals (x,y) 522_3; (ii) supervised learning for performing data regression, as illustrated by the adaptive controller 520_3 receiving output signal 522_1 and teaching signal 504_1 of FIG. 5A; and/or (iii) simultaneous supervised and unsupervised learning for performing platform stabilization, as illustrated by the adaptive controller 520_2 of FIG. 5A, receiving input 522_2 and learning 504_2 signals.

[0179] FIGS. 5B-5C illustrate dynamic tasking by a user of the adaptive controller apparatus (e.g., the apparatus 320 of FIG. 3A or 520 of FIG. 5, described supra) in accordance with one or more implementations.

[0180] A user of the adaptive controller 520_4 of FIG. 5B may utilize a user interface (textual, graphics, touch screen, etc.) in order to configure the task composition of the adaptive controller 520_4, as illustrated by the example of FIG. 5B. By way of illustration, at one instance for one application the adaptive controller 520_4 of FIG. 5B may be configured to perform the following tasks: (i) task 550_1 comprising sensory compressing via unsupervised learning; (ii) task 550_2 comprising reward signal prediction by a critic block via supervised learning; and (iii) task 550_3 comprising implementation of optimal action by an actor block via reinforcement learning. The user may specify that task 550_1 may receive external input $\{X\}$ 542, comprising, for example raw audio or video stream, output 546 of the task 550_1 may be routed to each of tasks 550_2, 550_3, output 547 of the task 550_2 may be routed to the task 550_3; and the external signal $\{r\}$ (544) may be provided to each of tasks 550_2, 550_3, via pathways 544_1, 544_2, respectively as illustrated in FIG. 5B. In the implementation illustrated in FIG. 5B, the external signal $\{r\}$ may be configured as $\{r\}=\{y^d(t), r(t)\}$, the pathway 544_1 may carry the desired output $y^d(t)$, while the pathway 544_2 may carry the reinforcement signal $r(t)$.

[0181] Once the user specifies the learning type(s) associated with each task (unsupervised, supervised and reinforcement, respectively) the controller 520_4 of FIG. 5B may automatically configure the respective performance functions, without further user intervention. By way of illustration, performance function F_u of the task 550_1 may be determined based on (i) ‘sparse coding’; and/or (ii) maximization of information. Performance function F_s of the task 550_2 may be determined based on minimizing distance between the actual output 547 (prediction pr) $d(r, pr)$ and the external reward signal r 544_1. Performance function F_r of the task 550_3 may be determined based on maximizing the difference $F=r-pr$. In some implementations, the end user may select performance functions from a predefined set and/or the user may implement a custom task.

[0182] At another instance in a different application, illustrated in FIG. 5C, the controller 520_4 may be configured to perform a different set of task: (i) the task 550_1, described above with respect to FIG. 5B; and task 552_4, comprising pattern classification via supervised learning. As shown in FIG. 5C, the output of task 550_1 may be provided as the input 566 to the task 550_4.

[0183] Similarly to the implementation of FIG. 5B, once the user specifies the learning type(s) associated with each

task (unsupervised and supervised, respectively) the controller **520_4** of FIG. 5C may automatically configure the respective performance functions, without further user intervention. By way of illustration, the performance function corresponding to the task **550_4** may be configured to minimize distance between the actual task output **568** (e.g., a class {Y} to which a sensory pattern belongs) and human expert supervised signal **564** (the correct class y^d).

[0184] Generalized learning methodology described herein may enable the learning apparatus **520_4** to implement different adaptive tasks, by, for example, executing different instances of the generalized learning method, individual ones configured in accordance with the particular task (e.g., tasks **550_1**, **550_2**, **550_3**, in FIG. 5B, and **550_4**, **550_5** in FIG. 5C). The user of the apparatus may not be required to know implementation details of the adaptive controller (e.g., specific performance function selection, and/or gradient determination). Instead, the user may 'task' the system in terms of task functions and connectivity.

Partitioned Network Apparatus

[0185] FIGS. 6A-6B illustrate exemplary implementations of reconfigurable partitioned neural network apparatus comprising generalized learning framework, described above. The network **600** of FIG. 6A may comprise several partitions **610**, **620**, **630**, comprising one or more of nodes **602** receiving inputs **612** {X} via connections **604**, and providing outputs via connections **608**.

[0186] In one or more implementations, the nodes **602** of the network **600** may comprise spiking neurons (e.g., the neurons **730** of FIG. 9, described below), the connections **604**, **608** may be configured to carry spiking input into neurons, and spiking output from the neurons, respectively. The neurons **602** may be configured to generate responses (as described in, for example, U.S. patent application Ser. No. 13/152,105 filed on Jun. 2, 2011, and entitled "APPARATUS AND METHODS FOR TEMPORALLY PROXIMATE OBJECT RECOGNITION", incorporated by reference herein in its entirety) which may be propagated via feed-forward connections **608**.

[0187] In some implementations, the network **600** may comprise artificial neurons, such as for example, spiking neurons described by U.S. patent application Ser. No. 13/152,105 filed on Jun. 2, 2011, and entitled "APPARATUS AND METHODS FOR TEMPORALLY PROXIMATE OBJECT RECOGNITION", incorporated supra, artificial neurons with sigmoidal activation function, binary neurons (perceptron), radial basis function units, and/or fuzzy logic networks.

[0188] Different partitions of the network **600** may be configured, in some implementations, to perform specialized functionality. By way of example, the partition **610** may adapt raw sensory input of a robotic apparatus to internal format of the network (e.g., convert analog signal representation to spiking) using for example, methodology described in U.S. patent application Ser. No. 13/314,066, filed Dec. 7, 2001, entitled "NEURAL NETWORK APPARATUS AND METHODS FOR SIGNAL CONVERSION", incorporated herein by reference in its entirety. The output {Y1} of the partition **610** may be forwarded to other partitions, for example, partitions **620**, **630**, as illustrated by the broken line arrows **618**, **618_1** in FIG. 6A. The partition **620** may implement visual object recognition learning that may require training input signal $y^d_j(t)$ **616**, such as for example an object template and/or a class designation (friend/foe). The output

{Y2} of the partition **620** may be forwarded to another partition (e.g., partition **630**) as illustrated by the dashed line arrow **628** in FIG. 6A. The partition **630** may implement motor control commands required for the robotic arm to reach and grasp the identified object, or motor commands configured to move robot or camera to a new location, which may require reinforcement signal $r(t)$ **614**. The partition **630** may generate the output {Y} **638** of the network **600** implementing adaptive controller apparatus (e.g., the apparatus **520** of FIG. 5). The homogeneous configuration of the network **600**, illustrated in FIG. 6A, may enable a single network comprising several generalized nodes of the same type to implement different learning tasks (e.g., reinforcement and supervised) simultaneously.

[0189] In one or more implementations, the input **612** may comprise input from one or more sensor sources (e.g., optical input {Xopt} and audio input {Xaud}) with each modality data being routed to the appropriate network partition, for example, to partitions **610**, **630** of FIG. 6A, respectively.

[0190] The homogeneous nature of the network **600** may enable dynamic reconfiguration of the network during its operation. FIG. 6B illustrates one exemplary implementation of network reconfiguration in accordance with the disclosure. The network **640** may comprise partition **650**, which may be configured to perform unsupervised learning task, and partition **660**, which may be configured to implement supervised and reinforcement learning simultaneously. The network configuration of FIG. 6B may be used to perform signal separation tasks by the partition **650** and signal classification tasks by the partition **660**. The partition **650** may be operated according to unsupervised learning rule and may generate output {Y3} denoted by the arrow **658** in FIG. 6B. The partition **660** may be operated according to a combined reinforcement and supervised rule, may receive supervised and reinforcement input **656**, and/or may generate the output {Y4} **668**.

