(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 104685516 B
(45)授权公告日 2018.03.30

(21)申请号 201380043294.X
(22)申请日 2013.08.16
(65)同一申请的已公布的文献号
申请公布号 CN 104685516 A
(43)申请公布日 2015.06.03
(30)优先权数据
13/588,774 2012.08.17 US
(85)PCT国际申请进入国家阶段日
2015.02.13
(86)PCT国际申请的申请数据
PCT/US2013/055381 2013.08.16
(87)PCT国际申请的公布数据
WO2014/028855 EN 2014.02.20
(73)专利权人 高通股份有限公司
地址 美国加利福尼亚州
(72)发明人 O·西尼亚维斯基 E·伊日科维奇
(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 张立达 王英
(51)Int.Cl.
G06N 3/02(2006.01)
(56)对比文件
US 20120084241 A1,2012.04.05,
US 6014653 ,2000.01.11,
US 7849030 B2,2010.12.07,
US 8015130 B2,2011.09.06,
Izhikevich等.《simple model of spiking neurons》.《IEEE》.2003,1569-1572.
Paugam-Moisy等.《Delay learning and polychronization for reservoir computing》.《google》.2007,
审查员 王德方

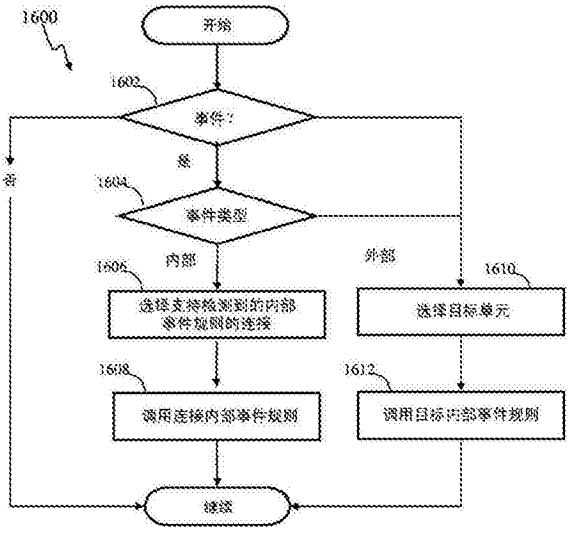
权利要求书3页 说明书61页 附图39页

(54)发明名称

用于在尖峰神经元网络中实现基于事件的更新的装置和方法

(57)摘要

可以执行人工神经元网络中的基于事件的更新。可以定义内部事件，以便更新神经元的输入连接。内部事件可以由外部信号触发和/或由神经元内部地触发。强化信号可以用于触发神经元的内部事件，以便执行突触更新，而不必具有突触后响应。可以定义外部事件，以便向期望的目标传送神经元的响应。外部事件和内部事件可以组合成复合事件，该复合事件被配置为实行连接更新以及向突触后目标传送尖峰。内部事件的范围可以包括各神经元，但不扩展到网络的其它神经元。相反地，外部事件的范围可以例如经由突触后尖峰传送扩展到网络的其它神经元。



CN 104685516 B

1.一种操作具有输入连接和输出连接的人工尖峰神经元的计算机实现的方法，所述方法由被配置为执行计算机程序模块的一个或多个处理器执行，所述方法包括：

提供第一事件类型的第一事件，所述第一事件基于经由被配置与所述输入连接分离的接口而被提供给所述神经元的外部信号，所述第一事件被配置为独立于对所述神经元的响应的传送来实行所述输入连接的可塑性的更新，所述输入连接的所述可塑性被用来更新与所述输入连接相关联的参数，所述更新的特征在于时间段；以及

提供第二事件类型的第二事件，所述第二事件被配置为经由所述输出连接实行对所述神经元的所述响应的传送，所述第二事件类型不同于所述第一事件类型。

2.如权利要求1所述的方法，其中：

所述参数包括输入连接权重。

3.如权利要求2所述的方法，其中：

所述输入连接被配置为向所述神经元提供前馈激励；

所述神经元根据被配置为产生目标结果的学习过程可操作；以及

所述外部信号包括强化信号，所述强化信号基于所述响应是否处于所述目标结果的预定范围内而被配置。

4.如权利要求3所述的方法，其中：

所述前馈激励包括以尖峰间间隔ISI为特征的多个尖峰，所述ISI包括所述多个尖峰中的个体尖峰之间的间隔；以及

所述时间段超过针对所述多个尖峰中的连续个体尖峰所确定的ISI。

5.如权利要求1所述的方法，其中，所述外部信号被配置为独立于所述第二事件类型的所述第二事件来引起所述第一事件类型的所述第一事件。

6.如权利要求1所述的方法，其中，所述第一事件类型的给定事件被配置为独立于所述第二事件类型的给定事件而被触发，以便启用所述第一事件类型的所述给定事件被触发后的无响应时间间隔。

7.如权利要求6所述的方法，其中：

输入连接权重的修改以可塑性时间窗为特征；以及

所述无响应时间间隔被配置为超过所述可塑性时间窗。

8.如权利要求1所述的方法，其中：

所述输入连接被配置为向所述神经元提供前馈激励。

9.如权利要求8所述的方法，其中，所述输入连接的所述可塑性更新基于提供所述外部信号的时间而被配置。

10.如权利要求9所述的方法，其中：

所述前馈激励包括至少一个尖峰；以及

所述可塑性更新还基于与所述至少一个尖峰相关联的时间而被配置。

11.如权利要求10所述的方法，其中：

所述可塑性更新包括调整通过所述输入连接的数据传输的可能性，所述调整基于以下两项中的一项或两项：(i)提供所述外部信号与神经元响应时间之间的时间间隔，或(ii)所述至少一个尖峰与所述神经元响应时间之间的时间间隔；以及

所述神经元响应时间与在提供所述外部信号之前所产生的神经元的响应相关联。

12.如权利要求10所述的方法，其中：

所述可塑性更新包括基于以下两项中的一项或两项来调整与所确定的输入连接相关联的效力：(i)提供所述外部信号与神经元响应时间之间的时间间隔，或(ii)所述至少一个尖峰与所述神经元响应时间之间的时间间隔；以及

所述神经元响应时间与在提供所述外部信号之前所产生的神经元的响应相关联。

13.如权利要求10所述的方法，其中：

所述可塑性更新包括调整以下项中任意项：(a)通信延迟以及(b)与所述输入连接相关联的权重，所述调整基于以下两项中的一项或两项而被确定：(i)提供外部信号与神经元响应时间之间的时间间隔，或(ii)所述至少一个尖峰与所述神经元响应时间之间的时间间隔；以及

所述神经元响应时间与提供所述外部信号之前所产生的神经元的响应相关联。

14.如权利要求1所述的方法，其中：

所述神经元被配置为根据以状态为特征的神经元过程而进行操作，所述状态具有与之相关联的响应产生阈值，所述响应产生阈值与产生所述响应的所述神经元相关联；

所述响应基于经由到所述神经元的所述输入连接而提供给所述神经元的一个或多个输入尖峰；以及

所述响应的特征在于突破所述响应产生阈值的状态。

15.如权利要求14所述的方法，还包括：

在与所述神经元相关联的存储位置存储所述状态的当前值；

其中，所述第一事件类型的所述第一事件被配置为将对所述存储位置的只读访问提供给所述输入连接。

16.如权利要求14所述的方法，其中：

所述第一事件基于以下各项中的一项或多项而被配置：(i)计时器到期；(ii)缓冲器溢出信号；或(iii)与所述神经元过程相关联的触发；以及

缓冲器被配置为存储所述一个或多个输入尖峰的时间关系曲线。

17.如权利要求14所述的方法，其中，所述第一事件类型的所述第一事件被配置为将对所述一个或多个输入尖峰的所述时间关系曲线的只读访问提供给所述输入连接。

18.如权利要求17所述的方法，其中，所述第一事件类型的所述第一事件还被配置将对一个或多个所述神经元的响应的时间关系曲线的只读访问提供给所述输入连接。

19.如权利要求1所述的方法，还包括：提供第三事件类型的第三事件，所述第三事件与突触前事件相关联，所述突触前事件引起所述输入连接的所述可塑性更新。

20.如权利要求1所述的方法，其中，所述第一事件和所述第二事件被配置为彼此同时被提供。

21.如权利要求1所述的方法，其中，所述第二事件触发引起所述输出连接的可塑性更新。

22.一种用于人工尖峰神经元网络的强化学习的计算机实现的方法，所述方法由被配置为执行计算机程序模块的一个或多个处理器执行，所述方法包括：

根据强化学习过程来操作所述神经元，所述强化学习过程被配置为基于输入信号和强化信号来产生输出；以及

基于所述强化信号，选择性地触发第一事件类型的第一事件，所述第一事件被配置为引起所述神经元的连接的可塑性更新；

其中：

所述选择性触发被配置为能够独立于所述输出的所述产生来应用所述可塑性更新；以及

所述输入信号传递关于所述神经元外部的环境的信息。

23.如权利要求22所述的方法，其中：

所述连接被配置为向所述神经元提供所述输入；

所述强化学习过程被配置为实现目标结果；以及

所述强化信号基于用于表征期望结果与所述输出之间的差异的测量。

24.如权利要求23所述的方法，其中，所述强化信号包括基于处于预定值之外的差异所确定的负回报。

25.一种被配置为实现网络中基于事件的更新的计算机化的尖峰神经元网络系统，所述系统包括：

被配置为执行计算机程序模块的一个或多个处理器，所述计算机程序模块的执行使得所述一个或多个处理器基于第一事件来更新分别与所述网络的第一尖峰神经元和第二尖峰神经元相关联的第一接口和第二接口；

其中，所述第一事件的特征在于不产生第一输出和第二输出，所述第一输出和所述第二输出分别由响应于所述事件的所述第一尖峰神经元和所述第二尖峰神经元产生。

26.如权利要求25所述的系统，其中，所述计算机程序模块的执行使得所述一个或多个处理器产生第二事件，所述第二事件被配置为使得经由分别与所述第一尖峰神经元和所述第二尖峰神经元相关联的第一输出接口或第二输出接口中的一个或两个来传送所述第一输出或所述第二输出中的一个或两个。

27.如权利要求25所述的系统，其中，所述计算机程序模块的执行使得所述一个或多个处理器产生定制事件，所述定制事件被配置为执行所述第一尖峰神经元或所述第二尖峰神经元中的一个或两个的用户定义的更新。

用于在尖峰神经元网络中实现基于事件的更新的装置和方法

[0001] 相关申请的交叉引用

[0002] 本申请涉及以下申请:2011年9月21日递交的、并且名称为“ELEMENTARY NETWORK DESCRIPTION FOR NEUROMORPHIC SYSTEMS(针对神经形态系统的初级网络描述)”(代理人案卷号021672-0399229)的美国专利申请No.13/239,123;2011年9月21日递交的、并且名称为“ELEMENTARY NETWORK DESCRIPTION FOR EFFICIENT LINK BETWEEN NEURONAL MODELS AND NEUROMORPHIC SYSTEMS(针对神经元模型与神经形态系统之间的有效链接的初级网络描述)”(代理人案卷号021672-0399230)的美国专利申请No.13/239,148;2011年9月21日递交的、并且名称为“ELEMENTARY NETWORK DESCRIPTION FOR EFFICIENT MEMORY MANAGEMENT IN NEUROMORPHIC SYSTEMS(针对神经形态系统中有效存储管理的初级网络描述)”(代理人案卷号021672-0399232)的美国专利申请No.13/239,155;2011年9月21日递交的、并且名称为“ELEMENTARY NETWORK DESCRIPTION FOR EFFICIENT IMPLEMENTATION OF EVENT-TRIGGERED PLASTICITY RULES IN NEUROMORPHIC SYSTEMS(针对神经形态系统中事件触发的可塑性规则的有效实现的初级网络描述)”(代理人案号021672-0399233)的美国专利申请No.13/239,163;2011年9月21日递交的、并且名称为“APPARATUS AND METHOD FOR PARTIAL EVALUATION OF SYNAPTIC UPDATES BASED ON SYSTEM EVENTS(用于局部评估基于系统事件的突触更新的装置和方法)”的美国专利申请No.13/239,259;以及2011年9月21日递交的、并且名称为“APPARATUS AND METHODS FOR SYNAPTIC UPDATE IN A PULSE-CODED NETWORK(用于编码脉冲网络中突触更新的装置和方法)”的美国专利申请No.13/239,255;2012年7月27日递交的、名称为“"APPARATUS AND METHODS FOR GENERALIZED STATE-DEPENDENT LEARNING IN SPIKING NEURON NETWORKS(用于尖峰神经元网络中广义情景关联学习的装置和方法)”(代理人案号021672-0410044,客户案号BC201212A)的美国专利申请序列No.13/560,902;以及2012年7月27日递交的、名称为“APPARATUS AND METHODS FOR EFFICIENT UPDATES IN SPIKING NEURON NETWORKS(用于尖峰神经元网络中的有效更新的装置和方法)”(代理人案号021672-040936,客户案号BC201221A)的美国专利申请序列No.13/560,891;前述申请中的每一个都是共同未决的、共有的,并通过引用方式将其整体并入本文。

[0003] 版权

[0004] 本专利文件的公开内容的一部分包含受版权保护的材料。版权所有者不反对任何人对该专利文件或专利公开进行复制,如同其出现在专利商标局的专利文档或记录中一样,但除此之外保留所有的版权。

技术领域

[0005] 本公开内容涉及用于模拟神经元网络的并行分布式计算机系统,其中,神经元网络执行神经计算,例如,视觉感知、电机控制和/或其它神经计算。

背景技术

[0006] 大部分神经元模型和系统由被简单单元(称为神经元)的网络组成，神经元彼此交互，并经由被称为突触的连接与外部世界交互。在这种神经元系统中的信息处理可以并行执行。

[0007] 一些现有的并行硬件架构和相应语言(其可以针对神经元模型的并行执行和仿真而被优化)，可以采用事件驱动的网络更新。当在神经网络中实现学习时，理想的是，能够采用选择性地修改网络的某些方面(例如，改变突触权重)的事件，同时保持网络的其它方面不受影响(例如，修改这些权重，而不产生神经元输出)。

[0008] 然而，现有的实施方式采用将连接权重调整与针对网络的神经元的突触后响应产生组合起来的事件。

[0009] 因此，格外需要额外的机制来将网络范围的连接调整从网络中的突触后响应产生去耦合，从而允许尖峰神经元网络中的学习算法能够更灵活地实现。

发明内容

[0010] 除其他方式外，本技术还通过提供用于实现灵活的基于事件的更新的装置和方法来满足前述需求。

[0011] 本公开内容的一个方面涉及一种操作具有输入连接和输出连接的人工尖峰神经元的方法。该方法发包括提供第一事件类型的一个或多个事件，其包括第一事件。第一事件可以被配置为实行输入连接的可塑性的更新。输入连接的可塑性可以被配置为调整与输入连接相关联的参数。所述调整的特征在于时间段。该方法可以包括提供第二事件类型的一个或多个事件，其包括第二事件。该第二事件可以被配置为实行经由输出连接的神经元的响应的传送。

[0012] 在一些实施方式中，第一事件可以基于提供给神经元的外部信号。参数可以包括输入连接权重。

[0013] 在一些实施方式中，输入连接可以被配置为向神经元提供前馈激励。神经元可以根据被配置为产生目标结果的学习过程进行操作。外部信号可以包括强化信号，其基于所述响应是否处于所述目标结果的预定范围内而被配置。

[0014] 在一些实施方式中，前馈激励可以包括多个以峰间间隔(ISI)为特征的多个尖峰，所述ISI包括多个尖峰的个体之间的间隔。该时间段可能超过针对多个尖峰的连续个体尖峰所确定的ISI。

[0015] 在一些实施方式中，外部信号可以被配置为独立于所述第二事件类型的所述一个或多个事件来引起所述第一事件类型的所述一个或多个事件。

[0016] 在一些实施方式中，第一事件类型的给定事件可以被配置为独立于第二事件类型的给定事件而被触发，以便启用第一事件类型的给定事件被触发后的无响应时间间隔。

[0017] 在一些实施方式中，权重的修改可以以可塑性时间窗为特征。免响应时间间隔可以被配置为超过可塑性时间窗。

[0018] 在一些实施方式中，输入连接能够向神经元提供前馈激励。外部信号可以经由被配置为与所述输入连接分离的接口而被提供。

[0019] 在一些实施方式中，输入连接的可塑性更新可以基于外部信号被提供的时间而被配置。

[0020] 在一些实施方式中，前馈激励可以包括至少一个尖峰。可塑性更新还可以基于与至少一个尖峰相关联的时间而被配置。

[0021] 在一些实施方式中，可塑性更新可以包括调整通过输入连接的数据传输的可能性。调整可以基于以下两项中的一项或两项：(i)提供外部信号与神经元响应时间之间的时间间隔；或者(ii)至少一个尖峰与响应时间之间的时间间隔。响应时间可以与提供外部信号之前所产生的神经元的响应相关联。

[0022] 在一些实施方式中，可塑性更新可以包括基于以下两项中的一项或两项来调整与所确定的输入连接相关联的效力：(i)提供外部信号与神经元响应时间之间的时间间隔；或者(ii)至少一个尖峰与响应时间之间的时间间隔。响应时间可以与提供外部信号之前所产生的神经元的响应相关联。

[0023] 在一些实施方式中，可塑性更新可以包括调整以下两项中的一项或两项：(a)通信延迟；或(b)与输入连接相关联的权重。调整可以基于以下两项中的一项或两项而被确定：(i)提供外部信号与神经元响应时间之间的时间间隔；或者(ii)至少一个尖峰与响应时间之间的时间间隔。响应时间可以与提供外部信号之前所产生的神经元的响应相关联。

[0024] 在一些实施方式中，神经元可以被配置为根据以状态为特征的神经元过程而进行操作。状态可以具有与之相关联的响应产生阈值。响应产生阈值可以与产生响应的神经元相关联。响应可以基于经由到神经元的输入连接而提供给神经元的一个或多个输入尖峰。响应的特征在于突破响应产生阈值的状态。

[0025] 在一些实施方式中，该方法可以包括在与神经元相关联的存储位置存储状态的当前值。第一事件类型的一个或多个事件可以被配置为将对存储位置的只读访问提供给输入连接。

[0026] 在一些实施方式中，第一事件可以基于以下各项中的一项或多项而被配置：(i)定时器过期，(ii)缓冲器溢出信号，或(iii)与神经元过程相关联的触发。缓冲器可以被配置为存储一个或多个输入尖峰的时间关系曲线。

[0027] 在一些实施方式中，第一事件类型的一个或多个事件可以被配置为将对一个或多个输入尖峰的时间关系曲线的只读访问提供给给输入连接。

[0028] 在一些实施方式中，第一事件类型的一个或多个事件可以被配置为将对一个或多个神经元响应的时间关系曲线的只读访问提供给输入连接。

[0029] 在一些实施方式中，与第一事件类型相关联的规定可以独立于第二事件类型而被配置，以便可塑性更新能够在不引起响应和/或与响应相关联的情况下实行。

[0030] 在一些实施方式中，该方法可以进一步包括提供第三事件类型的一个或多个事件，其包括第三事件。第三事件可能与突触前事件相关联。突触前事件能够引起输入连接的可塑性更新。

[0031] 在一些实施方式中，第一事件和第二事件可以被配置为彼此同时被提供。

[0032] 在一些实施方式中，第二事件能够引起输出连接的可塑性更新。

[0033] 本公开的另一方面涉及用于人工尖峰神经元网络的强化学习的方法。该方法可以包括根据强化学习过程来操作神经元。强化学习过程可以被配置为基于输入信号和强化信号来产生输出。该方法可以包括基于强化信号来选择性地触发第一事件类型的第一事件。第一事件可以被配置为引起所述神经元的连接的可塑性更新。选择性地触发可以被配置为

能够独立于输出的产生来应用可塑性更新。输入信号可以传送关于神经元外部的环境的信息。

[0034] 在一些实施方式中，连接可以被配置为向神经元提供输入。所述强化学习过程可以被配置为实现目标结果。所述强化信号可以基用于表征期望结果与所述输出之间的差异的测量。

[0035] 在一些实施方式中，所述强化信号可以包括基于处于预定值外的差异所确定的负回报。

[0036] 本公开的又一个方面涉及计算机化的尖峰神经元网络系统，其被配置为实现网络中的基于事件的更新。该系统可以包括一个或多个处理器，其被配置为执行计算机程序模块。所述计算机程序模块的执行可以使得一个或多个处理器基于第一事件来更新分别与网络的第一尖峰神经元和第二尖峰神经元相关联的第一接口和第二接口。第一事件的特征在于不产生第一输出和第二输出，所述第一输出和第二输出分别由响应于所述事件的第一尖峰神经元和第二尖峰神经元产生。

[0037] 在一些实施方式中，计算机程序模块的执行可以使得一个或多个处理器产生第二事件，第二事件被配置为经由分别与第一尖峰神经元和第二尖峰神经元相关联的第一输出接口或所述第二输出接口中的一个或两个来传送第一输出或第二输出中的一个或两个。

[0038] 在一些实施方式中，计算机程序模块的执行可以使得一个或多个处理器产生定制事件，所述定制事件被配置为执行第一尖峰神经元或第二尖峰神经元中的一个或两个的用户定义的更新。

[0039] 在参照附图来考虑以下描述和所附权利要求书时，本技术的这些和其它目的、特征和特性，以及结构的相关单元的操作方法和功能、以及部件的组合以及制造的经济，将变得更加显而易见，附图和以下描述以及所附权利要求书这些全部构成本说明书的一部分，其中，相同的附图标记在各个附图中指代相应的部分。然而，应该明确地理解，附图仅出于图示和描述的目的，并不旨在作为对本技术的限制的限定。如说明书和权利要求书中所使用，单数形式的“一个”、“一”和“所述”包括多个指示物，除非上下文明确地做出相反的描述。