[0191] The dynamic network learning reconfiguration illustrated in FIGS. 6A-6B may be used, for example, in an autonomous robotic apparatus performing exploration tasks (e.g., a pipeline inspection autonomous underwater vehicle (AUV), or space rover, explosive detection, and/or mine exploration). When certain functionality of the robot is not required (e.g., the arm manipulation function) the available network resources (i.e., the nodes **602**) may be reassigned to perform different tasks. Such reuse of network resources may be traded for (i) smaller network processing apparatus, having lower cost, size and consuming less power, as compared to a fixed pre-determined configuration; and/or (ii) increased processing capability for the same network capacity.

[0192] As is appreciated by those skilled in the arts, the reconfiguration methodology described supra may comprise a static reconfiguration, where particular node populations are designated in advance for specific partitions (tasks); a dynamic reconfiguration, where node partitions are determined adaptively based on the input information received by the network and network state; and/or a semi-static reconfiguration, where static partitions are assigned predetermined life-span.

Spiking Network Apparatus

[0193] Referring now to FIG. 7, one implementation of spiking network apparatus for effectuating the generalized learning framework of the disclosure is shown and described in detail. The network **700** may comprise at least one stochas-

tic spiking neuron **730**, operable according to, for example, a Spike Response Model, and configured to receive n-dimensional input spiking stream $X(t)$ **702** via n-input connections **714**. In some implementations, the n-dimensional spike stream may correspond to n-input synaptic connections into the neuron. As shown in FIG. 7, individual input connections may be characterized by a connection parameter **712** w_{ij} that is configured to be adjusted during learning. In one or more implementations, the connection parameter may comprise connection efficacy (e.g., weight). In some implementations, the parameter **712** may comprise synaptic delay. In some implementations, the parameter **712** may comprise probabilities of synaptic transmission.

[0194] The following signal notation may be used in describing operation of the network **700**, below:

$$y(t) = \sum_i \delta(t - t_i)$$

denotes the output spike pattern, corresponding to the output signal **708** produced by the control block **710** of FIG. 3, where t_i denotes the times of the output spikes generated by the neuron;

$$y^d(t) = \sum_{t_i^d} \delta(t - t_i^d)$$

denotes the teaching spike pattern, corresponding to the desired (or reference) signal that is part of external signal **404** of FIG. 4, where t_i^d denotes the times when the spikes of the reference signal are received by the neuron;

$$y^+(t) = \sum_{t_i^+} \delta(t - t_i^+);$$

$$y^-(t) = \sum_{t_i^-} \delta(t - t_i^-)$$

denotes the reinforcement signal spike stream, corresponding to signal **304** of FIG. 3. and external signal **404** of FIG. 4, where t_i^+ , t_i^- denote the spike times associated with positive and negative reinforcement, respectively.

[0195] In some implementations, the neuron **730** may be configured to receive training inputs, comprising the desired output (reference signal) $y^d(t)$ via the connection **704**. In some implementations, the neuron **730** may be configured to receive positive and negative reinforcement signals via the connection **704**.

[0196] The neuron **730** may be configured to implement the control block **710** (that performs functionality of the control block **310** of FIG. 3) and the learning block **720** (that performs functionality of the control block **320** of FIG. 3, described supra.) The block **710** may be configured to receive input spike trains $X(t)$, as indicated by solid arrows **716** in FIG. 7, and to generate output spike train $y(t)$ **708** according to a Spike Response Model neuron which voltage $v(t)$ is calculated as:

$$v(t) = \sum_{i,k} w_i \cdot \alpha(t - t_i^k),$$

where w_i w_i represents weights of the input channels, t_i^k represents input spike times, and $\alpha(t) = (1/\tau_\alpha) e^{1-(t/\tau_\alpha)}$ represents an alpha function of postsynaptic response, where τ_α represents time constant (e.g., 3 ms and/or other times). A probabilistic part of a neuron may be introduced using the exponential probabilistic threshold. Instantaneous probability of firing $\lambda(t)$ may be calculated as $\lambda(t) = e^{(v(t) - Th)/\kappa}$, where Th —represents a threshold value, and κ represents stochasticity parameter within the control block. State variables g (probability of firing $\lambda(t)$ for this system) associated with the control model may be provided to the learning block **720** via the pathway **705**. The learning block **720** of the neuron **730** may receive the output spike train $y(t)$ via the pathway **708_1**. In one or more implementations (e.g., unsupervised or reinforcement learning), the learning block **720** may receive the input spike train (not shown). In one or more implementations (e.g., supervised or reinforcement learning) the learning block **720** may receive the learning signal, indicated by dashed arrow **704_1** in FIG. 7. The learning block determines adjustment of the learning parameters w , in accordance with any methodologies described herein, thereby enabling the neuron **730** to adjust, inter alia, parameters **712** of the connections **714**.

Exemplary Methods

Generalized Learning Rules

[0197] Referring now to FIG. 8A one exemplary implementation of the generalized learning method of the disclosure for use with, for example, the learning block **420** of FIG. 4, is described in detail. The method **800** of FIG. 8A may allow the learning apparatus to: (i) implement different learning rules (e.g., supervised, unsupervised, reinforcement, and/or other learning rules); and (ii) simultaneously support more than one rule (e.g., combination of supervised, unsupervised, reinforcement rules described, for example by Eqn. 43) using the same hardware/software configuration.

[0198] At step **802** of method **800** the input information may be received. In some implementations (e.g., unsupervised learning) the input information may comprise the input signal $x(t)$, which may comprise raw or processed sensory input, input from the user, and/or input from another part of the adaptive system. In one or more implementations, the input information received at step **802** may comprise learning task identifier configured to indicate the learning rule configuration (e.g., Eqn. 43) that should be implemented by the learning block. In some implementations, the indicator may comprise a software flag transited using a designated field in the control data packet. In some implementations, the indicator may comprise a switch (e.g., effectuated via a software commands, a hardware pin combination, or memory register).

[0199] At step **804**, learning framework of the performance determination block (e.g., the block **424** of FIG. 4) may be configured in accordance with the task indicator. In one or more implementations, the learning structure may comprise, inter alia, performance function configured according to Eqn.

43. In some implementations, parameters of the control block, e.g., number of neurons in the network, may be configured as well.

[0200] At step 808, the status of the learning indicator may be checked to determine whether additional learning input is required. In some implementations, the additional learning input may comprise reinforcement signal $r(t)$. In some implementations, the additional learning input may comprise desired output (teaching signal) $y^d(t)$, described above with respect to FIG. 4.

[0201] If required, the external learning input may be received by the learning block at step 808.

[0202] At step 812, the value of the present performance may be computed performance function $F(x, y, r)$ configured at the prior step. It will be appreciated by those skilled in the arts, that when performance function is evaluated for the first time (according, for example to Eqn. 35) and the controller output $y(t)$ is not available, a pre-defined initial value of $y(t)$ (e.g., zero) may be used instead.

[0203] At step 814, gradient $g(t)$ of the score function (logarithm of the conditional probability of output) may be determined by the GD block (e.g., The block 422 of FIG. 4).

[0204] At step 816, learning parameter w update may be determined by the Parameter Adjustment block (e.g., block 426 of FIG. 4) using the performance function F and the gradient g , determined at steps 812, 814, respectively. In some implementations, the learning parameter update may be implemented according to Eqns. 22-31.