附图说明

[0040] 图1是描绘了根据一个实施方式的神经仿真器的框图。

[0041] 图2是示出了根据一个实施方式的模型开发工作流程的框图。

[0042] 图3是示出了根据本公开内容的包括ENI通信信道的END仿真的一个实施方式的框图。

[0043] 图4是示出了根据一个实施方式的ENI通信信道的框图。

[0044] 图5是详述了根据一个实施方式的ENI中的内容、格式和预期行为的框图。

[0045] 图6是描绘了根据一个实施方式的ENI(方案)的操作模式的图示。

[0046] 图7A是示出了根据一个实施方式的通过ENI信道的完全同步通信的框图。

[0047] 图7B是示出了根据一个实施方式的、具有移位(在两个方向上S＝1)的图7A的实施方式的同步通信的框图。

[0048] 图8是示出了根据一个实施方式的具有接合点和突触的样本通信模型的框图。

[0049] 图9是示出了根据一个实施方式的包括ENI信道的通信模型的框图。

[0050] 图10是示出了根据一个实施方式的END引擎的各种示例的框图。

[0051] 图11A是示出了根据一个或多个实施方式的计算机化系统的框图，其中，该计算机化系统对尖峰网络中基于事件的更新机制是有用的。

[0052] 图11B是示出了根据一个或多个实施方式的神经形态计算机化系统的框图，其中，该神经形态计算机化系统对尖峰网络中基于事件的更新机制是有用的。

[0053] 图11C是示出了根据一个或多个实施方式的分层的神经形态计算机化系统架构的框图，其中，该分层的神经形态计算机化系统架构对尖峰网络中基于事件的更新机制是有用的。

[0054] 图11D是示出了根据一个或多个实施方式的细胞型神经形态计算机化系统架构的框图，其中，该细胞型神经形态计算机化系统架构对尖峰网络中基于事件的更新机制是有用的。

[0055] 图12是示出了根据一个实施方式的用于实现尖峰时间相关可塑性(STDP)的成对事件(doublet-event)规则的框图。

[0056] 图13包括根据一个实施方式的列出了通信参数的可能值的表。

[0057] 图14是示出了根据一个或多个实施方式的与事件驱动的更新一起使用的尖峰神经网络。

[0058] 图15A是示出了根据一个或多个实施方式的由于外部触发的树突状事件所引起的网络更新的程序列表。

[0059] 图15B是示出了根据一个或多个实施方式的由于内部触发的树突状事件所引起的网络更新的程序列表。

[0060] 图15C是示出了根据一个或多个实施方式的由于轴突和树突状事件所引起的网络更新的程序列表。

[0061] 图15D是示出了根据一个或多个实施方式的由于外部触发的定制事件所引起的网络更新的程序列表。

[0062] 图15E是示出了根据一个或多个实施方式的使用内部触发的定制事件的网络更新的程序列表。

[0063] 图15F是示出了根据一个或多个实施方式的使用包括多个触发条件的定制事件的网络更新的程序列表。

[0064] 图15G是示出了根据一个实施方式的网络更新的程序列表，其中，该网络更新包括被配置为不触发尖峰传送的树突状事件。

[0065] 图15H是示出了根据一个实施方式的使用定制事件的网络更新的程序列表，其中，该定制事件被配置为执行突触规则。

[0066] 图16A是示出了根据一个或多个实施方式与图14的网络连接一起使用的有效更新方法的逻辑流程图。

[0067] 图16B是示出了根据一个或多个实施方式与图14的神经网络一起使用的、包括时不变连接动态的有效更新方法的逻辑流程图。

[0068] 图16C是示出了根据一个或多个实施方式与图14的神经网络一起使用的突触权重的有效更新的逻辑流程图。

[0069] 图17是示出了根据一个或多个实施方式的被配置为在尖峰网络中实现事件驱动的更新机制的传感处理装置的框图。

[0070] 图18是示出了根据一个实施方式的成对事件规则的逻辑流程图的框图。

[0071] 图19是示出了根据一个或多个实施方式的END事件框架扩展的框图。

[0072] 本文所公开的所有附图的版权归全脑有限公司(brain corporation)(2013年)所有。保留所有权利。

具体实施方式

[0073] 现在将参考附图来详细描述本技术的实施方式,附图作为图示性的例子而被提供,以使得本领域的技术人员能够实践该技术。尤其是,下面的附图和例子并不意味着将本技术的范围限制于单个实施方式,而是通过与所描述或示出的元件中的一些或所有进行互换或组合的方式,其它实施方式也是可能的。在方便的情况下,将贯穿附图使用相同的附图标记来指示相同或相似的部件。

[0074] 在这些实施方式的某些元件可以部分地或全部地使用已知组件来实现的情况下,将仅描述这些已知部件中对于本技术的理解必要的那些部分,而这些已知组件的其它部分的详细描述将省略,以便模糊本技术。

[0075] 在本说明书中,示出单个组件的实施方式不应被视为是限制性的;相反,除非本文中明确地陈述,否则本技术旨在涵盖包括多个相同组件的其它实施方式,反之亦然。

[0076] 此外,本技术涵盖在本文中通过示例的方式提及的组件的当前知晓的等同物和未来知晓的等同物。

[0077] 如本文所使用的,词语“计算机”、“计算设备”和“计算机化设备”可以包括以下各项中的一项或多项:个人计算机(PC)和/或小型计算机(例如,台式机、膝上型计算机和/或其它PC)、大型计算机、工作站、服务器、个人数字助理(PDA)、手持式计算机、嵌入式计算机、可编程逻辑设备、个人通信器、平板计算机、便携式导航助手、配备J2ME的设备、蜂窝电话、智能电话、个人集成通信设备或娱乐设备和/或能够执行一组指令并且处理输入数据信号的任何其它设备。

[0078] 如本文所使用的,词语“计算机程序”或“软件”可以包括执行某种功能的任何人工和/或机器可识别步骤的序列。这种程序可以实质上呈现为任何编程语言或环境,其包括以下各项中的一项或多项:C/C++、C#、Fortran、COBOL、MATLAB™、PASCAL、Python、汇编语言、标记语言(例如,HTML、SGML、XML、VoXML)等,以及面向对象的环境,例如公共对象请求代理体系架构(CORBA)、Java™(包括J2ME、Java Beans)、二进制运行环境(例如,BREW),和/或其它编程语言和/或环境。

[0079] 如本文所使用的,词语“连接”、“链接”“传输信道”、“延迟线”、“无线”可以包括两个或多个实体(无论是物理的或逻辑/虚拟的)之间的因果关系,这使得能够在实体之间交换信息。

[0080] 如本文所使用的,词语“存储器”可以包括任何类型的集成电路或其它适于存储数字数据的存储设备。举个非限制的例子,存储器可以包括以下各项中的一项或多项:ROM、PROM、EEPROM、DRAM、SDRAM、DDR/2SDRAM、EDO/FPMS、RLDRAM、SRAM、“闪速”存储器(例如,NAND/NOR)、忆阻器(MEMRISTER)存储器、PSRAM、和/或其它类型的存储器。

[0081] 如本文所使用的，词语“微处理器”和“数字处理器”通常意味着包括所有类型的数字处理设备。数字处理设备的例子包括数字信号处理器(DSP)、精简指令集计算机(RISC)、通用(CISC)处理器、微处理器、门阵列(例如，FPGA)、PLD、可重构计算机结构(RCF)、阵列处理器、安全微处理器、特定用途集成电路(ASIC)和/或其它数字处理设备。这种数字处理器可以被包含在单个统一的IC管芯上或跨越多个组件来分布。

[0082] 如本文所使用的，词语“事件”、“动作电位”、“脉冲”、“尖峰”、“尖峰群”和“脉冲群”通常意味着不局限于任何类型的脉冲信号，例如，信号的某些特征的快速变化，例如振幅、强度、相位或频率从基线值到更高或更低的值，接着快速返回基线值，并可以指任何单个尖峰、尖峰群、电子脉冲、电压脉冲、电流脉冲、脉冲和/或脉冲群的软件表示、脉冲延迟或时序的软件表示，和与脉冲传输系统或机制相关联的任何脉冲或脉冲类型。如本文所使用的，词语“spnet”可以包括尖峰网络。

[0083] 本文公开了本技术的某些实施方式的详细说明，包括系统、装置和方法。尽管本技术的某些方面在并行仿真引擎架构的上下文中可以很好地被理解、以软件和硬件来执行(这能够有效地仿真大规模神经元系统)，但该技术的实施方式可以用于实现指令集-基本网络描述(END)格式，该格式被优化以用于以独立于硬件的形式来有效表示神经元系统。

[0084] 例如，可以以神经形态计算机系统的硬件和/或软件实施方式来部署一些技术。在一种这样的实施方式中，图像处理系统可以包括嵌入特定应用集成电路(ASIC)的处理器，其可以适用于或配置用于嵌入式应用中，例如假肢设备。

[0085] 除此之外，本公开还提供计算机化的装置和方法，用于由尖峰神经元网络通过执行基于事件的更新来促进学习。在一个或多个实现中，可以使用用于神经形态系统的基本网络描述(END)来配置网络。在一些实施方式中，END描述可以包括被配置为执行基于事件的功能的多个事件类型。

[0086] 在一些实施方式中，突触前事件可以被定义，例如以便修改输入连接(例如，基于连接的突触前活动，使用神经元的突触前可塑性规则来调整突触前权重)。在一个或多个实现中，可以基于经由连接传送的突触前输入(例如，突触前尖峰)来配置突触前事件。

[0087] 在一些实施方式中，树突状(或内部)事件可以被定义，例如以便修改神经元的输入连接。内部事件可以由外部信号触发和/或由神经元内部触发。在一个或多个实现中，树突状事件可以由定时器触发、由与可以存储神经元尖峰历史(突触前和/或突触后)的缓冲器相关联的溢出指示器触发；由神经元内部地生成，或经由例如强化连接来外部地提供给神经元。在一个或多个实现中，树突状事件可以被配置为不会引起由神经元进行的突触后响应，而是连接可塑性适应。

[0088] 在一些实施方式中，轴突(或传送)事件可以被定义，例如以便引起向合适的目的地分配神经元的突触后响应。在一个或多个实现中，轴突事件可以由神经元尖峰响应过程(例如，超阈值条件)触发。在一些实施方式中，树突状事件和轴突事件可以作为单个复合事件而彼此顺序地实现。

[0089] 在一些实施方式中，内部事件的范围可以包括各个神经元，但不延伸到网络的其它神经元。相反地，外部事件的范围可以例如经由突触后尖峰传送延伸到网络的其它神经元。

[0090] 在一些实施方式中，定制事件可以被定义，例如以便实现END的定制事件规则。

[0091] 在一些实施方式中，可以定义两个或多个定制事件。在一个或多个实现中，两个或多个定制事件可以包括特定的名字(例如，事件标签)。在一些实施方式中，连接(例如，突触)可以被配置为基于标签来订阅一个或多个这种定制事件，以便合适地处理定制事件。

[0092] 在一个或多个实现中，可以使用树突状事件，以便促进由网络进行的强化学习。举例而言，在接收到强化输入(树突状尖峰)时，神经元的输入突触的可塑性可以被调整而不会将突触后响应传送给突触后目标。随后，神经元状态达到了超阈值条件，轴突事件可以用于向期望的目的地传送神经元突触后响应。

[0093] 在一个或多个实现中，树突状事件和轴突事件可以组合以获得复合事件，例如，举例来说，所使用的突触后尖峰，其例如用在初级神经态描述(END)框架V1、V2和V3中。

[0094] 现在参考图1所描绘的例子，显示并详细描述了神经仿真器开发环境100的一个配置。在该例子中，网络设计框、连接扩展框、监控和可视化框、引擎特定初始化框和引擎框可以包括软件工具，而带标签的库、高级描述框、调试表框、低阶描述框和引擎专用数据格式框可以实现作为特定格式的数据结构，这些特定格式的数据结构将在本文中更详细地描述。

[0095] 图1的神经仿真器开发环境允许用户定义任意的神经系统模型，并在任意的计算机平台(引擎)上执行该模型。神经仿真器开发100可以包括经由诶配置为某些格式的数据结构来彼此交互的数个软件工具(图1中的透明框)和可以体现在单个计算机、计算机群、GPU或专用硬件中的计算引擎104。在一些实施方式中，计算引擎104可以是计算机化控制/传感输入处理装置的一部分，并经由图1中的一对箭头102与该装置的其余部分交换数据。举例而言，计算引擎104经由输入102从“真实世界”接收传感信息，并经由输出103向控制装置的任何合适的执行器(未示出)发送电机(motor)命令，使得经由驱动器的集合对传感输入进行控制/处理响应。

[0096] 用户可以使用GUI网络设计工具(例如类似于Microsoft Visual Studio™的工具)来指定神经仿真器100的期望网络布局。在一些实施方式中，神经仿真器采用专用的库，该库被配置为执行各种专用函数。一些特定的函数库模块可以例如简单描述为“视网膜+丘脑+具有1M神经元的V1”。在一些实施方式中，可以更详细地描述库模块，(酌情)提供各种默认参数的初始化，所述参数定义了可塑性、神经元动态、皮质微电路和/或与默认参数相关联的其它信息中的一个或多个。GUI网络设计工具以“高级描述”格式保存了神经仿真器100的网络布局。在一些实施方式中，GUI网络设计工具被配置为修改图1的库106。

[0097] 使用库106来将网络布局的高级描述编译成低阶描述(基本网络描述-END)108。例如，高级描述可以包括皮质区域V1和V2(未示出)的描述，并可以根据存储在库106中的适当连接性规则来要求将它们进行连接。编译器分配神经元、建立神经元之间的连接，并以类似于汇编语言的低阶描述来保存网络布局100。在一些实施方式中，编译器可以提供用于监控的合适表格，和在调试过程中的可视化工具。

[0098] 基本网络描述(END)表示用作仿真器工具和图2所示的硬件平台实施方式之间的中间瓶颈(即，链接)。END表示提供了将开发环境从底层硬件隔离出来的抽象层。END的一个目的是从硬件模拟工具隔离出用于开发神经网络模型的步骤，如图2所示。END方法可以操作为将神经模型(例如，图1的模型)的实现分成两个步骤。在第一步，神经科学家使用高级描述语言和END表示来创建各种复杂度的神经模型。在第二步，开发者(程序员和硬件工程

师)修改和调整底层实现框,以适应和优化用于具体硬件/软件平台的模型操作。在这种架构中,END格式执行与LLVM(低阶虚拟机)或Java字节代码相同的服务;然而,END格式可以被优化用于神经元系统的并行表示和执行。

[0099] 模型的低阶描述被转换成适于向计算引擎104(如图1所示)上载的引擎特定二进制形式。计算引擎能以相同的格式保存其当前状态,用于后续重新上载。在系统的调试和计算结果的可视化以及监控过程中,二进制数据由图1的监控和可视化模块110使用。如上所述,计算引擎经由传感器和执行器(由驱动器连接到硬件)与真实世界互动,如由图1中的箭头102所描绘的。

[0100] 网络的基本网络描述(END)可以包括最低阶的独立于平台的模型描述。在一个实施方式中,这种描述被配置为类似于通常用在计算机编程技术中的汇编语言描述。然而,尽管大部分现有的计算机编译语言实现可能是依赖于处理器的,但END描述是跨硬件(hardware-agnostic)的。

[0101] END描述可以操作作为神经网络的高级描述与平台特定实现之间的独立于平台的链接,如图2所示。在图2中,方框210(神经仿真器1-3)指示各种网络开发工具(例如,神经元、GENESIS、NEST),而方框220(硬件平台1-3)指示不同的硬件实现(例如,CPU、多处理计算机、工作站、台式机、服务器、大型计算机、ASIC、FPGA和/或其它硬件实现),其用于执行相应的神经仿真器模型。

[0102] 在END实现的一个实施方式中,输入神经仿真器模型数据可以以XML格式(或任何其它方便的结构化数据格式)或以相关数据库通常格式来提供,目的在于提供足以指定神经仿真模型的一个或多个方面的输入数据的最小集合。神经仿真模块的给定方面可以包括以下各项中的一项或多项:神经元、树突状树、突触、神经元和突触类、可塑性规则、神经元动态和/或其它方面。该输入数据的集合经由以上项目之间的多个关系而被配置。该输入数据的集合以多种方式被配置:(i)多个文件的集合,其中的各个文件描述单个数据结构(例如,神经元);(ii)单个文件(其可以被压缩);或(iii)分层目录/文件夹/文件结构;或其组合。

[0103] 在一些实施方式中,网络仿真模型的基本(原子)计算单元是神经元,其被称为“单元”。在一些实施方式中,单元可以包括神经元区室(compartment),在此处,单元可以由接合点链接以形成树突状树,该树突状树形成神经元。在这些例子中,突触包括一个单元到另一个单元的连接,从而使得能够经由连接图来描述单元(节点)互连。这种图不必包括通过来自体细胞的突触来连接到树的树。

[0104] 为了获得可操作的网络描述,各个单元(例如神经元,区室)和各个突触可能服从管理其动态的规则的集合。在一些实施方式中,这些规则的一些包括基于时钟的规则,其适用于神经元单元和接合点,而其它规则可以是基于事件的,并适用于突触。

[0105] 举个例子,各个神经元单元可以服从描述了针对相应单元的尖峰产生机制的规则集合,该集合包括:(i)用于发射尖峰的条件;和(ii)变型的集合,其在发射尖峰后适用于单位动态状态变量。类似地,各个突触可以服从尖峰规则,该尖峰规则确定当突触前单元发射时对突触执行的动作的集合,以及当突触后单元发射时对突触执行的动作的集合。

[0106] 在一个实施方式中,END格式可以用于产生C代码,该C代码实现计算引擎(例如,图1中的引擎104)。在这个实施方式(其在本文中称为END1.0)中,规则的描述可以包括代码

串，其可以插入C代码模板，以提供具有任意可扩展功能的神经元模型，如由下面的例子所示出的。

[0107] END 1.0可以实现对象继承结构，其可以包括具有可能的子类的对象类单元类、接合点类、突触类和事件规则。这些类的各个类可能具有实例：即，单元、接合、突触和规则。

[0108] END 1.0可以被配置为将数据(单元、接合点、突触)从方法(更新和尖峰规则)中分离出来，从而使得计算引擎(例如，图1中的引擎102的链接器112)，以实现数据与方法(数据↔方法)的互连。在计算引擎的一些实施方式中，计算操作可以由数据来分组(例如，对于个体突触，一些或所有动作(规则)针对该突触而被执行)。在计算引擎的其它通用实现中(例如，与GPU硬件一起使用)，计算操作可以由方法来分组(例如，对于个体更新规则，针对服从更新规则的一些或所有突触的未解决请求被执行)。END可以被配置为与上述可操作分组配置中的任何一个良好地等同操作。

[0109] 当执行复杂的真实生活系统的大规模模型(例如，哺乳动物的视觉系统)时，由END格式描述的一些数据结构可以占用网络模型资源(存储器或CPU，或二者)的大多数(在一些实施方式中，高达99％)。这些通常称为“规范结构”的数据结构的实现，极大地得益于专用硬件的使用，例如，被优化以仿真这种规范结构的ASIC或FGPA。类似地，在一些实施方式中，在一些规则和方法占用大部分CPU处理资源(例如，占用最多的时间来执行)的情况下；专用硬件加速器的开发可以在规范方法的处理中提供实际的提高。不同的硬件实现可以硬连线(hard-wire)不同的方法，从而导致硬件平台的多样性。

[0110] 利用END描述取得的目标之一可以是提供足以用于描述任何复杂度的神经元模型的最小指令集。此处，以下注释用于描述END格式：类或类型定义类型被包括在三角括号中<...>；类(或类型)内的字段可以关于类定义而缩进，如下面的定义1例子中所示。

[0111] 在以上定义中，声明<unit of(unit_class)>指示具有如下字段"unit_id"和"Initialization"的类"unit_class"的实例的定义：

[0112] <unit of(unit class)>

[0113] unit id

[0114] Initialization

[0115] 该类定义神经元或间，但原则上，其可以是任何的神经单元，其可以在预定模型执行时间步长(例如，1ms)处由模型执行。该单元可以是对象unit_id的实例，其指定在各个模型时间步长处执行的操作。unit_class的字段可以定义如下：

[0116] unit_id

[0117] 是唯一标签，其例如是标识各个单元的数字或字符串。unit_id可以是Perl或PHP或RegEx或MATLAB表达式，其规定有效ID的范围，且实际ID可以在模型构建期间由链接器分配。例如，'exc(1:1000)'或'excl:excl000'。

[0118] Initialization

[0119] 可以是包含类字段的初始值的结构，该初始值不同于类定义中的默认值。这些中的某些或所有可以被声明和初始化为具有unit_class中的默认值。

<unit_class of (unit_class)>

unit_class_id

execution_condition

[0120] update_rule

event_condition

after_event_rule

initialization

[0121] 提供了神经元模型类的一般定义，其规定神经元动态、分配神经元变量，定义尖峰处理规则和/或与神经元模型类相关联的其它信息。这个类<unit_class>是能够从unit_class的另一个对象导出的对象(如在面向对象的编程中)。<unit_class>对象的字段可以定义如下：

[0122] unit_class_id

[0123] 是对象类(字符串)的唯一名字，例如，'exc'或'p23soma'或'p4cmprtimnt3'。这些对象的名字可以由神经元模型使用作为矢量的名称来指代单元，例如，在后台处理中或在I/O处理中。

[0124] execution_condition

[0125] 该条件在各个模型执行时间步长处被评估以便确定是否执行这个类的单元，例如：'now%10==0';或'DA>0.1'。如果没有execution_condition(字段为空)，那么可以在各自的模型时间步长处执行(即，系统时钟步长循环)各单元。当类(子类)可能从基类(超类)中导出时，则所导出的类执行条件将重写基类执行条件。

[0126] update_rule

[0127] 定义了将由链接器转换成可执行代码的代码串。在一些实施方式中，代码串可以包括前向欧拉数值方法注释，例如，举例来说：

[0128] 'x(t+1)=x(t)+tau*(F(x(t))).'I+=g*(E-V);g+=tau*(-g)/5;

[0129] v+=tau*(2*(v+70)*(v+50)-u+I)/20;u+=tau*0.1*(2*(v+70)-u);'.