[0205] At step 814, gradient $g(t)$ of the score function may be determined according, by the GD block (e.g., block 422 of FIG. 4). The learning parameter update may be subsequently provided to the control block (e.g., block 310 of FIG. 3).

[0206] At step 818, the control output $y(t)$ of the controller may be updated using the input signal $x(t)$ (received via the pathway 820) and the updated learning parameter A_w .

[0207] FIG. 8B illustrates a method of dynamic controller reconfiguration based on learning tasks, in accordance with one or more implementations.

[0208] At step 822 of method 830, the input information may be received. As described above with respect to FIG. 8A, in some implementations, the input information may comprise the input signal $x(t)$ and/or learning task identifier configured to indicate the learning rule configuration (e.g., Eqn. 43) that should be implemented by the learning block.

[0209] At step 834, the controller partitions (e.g., the partitions 520_6, 520_7, 520_8, 520_9, of FIG. 5B, and/or partitions 610, 620, 630 of FIG. 6A) may be configured in accordance with the learning rules (e.g., supervised, unsupervised, reinforcement, and/or other learning rules) corresponding to the task received at step 832. Subsequently, individual partitions may be operated according to, for example, the method 800 described with respect to FIG. 8A.

[0210] At step 836, a check may be performed as to whether the new task (or task assortment) is received. If no new tasks are received, the method may proceed to step 834. If new tasks are received that require controller repartitioning, such as for example, when exploration robotic device may need to perform visual recognition tasks when stationary, the method may proceed to step 838.

[0211] At step 838, current partition configuration (e.g., input parameter, state variables, neuronal composition, connection map, learning parameter values and/or rules, and/or other information associated with the current partition configuration) may be saved in a nonvolatile memory.

[0212] At step 840, the controller state and partition configurations may reset and the method proceeds to step 832,

where a new partition set may be configured in accordance with the new task assortment received at step 836. Method 800 of FIG. 8B may enable, inter alia, dynamic partition reconfiguration as illustrated in FIGS. 5B, 6A-6B, supra.

Automatic Derivation of Eligibility Traces

[0213] FIG. 18 illustrates exemplary data flow of automatic determination of eligibility traces for use with spiking neuron networks (e.g., the network 600 of FIG. 6A), in accordance with one or more implementations.

[0214] At step 1802 of method 1800, the state vector q , describing dynamic model of the neuron, may be provided. As described above with respect to Eqn. 7, the state vector may comprise membrane voltage and/or current and may be provided as user input, in some implementations.

[0215] At step 1804, the dynamic state of the neuron is described, such as, for example the structure of IF model or IZ neuron model. In one or more implementations, neuron state dynamics may be specified in generalized symbolic form as:

$$\begin{aligned} V(q)^T &= (V_1(q), V_2(q), \dots, V_n(q)), \\ R(q)^T &= (R_1(q), R_2(q), \dots, R_n(q)), \\ G(q)^T &= (G_1(q), G_2(q), \dots, G_n(q)), q^T = \{q_0, \dots, q_n\}. \end{aligned} \quad (\text{Eqn. 61})$$

[0216] At step 1806, partial derivatives of the state functions of Eqn. 61 may be determined as:

$$\frac{\partial V_i}{\partial q_j}, \frac{\partial R_i}{\partial q_j}, \frac{\partial G_i}{\partial q_j}. \quad (\text{Eqn. 62})$$

[0217] At step 1808, Jacobian matrices $J_v(q)$, $J_R(q)$, $J_G(q)$, associated with the respective dynamic neuronal model may be constructed. In some implementations of the IF neurons, the Jacobian matrices may be determined according to Eqn. 23. In one or more implementations of the IZ neuronal neurons, the Jacobian matrices may be determined according to Eqn. 26.

[0218] At step 1818, state traces SP , associated with the respective dynamic neuronal model, may be determined.

[0219] At step 1810, instantaneous probability density (IPD) $\lambda(q(t))$ of the neuron may be constructed. In some implementations of the IF neurons and/or IZ neuronal neurons, the IPD may be determined according to Eqn. 2-Eqn. 4.

[0220] At step 1812, partial derivatives

$$\left(\frac{\partial \lambda}{\partial q} \right)^T$$

of the IPD with respect to the state vector q may be determined.

[0221] At step 1820, the instantaneous PDF derivative

$$\frac{\partial \lambda}{\partial w_i}$$

may be determined using the partial derivatives of IPD from the step 1812 and the gradient from step 1818. In one or more implementations, the instantaneous PDF derivative may be determined using Eqn. 19.

[0222] At step 1822, the score function

$$g_i = \frac{\partial h_M}{\partial w_i}$$

may be determined using, for example, Eqn. 17. In one or more implementations of IF neuronal model, the exponential stochastic threshold may be implemented using Eqn. 2 and the score function g_i may be determined using Eqn. 32.

[0223] In some implementations of IZ neuronal model, the exponential stochastic threshold may be implemented using Eqn. 28 and the score function g_i may be determined using Eqn. 17 and Eqn. 29.

[0224] FIG. 19A presents one exemplary implementation of Python script configured to effectuate automatic derivation of eligibility traces of the method 1800 of FIG. 18. In FIG. 19A, the designators #18XX refer to the respective steps of the method 1800 of FIG. 18, according to one or more implementations.

[0225] FIG. 19B is a python script illustrates exemplary object constructs for use with the python script of FIG. 19A. The script shown in FIG. 19B is configured to interface with MATLAB® symbolic computations engine, according to one implementation. It will be appreciated by those skilled in the arts, that various other symbolic computations computer aided design (CAD) tools (e.g., Mathematica, etc.) may be used with the methodology described with respect to FIGS. 18-19B.

Performance Results

[0226] FIGS. 9A through 17B present performance results obtained during simulation and testing by the Assignee hereof, of exemplary computerized spiking network apparatus configured to implement generalized learning framework described above with respect to FIGS. 3-6B. The exemplary apparatus, in one implementation, comprises learning block (e.g., the block 420 of FIG. 4) that implemented using spiking neuronal network 700, described in detail with respect to FIG. 7, supra.

[0227] The average performance (e.g. the function $\langle F \rangle_{x,y,r}$ average of Eqn. 33-Eqn. 43) may be determined over a time interval T_{av} that is configured in accordance with the specific application. In some implementations, the spike rate of the network output $y(t)$ may be configured between 5 and 100 Hz. In one or more implementations, the T_{av} may be configured to exceed the spike rate of output by a factor of 5 to 10000. In one such implementation, the spike rate may comprise 70 Hz output, and the averaging time may be selected at about 100 s.

Combined Supervised and Reinforcement Learning Tasks

[0228] In some implementations, in accordance with the framework described by, inter alia, Eqn. 43, the cost function F_{sr} , corresponding to a combination of supervised and reinforcement learning tasks, may be expressed as follows:

$$F_{sr} = aF_{sup} + bF_{reinf}$$

where F_{sup} and F_{reinf} are the cost functions for the supervised and reinforcement learning tasks, respectively, and a , b are coefficients determining relative contribution of each cost component to the combined cost function. By varying the coefficients a , b during different simulation runs of the spik-

ing network, effects of relative contribution of each learning method on the network learning performance may be investigated.

[0229] In some implementations, such as those involving classification of spiking input patterns derived from speech data in order to determine speaker identity, the supervised learning cost function may comprise a product of the desired spiking pattern $y^d(t)$ (belonging to a particular speaker) with filtered output spike train $y(t)$. In some implementations, such as those involving a low pass exponential filter kernel, the F_{sup} may be computed using the following expression:

$$F_{sup} = \int_{-\infty}^{\infty} (y(s)e^{-(t-s)/\tau_d} ds)(y^d(t) - C), \quad (\text{Eqn. 63})$$

where τ_d is the trace decay constant, and C is the bias constant configured to introduce penalty associated with extra activity of the neuron does not corresponding to the desired spike train.