[0130] 在一些实施方式中，代码串指定映射x(t+1)='x(t)+tau*(f(x(t))'或C-可执行代码(功能或子例程)，其可以在各自的模型执行步长处执行。当类从基类导出时，则子类的对象update_rule可以先执行，随后是基类规则，从而允许由子类规则执行的某变量的更新。

[0131] event_condition

[0132] 定义检测尖峰所需的逻辑声明。例如，'v>30'。其可以在各自的时钟循环(模型时间步长)处执行。当类从基类导出时，则子类的event_condition替换基类的event_condition。

[0133] after_event_rule

[0134] 是当检测到尖峰时执行的代码。例如，'v=-65;u=u-8'。对于导出的子类，子类after_event_rule可以在基类的after_event_rule前执行。

[0135] initialization

[0136] 设置单元的一些或所有变量和参数的初始值。例如，'float v＝0;float u＝0;float g_AMPA＝0'。不同的单元类可以具有不同数量的突触传导类型和/或所分配的不同变量名称。由链接器解析初始串，以便在构建期间获得一些或所有变量的名称。导出的目标类可以添加新的变量。对于导出的子类，基类的初始化可以首先执行，随后执行该导出的子类的初始化，因为它可能会改变一些默认参数。初始化可以改变从基类继承的默认参数值。举例而言，初始串'float g_NMDA＝1;g_AMPA＝1'创建新的变量g_NMDA并重置现有变量g_AMPA的值。

[0137] 类似地，接合点和突触可以使用与上述相同的方法来描述。

[0138]
<junction of (junction_class)>
unit_idl
unit_id2
initialization

[0139] 提供了在单元对之间提供连接的神经元模型类的一般定义。字段junction_class是指父对象类的名称，例如，"gap_junction"或"dendritic_tree"。这个类字段可能如下：

[0140] unit_idl,unit_id2

[0141] 规定两个连接的神经元单元的ID。

[0142] initialization

[0143] 设置参数和变量的初始值。这个类junction_class可以用作定义子类<junction_class of(junction_class)>的基础，并可以声明如下：

[0144]
<junction_class of (junction_class)>
junction_class_id
execution _condition
unit_class_1
unit_class_2
update_rule
initialization

[0145] 其中，

[0146] junction_class_id

[0147] 是唯一的标识符，其规定该类能够从另一类导出。

[0148] execution_condition

[0149] 该条件在各个模型执行时间步长处被评估，以确定是否执行该类的单元，例如：'now%10＝＝0';或'DA>0.1'。如果没有execution_condition(该字段为空)，则各个单元可以在各个模型执行时间步长处执行。当类(子类)从基类(超类)中导出时，则所导出的类执行条件重写基类执行条件。

[0150] unit_class_1

[0151] unit_1的类标识符；

[0152] unit_class-2

[0153] unit_2的类标识符;如果省略的话,则假定接合点适用于一些或所有的单元类。

[0154] update_rule

[0155] 定义将由链接器转换成可执行代码的代码串。在一些实施方式中,代码串可以包括前向欧拉数值方法注释,例如,举例来说:

[0156] 'g_2tol*(V2-V1),g_1to2*(V1-V2)'

[0157] 在适用于电阻连接一些实施方式中,可以跳过_class部分,并仅将<junction>中的传导率g_2规定为1并且将g_1规定为2。当类从基类导出时,子类的对象update_rule可以首先执行,随后是基类规则,从而允许由子类规则执行的某些变量的更新。

[0158] Initialization

[0159] 设置变量的初始值(如果有的话),用于执行update_code。

[0160] 突触类可以声明如下:

<synapse of (synaptic class)>

pre

[0161] post

delay

initialization

[0162] 其中,

[0163] pre

[0164] post

[0165] 分别是突触前单元和突触后单元的标识符。

[0166] delay

[0167] 指定轴突传导延迟值(在仿真时间步长中)。

[0168] initialization

[0169] 设置初始参数和变量值。类似于接合点类,从基突触类导出的突触类可以声明如下:

[0170] <synaptic_class of(synaptic_class)>

[0171] synaptic_class_id

[0172] initialization

[0173] 其中,

[0174] synaptic_class_id

[0175] 是规定基类的唯一标签,其可以用作矢量标识符,以便指代后台处理(尽管顺序可以打乱)中的各个突触。该类可以从另一类中导出。

[0176] initialization

[0177] 设置各个突触的参数和变量的初始值,例如,'float w=5;float sd=0;float p=0;'。用于定义突触前事件规则(其访问突触变量和post_unit变量)的类可以声明如下:

[0178] <presynaptic_event of(event_rule)>

[0179] synaptic_class

[0180] post_unit_class

[0181] 其中，

[0182] synaptic_class

[0183] 指示该可塑性规则适用的突触类的名字。对象event_rule可以用于规定当突触前单元发射时执行的动作。synaptic_class的规则定义了用于突触前神经元(单元)的STDP规则的短期可塑性(STP)和长期增强(LTP)部分。

[0184] post_unit_class

[0185] 规定突触后单元的类，从而定义了event_rule的动作的域，进而提供STDP规则执行(例如，该类用于传送突触后电位(PSP)以及用于访问诸如last_spike的变量)可能需要的信息。该类post-synaptic_event可以声明如下：

[0186] <post-synaptic_event of(event_rule)>

[0187] synaptic_class

[0188] post_unit_class

[0189] 其中，

[0190] synaptic_class

[0191] 规定该可塑性规则适用的突触类的名字。对象event_rule可以用于定义当突触后单元(关于这些突触所引用的)发射时可以利用突触变量执行的动作的列表，从而实现STDP可塑性规则的LTP部分，其在突触后单元发射尖峰时可以执行。

[0192] post_unit_class

[0193] 规定突触后单元的类，从而定义用于该规则的动作域。在一个实施方式中，该规则可以被配置为访问突触变量。该event_rule类可以如下从基类导出：

[0194]
```
<event_rule of (event_rule)>
    event_rule_id
    rule
    initialization
```

[0195] 其中，

[0196] event_rule_id

[0197] 是规定规则类的唯一标签。通常，该规则类可以从另一对象类event_rule导出。

[0198] Rule

[0199] 是代码的字符串表示，其规定在发生事件时执行的动作。通过示例的方式，突触后电位(PSP)的传送可以规定为

[0200] “’g+=w’或’I+=w’或’g+=g*p’”。

[0201] 类似地，与presynaptic_event相关联的动作可以定义为：

[0202] ’p*=STP(now-last_active);w-=LTD(now-last_spike);’

[0203] 而与post-synaptic_event相关联的动作可以定义为：

[0204] ’w+=STDP(now-last_active)’

[0205] 其中，"last_active"是从突触被激活的先前时机经过的时间；

[0206] last_spike是突触后单元的上一个尖峰的时间，并且

[0207] 'now'是当前时间。

[0208] 此外,event_rule类可以对表进行规定,例如,STDP、LTP或其它生物激励的表。

[0209] initialization

[0210] 设置表的初始值,例如:'STDP={,..,};'或'STP={,..,};LTD={...}'。

[0211] <background_process>

[0212] execution_condition

[0213] 该条件在各个仿真时间步长处被评估,以确定是否运行该过程。例如,'now%10==0'或'DA>0.1'。如果缺省,那么该过程可以在各个时间步长处运行。unit_class或synaptic_class的局部变量的名称可以被访问。以下代码将运行在具有任意顺序(可能并行)的类成员内的环中。如果缺省,可以假定该过程在每次execution_condition为"真"时运行一次,并且可以使用单元或突触类名程来访问各个单元或突触。例如,单元类'exc'包括单元exc[i],其具有不必对应于他们被列出的顺序的、可能未知的顺序。

[0214] update_rule

[0215] 待执行的代码,例如'DA*=0.9'或在synaptic_class域中'w+=DA*sd;sd*=0.995',或不具有仅'exc[rand()].I=100'的域。

[0216] initialization

[0217] 全局变量的初始化,例如,'float DA=0'。

[0218] 可以规定仿真的时间步长和其它运行时参数。可以有若干全局变量,其可以被每个人访问,例如,"now"-当前时间。

[0219] 由参数区分的类可以被声明为单个类,并且参数值可以在实例化中被规定。如果仅存在参数的若干不同值(例如,2个、3个或其它某个数值),那么规定两个不同的类是合理的,每一个类在等式中都具有硬连线的参数。

[0220] END框架的外部接口描述了向神经元网络(例如,图1的网络100)提供外部传感输入以及将输出传送(例如经由图1的路径102)给外部机器人装置。END外部接口可以包括两个主要的逻辑框:传感输入框,以及输出和监控框,如下所详细描述的。

[0221] 这个框定义了外部输入到网络模型的各个单元的连接。举个例子,用于N-通道(编号1-N)尖峰视网膜输入的传感类可以声明如下:

[0222] <unit_class>

[0223] unit_class_id='retina'

[0224] after_event_rule //可以为空

[0225] 以上类声明通知输入驱动器和计算引擎来自视网膜的尖峰将去往何处。输入的结构可以使用以下N个空单元的声明来定义:

[0226] <unit>

[0227] unit_id='retina(1:N)'

[0228] 在一些实施方式中,可能不存在需要在各个网络时间步长处执行的更新规则。因此,计算引擎可能不会在这些空单元上耗费计算资源。然而,当来自输入信道的被称为'retina'的尖峰到达时,输入驱动器可以将单元索引放入尖峰队列,如同它实际上发射尖峰一样(如果不为空,这可以由引擎触发after_event_rule执行)。从该单元到其它单元的突触可以告知网络怎样处理尖峰。在对LGN输入视网膜输入的情况下,视网膜单元具有某些

LGN细胞上的1-3个突触。

[0229] 如果输入信道馈送连续信号，那么信号可以在给定的时间间隔(例如，每毫秒)更新各个单元中的变量"I"。在该情况下，可以指定更新规则和event_condition。

[0230] 输出和监控框为网络模式提供输出接口。在适用于来自模型的发动机输出的一个实施方式中，输出框规定网络单元之间的连接和外部发动机接口或某些其它执行器。在一些实施方式中，发动机接口可以包括肌肉接口。在一些实施方式中，发动机接口可以包括被配置为控制外部机器人装置的发动机接口。单元'neck muscles'包括用于与例如颈部肌肉交互的N-信道发动机输出对象，颈部肌肉可以使用如下的END框架来声明：

[0231] <unit class>

[0232] asunit_class_id＝'neck muscles'

[0233] initialization＝'float I＝0;'

[0234] 以上声明通知输出驱动器哪个神经元(单元)用于监控当前I值中的变化。相应的N个空单元对象可以随后创建如下：

[0235] <unit>

[0236] unit_id＝'neck muscles'

[0237] 在模型的执行期间，计算引擎确保至少一些发动机神经元(单元)颈部肌肉(neck_muscles)具有向这些神经元突出的非-零(例如，正)突触，从而在发动机神经元发射时，单元对象内的变量I可以被设置为正值。因此，输出驱动器在各个模型时间步长处监控变量I，并将其重置为I＝0(如果有必要的话)。由于发动机输出接口不需要(在各个模型执行时间步长处)执行更新规则，则计算引擎在维持'neck_muscles'单元上花费最少的资源。

[0238] 在适用于监控神经网络执行的一些实施方式中，输出框规定网络单元和外部监控接口之间的连接。

[0239] 如上所描述的，神经元模型的某些要素得益于可以由特定的硬件框(硬件加速器)执行的计算。通过示例的方式，考虑单元unit_class的方法update_rule消耗引擎计算资源的大部分(例如99％)：

```
<unit class>
  unit_class_id='simple_model'
  update_rule = 'v+=(0.04*v+5).*v+140-u+I; u+= a*(b*v-u);'
```

[0240]

```
  event_condition = 'v>30'
  afte _event_rule = 'v=-65;u+=d'
  initialization='float v=-65; float u=0; float a=0.1;
                  float b=0.5; float d=8;'
```

[0241] 假定update_rule的实现从单元到单元和/或从一个模型运行至另一模型都不变，那么与update_rule相关联的计算操作可以由专用的硬件加速器(其可以例如实现在ASIC、FPGA或专用硅中)更有效地执行。在END框架内，'simple_model'类可以用于指示编译器将例如与上述列出的update_rule相关联的代码的执行指示给对应于硬件加速器接口的适当位置。为了创建这种映射，simple_model类的实例可以如下地例示：

[0242] <unit of(simple_model)>

[0243] unit_id＝509

[0244] initialization'a＝0.01;d＝2'

[0245] 这种硬件加速器(simple_model对象)可以由END使用作为用于构建多个复杂对象的构造框。举例而言，例如，具有一百万个(1M)simple_model单元(神经元)以及例如具有额外慢神经元变量的一千个(1K)神经元的神经元模型，'w'可以使用类继承机制来声明如下：

```
<unit_class of (simple_model)>
  unit_class_id='complex_model'
```

[0246]

```
  update_rule = 'w+=0.001*(0.3*v-w); I-=w'
  after_even_rule = 'w+=1'
  initialization='float w=0'
```

[0247] 附接到专用硬件的计算引擎(例如ARM芯片)中的处理器可以处理以上类型的1K个单元，以评估变量'w'的动态，并将其并入变量I的值。随后，硬件加速器(模拟的或数字的)硬件可以执行simple_model的1M+1K个实例，而不用意识到一些实例对应于更复杂的模型。专用硬件可以包括神经元模型、突触动态最通用的实施方式。用户可以自由组合和匹配这些规范能力，或者无论需要什么额外的能力都将该额外的能力加入它们。

[0248] Spnet可以包括N＝Ne+Ni＝1000个神经元，其中，Ne＝800个兴奋性神经元和Ni＝200个抑制性神经元。各神经元可以包括每神经元M＝100个突触。一些或所有兴奋性突触可能是可塑性的(STDP)，具有范围在1ms与D＝20ms之间的随机延迟。抑制性突触->兴奋性突触可能是非可塑性的，具有D＝1ms的延迟。可能不具有抑制性突触->抑制性突触。模型的下层END描述可以表示如下。第一Ne单元可以由Ne兴奋性单元填充：

[0249] <unit of(exc)>

[0250] unit_id＝1:800

[0251] 接下来，Ni抑制性单元可以是如下填充的类的记录：

[0252] <unit of(inh)>

[0253] unit id＝801:1000

[0254] Spnet类可以随后如在下面的列表1中示出的那样被声明：

<unit class>

unit_class_id='QIF2'

update_rule = 'v+=0.5*((0.04*v+5).*v+140-u+I);

v+=0.5*((0.04*v+5).*v+140-u+I);'

event_condition = 'v>30'

after_event_rule = 'v=-65;'

initialization = 'float v=-65; float u=0.2*v; float 1=0'

<unit_class of (QIF2)>

unit _class _id='exc'

update_rule = 'u+=0.02.*(0.2*v-u);'

after _event _rule = 'u+=8; last spike=now'

initialization = 'int last _spike=-1000;'

[0255]

<unit _class of (QIF2)>

unit _class _id='inh'

update_rule = 'u+=0.1.*(0.2*v-u);'

after _event _rule = 'u+=2;'

Ne*M records of the class (exc->exc)

<synapse of (GLU)>

pre = i // i=1:Ne

post = j // 1 和 N 之间的随机数

delay // 1 和 D 之间的随机数

Ni*M records of the class

<synapse of (GABA)>

pre = i // i=Ne+1:N

post = j // 1 和 Ne 之间的随机数

<synaptic _class>
synaptic _class _id = 'GLU'
initialization = 'float w=6; float sd=0; int last _active-
1000;'
delay=1 // 默认延迟 (如果没有在实例中规定的话)
<synaptic _class>
synaptic_class _id='GABA'
delay=1 // 对于所有突触来说是相同的
<presynaptic _event of (EPSP _plus _LTD)>
synaptic_class = 'GLU'
post _unit _class = 'QIF2'// 即, exc 和 inh
<presynaptic _event of (Inhibition)>
synaptic _class = 'GABA'

[0256]

post _unit _class = 'QIF2'// 即, exc 和 inh
<post-synaptic _event of (LTP)>
synaptic_class = 'GLU'
post _unit _class='exc'
<event _rule>
event _rule _id = 'EPSP _plus _LTD'
rule = 'I+=w; sd-=LTD(now-last _spike); last _event=now'
initialization = 'LTD = {array};'
//从 LTP A=1.2 至零的指数衰减，其中 LTD _tau=20ms
<event rule>
event _rule _id = 'LTP'
rule = 'sd+=LTP(now-last _active);'
initialization = 'LTP = {array};'
<event _rule>
event _rule _id = 'Inhibition'

```
rule = 'I-=5;'
initialization = '    '
<background process>
id = 'Weight update';
execution _condition = 'now%1000==0'
```

[0257]
```
domain = 'GLU'
update rule = 'w += 0.01+sd; if (w<O) w=0; if (w>10) w=10;
sd *= 0.9'
<background process>
id = 'randominput'
update _rule = 'exc[int(floor(rand()/RAND MAX*N))].v=30;'
```

[0258] 列表1

[0259] 模型的低级描述(在上述列表1中这样示出)可以包括对唯一定义网络架构来说必要的信息。示出在列表1中的描述可能除此之外不适用于执行模型仿真,因为它不能为用于各个单元的一个或多个突触连接、突触前连接、突触后目标提供足够等级的细节和/或用于执行模型仿真的其它信息。链接器(例如,图1中的链接器108)使用低级END描述并且(i)填充单元之间的一些或所有链接;(ii)保存二进制或一些其它低级(机器)格式的代码和数据,以便促进由计算引擎在模型仿真期间的数据和代码加载。在仿真执行(运行时间)期间,引擎可以创建相同(或不同)格式的保存点(也就是保存引擎执行状态,其包括例如寄存器和存储内容、程序计数器,和/或与引擎执行状态相关联的信息),以使得从任何保存点都可以快速重启模型执行。

[0260] 在一个实施方式中,计算引擎可以包括单个处理器计算机。引擎在预定的时间步长处执行数个计算循环(通过网络的步长)。在一些实施方式中,时间步长可以被设置为1毫秒。

[0261] 在一些实施方式中,计算引擎可以在多核处理平台、单个/多核处理器的阵列、具有一个或多个嵌入式处理器内核的FPGA或可编程逻辑结构上执行。

[0262] 在低级END中描述的各个<_class>实例可以对应于分别的环执行环。各个执行环/循环内的计算可以以针对多核实施方式优化的顺序来并行执行。一些循环可以与其它循环并行执行。given_code可以是预编译的,并被包括在引擎中的适当位置。

[0263] 为了在模型仿真期间实现执行效率,用于实现计算引擎的神经形态硬件可以具有一个或多个以下特征:(i)对神经元动态和基本突触事件的快速高效专用处理,例如突触释放;(ii)用于执行计算后台处理的通用处理器(例如,ARM内核),所述后台处理例如慢突触更新、翻转、重新连线、短期可塑性和/或其它后台处理;和/或其它特征。这种配置实现了针对多数突触的基本突触处理(不太可能需要由用户进行频繁修改)的快速执行,同时允许实现针对少部分突触的专有精品处理(proprietary boutique processing)。

[0264] "最小指令集"实施方式的一个目标是提供低级描述格式,其可以包括(i)单元(神

经元)定义,其声明存储器分配,但不包含动作;(ii)接合点,其耦合单元,但不分配其自身的存储器;和(iii)规则,其将动作与单元或接合点相链接。在一些实施方式中,动作可以是时钟驱动的(也就是说,针对合适的单元,在神经元模式仿真执行的各个时间步长处执行)。在一些实施方式中,动作可以是事件驱动的,(也就是说,动作可以由单元例如经由针对该单元类声明的event_condition来触发,event_condition通知仿真器关于当单元发射尖峰时执行的动作。因此,这种事件(尖峰)触发基于事件的规则的执行,该规则可以适用于其它单元或接合点。

[0265] 在END框架内,可以将突触声明为单元,所述单元包括被配置为存储与突触功能相关联的各种变量的存储器,所述变量例如突触权重、延迟、目标目的地和/或与突触功能相关联的其它信息。突触可以被视为准备释放传递素(transmitter)的突触前机构。正如单元更新一样,突触更新规则(例如,保持'w+=sd;sd*=0.9')可以是基于时钟的或基于事件的。突触动作(例如,神经传递素的释放)可以由位于与突触前神经元相对应的单元处的尖峰事件触发。用于描述该释放的规则可以执行STDP的压缩部分和任何短期可塑性过程。STDP的LTP部分可以由单独的其它规则来实行,该规则可以由与突触后神经元相对应的单元触发。接合点规定突触和突触后单元之间的连接。

[0266] END的最小指令集的例子能够构造简单的基于事件的计算机,该计算机具有足够的描述能力来定义任意的神经网络。

[0267] END最小指令集的例子的类和变量的排列和结构描述如下,它们与上述的END 1.0格式类似。

[0268]
<rule of rule class>
subject id
<unit of unit _class>
unit _id
initialization
<unit _class (of unit _class)>
unit _class _id
initialization
<junction of junction _class>
junction _id
unit _1
unit _2
<junction _class (of junction _class)>
junction _class _id
unit _class 1
unit _class 2
<rule of rule _class>
subject _id

[0269] 单元或接合点的ID服从该规则。

[0270] trigger_id

[0271] 该单元的ID,其触发针对该主题的规则(用于基于事件的规则)。如果省略,则这个规则可能是基于时钟的。

[0272] Delay

[0273] 延迟,这个规则必须利用该延迟来执行。如果省略,则没有延迟。

[0274] <rule_class(of rule_class)>

[0275] rule_class_id

[0276] execution_condition

[0277] 例如,'now%10-0'。如果省略,则该规则可以在各个时间步长处被执行。

[0278] subject_class

[0279] 该规则适用的类。要注意的是,所述类可能是单元或接合点。

[0280] code

[0281] event_condition

[0282] initialization

[0283] 示例2:spnet

[0284]

```
//首先，为神经元分配存储器
<unit class>
   unit _class _id = 'neuron'
   initialization = 'float v=-65; float u=0; float I=0; int
last _spike=-1000;'
<unit _class of neuron>
   unit _class _id = 'exc'
<unit _class of neuron>
   unit _class _id = 'inh'
<unit of exc>
   unit _id = 1:800
<unit of inh>
   unit _id = 1:200
//现在，向神经元（基于时钟的规则）分配动态
<rule_class>
   rule _class _id = 'QIF'
   subject_class = 'neuron'
   code = 'v+=0.5*((0.04*v+5).*v+140-u+I);
v+=0.5*((0.04*v+5).*v+140-u+I);'
event _condition = 'v>30'
<rule_class of QIF>
rule _class _id = 'RS'
```

[0285]

```
subject_class = 'exc'
code = 'u+=0.02.*(0.2*v-u);'
<rule of RS>
subject_id = exc(1:800)
<rule_class of QIF>
rule _class _id = 'FS'
subject_class = 'inh'
code = 'u+=0.1.*(0.2*v-u);'
<rule of FS>
subject _id=inh(1:200)

//规定尖峰后的静默（基于事件的规则）
<rule_class>
  rule _class _id = 'reset'
  code = 'v=-65; last spike=now;'
<rule _class of reset>
rule _class _id = 'RS _reset'
subject_class = 'exc'
code = 'u+=8;'
<rule of RS _reset>
subject_id = exc(1:800)
trigger _id = exc(1:800)
<rule_class of reset>
rule _class _id = 'FS _reset'
subject_class = 'inh'
<rule of FS _reset>
subject _id = inh(1:200)
trigger _id = inh(1:200)

//规定突触
```

```
//抑制性突触是不可塑的。
  <rule_class>
     rule _class _id = 'inhibition'
     subject_class = 'neuron'
     rule = 'I-=6'
//下面表格的 200*100 个记录，每个都规定了抑制性突触
<rule of inhibition>
   subject_id = {an exc or inh unit}
   trigger _id = {an index of inh unit}

//兴奋性突触是可塑性的，因此存储它们的权重
<unit _class>
unit _class _id = 'synapse'
initialization = 'float w=5; float sd=0; int last active-
1000;'
```

[0286]

```
<unit of synapse>
unit _id = 1:80000
//对于每个突触，规定其目标。这可以经由接合点来进行
<junction _class>
junction _class _id = 'GLU'
unit _class _1 = 'synapse'
unit _class _2 = 'neuron'

//分配突触，即，创建格式的 80000 个记录
<junction of GLU>
   junction _id = 1:80000
   unit _1 = 1:80000
   unit _2 = {random neuron index}
//规定哪个突触前神经元触发哪个突触
<rule_class>
```

[0287]

rule _class _id = 'EPSP _plus _LTD'

subject_class = 'GLU'

code = 'I+=w;sd-=LTD(now-last spike);last active=now;'

initialization = 'LTD={array}'

<rule of EPSP _plus _LTD>

subject_id = 1:80000

trigger _id = exc(rand(800))

delay = rand(20)

//规定当后单元发射时的 LTP

<rule_class>

rule _class _id = 'LTP'

subject_class = 'synapse'

rule = 'sd += LTP(now-last _active)'

initialization = 'LTP={array}'

<rule of LTP>

subject _id = 1:80000

trigger _id = {对应突触后神经元，尽管这在原则上可以是任何其它的东西}

<rule of LTP>

subject id = 1:80000

trigger id = {对应于突触后神经元，尽管这在原则上可以是任何其它的东西}

//执行 sd-更新（维护）

<rule class>

rule _class _id = 'synaptic maintenance'

subject _class = 'synapse'

execution _condition = 'now%1000==0'

code = 'w += 0.01+sd; if (w<0) w=0; if (w>10) w=10; sd *=0 9

<rule of synaptic maintenance>

subject _id = 1:80000

//最后，随机丘脑输入，该输入每毫秒的时间步长发射神经元

<rule _class>

[0288] rule _class _id = ' input '

code = 'fire({random neuron});'

<rule of input > //没有参数；仅需要其来例示输入

[0289] 列表2

[0290] END 2.0格式与上述的1.0格式相比，可以包括下列特征。

[0291] • 无继承(没有对象类)；

[0292] • 无后台处理(替代地，使用全局变量)；

[0293] • 无单独的突触前/突触后规则(一些或全部规则可能是突触类型的一部分)；

[0294] • 各规则可以包括其它规则和代码串，以及执行条件。

[0295] <rule>

[0296] name

[0297] exec_condition

[0298] 该条件可以在各个步长处被评估以确定是否执行该规则。这可以是字符串，例如，'now%10-0'或'DA>0.1'，或对规则名称的引用(如果条件需要一些全局表，则这是有用的)。如果缺省，那么该规则适用各个执行时间步长。该条件可以访问在下列代码中定义的任何变量。

[0299] Code

[0300] 规定当规则发生时该做什么的代码串或其它规则名称。例如，'I+=w'或`v+=tau*(2*(v+70)*(v+50)-u+I)/20;u+=tau*0.1*(2*(v+70)-u);'。另外，规则的输出可能是可以用在尖峰条件中的逻辑(真/假)陈述。这里可以提供多个规则名称;它们可以按照所提供的顺序执行。

[0301] init

[0302] 声明和设置了该规则中使用的全局变量的值。例如，'STDP={...};DA=0'或'STP=1,...,1;LTD={...}'。任何实例变量都可以在unit_type和synaptic_type中定义;链接器检查在不同unit_type和synaptic_type中定义的各个数据类型是否与规则一致。

[0303] <unit type>

[0304] name

[0305] update_rule

[0306] 在各个时间步长处执行的代码串或规则名称。这里可以提供多个规则;它们可以按照规定的顺序来执行。

[0307] event_condition

[0308] 检测尖峰所需的逻辑声明或规则名称，例如，'v>30'，其在各个时间步长处执行。

[0309] event_rule

[0310] 当检测到尖峰时可能执行的代码或规则名称。例如，'v=-65;u=u-8'。

[0311] Init

[0312] 声明并设置了用在单元(即,实例变量)中的各个变量和参数的初始值。链接器(编译器)检查这些变量与使用相同规则的各单元类型具有一致的数据类型,例如,'analog v=0;analog g_AMPA=0;'

[0313] <unit of unit type>

[0314] unit_id

[0315] init

[0316] 设置与类型的定义中的默认值不同的参数和变量值。各个值或这些值必须被声明,并利用unit_type的定义中的默认值来初始化。

[0317] <junction_type>

[0318] name

[0319] update_rule

[0320] init

[0321] <junction of junction_type>

[0322] unit_1

[0323] unit_2

[0324] init

[0325] <synaptic_type>

[0326] type

[0327] presynaptic_event_rule

[0328] 当突触前神经元发射时可能被触发的代码或规则名称。这照顾了LTP和PSP。

[0329] post-synaptic_event_rule

[0330] 由突触后单元的发射所触发的代码或规则名称。这照顾了STDP的LTD部分。

[0331] update_rule

[0332] 在各时间步长处可能执行的代码串或规则名称(例如,可以具有execution_condition并可能很少执行)。这里可以提供多个规则;它们可以按照规定的顺序来执行。这对于突触维护来说是需要的,以代替后台处理。

[0333] <synapse of synaptic type>

[0334] pre

[0335] post

[0336] delay

[0337] init

[0338] <global_variable>

[0339] update_rule

[0340] 规则名称,其初始化全局变量并执行对它们进行更新的代码。代码可以访问单元、接合点或突触的特定实例。在最简单的情况下,规则可以仅是基于实例的值而对全局变量的值的分配。

[0341] 需要注意的是,实例变量可以在<rules>中使用,但它们可以在<unit_type>、<junction_type>和<synaptic_type>中定义。假定实例变量的各个声明可以与一些规则或所有规则一致。这可能存在两个问题:

[0342] 变量可能在规则中使用但没有在unit_type或junction_type或synaptic_type中定义的情况可以如下地处理：

[0343] • 链接器可能产生错误

[0344] • 链接器使用变量的默认类型，例如，"analog"

[0345] • 可以指示链接器来查看规则的其它用法，并且如果变量在某处定义，则延伸其定义。

[0346] 在一些实施方式中，变量可以在使用相同规则的两个或多个unit_type中被不同地定义。再一次地，对此可能有多种可能的解决方案：

[0347] • 链接器产生错误

[0348] • 链接器将定义之一转换为另一个。例如，可能存在部分排序的定义集，例如，int8<intl 6<int32<int64<analog，从而两个定义可以转换成共同的一个

[0349] • 链接器将规则分为两个规则：rule_a和rule_b，这两个规则对变量的不同类型起作用。

[0350] 示例2：全局变量

[0351] <rule>

[0352] name＝'DA_update'

[0353] code＝'DA＝exc[3].v+inh[1].u'

[0354] init＝'analog DA＝0'

[0355] <global_variable>

[0356] update_rule＝'DA_update'

[0357] 示例3：spnet

[0358] 标准spnet网络具有N＝1000个神经元；其中，Ne＝800个兴奋性神经元以及Ni＝200个抑制性神经元，其中，每个神经元具有M＝100个突触。一些或所有的兴奋性突触可能是可塑性的(STDP)，其中，随机延迟在1与D＝20ms之间。在延迟D＝1ms的情况下，抑制性→兴奋性突触可能是非可塑性的。可能没有抑制性→抑制性突触。该模型的低级描述如下。

[0359] <rule>

[0360] name＝'QIF2'//仅v等式被定义

[0361] code＝'v+＝(0.04*v+5).*v+140+I;I＝4*rand()/MAX RAND'

[0362] <rule>

[0363] name＝'spike'

[0364] code＝'v>30'

[0365] <unit_type>

[0366] name＝'exc'

[0367] update_rule＝'u+＝0.02*(0.2*v-u);I-＝u;'

[0368] update_rule＝'QIF2'

[0369] event_condition＝'spike'

[0370] after_event_rule＝'v＝-65;u+＝8;last_spike＝now'

[0371] init＝'analog v＝-70,u＝-14,I＝0;int last_spike＝-1000'

[0372] <unit_type>

[0373] name＝'inh'//抑制性神经元

[0374] update_rule＝'u+＝0.1.*(0.2*v-u);I-＝u;'

[0375] update_rule＝'QIF2'

[0376] event_condition＝'spike'

[0377] after_event_rule＝'v＝-65;u+＝2;'

[0378] init＝'analog v＝-70,u＝-14,I＝0;int last_spike＝-1000'

[0379] //类的Ne个记录

[0380] <unit of exc>

[0381] unit_id＝1:800//i＝1:Ne

[0382] //类的Ni个记录

[0383] <unit of inh>

[0384] unit_id＝1:200//i＝1:Ni

[0385] <rule>

[0386] name＝'EPSP_plus_LTD'

[0387] code＝'I+＝w;sd-＝LTD(now-last_spike);last_active＝now'

[0388] init＝'LTD＝{array};'

[0389] <rule>

[0390] name＝'LTP'

[0391] code＝'sd+＝LTP(now-last_active);'

[0392] init＝'LTP＝{array};'

[0393] <rule>

[0394] name＝'Synaptic_Maintenance'

[0395] execution_condition＝'now%1000-0'

[0396] update_rule＝'w+＝0.01+sd;if(w<0)w＝0;if(w>10)w＝10;

[0397] sd*＝0.9'

[0398] <synaptic_type>

[0399] name＝'GLU'

[0400] presynaptic_event_rule＝'EPSP_plus_LTD'

[0401] post-synaptic_event_rule＝'LTP'

[0402] update_rule＝'Synaptic_Maintenance'

[0403] init＝'analog w＝6,sd＝0;int last_active＝-1000;'

[0404] <synaptic_type>

[0405] name＝'GABA'

[0406] presynaptic_event_rule＝'I-＝5'

[0407] delay＝1//默认延迟;对所有的突触来说是相同的

[0408] //类的Ne*M个记录(兴奋性->兴奋性或抑制性)

[0409] <synapse of GLU>

[0410] pre＝exc[i]//i＝1:Ne

[0411] post＝exc[j]or inh[j]//随机数,1与Ne之间或1:Ni

[0412] delay//1与D之间的随机数

[0413] //类的Ni*M个记录(抑制性->兴奋性)

[0414] <synapse of GABA>

[0415] pre＝inh[i]//i＝1:Ni

[0416] post＝exc[j]//1与Ne之间的随机数

[0417] 与END 2.0格式相比,END格式3.0实现了若干个主要变化。这些可以包括:

[0418] •对单向接合点的介绍

[0419] •将接合点update_rule分为两个规则:update_rule(被配置为修改接合点变量);和delivery_rule(被配置为修改突触后单元变量)

[0420] •移除post-synaptic_event_rule

[0421] •将presynaptic_event_rule分成:prepost_rule(用于STDP的LTP部分);postpre_rule(用于STDP的LTD部分);和delivery_rule(用于传送PSP)

[0422] •用于delivery_rule的原子加法(用于接合点和突触)

[0423] •移除基于时钟的突触更新规则(其可以经由postpre_rule来实现)

[0424] •经由到实例变量的链接来实现全局变量。

[0425] END 3.0中的各名称和陈述可以与END 2.0中的相似,除非做了相反的陈述。对于END 3.0,句法exc:3可以用于指示类型'exc'的实例3。

[0426] presynaptic_event_rule可以包括多个独立的规则。出于描述的目的,指示符t1和t2指示突触前神经元的尖峰到达突触后单元的时刻(即,传导延迟可能已经并入t1和t2),其中,t2对应于当前的仿真时间(也称为"now")。

[0427] prepost_rule可以在t1之前的任意时间执行,以实现STDP的LTP部分,其对应于突触前神经元在t1处发射而后续的突触后神经元在t1之后但在t2之前发射的一对脉冲。虽然prepost_rule规则访问了系统变量prepre(now-t1)和prepost(post_spike-t1),但其不必在时刻t2('now')处访问任何后续单元变量或系统变量,因为可能不清楚何时调用这个规则。如果prepost_mode＝11(1-to-1),那么各突触前尖峰触发用于prepost_rule的1个调用。如果prepost_mode＝1A(1-to-all),那么各突触前尖峰触发用于后续突触后尖峰的prepost_rule(具有其自身的prepost变量),直到下一突触前尖峰的时刻。参数prepost_max(如果给出的话)可以限制突触前尖峰之后的时间跨度,在这个时间跨度期间,考虑用于prepost_rule的突触后尖峰。例如,如果STDP的LTP窗只有50,那么没有必要考虑间隔大于50的pre-post尖峰对。在一些实施方式中,当t1之后的最早的post尖峰在晚于t1+prepost_max发生时,可以调用prepost_rule。在一些实施方式中,如果在t1与t2之间的时间段期间从不发生后尖峰,那么可能不调用规则。

[0428] postpre_rule可以刚好在t1时刻前执行,以便基于先前的突触前尖峰(prepre＝t2-t1)和最后的突触后尖峰(postpre)的时序来更新突触变量。可以提供之后的变量,即使自从先前的后尖峰发生已经经过了长的时间段。变量'now'指向当前时间,并且来自突触后单元的各变量可能是可用于读取的。

[0429] 可以在t1时刻,但在postpre_rul更新了突触权重之后调用delivery_rule。delivery_rule访问之后规则的各个变量,而且针对原子加法对后单元的变量具有写访问。

[0430] <rule>

[0431] name

[0432] exec_condition

[0433] code

[0434] init

[0435] <unit_type>

[0436] name

[0437] update_rule

[0438] event_condition

[0439] after_event_rule

[0440] init

[0441] <unit of unit_type>

[0442] unit_id

[0443] init

[0444] <junction_type>

[0445] name

[0446] update_rule

[0447] delivery_rule

[0448] init

[0449] <junction of junction_type>

[0450] pre

[0451] post

[0452] init

[0453] <synaptic_type>

[0454] type

[0455] prepost_rule

[0456] 列表3

[0457] 在列表3所示的代码例子中，可以在t2之前触发规则，并且该规则修改突触。该规则可以对突触变量进行读和写，但不能访问来自突触后单元的任何变量。规则可以访问prepre＝t2-t1和prepost(后尖峰-t1)。

[0458] prepost_mode

[0459] 可以支持两种模式：11(1-to-1)和1A(1-to-all)。前者最多调用一次规则，而后者在前单元的最后一个尖峰之后针对各个后单元尖峰来调用多次。默认：11

[0460] prepost_max

[0461] 限制了前单元尖峰之后的时间范围，以考虑形成前-后对。可以被忽略t1+prepost_max之后的一些或所有后尖峰。

[0462] postpre_rule

[0463] 代码或规则名称，其可以在t2处被触发，并修改突触。它能够对突触变量进行读和写，并对来自突触后单元的变量进行读访问。它可以访问prepre、postpre和now＝t2。

[0464] delivery_rule

[0465] 代码或规则名称，其可以在t2处被触发，并修改后单元的变量。它对突触变量、prepre、postpre和now＝t2进行读访问。该代码可以是原子加法的。

[0466] <synapse of synaptic_type>

[0467] pre

[0468] post

[0469] delay

[0470] init

[0471] 在一些实施方式中，希望对短期突触可塑性进行建模，其可以由突触前尖峰触发。通常，这要求具有变量或变量的矢量，变量可以由各个突触前尖峰修改，然后根据某个等式演变，但可能仅需要该变量在前脉冲时刻的值。在这种情况下，变量可以是各突触的一部分。然而，由于一些或所有这种变量的值对于所有突触来说是相同的，所以从END到引擎的编译器(链接器)能够从突触中移除这些变量，并替代地使用服从"pre-rule"的单个突触前变量。END格式可以具有特殊标签或标记或者语句，它们能够帮助编译器以突触事件规则或pre-synaptic(突触前)单元事件规则来识别这种pre-event触发的变量。

[0472] 如果END程序分散在多个引擎之间，那么各个引擎可以转移具有各突触前尖峰的这种变量的值。在一些实施方式中，接收来自这种突触前单元的突触的各引擎可以在保留变量的本地副本，按照它在主办突触前单元的引擎中被更新的方式进行更新。

[0473] spnet网络可以包括N＝Ne+Ni＝1000神经元，其中，Ne＝800个兴奋性神经元和Ni＝200个抑制性神经元。每个神经元可以包括M＝100个突触。一些或所有的兴奋性突触可能是可塑的，其中，随机延迟处于1ms与D＝20ms之间。抑制性→兴奋性突触可能是非可塑性的，其中，延迟D＝1ms。可能不存在抑制性→抑制性突触。模型的低级END 3.0描述可以被表达如下。

<rule>
name='QIF2' //只有 v 等式被定义
code = 'v+=(0.04*v+5).*v+140+I; I=4*rand()/MAX _RAND'
<rule>
name='spike'
code = 'v>30'
<unit type>
name = 'exc'
update _rule = 'u+=0.02*(0.2*v-u); I-=u;'
update_rule = 'QIF2'
event condition = 'spike'
after event rule = 'v=-65; u+=8;'

[0474]

init = 'analog v=-70, u=-14, I=0;'
<unit_type>
name='inh' //抑制性神经元
update_rule = 'u+=0.1.*(0.2*v-u); I-=u;'
update_rule = 'QIF2'
event _condition = 'spike'
after _event _rule = 'v=-65; u+=2;'
init = 'analog v=-70, u=-14, I=0;'
//Ne 个类的记录
<unit of exc>unit _id = i // i=1：Ne
//Ni 个类的记录
<unit of inh>

[0475]

unit_id=i //i=1:Ni

<rule>

name = 'LTD'

code = 'sd-=LTD(postpre); w += 0.00001*prepre + sd*10*(1- 0.9999^prepre); if (w<0) w=0; if (w>10) w=10; sd *=0.9999^prepre)'

init = 'LTD = {array};'

//sd'=-sd/10 sec; 0.9999Ax = exp(-x/10000);可以优化规则以便仅计算一次 0.9999^x： w+10*sd; sd*=..; w-=10*sd;' 还可以进行优化以使移除 10*（但是 LTP 和 LTD 表可以通过 10 进行缩放

<rule>

name = 'LTP'

code = 'sd+=LTP(prepost);'

init = 'LTP = {array};'

<synaptic _type>

name = 'GLU'

prepost _rule = 'LTP'

postpre _rule = 'LTD'

delivery _rule = 'I+=w'

init = 'analog w=6, sd=0;'

<synaptic_type>

name='GABA'

delivery _rule = 'I-=5'

delay=1 //默认延迟；对所有突触来说都相同

//Ne*M 个类的记录（兴奋性->兴奋性或抑制性）

<synapse of GLU>

pre = exc:i// i=1:Ne

post = exc:j or inh:j //随机数，1 与 Ne 或 1：Ni 之间

delay// 1 与 D 之间的随机数

[0476]
```
//Ni*M 个类的记录（抑制性->兴奋性）
<synapse of GABA>
    pre = inh:i// i=1:Ni
    post = exc:j//1 与 Ne 之间的随机数
```