[0230] The cost function for reinforcement learning may be determined as a sum of positive and negative reinforcement contributions that are received by the neuron via two spiking channels ($y^+(t)$ and $y^-(t)$):

$$F_{reinf} = y^+(t) - y^-(t), \quad (\text{Eqn. 64})$$

where the subtraction of spike trains is understood as in Eqn. 65. Reinforcement may be generated according to the task that is being solved by the neuron.

[0231] A composite cost function for simultaneous reinforcement and supervised learning may be constructed using a linear combination of contributions provided by Eqn. 63 and Eqn. 64:

$$F = aF_{sup} + bF_{reinf} == \quad (\text{Eqn. 65})$$

$$a \int_{-\infty}^{\infty} \left(\sum_i \delta(t - t_i) e^{-(t-s)/\tau_d} ds \right) \left(\sum_i \delta(t - t_i^d) - C \right) + b \left(\sum_i \delta(t - t_i^+) - \sum_j \delta(t - t_j^-) \right).$$

Using the description of Eqn. 65, the spiking neuron network (e.g., the network 700 of FIG. 7) may be configured to maximize the combined cost function F_{sr} using one or more of the methodologies described in a co-owned and co-pending U.S. patent application entitled "APPARATUS AND METHODS FOR IMPLEMENTING LEARNING RULES USING PROBABILISTIC SPIKING NEURAL NETWORKS" filed contemporaneously herewith, and incorporated supra.

[0232] FIGS. 9A-9F present data related to simulation results of the spiking network (e.g., the network 700 of FIG. 7) configured in accordance with supervised and reinforcement rules described with respect to Eqn. 65, supra. The input into the network (e.g., the neuron 730 of FIG. 7) is shown in the panel 900 of FIG. 9A and may comprise a single 100-dimensional input spike stream of length 600 ms. The horizontal axis denotes elapsed time in milliseconds, the vertical axis denotes each input dimension (e.g., the connection 714 in FIG. 7), each row corresponds to the respective connection, and dots denote individual spikes within each row. The panel 902 in FIG. 9A, illustrates supervisor signal, comprising a sparse 600 ms-long stream of training spikes, delivered to the

neuron **730** via the connection **704**, in FIG. 7. Each dot in the panel **902** denotes the desired output spike $y^d(t)$.

[0233] The reinforcement signal may be provided to the neuron according to the following protocol:

[0234] If the network (e.g., the network **700** of FIG. 7) generates one spike within a [0.50 ms] time window from the onset of the input, then it may receive the positive reinforcement spike, illustrated in the panel **904** in FIG. 9A.

[0235] If the network does not generate outputs during that interval or generates more than one spike, then it may receive negative reinforcement spike, illustrated in the panel **906** in FIG. 9A.

[0236] If the network is active (generates output spikes) during time intervals [200 ms, 250 ms] and [400 ms, 450 ms], then it may receive negative reinforcement spike.

[0237] Reinforcement signals may not be generated during one or more other intervals.

A maximum reinforcement configuration may comprise (i) one positive reinforcement spike and (ii) no negative reinforcement spikes. A maximum negative reinforcement configuration may comprise (i) no positive reinforcement spikes and (ii) three negative reinforcement spikes.

[0238] The output activity (e.g., the spikes $y(t)$) of the network **660** prior to learning, illustrated in the panel **910** of FIG. 9A, shows that output **910** comprises few output spikes generated at random times that do not display substantial correlation with the supervisor input **902**. The reinforcement signals **904**, **906** show that the untrained neuron does not receive positive reinforcement (manifested by the absence of spikes in the panel **904**) and receives two spikes of negative reinforcement (shown by the dots at about 50 ms and about 450 ms in the panel **906**) because the neuron is quiet during [0 ms-50 ms] interval and it spikes during [400 ms-450 ms] interval.

[0239] FIG. 9B illustrates output activity of the network **700**, operated according to the supervised learning rule, which may be effected by setting the coefficients (a,b) of Eqn. 65 as follows: $a=1$, $b=0$. Different panels in FIG. 9B present the following data: panel **900** depicts feed-forward input into the network **700** of FIG. 7; panel **912** depicts supervisor (training) spiking input; and panels **914**, **916** depict positive and negative reinforcement input spike patterns, respectively.

[0240] The output of the network shown in the panel **910** displays a better correlation (compared to the output **910** in FIG. 9A) of the network with the supervisor input. Data shown in FIG. 9B confirm that while the network learns to repeat the supervisor spike pattern it fails to perform reinforcement task (receives 3 negative spikes—maximum possible reinforcement).

[0241] FIG. 9C illustrates output activity of the network, operated according to the reinforcement learning rule, which may be effected by setting the coefficients (a,b) of Eqn. 65 as follows: $a=0$, $b=1$. Different panels in FIG. 9C present the following data: panel **900** depicts feed-forward input into the network; panel **922** depicts supervisor (training) spiking input; panels **924**, **926** depict positive and negative reinforcement input spike patterns, respectively.

[0242] The output of the network, shown in the panel **920**, displays no visible correlation with the supervisor input, as expected. At the same time, network receives maximum possible reinforcement (one positive spike and no negative spikes) illustrated by the data in panels **924**, **926** in FIG. 9C.

[0243] FIG. 9D illustrates output activity of the network **700**, operated according to the reinforcement learning rule augmented by the supervised learning, effected by setting the coefficients (a,b) of Eqn. 65 as follows: $a=0.5$, $b=1$. Different panels in FIG. 9D present the following data: panel **900** depicts feed-forward input into the network; panel **932** depicts supervisor (training) spiking input; and panels **934**, **936** depict positive and negative reinforcement input spike patterns, respectively.

[0244] The output of the network shown in the panel **930** displays a better correlation (compared to the output **910** in FIG. 9A) of the network with the supervisor input. Data presented in FIG. 9D show that network receives maximum possible reinforcement (panel **934**, **936**) and begins starts to reproduce some of the supervisor spikes (at around 400 ms and 470 ms) when these do not contradict with the reinforcement learning signals. However, not all of the supervised spikes are echoed in the network output **930**, and additional spikes are present (e.g., the spike at about 50 ms), compared to the supervisor input **932**.

[0245] FIG. 9E illustrates data obtained for an equal weighting of supervised and reinforcement learning: ($a=1$; $b=1$ in of Eqn. 65). The reinforcement traces **944**, **946** of FIG. 9E show that the network receives maximum reinforcement. The network output (trace **940**) contains spikes corresponding to a larger portion of the supervisor input (the trace **942**) when compared to the data shown by the trace **930** of FIG. 9E, provided the supervisor input does not contradict the reinforcement input. However, not all of the supervised spikes of FIG. 9E are echoed in the network output **940**, and additional spikes are present (e.g., the spike at about 50 ms), compared to the supervisor input **942**.

[0246] FIG. 9F illustrates output activity of the network, operated according to the supervised learning rule augmented by the reinforcement learning, effected by setting the coefficients (a,b) of Eqn. 65 as follows: $a=1$, $b=0.4$. The output of the network shown in the panel **950** displays a better correlation with the supervisor input (the panel **952**), as compared to the output **940** in FIG. 9E. The network output (**950**) is shown to repeat the supervisor input (**952**) event when the latter contradicts with the reinforcement learning signals (traces **954**, **956**). The reinforcement data (**956**) of FIG. 9F show that while the network receive maximum possible reinforcement (trace **954**), it is penalized (negative spike at 450 ms on trace **956**) for generating output that is inconsistent with the reinforcement rules.