[0477] 列表4

[0478] 初级网络接口(ENI)可以被实现为在低级描述END格式中描述的、用于实现两个仿真之间的数据交换的通信协议，或可能需要用于向仿真发送/从仿真接收数据的任何其它实体(例如，输入设备、可视化/调试工具和/或其它实体)。ENI可以与END本身紧密相连，并可以用于将大的END文件分成较小的片段，以确保正确的结果。END的某些部分可能需要END引擎处理通信事件的详细知识。

[0479] 现在参见图3，示出并描述了两个计算引擎302(每一个都运行END模型仿真)之间的通用通信框架300。ENI通信协议可以以某种物理通信手段来实现(例如，用于外围或以太网/无限宽带技术的USB/蓝牙，用于有更多需求的情况)。各计算引擎302可以在可用的传输层(如图3中的308所指示的通信管路)的顶部实现ENI(如图3中的306所指示的通信管路)。要注意的是，ENI形成由END描述的模型之间的、但不在引擎302自身之间的通信层。

[0480] 在一些实施方式中，可以由低级传输层连接的引擎302可以发现彼此并自动配对。手动建立是可能的(例如，对于经由IP协议连接的引擎)。通信说明文档(ENI文件)可以与END文件一起提供给引擎。一旦引擎发现它与运行正确END文件的另一引擎配对，就可以协商和建立ENI信道(在图1中的绿色管道)。ENI通信说明文档可以被视为映射，其帮助双方互相理解并定位正确类的正确单元。

[0481] 图4示出了图3中ENI通信信道306的框图。信道306可以由其端点(低级END描述的ID，例如图3中的END A和END B)和参数集来定义，该参数集描述了如何执行通信以及可以传输什么类型的信息。通信信道306可能是单向的，从左向右，并可以由发送实体402和接收实体404来定义。数据传输可以是单向的(尽管不限制传输层中的协商和元信息)。

[0482] ENI信道参数可以包括以下各项中的一项或多项：

[0483] 发射方与接收方END ID和唯一标识信道的标识符；

[0484] 发射方与接收方的类；

[0485] 发射方与接收方的类实例(由它们在类中的编号枚举)；

[0486] 内容：事件/尖峰通知或某些参数的值；

[0487] 发送的数据的格式；

[0488] 操作模式(同步、异步)和它们的参数(频率、相移和/或其它参数)；

[0489] 和/或其它参数。

[0490] 在一个实施方式中，ENI通信信道可以用于交换(i)尖峰(事件)通知和(ii)所选择的类参数(变量)的值。这两种内容类型可以获得通过信道发送的不同数据，即：

[0491] • 单元索引(用于尖峰通知)

[0492] • (由信道传送的一些或所有实例的)变量的实际值

[0493] 在一个实施方式中，网络单元索引与值变量交织。当值需要发送给经历了事件(尖峰)的单元时，这种实施方式可能是适用的，以便最小化网络业务。总之，可以支持下列数据

格式：

[0494] • 有值或无值的索引(一些单元)。如果值丢失，那么数据可以解释为尖峰通知。

[0495] • 没有索引(一些或所有单元)但是只有值。(如果值丢失，那么数据可以解释为所涉及的各单元的尖峰通知)。

[0496] 这两种内容类型(即，事件和数据)未混合在单个ENI信道实例中。在一些实施方式中，无论何时将ENI信道设置成发送事件/尖峰通知，目标单元都可以("人工地")发射；也就是说，事件可以在接收方尖峰队列上进行调度，但是像LTP或重置之类的任何后事件动作都不执行。在一些实施方式中，可以外部地发射的单元不具有任何局部输入突触，该突触可能服从可塑性规则(例如，LTP)。在一些实施方式中，单元可以被配置成既响应外部发射触发器又响应可塑性规则，从而在外部地发射它们时，不调用后事件规则。这种配置确保仿真一致性，并使得分开的仿真产生像"单个仿真"一样的结果。因此，当适当分隔时，利用分开的仿真来获得像利用单个仿真一样的结果是可能的(考虑到上述限制)。在一些实施方式中，经由按要求介绍特殊假/接收单元，可以促进模型分隔。

[0497] 图4示出了ENI通信信道中的内容、格式和期望的行为。在发送变量的值的情况下，信道描述规定了发送方和接收方的类的变量集。换句话说，ENI规定源类中的哪个变量可以映射到目标类中的哪个变量。一致性要求可以是通过信道来发送的变量的内部表示在发送方和接收方之间兼容或者是通信信道执行必要的会话。在各通信事件后，目标类中的值可以更新至新的值。

[0498] 在一些实施方式中，ENI信道传输与END格式中的单元相关的数据。在一些实施方式中，可以传输与突触或接合点相关的数据(例如，突触权重或其它变量)。

[0499] ENI信道可以引入从END格式的源单元类的某个元素到目标类的某个元素的映射。各信道建立一个发送方类与一个接收方类之间的通信。如果尖峰通知被发送的话，或者如果(如果传输了参数值的话)值的顺序被发送的话，映射建立()可以通过线来发送的索引(参见图4)。

[0500] ENI信道不需要在各模型仿真循环中发送信息。ENI文件规定了发送/接收的时间段T1、T2(在引擎模型仿真循环中表述)。标志符T1对应于发射侧时间段(因此数据仅在每个T1个循环被发送)，T2对接收机侧(期望将数据每个T2个循环传送给仿真)进行描述。数据可以在任意时间点传送给引擎，但是引擎可以保留(缓存)它，并可以使它在合适的接收点对运行的仿真来说是可用的。

[0501] ENI通信信道可配置为以两个模式操作-同步和异步。在同步模式，发送和传送可以同步到达最近的仿真循环。在异步模式，(输入)数据可以连续地提供给计算引擎并传送到仿真实例的各单元(当其准备好的时候)。虽然同步模式确保及时地传送数据，但它可能引起高输入数据速率的严重性能瓶颈。相反地，非同步模式由于可变的数据传送时间，可能引起不明确的模型行为。

[0502] 图5呈现了ENI(方案)的操作模式的例子。合并了两种格式，两个内容类型和两个操作模式来形成图6的实施方式中所示的立方体。在一些实施方式中，就可能的数据格式而言，ENI可以扩展，并可以包括内容感知压缩。

[0503] 在同步模式的一个实施方式中，可能假定接收引擎不能通过数据接收点，除非必要的数据已到达(因此接收方与发送方同步)。该模式下的信道规定额外属性-即，相移S。如

果相移S>0且如果S循环之前发送的数据分组已经到达，则引擎可以进行仿真(因为引擎是同步的，所以无所谓是发射方循环还是接收方循环)。当神经元仿真的实际结构允许时，相移允许更好地利用通信信道(也就是说，通过ENI信道发送的某些投射具有延迟，该延迟可以用于放松对信道的需求，并引入相移，见图7A和7B)。潜在传输可以基于它们的相移(传输数据所需的时间)将优先顺序引入给信道。

[0504] 在数据不用等待任何传送通知就发送之后，发射方可以进行其它任务，然而，如果没有传送该消息，则通过信道发送数据的下一次尝试可以保持仿真。(如果超过S个非确认传送存在于具有相移S的同步信道中，则送射引擎可能不继续)。

[0505] 在一些实施方式中，特别是适用于非同步(即，异步)模式的情况下，可以规定用于发送和接收数据的频率，但是直到数据被发送为止，引擎不会停止其执行。在接收侧，异步模式不会施加任何数据传送时序的限制。在一些实施方式中，计算引擎可以被配置为接收以不可分割块(到达的数据块传送)。在一些实施方式中，数据可以通过流来传输(流传送)。其它例子是可能的，例如，举例来说，块传送和流传送的组合。在块传送子模式中，假定所发送的信息在块内的数据已经传送到接收机(例如，图4中的接收机404)后被传送，并且通信交易可以完成。在流式子模式，在接收机处可获得的数据可以逐步发送到引擎，而不管传输交易是否完成。

[0506] 在异步块的模式，假定最近的消息实际上传送到引擎，而在引擎遇到接收点之前接收到的消息可以被丢弃(当实时输入装置(如照相机)以比引擎能够进行处理还要快的异步模式来发送数据时，这可能是有用的)。异步流模式对数据进行累加，并在最接近的接收点传送它。

[0507] 图7A示出了两个计算引擎702、704之间的完全同步通信的一个例子。图7A的实现可以确保引擎702、704之间及时的交换数据，并能减小通信延迟，从而简化由于延迟造成的潜在的通信信道拥堵，该延迟可能与大的、实时神经元模型仿真运行中的大量并且不均匀的数据流相关联。在一些实施方式中，可以使用各种数据传送优化，这能显著扩大通信带宽。在图7B中，事件索引和数据值可以通过单独的ENI信道发送(在图7A中，分别由加重的细箭头706、708指示)，并具有分别的接收点。

[0508] 图7B示出了同步通信的一个例子，其具有一个系统时钟循环(仿真步长)(在两个方向上S＝1)的相移。由于窗口重叠(如图7B中由竖直箭头752、754所示)，通信信道被更平均地使用，但是尖峰和值具有每步长1ms的延迟而到达。

[0509] 发送点指的是仿真模型的逻辑压缩，仿真模型可用于描述数据和事件，所述数据和事件在计算引擎执行模型期间对单元进行处理之后立即变得可用。类似地，接收点可以用于描述在各仿真循环期间对接合点进行处理之前使用的数据分级容器。这种设置可以留下短的通信窗，但下列可选的优化是可能的：

[0510] 发送驱动器按照优先级顺序来处理单元，并一旦数据变的可用之后就发出数据，而其它单元仍由引擎并行处理。

[0511] 接收驱动器执行局部接合点，同时等待必要的数据到达。

[0512] 如果信道发送尖峰通知，则计算引擎能在从非本地单元接收尖峰上的数据之前处理来自本地单元的突触。在这种情况下，通信窗可以显著地扩展(如由图7B中虚线箭头所示)。

[0513] 列表7示出了用于在使用END的网络仿真中的ENI定义文件的一个例子(为发射方/接收方驱动器所知)。本领域技术人员应该理解,虽然以下例子表明可能需要一些或所有数据来建立信道,但该文件的实际格式可能转换成XML或一些其它格式。

[0514] 示例4:

[0515] 示例4描述了引擎仿真,其中,'Retina'END文件从发射的类RGC向运行'LGN'END文件的引擎发送所选择的单元的索引。类'exc'中所选择的元素被发射。通信可以是同步的,同步点出现在发射方或接收方的一者或二者的各循环处,并且信道可以没有相移(延迟)。

[0516] ENI ID＝'CommC'

[0517] SOURCE_ID＝'Retina'

[0518] //源END文件(仿真)的id

[0519] TARGET_ID＝'LGN'

[0520] //目标END文件(仿真)的id

[0521] MODE＝SYNC

[0522] T1＝1

[0523] //发射方的同步频率-何时发送数据(在仿真循环)

[0524] T2＝1

[0525] //接收方的同步频率-何时预期数据(在仿真循环)

[0526] S＝0

[0527] //接收方相移

[0528] FORMAT＝INDICES

[0529] //信道仅观测到尖峰单元

[0530] SENDER_VALUES＝NONE

[0531] //将仅发送索引,也就是说,信道将发送目标单元

[0532] RECEIVER_VALUES＝NONE

[0533] SENDER_CLASS＝'RGC'

[0534] RECEIVER_CLASS＝'exc'

[0535] NUMBER_OF_UNITS＝500

[0536] //信道将观测到多少单元?

[0537] SENDER_UNITS＝84233953920134...

[0538] RECEIVER_UNITS＝33456798183419...

[0539] 列表7

[0540] 示例5:

[0541] 示例5示出了引擎仿真,其中,'Camera'END将类Pixel中的'R,G,B'变量值异步发送到Retina END文件中的类RGC的变量'Red,Green,Blue'中。

[0542] ENI_ID＝'ENI2'

[0543] SOURCE ID＝'Camera'

[0544] //源END文件(仿真)的id

[0545] TARGET ID＝'Retina'

[0546] //目标END文件(仿真)的id

[0547] MODE＝ASYNC

[0548] STREAM＝NO

[0549] //接收方将不累积数据(仅将最新的数据传送到引擎)

[0550] T1＝1

[0551] //发送方的同步频率-何时发送数据(在仿真循环中)

[0552] T2＝2

[0553] //接收方的同步频率-何时预期数据(在仿真循环中)

[0554] FORMAT＝NOINDICES

[0555] //将每次发送所有的值

[0556] SENDER_VALUES＝'R,G,B'

[0557] //将发送那些变量的值

[0558] RECEIVER_VALUES＝'Red,Green,Blue'

[0559] //将更新那些变量的值

[0560] SENDER_CLASS＝'Pixel'

[0561] RECEIVER_CLASS＝'RGC'

[0562] NUMBER_OF_UNITS＝1024

[0563] //信道将观测到多少个单元?

[0564] SENDER_UNITS＝1234567891011……

[0565] RECEIVER_UNITS＝33456798183419……

[0566] 列表8

[0567] 图13所示的表列出了在列表8中使用的通信参数的可能值。

[0568] 在一些实施方式中,存在许多方式来处理大的神经仿真模型(例如使用高级描述格式来定义的)。在一些实施方式中,可以由单个处理计算引擎来执行处理。在一些实施方式中,处理可以分布在若干个计算引擎(计算节点)的集合中。为了实现有效的工作量分布,需要分割模型。在一些实施方式中,可以产生模型的一个大的低级描述(END文件),并且由所分布的计算引擎来执行分割。该示例提供了实时负载调整和再平衡的益处,但是技术上更复杂,并且可能需要更先进的负载分配控制器。在一些实施方式中,模型可以被分割成END文件的集合和ENI通信信道,它们可以彼此分开地执行。存在使用ENI将神经仿真分成若干部分的方式,这可能通过引入额外的“假”单元。

[0569] 图8和图9示出了根据本技术的某些方面进行模型分割的一个例子。图8示出了END模型800,其包括若干个单元802、突触804和接合点806,并使用END文件END0进行描述。沿着突触804的数字指示突触延迟。模型800仿真可以使用单个计算引擎0来执行。

[0570] 图9示出了将模型800分割成两个分区910和920,其分别使用END文件END1、END2来描述。分区910可以使用三个ENI接口来互连:在图9中分别由912、914和916指示的ENI1、ENI2、ENI3。已经引入了一些额外的“假”单元(如902标记的矩形)。因为ENI 1信道具有相移S＝2,所以可以使用目标突触的较小延迟来补偿信道通信延迟,如图8和图9中分别由突触804和904的比较所示出的。需要注意的是,当沿着分割线在原始模型(模型800)中没有最小延迟(例如,0)的突触时,这种实施方式是可能的,其中,分割线可以由ENI信道替换。根据高级定义语言的编译器可以考虑延迟,同时分割仿真,以优化通信(例如,扩大通信窗、平衡所

发送的数据量，和/或用于优化通信的其它操作）。模型900的仿真可以使用两个计算引擎来执行：图9中所示的引擎1和引擎2。假定所分配的计算引擎1/引擎2具有足够的数字精度来避免化整误差，与单个分区仿真实施方式800相比，多分区仿真900能够产生相同的仿真结果。

[0571] 在一些实施方式中，可以实现被分割的模型（未示出）的下列特征：

[0572] • 空中ENI信道的建立和链接协商（用于动态调试和监控）；

[0573] • 对额外数据格式的支持，例如，模拟；

[0574] • 对专用硬件基础架构的支持以运行ENI；

[0575] • 用于规定ENI格式的方便并且直观的方式。

[0576] 如以上关于分布式模型仿真所描述的（例如，图9的分割模式），可以将模型划分成若干个分区（例如，分区910、920）。各分区可以在单个计算引擎上执行（图9的引擎1、引擎2）。在一些实施方式中，计算引擎可以包括单个处理单元（CPU）实施方式。各计算引擎可以包括并行运行的多个处理单元（PU），例如，CPU、FPGA、MCU。这种并行使得模型仿真的吞吐量（其可以利用并行PU的数量进行缩放）大量增加。为了启用模型并行执行，仿真域可以分成若干个处理块（处理域），各个块可以分配给各个PU。各处理块可以被配置为（i）保持跟踪本地区室、本地尖峰队列和/或其它本地信息；（ii）存储输入“突触”的列表（表）（不适用于END 3.0）；以及存储可以被处理但没有锁定的本地接合点的s个列表。区室可以利用域ID-s来标记，当其进行发射时可以通知域ID-s。当这种外部可视区室发射时，适当的信息可以被发送给相邻域，相邻域进行剩余的处理（例如，突触队列）。

[0577] 对于各个步骤，经由外部（远程）接合点连接的区室的电压可以发送到它们的目标域。相应地，所接收的电压可以用于计算接合点电流。

[0578] 处理域可以交换的数据是下列各项中的一项或多项：（i）尖峰；或（ii）接合点电压（或一些跨接合点传输的其它变量）。由于大部分接合点是树突状类型的（本地），所以各交换中的数据量不大（如果域划分考虑了树突状接合点的话）。

[0579] 在一些实施方式中，异构并行化计算引擎可以使用多核对称多处理器（SMP）硬件来实现。在一些实施方式中，SMP实现可以包括要实现的图形处理单元。

[0580] 图10-12描述了技术的不同示例性实现，该技术不依赖于END语言的句法。图10示出了END引擎的三个基本结构，该引擎可以在标准的CPU、GPU或集成电路（例如，ASIC）上实现。这些可以是“单元”1001、“二元组（doublet）”1011和“三元组（triplet）”1021。END引擎处理单元、二元组和三元组规则的执行，并访问这些元素的存储器。上述END格式可以作为硬件描述语言，该语言对具有执行专用的神经元网络的单元、二元组和三元组的半导体电路进行配置。

[0581] 在一些实施方式中，各基本结构（单元、二元组和三元组）可以作为多线程处理器上的单线程来实现。在一些实施方式中，各结构可以实现为超单元、超二元组和超三元组，它们可以包括配置成分别使用时间复用来处理单元、二元组和三元组的专用电路（可能地，三个不同的电路用于单元、二元组和三元组）。

[0582] 在一些实施方式中，单元1001表示神经元或神经元的一部分，例如，树突状区室。在一些实施方式中，单元1001表示一群神经元，其中神经元的活动表示该群的“平均发射率”活动或该群的活动的一些其它平均场近似。各单元可以具有它们自己的存储变量和更

新规则，该规则描述在其存储上执行怎样的操作。操作可以是基于时钟的，即，在仿真的各时间步长处执行，或者它们可以是基于事件的，即，当某事件被触发时执行。单元更新规则可以不包括属于其它单元的变量。因此，单元更新规则的执行可以独立于其它单元的单元更新规则的执行顺序，从而使得并行执行单元更新规则。

[0583] 取决于单元变量的值，单元可以产生事件——脉冲或尖峰——其经由二元组来触发其它单元中的突触事件。例如，图10中的单元1002可以经由二元组1011来影响单元1003，该二元组1011代表从突触前神经元(突触前单元1002)到突触后神经元(突触后单元1003)的突触。

[0584] 单元可以具有后事件更新规则，该规则可以在事件被触发后触发。这些规则可以负责单元变量的修改，所述单元变量的修改是由于事件，例如电压变量的尖峰后重置。

[0585] 各二元组可以具有其自身的存储变量，并可以访问突触后单元的变量。访问可以包括读和写。各二元组可以具有二元组事件规则，该规则对二元组存储器进行修改，以实现突触可塑性和突触后单元存储，以实现脉冲的传送。二元组事件规则包括以上END格式中所描述的突触规则。

[0586] 由于多个二元组(例如，图10中的1016-1018)能够将相应的多个突触前单元1006-1008连接到单个突触后单元1009，因此，可以期望二元组并行或以任意顺序来修改突触后单元存储器，并且结果是独立于顺序的。如果对突触后单元存储器的操作是原子加法的(如在GPU中)、原子乘法(其通过逻辑变换等于加法)或重置为某个值(其中，各二元组试着重置成同一值)，则这很容易实现。在一些实施方式中，由二元组事件规则修改突触后单元变量可能不会在规则中使用。结果取决于二元组事件规则的执行顺序。

[0587] 在神经元计算的上下文中，可以期望具有轴突传导延迟，以便由突触前单元产生的事件与由单元触发的二元组事件规则的执行之间存在时间延迟。延迟可以实现为缓冲器，从而各二元组接收具有某延迟的事件。也就是说，END引擎登记由突触前单元产生的事件，并取决于延迟的幅度来将特殊的标记放入延迟线队列。可以对延迟进行倒数，然后将事件发送给二元组，用于二元组事件规则的执行。

[0588] 在一些实施方式中，二元组不访问突触前单元变量。然而，在一些实施方式中，二元组可以访问一些突触前单元变量。在一些实施方式中，突触前单元具有在事件由单元(即，基于事件的存储器)触发期间被修改的变量，并且修改不取决于其它突触前单元变量的值。在这种例子中，这种基于事件的存储器可以驻留在突触前单元存储器中，或等效地，其可以被处理，如同事件触发的变量的副本驻留在各二元组处或在二元组之间共享一样。

[0589] 二元组事件规则可以是二元组规则的类的一部分，这在一些实施方式中，可以包括后事件规则和二元组更新规则，如图11所示。后事件规则可以是由突触后单元触发并针对连接到突触后单元的二元组所执行的规则。在一些实施方式中，二元组的后事件规则可以取决于恰在突触前事件之前的时序。二元组更新规则可以是基于时钟的规则，其在一个或多个时间步长处针对各二元组执行(例如，一时间段内的每个系统时钟循环)。

[0590] 在图12所示的实现中，二元组事件规则可以包括定时事件规则，其(i)基于突触前和突触后单元的事件时序而进行配置；和(ii)控制对二元组存储器变量的修改。图12中所示的更新规则在于实现依赖于尖峰-时序的可塑性(STDP)—尖峰网络中的可塑性的标准格式。

[0591] 时序事件规则可以包括实现STDP的一部分的前后事件规则，STDP的一部分对应于首先发射的突触前神经元(或从突触前神经元先到达的尖峰)，然后，突触后神经元其后发射例如在脉冲1213、1214前产生的脉冲1202。在经典的STDP中，这对应于STDP曲线的长时增强(LTP)部分。该规则基于突触前和至少一个后续突触后发射的时序来修改二元组的存储器，例如，对1202、1213(线1223所示)；它基于时间差。

[0592] 时序事件规则可以包括后-前事件规则，其实现经典STDP的长期压缩部分，其在当突触后单元先发射，并且随突触后前单元在此之后发射时发生，例如，像脉冲1202之前产生的脉冲1211、1212。该规则基于突触前和突触后发射的相关时序来修改二元组存储器；其可以取决于事件差，即，取决于脉冲1212、1202之间的时间差。

[0593] 前后和后前事件规则可以取决于突触后单元存储器中的变量的值。

[0594] 在一些实施方式中，期望二元组的突触前单元来分配它们的存储器，以便具有公共突触前单元的二元物可以聚集在一起，并因此在系统存储器中分配。该方法可以最小化对系统存储器的随机访问。

[0595] 二元组事件规则可以包括脉冲传送规则，其基于二元组存储器和突触后单元存储器的值来修改突触后单元的变量的值。

[0596] 二元组的描述可以由突触变量的描述和END格式中的规则来提供。

[0597] 如上面在图10中所描绘的，三元组1021连接两个单元1004和1005。这种三元组表示END格式中的接合点。在一些实施方式中，三元组1021可以包括两个神经元之间的间隙接合点连接。在一些实施方式中，三元组1021对应于阻抗连接，其连接神经元的两个相邻树突状区室。在一些实施方式中，三元组1021可以是两个接合点类型的组合。三元组可以包括其自身的存储器，并且其访问两个单元的存储器。三元组可以包括三元组更新规则，其在各仿真时间步长处执行。三元组更新规则可以更新各三元组的存储器和至少一个单元的存储器。在一些实施方式中，三元组更新规则可以更新两个单元的存储器。由于期望这中更新是独立于顺序的，所以至少一个单元的存储器更新可以经由对单元变量的原子加法来执行，该单元变量不参与三元组更新规则。

[0598] 三元组的描述可以由END格式中的接合点的描述来提供。

[0599] 一些实施方式可以仅仅执行平均发射率模型，其中，没有事件，但是各单元经由三元组更新规则来向其它单元发射信号。

[0600] 三元组可以根据突触前单元来在系统存储器中分配，以便源自公共突触前单元的三元组聚集在一起。三元组可以根据突触后单元进行分配。在一些实施方式中，三元组可以根据神经网络的结构在系统存储器中分配。例如，如果神经网络具有多区室的树突，则当树突树的动态在系统时钟循环期间被评估时，负责连接树突分隔区的三元组可以被最优地分配成对存储器访问免疫。

[0601] 一些实施方式执行单纯的事件驱动的架构，其中，单元触发事件，该事件经由二元组向突触后单元传送脉冲。那些在时钟循环期间接收到至少单个脉冲的单元可以针对事件条件被测试，并且如果满足，则执行后事件规则，并可以执行相应的二元组事件规则，从而脉冲可以传送到其它单元。

[0602] 事件驱动的架构(EDA)可以定义为END格式的一般化，其用作被配置为将计算描述从模型的神经科学描述中隔离的抽象层。EDA定义存储管理、并行化和由事件触发的规则，

并使得END码直接编译成EDA码。

[0603] END格式中的事件和EDA架构对应于脉冲(无论是实际的还是虚拟的)、脉冲的软件表示、脉冲的典型突发,或其它分立的临时事件。

[0604] EDA存储器管理在以下方面不同于END框架(从变量的所有的角度):

[0605] 单元拥有其自身的变量。当执行单元A的更新规则时,可能不需要访问网络中任何其它单元的变量。相反地,由其它单元执行的规则不需要访问单元A的变量。

[0606] 突触拥有其自身的"突触"变量,例如权重,诸如last_active的变量和/或其它变量,并且它们可以指的是(读出和写入)突触后单元中的某些变量。当执行presynaptic_rule或post-synaptic_rule时,两个或多个突触可以试图访问和修改突触后单元的相同变量。然而,突触不会竞争它们自身的突触变量。

[0607] 接合点拥有它们的"接合点"变量,但是它们访问和修改unit_1和unit_2中的变量。当执行接合点时,并行访问它们的接合点变量可能不会有竞争,但是访问unit_1和unit_2中的变量可能会有竞争。

[0608] 因此,单元、突触和接合点就变量的所有而言可以被处理为单元、二元组和三元组。单元拥有变量集,突触拥有一个变量集并且指的是单元所拥有的另一集。接合点拥有一个变量集并且指的是两个单元所拥有的两个其他的变量集。这种命名法适用于描述END 1.0、2.0和3.0格式,以及下列的示例性实施方式。

[0609] 类成员event_condition触发下列规则的执行:

[0610] • 作用于已经触发事件的单元(触发器单元)的after_event_rule;

[0611] • 作用于从触发器单元指向其它单元的突触(二元组)的presynaptic_event_rule;

[0612] • 作用于从其它单元指向该单元的突触(二元组)的post-synaptic_event_rule。

[0613] 可以在网络仿真的各时间步长处执行类成员update_rule,并且它更新单元、突触(可能通过后台处理)和接合点中的变量,即,单元、二元组和三元组中的变量。

[0614] 作为元素的单元、二元组和三元组可以指网络元素。因此,END格式定义(i)元素,(ii)作用于元素的规则,以及(iii)由元素触发并引起作用于其它(目标)元素的其它规则的执行的事件。

[0615] 根据本技术某些方面的EDA框架的开发背后的对象包括:

[0616] • 具有自己的变量并涉及其它元素中的变量的元素(例如,单元、二元组、三元组和/或其它元素);

[0617] • 作用于元素中的变量的时钟驱动的规则;以及

[0618] • 由一些元素触发并作用于其它(目标)元素中的变量的事件驱动的规则(事件)。

[0619] EDA指令集开始于对作用于抽象(符号)变量的规则进行定义。规则可以包括先前定义的其它规则,以形成没有循环的规则的有向图。元素可以由作用于同一元素或不同元素的时钟驱动的规则来定义。事件可以由适用于其它元素的触发器条件和目标规则来定义。

[0620] EDA指令集的一个例子如下列表5中所示。列表5中的粗体关键字指示RND指令集的组件,而非粗体字指示用户定义的名称和值。

[0621] rule_name＝rule

[0622] code＝'....'

[0623] code＝another_rule_name

[0624] init＝'....'

[0625] 列表5

[0626] 在列表5中，标识符"code"可以指任何代码串或先前定义的另一规则的名称。虽然C-代码语法在列表5中使用，但是本领域技术人员应该理解，任何其它语言描述(例如，C#、Python、Perl和/或其它语言)等效地适用于本技术。可以有多种代码和规则包括在能按照包含顺序来执行的规则中。在一些实施方式中，可以分别定义在多个元素类型中使用的规则，以便引擎可以使用其加速方法来并行执行这种规则。语句"init"定义需要执行规则的静态变量(规则内的全局变量)，例如，其定义查找表。"code"可以指定义在"init"中的静态变量、定义在要素中的实例变量(见下)或定义在其它要素类型中的实例变量，例如，"I+＝A.w+B.w"指实例变量I和定义在要素A和B中的变量w。

[0627] element_name＝element

[0628] rule＝rule_name

[0629] rank＝a number or range of numbers

[0630] init＝'....'

[0631] 后者是要素类型的定义。此处，"rule"指的是此前定义的规则或指的是代码串。参数"rank"规定规则中时钟循环中规则执行的等级顺序。它采用来自间隔[01]的固定点值。例如，rank＝0.45意味着该规则可以在具有较低等级的各规则之后、具有较高等级的各规则之前执行，但是与等级0.45的规则并行执行。如果等级被给定为间隔，例如，rank＝min:max，则引擎具有在等级小于min的各规则之后以及等级大于max的各规则之前的任何时间执行该规则的自由度。如果丢失"rank"，则等效于默认值rank＝0:1，即，引擎对于选择何时执行该规则具有完全的自由度。如果等级大于1，那么引擎跳过循环来执行规则。例如，rank＝2.45可以使得引擎跳过2个循环，直到该规则的下一次执行。字符串"init"规定要素的实例(本地)变量的名称，并设置它们的默认初始值。

[0632] id＝element_name

[0633] A＝other_element_name.id

[0634] B＝other_element_name.id

[0635] variable_name＝value

[0636] 后者是元素类型实例的定义。此处，"element_name"是之前定义的元素类型的名称。"variable name＝value"这一行可以设置实例变量的值，它们不同于在元素定义中的"init"语句中定义的默认值。如果元素定义中的rule_name指的是其它元素(其为对于成对物和三对物中的情况)，那么可以在此规定这些元素的ID。需要注意的是，可以使用任何变量名称，而不必是A和B(或END中的unit_1和unit_2)来指示其它元素。

[0637] event_name＝event

[0638] trigger_condition＝rule_name

[0639] trigger_rank＝number or number range

[0640] target_code＝rule_name

[0641] target_rank＝number or number range

[0642] 后者是事件类型的定义。此处,"trigger condition"是返回真/假值的规则或字符串代码的名称。该条件(适用于元素;见以下)可以被评估处于由"trigger_rank"给定的等级。当条件为真时,其触发处于等级"target_rank"的目标元素(见以下)中的"target_code"的执行。

[0643] event_name

[0644] trigger=element_name.id

[0645] target=element_name.id

[0646] 后者是事件类型实例的定义。它规定哪个元素是"触发器",以及哪个元素是事件的"目标"。

[0647] 示例3:没有延迟的SPNET

[0648] 随机连接800个兴奋性神经元和200个抑制性神经元(兴奋性—>所有和抑制性—>兴奋性连接的100个)的网络可以利用服从STDP和没有传导延迟(对于传导延迟,参见下例)的兴奋性突触来定义。

[0649] QIF=rule

[0650] code='v+=0.5f*((0.04f*v+5.0f)*v+140.0f-u+I);

[0651] v+=0.5f*((0.04f*v+5.0f)*v+140.0f-u+I);I=0.0f;'

[0652] exc=element

[0653] code='u+=0.02f*(0.2f*v-u);'

[0654] code=QIF

[0655] rank=0

[0656] init='float v=-65.0f;float u=-13.0f;float I=0.0f;int

[0657] last spike=-1000'

[0658] 1:800=exc

[0659] inh=element

[0660] code='u+=0.01f*(0.2f*v-u);'

[0661] code=QIF

[0662] rank=0

[0663] init='float v=-65.0f;float u=-13.0f;float I=0.0f;int

[0664] last spike=-1000;'

[0665] 1:200=inh

[0666] spike=rule

[0667] code='v>30'

[0668] after_spike_reset_exc=event

[0669] trigger_condition=spike

[0670] trigger_rank=0.1

[0671] target_rule='u+=8;last spike=now'

[0672] target_rank=0.2

[0673] after_spike_reset_exc

[0674] trigger=exc.1:800

[0675] target＝exc.1:800

[0676] after_spike_reset_inh＝event

[0677] trigger_condition＝spike

[0678] trigger_rank＝0.1

[0679] target_rule＝'u+＝2;last_spike＝now'

[0680] target_rank＝0.2

[0681] after_spike_reset_inh

[0682] trigger＝inh.1:200

[0683] target＝inh.1:200

[0684] inh_syn_event＝event

[0685] trigger_condition＝spike

[0686] trigger_rank＝0.1

[0687] target_rule＝'I-＝5.0f'

[0688] target_rank＝0.3

[0689] //制作200*100这种条目

[0690] inh_syn_event

[0691] trigger＝inh.<random number between 1and 200>

[0692] target＝exc.<random number between 1and 800>

[0693] weight_update＝rule

[0694] code＝'w+＝0.01f+sd;if(w<0.0f)w＝0.0f;if(w>10.0f)

[0695] w＝10.0f;sd*＝0.9f'

[0696] exc_synapse＝element

[0697] code＝weight_update

[0698] rank＝999:1000//该循环内的任何时间

[0699] init＝'float w＝6.0f;float sd＝0.0f;int last active

[0700] 1000'

[0701] id＝exc_synapse

[0702] post＝<either exc.id or inh.id with id＝random>

[0703] PSP_LTD_rule＝rule

[0704] code＝'post.I+＝w;sd-＝LTD(now-post.last_spike);

[0705] last_active＝now'

[0706] init＝'float LTD＝{..}'

[0707] PSP_LTD_event＝event

[0708] trigger_condition＝spike

[0709] trigger_rank＝0.1

[0710] target_rule＝PSP_LTD_rule

[0711] target_rank＝0.3

[0712] PSP_LTD_event

[0713] trigger＝exc.<each id>

[0714] target＝exc_synapse.<each corresponding id>

[0715] LTP_rule＝rule

[0716] code＝'sd+＝LTD(now-last_active)'

[0717] init＝'float LTP＝{...}'

[0718] LTP_event＝event

[0719] trigger_condition＝spike

[0720] trigger_rank＝0.1

[0721] target_rule＝LTP_rule

[0722] target_rank＝0.4

[0723] LTP_event

[0724] trigger＝{for each exc.id and inh.id}

[0725] target＝exc_synapse.<each corresponding id>

[0726] 列表6

[0727] 链接器可以被配置为将由'尖峰'规则触发的事件(处于同一等级内)组成单个事件,以便在每一个针对各单元的模型仿真步长,计算引擎执行'尖峰'条件一次。

[0728] 在一些实施方式中,等级信息在规则的定义中提供;这样,可以不需要在事件的定义中重复等级。

[0729] 在尖峰神经元网络(例如,强化学习)的一个或多个实现中,由神经元将可塑性适应从突触后响应产生中解耦合可能是有利的。

[0730] 图14示出了根据一个或多个实现的事件驱动的网络更新机制。网络1400可以包括至少一个尖峰神经元1420,其根据例如尖峰响应处理(SRP)1422进行操作。神经元1420可以经由M输入连接1404来接收M-维输入尖峰流X(t)1402。在一些实施方式中,M-维尖峰流可以对应于连接至神经元1420的M-输入突触连接。如图14所示,各输入连接的特征在于连接参数1406w_{ii},其可以被配置为在学习期间进行调整。在一个或多个实现中,连接参数可以包括连接效力(例如,权重)。在一些实施方式中,参数1406可以包括突触延迟。在一些实施方式中,参数1406可以包括突触传输的可能性。

[0731] 在一些实施方式中,神经元1420可以被配置为经由连接1408接收外部输入。在一个或多个实现中,输入1408可以包括训练输入,例如,正和/或负强化信号,其由图14中的r^+,r^-表示,指示强化信号尖峰流,这可以表达如下:

[0732] $$r^+(t)=\sum_i \delta(t-t_i^+), r^-(t)=\sum_i \delta(t-t_i^-)$$

[0733] 其中,t^+_i、t^-_i表示例如分别与正和负强化相关联的尖峰时间。

[0734] 在被监控的学习的一些实施方式中,监控尖峰可以用于触发神经元突触后响应。

[0735] 在强化学习的一些实施方式中,外部信号(例如,强化尖峰)使得神经元进入探索域(例如,在一个实施方式中,通过在短时间内增加突触权重)。

[0736] 神经元1420可以被配置为产生输出Y(t)(例如,突触后尖峰),其可以经由输出连接1414传送到期望的目标(例如,其它神经元1420_1、1420_2、1420_3)。如图14所示,各输出连接1414的特征在于可以在学习期间进行调整的连接参数1416。在一个或多个实现中,连接参数1416可以包括连接效力(例如,权重)。在一些实施方式中,参数1416可以包括突触延

迟。在一些实施方式中，参数1416可以包括突触传输可能性。

[0737] 神经元1420可以被配置为实现控制器功能，例如，在前面并入的名称为“STOCHASTIC SPIKING NETWORK APPARATUS AND METHODS(随机尖峰网络学习装置和方法)”的美国专利申请序列No.13/487,533中所描述的，以便控制例如机器臂。输出信号y可以包括电机控制指令，该命令被配置为沿着期望的轨迹移动机器臂。处理1422的特征在于处理内部状态q。内部状态q可以例如包括神经元的膜电压、膜的传导率和/或其它参数。处理1422的特征在于一个或多个学习参数，该参数可以包括突触权重1406、1416、响应产生(发射)阈值、神经元的静态电位和/或其它参数。在一个或多个实现中，参数w可以包括网络的单元(例如，神经元)之间的信号传输可能性。

[0738] 输入信号x(t)可以包括用于解决特定控制任务的数据。在一个或多个实施方式中，例如，涉及机器臂或自主机器人的实施方式中，信号x(t)可以包括原始传感器数据的流(例如，接近度、惯性和/或地域成像)和/或预处理的数据(例如，从加速计中提取的速度、到障碍的距离、和/或位置)。在一些实施方式中，例如包括对象识别的一些实施方式中，信号x(t)可以包括输入图像中的像素值(例如，RGB、CMYK、HSV、HSL和/或灰度)的阵列，或预处理的数据(例如，用于脸部识别的Gabor滤波器的激活程度、轮廓和/或其它预处理的数据)。在一个或多个实现中，输入信号x(t)可以包括例如期望的运动轨迹，以便在当前状态和期望运动的基础上预测机器人的未来状态。

[0739] 在一个或多个实现中，网络1400可以使用高级神经形态描述(HLND)框架来实现，例如在以下申请中所描述的：2012年3月15日提交的、名称为“TAG-BASED APPARATUS AND METHODS FOR NEURAL NETWORKS(用于神经网络的基于标签的装置和方法)”的美国专利申请No.13/385,938，通过引用方式将其整体并入本文。在一个或多个实现中，HLND框架可以扩大到处理本文所述的基于事件的更新方法。

[0740] 神经元处理1422可以被配置为实行基于事件的更新。在一个或多个实现中，基于事件的更新可以包括多个事件类型(例如，树突状、轴突、定制和/或其它事件类型)，其被配置为执行如下面将要详细描述的基于事件的功能的各方面。

[0741] 在一些实施方式中，为了促进基于事件的更新功能，END 2.0实施方式可以按照如下面关于图19所描述的来扩展。在一些实施方式中，END 2.0格式1900包括POST-SYNAPTIC_EVENT(PE)。在扩展的END框架(例如，图19中的END 2.1实现1902)的一个或多个实现中，额外的单个DENRITIC_EVENT可以被执行，同时保留POST-SYNAPTIC_EVENT，以允许对当前的END 2.0格式向后兼容，如由以下将详细描述的图15A-15B的例子所示出的。

[0742] 在一些实施方式中，(例如，图19中的END 2.1实现1904)，可以提供CUSTOM_EVENT API，以便例如处理突触前和/或事件触发的更新，而不产生突触后响应，如上面详细描述的那样。END 2.1实现1904在图15C的列表中示出，下面进行详细描述。END 2.1实现1904向后兼容END 2.0格式。

[0743] 在一些实施方式中，(例如，图19中的END 2.1实现1906)可以移除POST-SYNAPTIC_EVENT，并提供CUSTOM_EVENT API，以便例如处理上述的前和/或后更新。END 2.1实现1906在图15D-15F的列表中示出，下面进行详细描述。虽然END 2.1实现1906不向后兼容当前的END 2.0格式，但是它提供统一且灵活的方式进行事件处理，并可以有利地引起更快的代码执行，这是由于例如处理较少的逻辑条件。然而，开发者需要特别的注意，特别是当实现每

单元的事件的变量数时。

[0744] 单元类型定义可以包括树突状和/或轴突事件条件：

[0745] <unit type>

[0746] ...members of END 2.0unit type definition(e.g.name,

[0747] update_rule)

[0748] dendritic_event_condition

[0749] after_dendritic_event_rule

[0750] 其中，

[0751] dendritic_event_condition

[0752] 定义了被配置成检测树突状事件的逻辑语句，例如，'denritic_event_flag==true'。在一些实施方式中，事件规则可以在各时钟循环(模型时间步长)处执行。当从基类导出类时，则子类的denritic_event_condition替换基类的denritic_event_condition。

[0753] 单元定义成员

[0754] after_dendritic_event_rule

[0755] 可以包括在检测到树突状事件时可以执行的代码。对于导出的子类，子类after_denritic_event_rule可以在基类的after_denritic_event_rule前执行。

[0756] 突触类可以由树突状事件扩大：

[0757] <synaptic type>

[0758] ...members of END 2.0synapse type(e.g.