Combined Supervised and Unsupervised Learning Tasks

[0247] In some implementations, in accordance with the framework described by, inter alia, Eqn. 43, the cost function F_{su} , corresponding to a combination of supervised and unsupervised learning tasks, may be expressed as follows:

$$F_{su} = aF_{sup} + c(-F_{unsup}). \quad (\text{Eqn. 66})$$

where F_{sup} is described by, for example, Eqn. 34, F_{unsup} is the cost function for the unsupervised learning tasks, and a, c are coefficients determining relative contribution of each cost component to the combined cost function. By varying the coefficients a, c during different simulation runs of the spiking network, effects of relative contribution of individual learning methods on the network learning performance may be investigated.

[0248] In order to describe the cost function of the unsupervised learning, an instantaneous Kullback-Leibler divergence between two point processes may be used:

$$F_{unsup} = \ln(p(t)) - \ln(p^d(t)) \quad (\text{Eqn. 67})$$

where $p(t)$ is the probability of the actual spiking pattern generated by the network, and $p^d(t)$ is the probability of a spiking pattern generated by Poisson process. The unsupervised learning task in this implementation may serve to minimize the function of Eqn. 67 such that when the two probabilities $p(t) = p^d(t)$ are equal at all times, then the network may generate output spikes according to Poisson distribution.

[0249] Accordingly, the composite cost function for simultaneous unsupervised and supervised learning may be expressed as a linear combination of Eqn. 63 and Eqn. 67:

$$F = aF_{sup} + c(-F_{unsup}) \quad (\text{Eqn. 68})$$

$$= a \left[\int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} \left(\sum_i \delta(t - t_i) e^{\frac{t-s}{\tau_d}} ds \right) \sum_i \delta(t - t_i^d) dt \right) - C \right] + c(\ln(p^d(t)) - \ln(p(t)))$$

[0250] Referring now to FIGS. 8A-8C, data related to simulation results of the spiking network 700 may be configured in accordance with supervised and unsupervised rules described with respect to Eqn. 68, supra. The input into the neuron 730 is shown in the panel 1000 of FIG. 10A-10C and may comprise a single 100-dimensional input spike stream of length 600 ms. The horizontal axis denotes elapsed time in ms, the vertical axis denotes each input dimension (e.g., the connection 714 in FIG. 7), and dots denote individual spikes.

[0251] FIG. 10A illustrates output activity of the network (e.g., network 700 of FIG. 7), operated according to the supervised learning rule, which is effected by setting the coefficients (a,c) of Eqn. 68 as follows: $a=1$, $b=0$. The panel 1002 in FIG. 10A, illustrates supervisor signal, comprising a sparse 600 ms-long stream of training spikes, delivered to the neuron 730 via the connection 704 of FIG. 7. Dot in the panel 1002 denotes the desired output spike $y^d(t)$.

[0252] The output activity (the spikes $y(t)$) of the network, illustrated in the panel 1010 of FIG. 10A, shows that the network successfully repeats the supervisor spike pattern which does not behave as a Poisson process with 60 Hz firing rate.

[0253] FIG. 10B illustrates output of the network, where supervised learning rule is augmented by 15% fraction of Poisson spikes, effected by setting the coefficients (a,c) of Eqn. 68 as follows: $a=1$, $c=0.15$. The output activity of the network, illustrated in the panel 1020 of FIG. 10B, shows that the network successfully repeats the supervisor spike pattern 1022 and further comprises additional output spikes are randomly distributed and the total number of spikes is consistent with the desired firing rate.

[0254] FIG. 10C illustrates output of the network 700, where supervised learning rule is augmented by 80% fraction of Poisson spikes, effected by setting the coefficients (a,c) of Eqn. 68 as follows: $a=1$, $c=0.8$. The output activity of the network 700, illustrated in the panel 1030 of FIG. 10B, shows that the network output is characterized by the desired Poisson distribution and the network tries to repeat the supervisor pattern, as shown by the spikes denoted with circles in the panel 1030 of FIG. 10C.

Combined Supervised, Unsupervised, and Reinforcement Learning Tasks

[0255] In some implementations, in accordance with the framework described by, inter alia, Eqn. 43, the cost function F_{sur} , representing a combination of supervised, unsupervised, and/or reinforcement learning tasks, may be expressed as follows:

$$F_{sur} = aF_{sup} + bF_{rein} + c(-F_{unsup}) \quad (\text{Eqn. 69})$$

[0256] Referring now to FIG. 11, data related to simulation results of the spiking network configured in accordance with supervised, reinforcement, and unsupervised rules described with respect to Eqn. 69, supra. The network learning rules comprise equally weighted supervised and reinforcement rules augmented by a 15% fraction of Poisson spikes, representing unsupervised learning. Accordingly, the weight coefficients of Eqn. 69 are set as follows: $a=1$; $b=1$; $c=0.1$.

[0257] In FIG. 11, panel 1100 depicts the input comprising a single 100-dimensional input spike stream of length 600 ms; panel 902 depicts the supervisor input; and panels 904, 906 depict positive and negative reinforcement inputs into the network 700 of FIG. 7, respectively.

[0258] The network output, presented in panel 1110 in FIG. 11, comprises spikes that generated based on (i) reinforcement learning (the first spike at 50 ms leads to the positive reinforcement spike at 60 ms in the panel 1104); (ii) supervised learning (e.g., spikes between 400 ms and 500 ms interval); and (iii) random activity spikes due to unsupervised learning (e.g., spikes between 100 ms and 200 ms interval).

Supervised Learning Tasks

[0259] FIG. 12 presents to simulation results of the spiking network (e.g., the network 700 of FIG. 7) configured in accordance with supervised learning rule. In one implementation, the costs function, comprising a measure between the desired $y^d(t)$ and the actual $y(t)$ output spike train of the neuron, may be determined using low-pass filtering of the desired and actual spike trains, as follows:

$$F(x, y) = d(y, y^d) = \left(\int_T y(s) a(t-s) ds - \int_T y^d(s) b(t-s) ds \right)^2 \quad (\text{Eqn. 70})$$

where,

$$y(t) = \sum_i \delta(t - t_i^{out}), \quad (\text{Eqn. 71})$$

$$y^d(t) = \sum_j \delta(t - t_j^d),$$

x is the input signal, δ is the Dirac delta function, t_i^{out} , t_j^d are the output and desired spike times, respectively, and $a(t)$, $b(t)$ are some positive finite-response kernels. In one or more implementations, one or both kernels (e.g., $a(t)$) may comprise an exponential trace:

$$a(t) = e^{-t/\tau_a}, \quad (\text{Eqn. 72})$$

where τ_a is the time interval, typically selected according to average output firing rate of a neuron.

[0260] The results, depicted in FIG. 12 correspond to the input signal $\{X\}$ comprised of 100 input spike trains (shown in the panel 1200 in FIG. 12); the network is able to reproduce

the desired spike train (shown in the panel **1202**), as illustrated by very close match of the network output shown in the panel **1210**.