[0759] presynaptic_event_rule)

[0760] dendritic_event_rule

[0761] 其中，denritic_event_rule的代码通过产生突触后单元的denritic_event来触发。在一些实施方式中，它可以执行经调制的STDP

[0762] 在一些实施方式中，可以定义内部(也称为树突状)事件，以便例如修改神经元1420的输入连接(例如，调整突触权重1406)，如由图14中的虚线箭头1426所示。

[0763] 在一个或多个实现中，树突状事件可以由外部触发器触发。在一个这种实现中，外部触发器可以包括图14中所示的强化信号1408。在一些实施方式中，外部信号1408可以包括由另一网络实体(例如定时器)产生的信号。在一个或多个实现中，信号1408可以包括与缓冲器相关联的溢出警报，所述缓冲器被配置为存储网络的输入和/或输出尖峰活动(例如，尖峰时序)。在2011年9月21日递交的、名称为“APPARATUS AND METHODS FOR SYNAPTIC UPDATE IN A PULSE-CODED NETWORK(用于脉冲编码网络中的突触更新的装置和方法)”的共有的美国专利申请No.13/239,255中描述了缓冲器溢出事件产生的一个实施方式，该申请通过引用方式整体并入本文。

[0764] 在一些实施方式中，树突状事件可以由图14的神经元1420的处理1422内部地触发。在一个或多个实现中，内部事件可以包括内部时间到期、与内部缓冲器相关联的溢出指示符，其中内部缓冲器存储神经元尖峰历史(突触前和/或突触后)。本领域人员应该意识到，内部和/或外部事件触发器的许多源可能存在，其例如与处理1422的状态相关。

[0765] 根据本公开的原理，内部事件可以用于实现突触更新，而不会使得突触后响应传播到目标(例如，图14中的神经元1420_1、1420_2、1420_3)。通过示例的方式，在一个或多个

实现中，外部强化信号1408可以用于基于神经元先前性能来触发一个或多个输入连接1404的可塑性的更新(例如，修改权重1406)。

[0766] 在一个或多个实现中，不具有突触后响应的突触更新可以减少不期望的尖峰活动(例如，噪声)。这种不期望的尖峰通常可能造成设备(例如，电机)故障。更快速的奖励信号(例如，1Hz-100Hz)可以用于与树突状事件协力，以便例如改善电机控制器的学习收敛，而不引起不必要(例如，重复)的电机控制指令。当学习错误足够小时，可以使用轴突事件，以便引起电机控制输出，从而增加设备寿命。

[0767] 在一些实施方式中，内部触发的树突状事件(例如，突触前缓冲器溢出)可以引起突触更新，以便缓冲器内容可以被清除，并且神经元处理1422的完整性得以保持。

[0768] 在一些实施方式中，树突状事件可以引起外部信号连接1408的可塑性更新，如图14中的虚线箭头1424所示。

[0769] 图15A呈现了使用外部触发的内部(即，树突状)事件的可塑性更新的一个示例性实现。图15A的树突状更新实现示出了包括DENDRITIC_EVENT_RULE的END和/或HLND扩展的使用。

[0770] 图15B示出了内部触发的内部(即，树突状)事件的可塑性更新的一个示例性实现。图15B的树突状更新实现示出了包括DENDRITIC_EVENT_RULE的END扩展的使用。图15B的更新规则使用内部条件以便调用规则。

[0771] 图15B的实现示出了自平衡机制，其可以被配置为当神经元活动超过阈值(例如，神经元产生太多尖峰)时，降低兴奋性(降低各权重)。

[0772] 在一些实施方式中，输入连接的可塑性更新可以使用STDP规则来实行，例如，所述STDP规则例如在以下申请中所描述的：2011年6月2日递交的、名称为“SENSORY INPUT PROCESSING APPARATUS AND METHODS(传感输入处理装置和方法)”的共有并且共同未决的美国专利申请序列No.13/152,119；2012年5月7日递交的、名称为“SPIKING NEURAL NETWORK FEEDBACK APPARATUS AND METHODS(尖峰神经网络反馈装置和方法)”的共有并且共同未决的美国专利申请序列No.13/465,924；2012年5月7日递交的、名称为“SENSORY INPUT PROCESSING APPARATUS IN A SPIKING NEURAL NETWORK(尖峰神经网络中的传感输入处理装置)”的共有并且共同未决的美国专利申请序列No.13/465,903；2012年5月7日递交的、名称为“SPIKING NEURAL NETWORK OBJECT RECOGNITION APPARATUS AND METHODS(尖峰神经网络对象识别装置和方法)”的共有的美国专利申请序列No.13/465,918；以及2012年6月4日递交的、名称为“SPIKING NEURON NETWORK APPARATUS AND METHODS(尖峰神经网络装置和方法)”的共有的美国专利申请序列No.13/488,106，通过引用方式将以上申请中的每一个整体并入本文。

[0773] 在一些实施方式中，用于连接更新的可塑性规则可以由外部信号1408的时序调制，例如2012年1月24日发布的美国专利No.8,103,602所描述的。在一个这样的实现中，第j个连接1404的权重1406可以调整如下：

[0774] $$w_j \propto \exp[-(t_{out}-t_{in}^j)]\exp[-(t_{out}-t_{ext})] \quad \text{(等式1)}$$

[0775] 其中：

[0776] t_{out}—指示由神经元产生的最近的输出(突触后尖峰)的时间；

[0777] t_{in}^j—指示由神经元经由连接j接收到最近输入(突触前尖峰)中的一个

[0778] 或多个的时间；

[0779] t_{ext}—指示外部(触发器)信号的时间。

[0780] 在一个或多个实现中，可以使用树突状事件以便促进通过网络进行的强化学习。通过示例的方式，在接收到强化输入(树突状尖峰)时，可以调整神经元的输入突触的可塑性，而不向突触后目标传送突触后响应。随后，神经元状态达到超阈值条件，轴突事件可以用于向期望的目的地传送神经元突触后响应。

[0781] 在一些实施方式中，内部事件可以向处理的状态(例如图14中的处理1422的电压)提供输入连接(例如，连接1404)的读访问。

[0782] 在一些实施方式中，在内部事件的执行期间，可以对输入连接提供对缓冲器内容的读访问，所述缓冲器用于存储神经元的突触前和/或突触后活动历史。

[0783] 图15A的实施方式示出了例如用于机器人控制应用中的强化学习的经调制的STDP。

[0784] 在一些实施方式中，可以定义外部(也称为轴突)事件，以便例如向期望的目标(神经元1420_1)传送神经元(例如，图14的神经元1420)的输出(突触后尖峰)，而不必更新输入连接1404。在一个或多个实现中，轴突事件可以由神经元尖峰响应处理(例如，超阈值条件)触发。

[0785] 在随机神经元的一些实施方式中，当以下各项发生时可以触发轴突事件：(i)随机数生成器的输出大于阈值；和/或(ii)当时间计数器超过某值，该值可能例如与最大静止间隔(例如，当神经元静默超过100ms时)相关联。

[0786] 在一些实施方式中，轴突事件可以引起输出1416的可塑性更新，如图14中的虚线箭头1426所示。

[0787] 在一些实施方式中，内部事件和外部事件可以组合成单个复合事件，以便实行可塑性更新和突触后输出传送，从而提供对常规更新事件的兼容性，如以上由END v1、END 2所描述的。

[0788] 在一些实施方式中，在树突状事件的执行期间，可以暂停轴突事件的执行。

[0789] 在一些实施方式中，树突状事件可以设置计数器，并且轴突可以检查该计数器，以便触发尖峰传送事件。

[0790] 在一些实施方式中，神经元处理(例如，处理1422)可以包括两种类型的事件：树突状事件(其触发可塑性更新)和公共END尖峰事件，其调用突触更新和尖峰传送。

[0791] 在一些实施方式中，单元描述可以包括两种类型的事件：(i)"树突状"尖峰，其影响输入突触的可塑性，但不会导致任何信令沿着轴突向突触后目标传播；和(ii)"轴突"尖峰，其向突触后目标传播，但不会对神经元上的输入突触做任何事。用在END 1、2、3中的标准"尖峰"，仅在树突状尖峰和轴突尖峰的组合中。

[0792] 图15C呈现了使用外部触发的外部(即，树突状)事件的可塑性更新的一个示例性实施方式。图15C的实施方式可以包括神经元响应条件(例如，方法尖峰())。明确定义的AxonicEvent(图15C的行28-40)可以用于使尖峰向期望目标(OutputNeuron)传播。

[0793] 在一些实施方式中，END格式可以如下地扩展。单元类型定义可以包括树突状和/或轴突事件条件：

<unit class>

unit _class _id

initialization

execution _condition

[0794] update _rule

list of

custom _event _type _id

custom _event _condition

after _custom _event _rule

[0795] 其中，

[0796] 突触类可以由树突状事件增大：

<synaptic class>

synapse _class _id

initialization

presynaptic _event _rule

[0797]

update _rule

list of

custom _event _type _id

custom _event _rule

[0798] 其中，具有某个ID的定制事件规则的代码可以通过产生具有相同ID的突触后单元的定制事件来触发。突触可以定义具有不同ID的若干事件规则。在一些实施方式中，它可以执行经调制的STDP。

[0799] 在一个或多个实现中，当使用事件HLND扩展和END框架来定义尖峰网络时，用户可以定义定制事件。在一些实施方式中，定制事件可以用于触发输入连接1404的一部分的更新。通过示例的方式，可以使用这种定制事件来更新向神经元提供一个类型输入(例如，臂位置)的连接。在一个或多个实现中，剩余的连接(例如，提供虚拟输入的连接)可以根据不同的可塑性机制来操作，并不会由相同的定制事件来更新。

[0800] 在一些实施方式中，用户可以定义多个事件和可以由事件调用的规则。这样，用户可以定义多种类型的事件，它们中的一些可以用作`树突状'事件，另一些可以用作"轴突"事件，还有一些可以用作"绿"、"蓝"和/或其它事件。

[0801] 图15D-E呈现了使用定制事件的可塑性更新的示例性实施方式。图15D的实施方式示出了规则触发器条件(例如，状态参数v>1)。图15D的实施方式可以包括基于状态变量和响应活动的多个规则触发器条件。

[0802] 图15F呈现了包括多个定制事件的可塑性更新的示例性实施方式。图15F的实施方

式，神经元的突触可以被配置为产生突触后神经元的TeacherEvent(教导事件)，而不定义具有特殊功能Unit.Event(事件ID)的条件标志(如图15A所示)。在一些实施方式中，列表(例如，图15F的列表)可以包括一个或多个用户定义的标签。

[0803] 在一些实施方式中，END格式可以如下扩展。单元类型定义可以包括树突状和/或轴突事件条件：

<unit class>
unit _class _id
initialization
execution _condition
update_rule

[0804]

event _condition
after _event _rule
list of
custom _event _type _id
custom _event _condition

[0805] after _custom _event _rule

[0806] 其中，

[0807] 突触类可以由树突状事件扩大：

<synaptic class>
synapse _class _id
initialization
presynaptic _event _rule

[0808]

post-synaptic _event _rule
update_rule
list of
custom _event _type _id
custom _event _rule

[0809] 其中，具有某个ID的定制事件规则的代码可以由产生具有相同ID的突触后单元的定制事件来触发。突触可以定义具有不同ID的若干事件规则。要注意的是，突触类仍然具有post-synaptic_event_rule，其由向后兼容END 2.0的普通单元(不是突触后单元中的定制事件规则)触发。在一些实施方式中，它可以执行经调制的STDP。

[0810] 图15G中的列表示出了网络更新，其包括被配置为不执行不会引起神经元输出的突触更新的树突状事件。

[0811] 图15H中的列表示出了定制事件的使用，其包括被配置为执行突触更新的CUSTOM_EVENT_RULE方法。

[0812] 图16A-16C示出了用于实现本公开的事件驱动的更新的示例性方法。在一些实施方式中，可以使用图16A-16C的方法来例如用于操作图14中的神经元1420。另外，图16A-16C的方法可以在网络装置中实现(例如，图11C的装置1130)。图16A-16C的方法可以在尖峰神经元网络的传感处理中实现，例如，举例来说，以下关于图17所描述的传感处理装置的网络，从而在处理传感输入时，尤其有利地辅助强化学习。

[0813] 现在返回图16A，在方法1600的步骤1602处，可以执行关于事件是否被触发的检测。当检查到时，可以在步骤1604确定事件类型。

[0814] 当检测到内部(即，树突状)事件时，方法进行到步骤1606，在该步骤处，可以选择用于支持所检测到的事件类型的一个或多个输入连接。在一个或多个实现中，可以由一个或多个输入连接来支持一个或多个不同的树突状事件类型。

[0815] 在步骤1608，可以针对在步骤1606处选择的一个或多个输入连接来调用适当的内部事件规则。

[0816] 当检测到外部(即，轴突)事件时，方法可以进行到步骤1610，在此处，可以选择用于支持所检测到的输出事件类型的一个或多个输出目标。

[0817] 在步骤1612，可以针对在步骤1610处选择的一个或多个目标来调用适当的外部事件规则(例如，图15A中的PRESYNAPTIC规则)。

[0818] 图16B示出了例如与上述的图16A的方法1600一起使用的内部事件规则的一个示例性实施方式。

[0819] 在方法1620的步骤1622，可以选择连接更新的类型。在一个或多个实现中，网络(例如，图14的网络1400)可以支持一个或多个不同的可塑性规则。

[0820] 在一些实施方式中，可塑性规则可以包括基于神经元的状态(例如，图14中的神经元的膜电压)的权重调整，如由方法1620的步骤1624处的权重调整所示。

[0821] 在一些实施方式中，可塑性规则可以包括权重调整，其基于由连接传送的输入尖峰之间的定时(t_{in})和由神经元(即，STDP规则)产生的输出响应(输出尖峰)的定时t_{out}，如由方法1620的步骤1626处的权重调整所示。

[0822] 在一些实施方式中，可塑性规则可以包括权重调整，其基于由连接传送的输入尖峰之间的定时(t_{in})、由神经元(即，STDP规则)产生的输出响应(输出尖峰)的定时t_{out}和与树突状事件相关联的触发器的定时，如由方法1620的步骤1628处的权重调整所示。在一个或多个实现中，树突状事件触发器可以包括经由图14中的连接1408传送到神经元1420的外部信号(例如，强化尖峰)。在一些实施方式中，步骤1628的可塑性可以使用等式2的方法来实行。

[0823] 在步骤1630，可以对连接(例如，在图16A中的步骤1606处选择的连接)施加适当的调整。

[0824] 图16C示出了与上述图16A的方法1600一起使用的外部事件规则的一个示例性实现。

[0825]]在方法1640的步骤1642，可以针对一个或多个目标(例如，图14中的神经元1420_1、1420_2、1420_3)来选择输出传送规则。在一个或多个实现中，网络(例如，图14中的网络

1400)可以支持一个或多个不同的输出(例如,尖峰传送)规则。

[0826] 在一个或多个实现中,可以在编译时间经由HLND框架来实行目标选择。

[0827] 在一些实施方式中,响应传送规则可以包括向合适的目标传播神经元突触后响应,如由方法1640的步骤1644的规则所示。在一个或多个实现中,突触后响应传播包括向目标施加输出信号(例如,到电机的驱动电流,或到电机控制器的控制信号(例如,脉冲-带宽调制的电压))。

[0828] 在一些实施方式中,响应传送规则可以包括传递权重调整,所述权重调整基于由连接传送的输入尖峰之间的定时(t_{in})和由神经元(即,STDP规则)所产生的输出响应(输出尖峰)的定时t_{out},如由方法1620的步骤1626处的权重调整所示。

[0829] 在一些实施方式中,响应传送规则可以包括传递权重调整,所述权重调整基于由连接传送的输入尖峰之间的定时(t_{in})、由神经元(即,STDP规则)所产生的输出响应(输出尖峰d)的定时t_{out}和与树突状事件相关联的触发器的定时,如由方法1620的步骤1628处的权重调整所示。在一个或多个实现中,树突状事件触发器可以包括经由图14中的连接1408传送到神经元1420的外部信号(例如,强化尖峰)。在一些实施方式中,步骤1628的可塑性可以使用等式3的方法来实行。

[0830] 在步骤1630,可以对连接(例如,在图16A中的步骤1606处选择的连接)施加适当的调整。

[0831] 在一些实施方式中(未示出),可塑性规则可以包括连接延迟和/或传输可能性调整。

[0832] 现在关于图17和图11A-11D来描述用于实现本文所阐述的一个或多个方法(例如,使用以上所述的示例性的事件驱动的更新机制)的各种示例性尖峰网络装置。

[0833] 图17示出了用于使用尖峰神经网络(包括本文所述的一个或多个事件驱动的更新机制)来处理传感信号信息(例如,视觉的、听觉的、体觉的)的一种装置。所示出的处理装置1700可以包括被配置为接收输入传感信号1720的输入接口。在一些实施方式中,该传感输入可以包括进入图像传感阵列(包括RGC、电荷耦合装置(CCD)、CMOS装置或有源像素传感器(APS))的电磁波(例如,可见光、IR、UV和/或其它波长)。在这种情况下的输入信号是经由接收机装置从CCD或CMOS相机接收到的图像(图像帧)序列。所述图像可以包括以24Hz帧速率刷新的二维RGB值阵列。本领域技术人员应该理解,上述图像参数和组件仅仅是示例性的,并且许多其它的图像表示(例如,位图、CMYK、灰度和/或其它图像表示)和/或帧速率可以与本技术一起等效地使用。

[0834] 本装置1700可以包括编码器1724,其被配置为转移(编码)输入信号,以形成编码信号1726。在一些实施方式中,编码信号可以包括被配置为对神经元行为进行建模的多个脉冲(也称为脉冲组)。该编码信号1726可以从编码器1724经由多个连接(也称为传输信道、通信信道或突触连接)1704传输到一个或多个神经元节点(也称为检测器)1702。

[0835] 在图17的实现中,同一级层的不同检测器可以用"_n"指示符表示,从而例如指示符1702_1表示层1702的第一检测器。尽管在图17中出于清楚的目的仅示出了两个检测器(1702_1、1702_n),但是应该理解,编码器能耦合到任意数量的检测器节点,该编码器可以与检测装置硬件和软件限制兼容。单个检测器节点可以耦合到任何实际数量的编码器。

[0836] 在一个实施方式中,各检测器1702_1、1702_n可以包含被配置为识别编码信号

1704中预定的脉冲模式的逻辑(其可以实现为软件代码、硬件逻辑或其组合),使用例如在以下专利中所描述的任何机制:2010年8月26日递交的、并且名称为“SYSTEMS AND METHODS FOR INVARIANT PULSE LATENCY CODING(用于不变量脉冲延迟编码的系统和方法)”的美国专利申请No.12/869,573;2010年8月26日递交的、并且名称为“INVARIANT PULSE LATENCY CODING SYSTEMS AND METHODS(不变量脉冲延迟编码系统和方法)”的美国专利申请No.12/869,583;2011年5月26日递交的、并且名称为“APPARATUS AND METHODS FOR POLYCHRONOUS ENCODING AND MULTIPLEXING IN NEURONAL PROSTHETIC DEVICES(用于神经元假肢装置中的多同步编码和复用的装置和方法)”的美国专利申请No.13/117,048;2011年6月2日递交的、并且名称为“APPARATUS AND METHODS FOR PULSE-CODE INVARIANT OBJECT RECOGNITION(用于脉冲编码不变量对象识别的装置和方法)”的美国专利申请No.13/152,084,它们中的每一个都通过引用整体并入本文,以产生通过通信信道1708发送的突触后检测信号。在图17中,指示符1708_1、1708_n分别指示检测器1702_1、1702_n的输出。

[0837] 在一个实施方式中,检测信号可以传送到检测器1712(包括检测器1712_1、1712_m、1712_k)的下一层,用于识别复杂对象特征和对象,这类似于在共同拥有并未决的2011年6月2日递交的、名称为“APPARATUS AND METHODS FOR PULSE-CODE INVARIANT OBJECT RECOGNITION(用于脉冲编码不变量对象识别的装置和方法)”的美国专利申请No.13/152,084中所描述的示例性配置,该专利申请通过引用整体并入本文。在该配置中,检测器的各后续层可以被配置为从先前的检测器层接收信号,并检测更多的复杂特征和对象(与由先前的检测器层检测的特征相比)。例如,一组边界检测器之后可以是一组条状检测器,之后是一组角落检测器等等,从而使得装置能够进行字母识别。

[0838] 检测器1702中的各个检测器可以输出通信信道1708_1、1708_n(具有适当的延迟)上的检测(突触后)信号,其利用不同的传导延迟传播到检测器1712。图17装置的检测器级联可以包含任意实际数量的检测器节点和检测器库,这由检测装置的软件/硬件源和所检测到的对象的复杂度来确定。

[0839] 在一些实施方式中,装置1700可以被配置为经由连接1734向一个或多个神经元1702、1712提供外部输入。在一个或多个实现中,输入1734包括训练输入,例如,正和/或负强化信号,如以上关于图14所描述的。

[0840] 图17所示的信号处理装置实施方式可以包括横向连接1706。在一些实施方式中,连接1706可以被配置为在同一层级的相邻神经元之间传送突触后活动指示,如由图17中的连接1706_1所示。在一些实施方式中,相邻神经元可以包含具有重叠输入的神经元(例如,图17中的输入1704_1、1704_n),以便神经元可以进行竞争从而不会学习相同的输入特征。在一个或多个实现中,相邻神经元可以包括例如在3-维(3D)和/或2-维(2D)空间上彼此处于某体积/区域内的空间近似神经元。

[0841] 装置1700还可以包括反馈连接1714,其被配置为从一个层级的层内的检测器向先前的层传送上下文信息,如由图17中的反馈连接1714_1所示。在一些实施方式中,反馈连接1714_2可以被配置为向编码器1724提供反馈,从而促进传感输入编码,如在共同申请并未决的、2011年6月2日递交的、名称为“APPARATUS AND METHODS FOR PULSE-CODE INVARIANT OBJECT RECOGNITION(用于脉冲编码不变量对象识别的装置和方法)”的美国专利申请No.13/152,084中所描述的,该申请在上文中被包括。

[0842] 计算机化的神经形态系统

[0843] 计算机化神经形态的处理系统的一个具体实施方式适于操作计算机化尖峰网络(并实现上述的示例性事件驱动的更新方法),在图11A中示出。图11A的计算机化系统1100包括输入接口1110,例如,图像传感器、计算机化尖峰视网膜、音频阵列、触摸传感输入设备和/或其它组件。输入接口1110可以经由输入通信接口1114耦合到处理模块(例如,单个或多个处理器模块)。系统1100可以包括随机存取存储器(RAM)1108,其被配置为存储神经元状态和连接参数(例如,图14中的权重1406),并促进突触更新。在一些实施方式中,突触更新可以根据在例如以下申请中所提供的描述来执行:2011年9月21日递交的、名称为"APPARATUS AND METHODS FOR SYNAPTIC UPDATE IN A PULSE-CODED NETWORK(用于脉冲编码网络中的突触更新的装置和方法)"的美国专利申请No.13/239,255,其通过引用并入上文。