Predictive Supervised Learning Tasks

[0261] The generalized learning framework described herein is not limited to the applications characterized by an immediate correspondence between the network activity and the cost function. In one or more implementations, the present framework may be applied to a spike prediction task, using spike train distance as the cost function configured as:

$$F(x,y)=\int_T \int_{-\infty}^t y(s)\alpha(t-s)ds \sum_j \delta(t-t_j^d)dt \quad (\text{Eqn. 73})$$

where $\alpha(t)$ is the alpha function. As shown by Eqn. 73, first, the actual output spike train $y(t)$ may be low-pass filtered via a convolution with the alpha-function kernel. This may create an output trace that reaches a maximum value after a time delay of τ_d from the output spike occurrence. Subsequently, the value of the filtered trace may be evaluated at the desired spike times t_j^d . The resulting cost function F may reach its maximum when spike output precede desired output (i.e., supervisory input) by the time interval of τ_d . Accordingly, the network learns to maximize the cost function of Eqn. 73 in order to predict the desired spikes with the exact delay. The predictive supervised learning may be used, in some implementations, for a variety of prediction tasks such as, for example, building forward models.

[0262] FIG. 13 presents simulation results of the spiking network (e.g., the network **700** of FIG. 7) configured in accordance with predictive supervised learning rule of Eqn. 73. The results depicted in FIG. 13 correspond to the input signal $\{X\}$ comprised of 100 input spike trains (shown in the panel **1300** in FIG. 13); the network is able to predict the desired spike train (shown in the panel **1302**), as illustrated close match of the network output shown in the panel **1310**, and indicated by the arrows **1304** in FIG. 13. Notice that the network predictive output $y(t)$ is not the exact shifted replica of the desired signal pattern $y^d(t)$, as may be seen from comparing data in the panels **1302**, **1310** in FIG. 13. This is due to, partly, the low-pass filter kernel, embodied in the performance function F which ‘smooth’ out the presence of multiple closely-spaced spikes, indicated by spike group marked with the arrow **1306** in FIG. 13.

Reciprocal Supervised and Learning Tasks

[0263] In some implementations, supervised learning may be used in order to cause pauses in the activity of the neuron prior to generating the desired spikes, also referred to as the reciprocal supervised learning. This task may be formalized as minimization of the performance function F of Eqn. 73 with a constant non-associative potentiation of the synaptic weights (e.g., the weights **712** in FIG. 7. The non-associative potentiation may lead to a gradual increase of the firing rate of the neuron (performing exploration), while the associative minimization of the function F may cause pauses with a certain delay before the supervised spike, as illustrated in FIG. 14. The results depicted in FIG. 14 correspond to the input signal $\{X\}$ comprised of 100 input spike trains (shown in the panel **1400** in FIG. 14). The network output (shown in the panel **1402**) comprises periods of inactivity (pauses) denoted by broken lines **1406_1**, **1406_2** in FIG. 14, preceding network output pulse associated with respective desired output pulse of the training signal $y^d(t)$ (the association being indicated by the arrows **1404** in FIG. 14). It is noteworthy that

multiple consecutive pulses in the training signal $y^d(t)$ (e.g., the pulse group **1408**) cause longer period of inactivity, as indicated by the inactivity period **1406_1**, as compared to the inactivity period **1406_2**, that is associated with a single pulse in the training signal $y^d(t)$.

Unsupervised Learning Tasks

Mutual Information Maximization

[0264] In one or more implementations, the learning framework of the disclosure may be applied to unsupervised learning where the cost function F is configured based on inputs x and outputs $y(t)$ of the network (e.g., the network **700** of FIG. 7). In some implementations of unsupervised learning, the mutual information $I(x,y)$ between the input x and output y spike trains of the networks may be used as the cost function F , so that:

$$\langle F \rangle = I(x,y) = \langle \ln(p(y|x)/p(y)) \rangle_{x,y} \Rightarrow F(x,y) = h(y) - h(y|x) \quad (\text{Eqn. 74})$$

where $h(y)$ is the unconditional per-stimulus entropy (surprisal), described by (Eqn. 13). Learning by the network **700** may be configured to maximize the cost function F of Eqn. 74. [0265] FIG. 15A presents simulation results of the spiking network (e.g., the network **700** of FIG. 7) configured in accordance with unsupervised learning rule of Eqn. 74. The results depicted in FIG. 15A correspond to the input signal $\{X\}$ comprised of 100 input spike trains (shown in the panel **1500** in FIG. 15). The network output (shown in the panel **1510**) show that neurons activity does not decay to zero and does not increase uncontrollably, so that the neuron is capable of extracting information from the input **1500**. The panel **1502** illustrates evolution of weights during learning. While some of the weights shown in the panel **1502** exhibit continuing growth, the resulting activity of the network (panel **1510**) is not adversely affected, as the average of the weights remains constrained.

Minimization of Kullback-Leibler Divergence

[0266] In some implementations of unsupervised learning, the performance function may be configured to minimize the Kullback-Leibler divergence (D_{KL}) between the output spike train distribution $p(y)$ and the desired probability distribution $p^d(y)$. The performance function may be expressed as:

$$\langle F \rangle = D_{KL}(p||p^d) = \langle \ln(p(y)/p^d(y)) \rangle_{x,y} \Rightarrow F(x,y) = h^d - h(y) \quad (\text{Eqn. 75})$$

[0267] In one or more implementations, the desired output $y^d(t)$ is characterized as the Poisson point process with average firing rate r and $p^d(y)$ comprises Poisson distribution. This configuration of results in a constrained average firing rate of the neuron (e.g., the neuron **730** in FIG. 7), thereby preventing the weights from growing infinitely large. Furthermore, Poisson distribution of the desired output, causes output $y(t)$ with exponential distribution of inter-spike intervals (ISI). The Poisson process is the point process with the largest entropy for that particular firing rate and therefore it is the most informative point process. This means that minimizing the Kullback-Leibler divergence between the output distribution and the Poisson distribution causes maximization of information transmission by the network subject to the firing rate (energy) constraint.

[0268] FIGS. 15B-15C present simulation results of the spiking network (e.g., the network **700** of FIG. 7) configured

in accordance with unsupervised learning rule of Eqn. 75, corresponding to the input signal $\{X\}$ comprised of 100 input spike trains, (shown in the panel 1540 in FIG. 15C). Each input spike train in the panel 1540 is characterized by Poisson distribution with 50-Hz average firing rate.

[0269] Network averaged performance and weight evolution with time are shown in the panels 1530, 1532, of FIG. 15B, respectively. The performance shown in FIG. 15B corresponds to maximization of negative divergence, hence the performance increases over time. The weight evolution in the panel 1532 illustrates weight stabilization after the best (stable) performance is achieved. As shown by the neuron weight evolution (panel 1532 in FIG. 15B) and network output) the KL-divergence minimization ensures that weight do not grow substantially after the desired performance (average firing rate) is achieved. Accordingly, firing rate of the neurons (panel 1542 in FIG. 15C) remains controlled and does not increase infinitely with time.

Reinforcement Learning Tasks

[0270] One or more implementations of reinforcement learning may require solving adaptive control task (e.g., AUV/UAV navigation) without having detailed prior information about the dynamics of the controlled plant (e.g., the plant 514 in FIG. 5). The reinforcement signal (e.g., the signal 504 in FIG. 5) is typically used to specify to the adaptive controller (e.g., the controller 520 of FIG. 5) whether prior behavior led to “desired” or “undesired” results.

[0271] FIG. 16 illustrates operation of the neural network of FIG. 7, configured to control navigation of an autonomous unmanned vehicle (AUV) along a trajectory using reinforcement learning. Data in the top panel 1600 illustrate mean distance $\langle d \rangle$ between the actual position of an AUV $y(t)$ and the desired position of the AUV. Data in the bottom panel 1610 present variance of the distance d . Data in FIG. 16 illustrate improved network operation, characterized by a decrease of the position mean error and variance with time due to learning rules that enable provides minimization of average costs (maximization of average performance).

[0272] FIGS. 17A-17B illustrate operation of the neural network of FIG. 7, configured to implement coincidence detector. The neuron 730 of FIG. 7 may be configured to receive two spiking inputs, presented by the traces 1700-1, 1700_2 in FIG. 17A. The output of the network and the reward signal are presented in the panels 1710, 1702 in FIG. 17A, respectively. The neuron 730 may be configured to generate an output spike only when it receives two input spikes simultaneously and remain silent otherwise. When the neuron performs coincidence detection (e.g., it spikes at the right time, as illustrated for example by the arrow 1712 in FIG. 17A), it receives a positive reward spike (reinforcement equals to one, illustrated by the arrow 1704 in FIG. 17A). If a neuron does not generate a spike when the two input spikes are present, or if it spikes after only one input, the reinforcement may be negative (e.g., reinforcement equals to zero).

[0273] The learning process is depicted in FIG. 17B, where the panel 1720 displays performance, determined by an average normalized reinforcement through epoch (epoch equals to 250 s); and the panel 1730 presents evolution of weights during learning. Performance measure in FIG. 17B is configured as a ratio of correct coincidence detections by neuron and it ranges from 0.5 (corresponding to random guesses: no detection), to 1.0, when each output spike is associated with coincident input (perfect detection). As seen from the data

shown in FIG. 17B, the performance gradually increases due partly to increase in weights and after about 2000 s weight change stabilizes and performance remains within a range between 0.75 and 0.95. The network operation illustrated in FIG. 17B, is advantageously enabled by learning rules that provide minimization of average costs (maximization of average performance).

Exemplary Uses and Applications of Certain Aspects of the Invention

[0274] Generalized learning framework apparatus and methods of the disclosure may allow for an improved implementation of single adaptive controller apparatus system configured to simultaneously perform a variety of control tasks (e.g., adaptive control, classification, object recognition, prediction, and/or clusterisation). Unlike traditional learning approaches, the generalized learning framework of the present disclosure may enable adaptive controller apparatus, comprising a single spiking neuron, to implement different learning rules, in accordance with the particulars of the control task.

[0275] In some implementations, the network may be configured and provided to end users as a “black box”. While existing approaches may require end users to recognize the specific learning rule that is applicable to a particular task (e.g., adaptive control, pattern recognition) and to configure network learning rules accordingly, a learning framework of the disclosure may require users to specify the end task (e.g., adaptive control). Once the task is specified within the framework of the disclosure, the “black-box” learning apparatus of the disclosure may be configured to automatically set up the learning rules that match the task, thereby alleviating the user from deriving learning rules or evaluating and selecting between different learning rules.

[0276] Even when existing learning approaches employ neural networks as the computational engine, each learning task is typically performed by a separate network (or network partition) that operate task-specific (e.g., adaptive control, classification, recognition, prediction rules, etc.) set of learning rules (e.g., supervised, unsupervised, reinforcement). Unused portions of each partition (e.g., motor control partition of a robotic device) remain unavailable to other partitions of the network even when the respective functionality of not needed (e.g., the robotic device remains stationary) that may require increased processing resources (e.g., when the stationary robot is performing recognition/classification tasks).

[0277] When learning tasks change during system operation (e.g., a robotic apparatus is stationary and attempts to classify objects), generalized learning framework of the disclosure may allow dynamic re-tasking of portions of the network (e.g., the motor control partition) at performing other tasks (e.g., visual pattern recognition, or object classifications tasks). Such functionality may be effected by, inter alia, implementation of generalized learning rules within the network which enable the adaptive controller apparatus to automatically use a new set of learning rules (e.g., supervised learning used in classification), compared to the learning rules used with the motor control task. These advantages may be traded for a reduced network complexity, size and cost for the same processing capacity, or increased network operational throughput for the same network size.

[0278] Generalized learning methodology described herein may enable different parts of the same network to implement different adaptive tasks (as described above with respect to

FIGS. 5B-5C). The end user of the adaptive device may be enabled to partition network into different parts, connect these parts appropriately, and assign cost functions to each task (e.g., selecting them from predefined set of rules or implementing a custom rule). The user may not be required to understand detailed implementation of the adaptive system (e.g., plasticity rules and/or neuronal dynamics) nor is he required to be able to derive the performance function and determine its gradient for each learning task. Instead, the users may be able to operate generalized learning apparatus of the disclosure by assigning task functions and connectivity map to each partition.

[0279] Furthermore, the learning framework described herein may enable learning implementation that does not affect normal functionality of the signal processing/control system. By way of illustration, an adaptive system configured in accordance with the present disclosure (e.g., the network 600 of FIG. 6A or 700 of FIG. 7) may be capable of learning the desired task without requiring separate learning stage. In addition, learning may be turned off and on, as appropriate, during system operation without requiring additional intervention into the process of input-output signal transformations executed by signal processing system (e.g., no need to stop the system or change signals flow).

[0280] In one or more implementations, the generalized learning apparatus of the disclosure may be implemented as a software library configured to be executed by a computerized neural network apparatus (e.g., containing a digital processor). In some implementations, the generalized learning apparatus may comprise a specialized hardware module (e.g., an embedded processor or controller). In some implementations, the spiking network apparatus may be implemented in a specialized or general purpose integrated circuit (e.g., ASIC, FPGA, and/or PLD). Myriad other implementations may exist that will be recognized by those of ordinary skill given the present disclosure.

[0281] Advantageously, the present disclosure can be used to simplify and improve control tasks for a wide assortment of control applications including, without limitation, industrial control, adaptive signal processing, navigation, and robotics. Exemplary implementations of the present disclosure may be useful in a variety of devices including without limitation prosthetic devices (such as artificial limbs), industrial control, autonomous and robotic apparatus, HVAC, and other electromechanical devices requiring accurate stabilization, set-point control, trajectory tracking functionality or other types of control. Examples of such robotic devices may include manufacturing robots (e.g., automotive), military devices, and medical devices (e.g., for surgical robots). Examples of autonomous navigation may include rovers (e.g., for extra-terrestrial, underwater, hazardous exploration environment), unmanned air vehicles, underwater vehicles, smart appliances (e.g., ROOMBA®), and/or robotic toys. The present disclosure can advantageously be used in other applications of adaptive signal processing systems (comprising for example, artificial neural networks), including: machine vision, pattern detection and pattern recognition, object classification, signal filtering, data segmentation, data compression, data mining, optimization and scheduling, complex mapping, and/or other applications.

[0282] It will be recognized that while certain aspects of the disclosure are described in terms of a specific sequence of steps of a method, these descriptions are only illustrative of the broader methods of the invention, and may be modified as

required by the particular application. Certain steps may be rendered unnecessary or optional under certain circumstances. Additionally, certain steps or functionality may be added to the disclosed implementations, or the order of performance of two or more steps permuted. All such variations are considered to be encompassed within the disclosure disclosed and claimed herein.

[0283] While the above detailed description has shown, described, and pointed out novel features of the disclosure as applied to various implementations, it will be understood that various omissions, substitutions, and changes in the form and details of the device or process illustrated may be made by those skilled in the art without departing from the disclosure. The foregoing description is of the best mode presently contemplated of carrying out the invention. This description is in no way meant to be limiting, but rather should be taken as illustrative of the general principles of the invention. The scope of the disclosure should be determined with reference to the claims.

What is claimed:

1. Stochastic spiking neuron signal processing system configured to implement task specific learning, the system comprising:

a controller apparatus configured to generate output control signal y based at least in part on input signal, said controller apparatus characterized by a controller state parameter, and a control parameter and

a learning apparatus configured to:

generate an adjustment signal based at least in part on said input signal, said controller state parameter, and said output signal; and

provide said adjustment signal to said controller apparatus, thereby effecting said learning;

wherein:

said control parameter is configured in accordance with said task; and

said adjustment signal is configured to modify said control parameter based at least in part on said input signal and said output signal.

2. The system of claim 1, wherein said output control signal comprises a spike train configured based at least in part said adjustment signal; and

said learning apparatus comprises:

a task-specific block, configured independent from the controller state parameter, the task-specific block configured to implement said task-specific learning; and

a controller-specific block, configured independent from said task-specific learning;

wherein said task-specific learning is characterized by a performance function, said performance function configured to effect at least unsupervised learning rule.

3. The system of claim 2, further comprising a teaching interface operably coupled to said learning apparatus and configured to provide a teaching signal;

wherein: said teaching signal comprises a desired controller output signal; and

wherein said performance function is further configured to effect a supervised learning rule, based at least in part on said desired controller output signal.

4. The system of claim 3, wherein:
said teaching signal further comprises a reinforcement spike train associated with current performance of the controller apparatus relative to desired performance; and
said performance function is further configured to effect a reinforcement learning rule, based at least in part on said reinforcement spike train.
5. The system of claim 4, wherein:
said current performance is based at least in part on adjustment of said control parameter from a prior state to current state;
said reinforcement is positive when said current performance is closer to desired performance of said controller; and
said reinforcement is negative when said current performance is farther from said desired performance.
6. The system of claim 4, wherein said task-specific learning comprises a hybrid learning rule comprising a combination of said reinforcement, said supervised and said unsupervised learning rules simultaneous with one another.
7. The system of claim 1, wherein said adjustment signal is determined as a product of controller performance function with a gradient of per-stimulus entropy parameter, said gradient is determined with respect to said control parameter; and
per-stimulus entropy parameter is configured to characterize dependence of said output signal on (i) said input signal; and (ii) said control parameter.
8. The system of claim 7, wherein said per-stimulus entropy parameter is determined based on a natural logarithm of $p(y|x, w)$, where p denotes conditional probability of said output signal given said input signal x with respect to said control parameter w .
9. Computer readable apparatus comprising a storage medium, said storage medium comprising a plurality of instructions to adjust a learning parameter associated with a computerized spiking neuron configured to produce output spike signal consistent with (i) an input spike signal, and (ii) a learning task, the instructions configured to, when executed:
construct time derivative representation of a trace of a neuron state, based at least in part on said input spike signal and a state parameter;
obtain a realization of said trace, based at least in part in integrating said time derivative representation; and
determine adjustment of said learning parameter, based at least in part on said trace;
wherein said adjustment is configured to transition said neuron state towards a target state, said target state associated with said neuron generating said output spike signal.
10. The apparatus of claim 9, wherein said integrating said representation is effected via symbolic integration operation.
11. The apparatus of claim 9, wherein:
said state parameter is configured to characterize time evolution of said neuron state;
said realization of said trace comprises an analytic solution of said time derivative representation; and
said construct of said time derivative representation enables to attain said integration via symbolic integration operation.
12. The apparatus of claim 9, wherein:
said state parameter q is configured to characterize time evolution of said neuron state in accordance with a state evolution process characterized by:
a response mode and a transition mode, said response mode is associated with generating a neuronal response;
state transition term V describing changes of neuronal state in said transition mode;
state transition term R describing changes of state set in said response mode; and
state transition term G describing changes of state set due to said input.
13. The apparatus of claim 12, wherein:
said state parameter is configured to characterize neuron membrane voltage; and
said input comprises analog signal and said state transition term G is configured to describe changes of said voltage due to said analog signal.
14. The apparatus of claim 12, wherein said input comprises a plurality of spikes and said state transition term G is configured to describe changes of said state parameters due to an integral effect of said plurality of spikes; and
said output comprises at least said neuronal response.
15. The apparatus of claim 12, wherein:
said state parameter q comprises neuron excitability and;
said time derivative representation comprises a sum of V , R , G state transition terms each multiplicatively combined with said trace.
16. The apparatus of claim 12, wherein said
state transition term V comprises said trace multiplicatively combined with a Jacobian matrix J_v configured in accordance with said transition mode of said evolution process;
state transition term R comprises said trace multiplicatively combined with a Jacobian matrix J_r configured in accordance with said response mode of said evolution process; and
state transition term G comprises said trace multiplicatively combined with a Jacobian matrix J_g configured in accordance with said input x .
17. The apparatus of claim 9, wherein:
said input comprises feed-forward input into said neuron via an interface; and
said learning parameter comprises efficacy associated with said interface.
18. The apparatus of claim 17, wherein said interface comprises synaptic connection and said learning parameter comprises connection weight.
19. The apparatus of claim 9, wherein said state parameter q is configured to describe time evolution of said neuron state in accordance with a state evolution process, said evolution process characterized by an instantaneous probability density (IPD) of generating neuronal response;
said instructions are further configured to, when executed,
determine derivative of said IPD, with respect to said learning parameter, based at least in part on said trace and
obtain an instantaneous score function value, based at least in part on said derivative of said IPD; and
said determine said adjustment is based at least in part on said instantaneous score function value.
20. The apparatus of claim 19, wherein:
said determine said realization of said trace, said determine said derivative of said IPD, and said obtain instantaneous score function value cooperate to produce said adjustment such that a next instance of said neuron state,

associated with an adjusted value, configured based on said current value and said adjustment, is closer to said target state.

21. A computerized apparatus configured to process input spike train using hybrid learning rule, the apparatus comprising stochastic learning block configured to produce learning signal based at least in part on said input spike and training signal;

wherein said hybrid learning rule is configured to simultaneously effect reinforcement learning rule and unsupervised learning rule.

22. The apparatus of claim **21**, wherein said stochastic learning block is operable according to a stochastic process characterized by a current state and a desired state, said process being described by at least a state variable configured to transition said learning block from current state to said desired state;

said training signal r comprises a reinforcement spiking indicator associated with current performance relative to desired performance of the apparatus, said current performance corresponding to said current state and said desired performance corresponding to said desired state; said current performance is effected, at least partly, by transition from a prior state to said current state;

said reinforcement learning is configured based at least in part on said reinforcement spiking indicator so that it provides:

positive reinforcement when a distance measure between said current state and said desired state is smaller compared to said distance measure between said prior state and said desired state; and

negative reinforcement when said distance measure between said current state and said desired state is

greater compared to said distance measure between said prior state and said desired state.

23. The apparatus of claim **22**, wherein:

said training signal further comprises desired output spike train;

current performance is effected, at least partly, by transition from prior state to said current state; and said reinforcement learning is configured based at least in part on said reinforcement spiking indicator so that: positive reinforcement when said current performance is closer to said desired performance; and said reinforcement is negative when said current performance is farther from said desired performance.

24. The apparatus of claim **22**, wherein:

said stochastic learning block is operable according to stochastic process characterized by current state and desired state, said process being described by at least state variable configured to transition said learning block from current state to said desired state;

said hybrid learning rule is further configured to simultaneously effect reinforcement learning rule, unsupervised learning rule, and supervised learning rule;

said hybrid learning rule is characterized by a hybrid performance function, comprising a simultaneous combination of reinforcement learning performance function, supervised learning performance function, and unsupervised learning performance function; and

said simultaneous combination is effectuated by at least in part on a value of said hybrid performance function determined at a time step t , said value comprising reinforcement performance function value, supervised learning performance function value, and unsupervised learning performance function value.

* * * * *