[0844] 在一些实施方式中,存储器1108可以经由直接连接(存储器总线)1116耦合到处理器1102。存储器1108可以经由高速处理器总线1112耦合到处理器1102。

[0845] 系统1100包括非易失性存储装置1106,其尤其包括计算机可读指令,所述指令被配置为实现尖峰神经元网络操作(例如,传感输入编码、连接可塑性、神经元的操作模型和/或尖峰神经元网络操作的其它方面)的各方面。非易失性存储装置1106可以用于例如当出现以下情况时存储神经元和连接的状态信息:存储/加载网络状态快照,或实现上下文交换时(例如,保存当前网络配置(尤其包括连接权重和更新规则、神经元状态和学习规则和/或其它信息)以便之后使用,以及加载先前存储的网络配置。

[0846] 在一些实施方式中,计算机化的装置1100可以经由I/O接口1120耦合到一个或多个外部处理/存储/输入装置,所述接口例如计算机I/O总线(PCI-E)、有线(例如,以太网)或无线(例如,Wi-Fi)网络连接。

[0847] 在一些实施方式中,装置1100可以被配置为向操作神经元网络的处理器提供外部输入。在一个或多个实现中,输入可以包括训练输入,例如,经由接口1114提供的正和/或负强化信号。

[0848] 在一些实施方式中,输入/输出接口包括被配置成接收和识别用户命令的语音输入(例如,麦克风)和语音识别模块。

[0849] 本领域技术人员应该意识到,各处理装置可以与计算机化系统1100一起使用,其包括但不限于:单核/多核CPU、DSP、FPGA、GPU、ASIC、其组合,和/或其它处理器。各用户输入/输出接口可以类似地适用于本技术的实施方式,该实施方式包括例如:LCD/LED监控器、触摸屏输入和显示设备、语音输入设备、手写笔、光电笔、跟踪球等。

[0850] 现在参见图11B,详细描述了神经形态计算机化系统的一个实施方式,其被配置为实现尖峰网络中的事件驱动的更新机制。图11B的神经形态处理系统1130可以包括多个处理块(微块)1140,其中,各微块可以包括计算逻辑内核1132和存储块1134。逻辑内核1132可以被配置为实现神经元节点操作的各方面(例如,节点模型),以及突触更新规则(例如,I-STDP)和/或与网络操作相关的其它任务。存储块可以被配置为存储(除了其它之外)神经元状态变量和连接1138的连接参数(例如,权重、延迟、I/O映射)。

[0851] 微块1140可以使用连接1138和路由器1136彼此互联。如本领域技术人员将意识到的,图11B中的连接布局是示例性的,并且许多其它的连接实施方式(例如,一对所有、所有

对所有，和/或其它连接实施方式)可以与本公开内容兼容。

[0852] 神经形态装置1130可以被配置为经由接口1142接收输入(例如，视觉输入)。在一个或多个实现中，适用于例如与计算机化的尖峰视网膜或图像阵列交互，装置1130可以经由接口1142提供反馈信息，以促进输入信号的编码。

[0853] 神经形态装置1130可以被配置为经由接口1144来提供输出(例如，所识别对象或特征的指示，或电机指令，例如，以缩放/跨越图像阵列)。

[0854] 在一些实施方式中，装置1130可以被配置为向一个或多个块1132提供外部输入。在一个或多个实现中，外部信号可以包括经由接口1142向一个或多个块提供的训练输入。

[0855] 在一个或多个实现中，装置1130可以经由带宽宽的存储器接口1148来接入外部快速响应存储器(例如，RAM)，从而能够存储中间网络操作参数(例如，尖峰时序)。装置1130可以经由较低带宽的存储器接口1146来接入外部慢速存储器(例如，闪存或磁(硬驱动))，以便促进程序加载、操作模式改变和重定向，其中，用于当前任务的网络节点和连接信息可以被保存用于将来使用和被排出(flushed)，并且预先存储的网络配置可以载入其位置。

[0856] 图11C示出了共享总线神经形态计算机化系统的实现，该系统包括耦合到共享互连的微块1140，其关于图11B在上文中被描述。图11C的装置1145采用一个(或多个)共享总线1146，以使微块1140彼此互连。

[0857] 图11D示出了基于细胞的神经形态计算机化系统架构的一个实施方式，其被配置为实现尖峰网络中的事件驱动的更新机制。图11D的神经形态系统1150包括处理块(细胞块)的层级。在一些实施方式中，装置1150的最低层L1细胞1152包括逻辑和存储器，并可以类似于图11B中所示的装置的微块1140进行配置。数个细胞块可以设置成集群，并利用本地互联1162、1164彼此通信。各集群可以形成高层级细胞，例如，细胞L2，其在图11d中被表示为1154。类似地，若干个L2集群可以经由第二及互连1166彼此通信，并形成超集群L3，其在图11D中被指示为1156。超集群1154可以例如经由第三级互联1168来通信，并可以形成下一级集群等等。本领域技术人员应该意识到，装置1150的层级结构包括每一级给定数量(例如，4个)的细胞，这仅仅是一个示例性实现，并且其它实现可以每一级包括更多或更少的细胞，和/或更少或更多层级，以及架构的其它类型。

[0858] 图11D的示例性装置1150的不同细胞层级(例如，L1、L2、L3)可以被配置为执行具有各种层级的复杂性的功能。在一个实施方式中，不同的L1细胞可以并行处理视觉输入的不同部分(例如，编码不同的框架宏块)，L2、L3细胞执行不断增加的高级功能(例如，边缘检测、对象检测)。不同的L2、L3细胞可以执行操作例如机器人的不同方面，其中，一个或多个L2/L3细胞处理来自相机的视觉数据，而其它L2/L3细胞操作电机控制块，用于当例如追踪对象时执行透镜运动，或执行透镜稳定功能。

[0859] 神经形态装置1150可以经由接口1160接收输入(例如，视觉输入)。在适于例如与计算机化尖峰视网膜和/或图像阵列交互的一个或多个实现中，装置1150可以经由接口1160提供反馈信息，以促进输入信号的编码。

[0860] 神经形态装置1150可以经由接口1170提供输出(例如，所识别对象或特征的指示，或电机指令，例如，以缩放/跨度图像阵列)。在一些实施方式中，装置1150可以使用单个I/O块(未示出)来执行I/O功能中的一些或所有。

[0861] 在一些实施方式中，装置1150可以被配置为向一个或多个块1152、1154、1156提供

外部输入。在一个或多个实现中，外部信号可以包括由装置经由接口1160接收到的训练输入。

[0862] 在一个或多个实现中，装置1150的一个或多个部分可以被配置为操作一个或多个学习规则，例如在所拥有的以下申请中所描述的：2012年6月4日递交的、名称为“DYNAMICALLY RECONFIGURABLE STOCHASTIC LEARNING APPARATUS AND METHODS(动态可重配置随机学习装置和方法)”的美国专利申请序列No.13/487,576，其过引用方式将其整体并入本文。在一个这样的实现中，一个块(例如，L3块1156)可以用于处理经由接口1160接收到的输入，并经由间隔互连1166、1168向另一块(例如，L2块1156)提供强化信号。

[0863] 在一个或多个实现中，装置1150可以经由宽带宽的存储器接口(未示出)接入外部快速响应存储器(例如，RAM)，从而能够存储中间网络操作参数(例如，尖峰时序和/或其它参数)。装置1150可以经由较低带宽的存储器接口(未示出)接入外部慢速存储器(例如，闪存或磁盘(硬驱动))，以促进程序加载、操作模式改变和重定向，其中，用于当前任务的网络节点和连接信息可以保存起来用于以后使用和排出，并且先前存储的网络配置可以加载到其位置。

[0864] 本公开内容的事件驱动的更新机制尤其有利地实现灵活的网络更新。在一些实施方式中，树突状事件可以用于响应于奖赏信号来及时地执行强化学习应用中输入连接的可塑性更新。在一些实施方式中，轴突事件可以用于执行向期望目标和/或外部连接更新的突触后响应传送。在一些实施方式中，定制事件可以启用定制更新规则，例如，举例来说，更新神经元连接的一部分，例如，使用连接的树突状更新，其连接到机器人装置的电机控制部分，同时允许其它连接(例如，处理视觉传感器数据)根据标准尖峰响应事件进行更新。

[0865] 与现有技术的方法相比，本公开内容的例如树突状和轴突事件的使用可以从突触后响应传送中解耦合可塑性更新。因此，在神经元接收到教导尖峰时，输入突触的权重就可以改变，而不影响内部状态(兴奋性)和/或神经元的输出。

[0866] 这种功能可以有利地实现网络学习的更准确的控制。通过示例的方式，在一个电机控制应用中，可以采用频繁(例如，1-100Hz)的树突状事件，以便提高控制器重新学习的速度。一旦控制器性能达到期望的等级(例如，以性能测量为特征)，就可以使用轴突事件以便向电机发送电机命令。奖赏驱动的更新和性能驱动的响应的分离可以有利地消除对设备(例如电机控制器)的过度网络输出，以增加设备的寿命。

[0867] 本领域技术人员应该意识到，以上实现是示例性的，并且本技术的框架可以等效地兼容，并适用于处理其它信息，例如，举例来说，使用数据库的信息分类，其中，特定模式的检测可以识别为类似于尖峰的分立信号，并且基于先前检测的历史，以类似于示例性尖峰神经网络的操作的方式，其它模式的重合检测影响特定模式的检测。

[0868] 有利地，本发明的示例性实现在各种设备中可能是有用的，其包括但不限于假肢设备、自主装置和机器人装置和需要传感处理功能的其它机电设备。这种机器人设备的示例包括制造机器人(例如，汽车)、军队、医疗(例如，显微镜、x射线、超声图记录仪、X线断层摄影术的处理)中的一个或多个。自主交通工具的示例包括巡航船、无人驾驶的空中交通工具、水下交通工具、智能家居设备(例如，ROOMBA®)和/或其它机器人设备。

[0869] 本公开内容的原理的实现适用于各种静态和便携式设备中的视频数据压缩和处理，例如，举例来说，智能电话、便携式通信设备、笔记本、上网本和平板计算机，侦查照相机

系统和实际上被配置成处理视觉数据的任何其它计算机化设备。

[0870] 本公开内容的原理的实现可以适用于宽的应用种类，包括人机交互(例如，手势、声音、姿态、面部的识别和/或其它应用)、控制处理(例如，工业机器人、自主交通工具和其它交通工具)、增强现实应用、信息的组织(例如，用于索引图像和图像序列的数据库)、访问控制(例如，基于手势开门、基于对授权人的检测来打开访问通路)、检测事件(例如，用于视觉监视、或人或动物的计数、追踪)、数据输入、金融交易(基于人或特殊支付符号的识别的支付处理)和许多其它应用。

[0871] 有利地，本公开内容可以用于简化与运动估计相关的任务，例如，其中，可以处理图像序列，以产生对在图像中或在3D场景中的各点处的对象位置的估计(以及因此的速率)，或者甚至对产生该图像的照相机的估计。这种任务的示例可能是：自身运动，即，根据照相机产生的图像序列来检测照相机的三维刚性运动(旋转和平移)；跟随图像序列中的感兴趣的点或对象集(例如，车辆或人)的运动并参考图像平面。

[0872] 在一些实施方式中，对象识别系统的一部分可以嵌入远程服务器，包括存储计算机可执行指令的计算机可读装置，该指令被配置为针对各应用程序执行数据流中的模式识别，例如，科学的、地理的开发、监控、导航、数据挖掘(例如，基于内容的图像获取)。在给出了本公开内容的情况下，本领域技术人员将会意识到存在无数的其它应用。

[0873] 应该意识到，虽然本技术的某些方面可以关于方法的步骤的特定序列进行描述，但是这些描述仅仅示出了本技术的较宽方法，并可以按照特定应用所需进行修改。在某些情况下，某些步骤可能被不必要地放弃或可选的。此外，某些步骤或功能可以加入到所公开的实现中，或者对两个或多个步骤的执行改变序列。所有这些变形都被认为应该包括在本文公开和声明的技术内。

[0874] 虽然以上详细的描述已经示出、描述和指出了适用于各种实现的本技术的新颖特征，但是应该理解，在不脱离本技术范围的情况下，本领域普通技术人员可以做出对所示出的设备或过程的形式以及细节上的各种省略、替换和改变。前述描述是当前预期的用于实现本技术的最佳模式。该描述绝不意味着限制，而是应该视为本技术的一般原理的示例。本技术的范围应该参照权利要求书来确定。

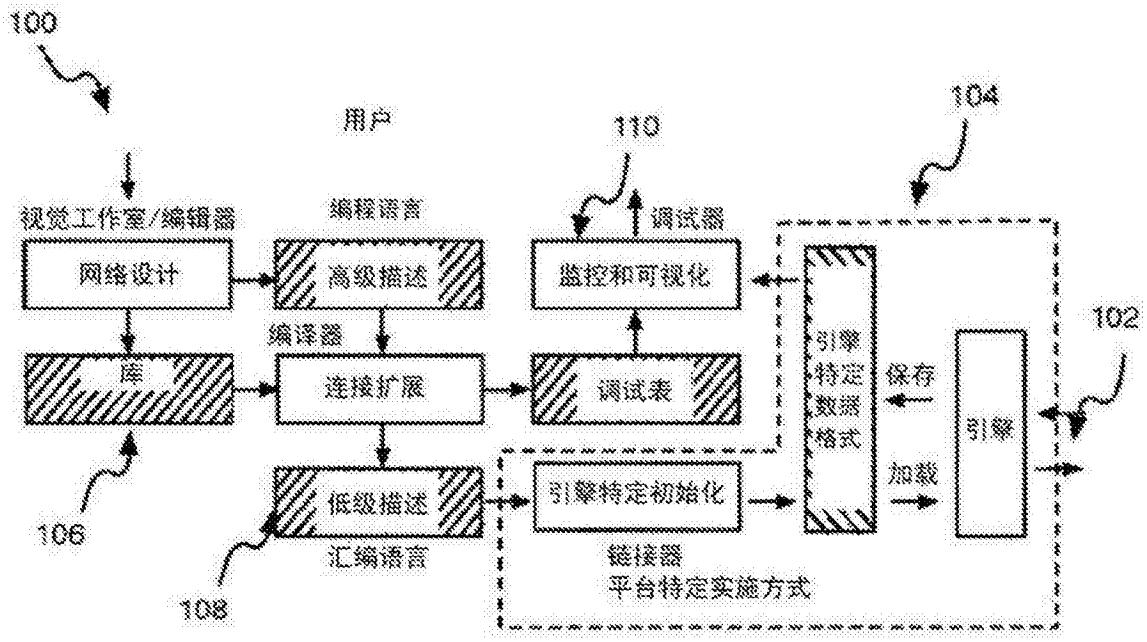

100
用户
104
110
视觉工作室/编辑器
编程语言
调试器
网络设计
高级描述
监控和可视化
编译器
102
库
连接扩展
调试表
引擎特定数据格式
保存
引擎
106
低级描述
引擎特定初始化
加载
汇编语言
链接器
108
平台特定实施方式

图1

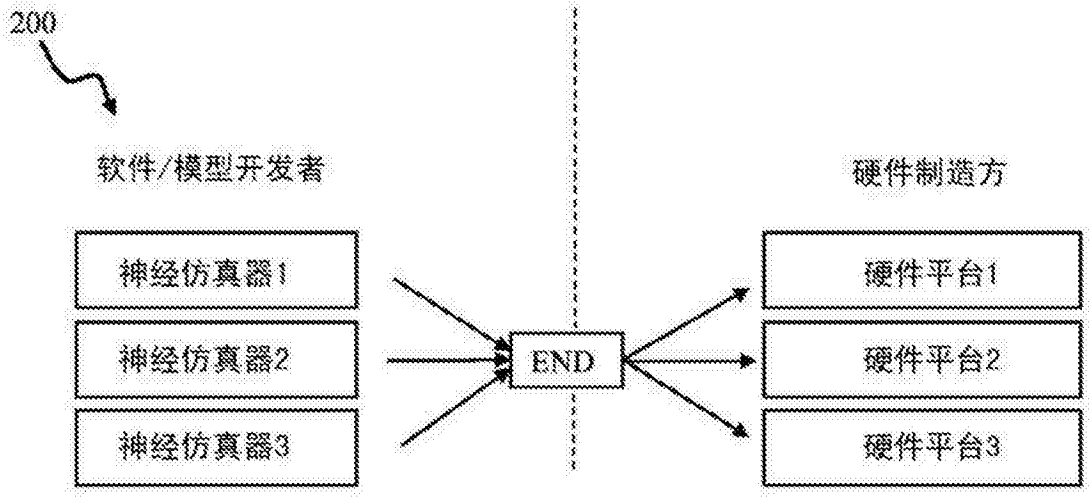

200
软件/模型开发者
硬件制造方
神经仿真器1
神经仿真器2
神经仿真器3
END
硬件平台1
硬件平台2
硬件平台3

图2

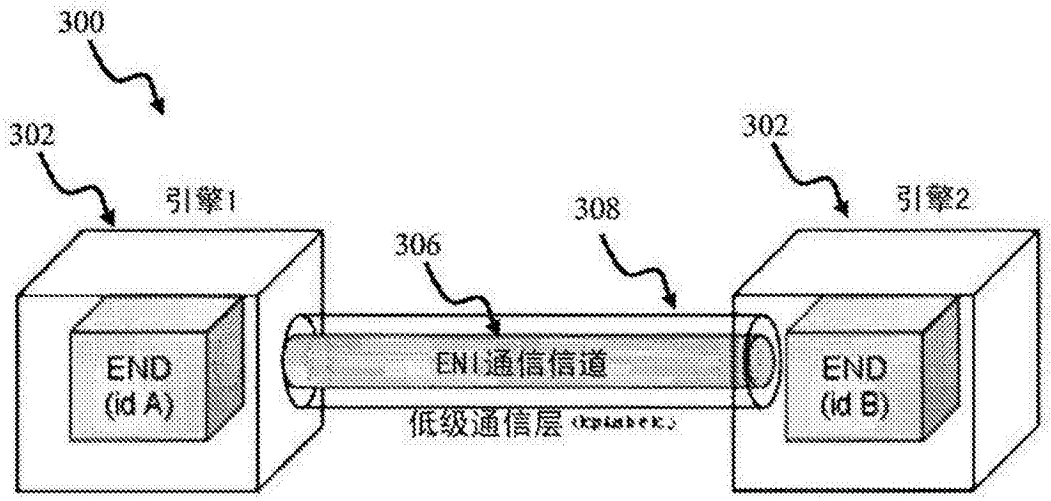

300
302
引擎1
302
引擎2
308
306
END
(id A)
ENI通信信道
END
(id B)
低级通信层

图3

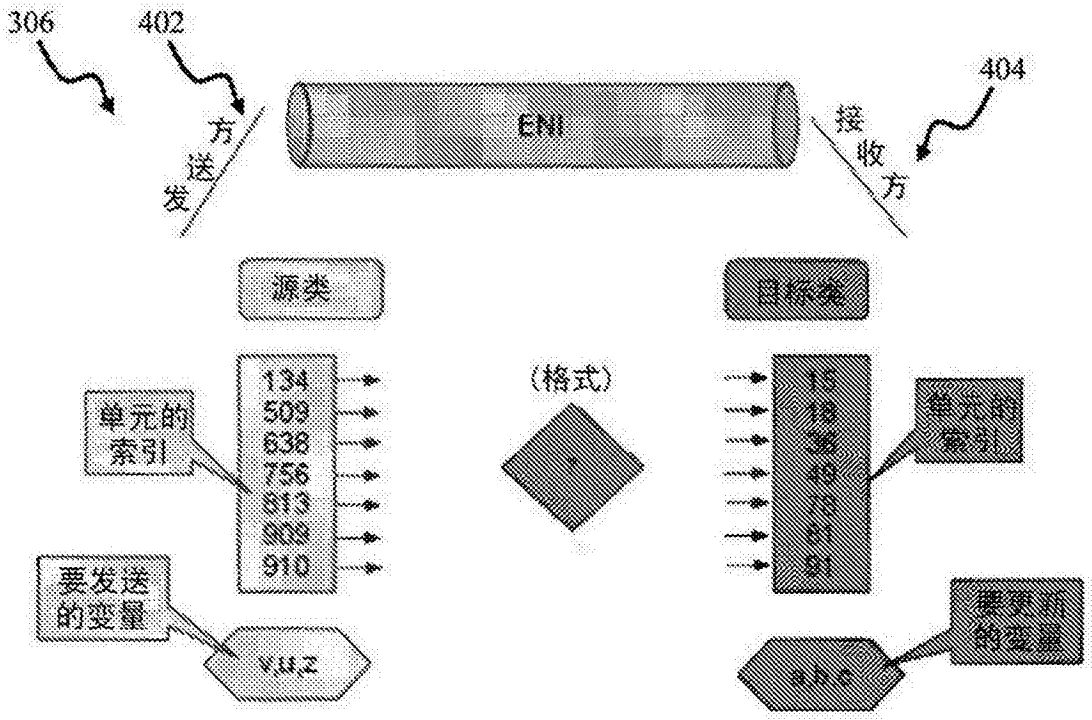

306
402
404
方
送
发
ENI
接
收
方
源类
目标类
(格式)
134
509
638
756
813
903
910
单元的
索引
要发送
的变量
v,u,z
单元的
索引
要更新
的变量
a,b,c

图4

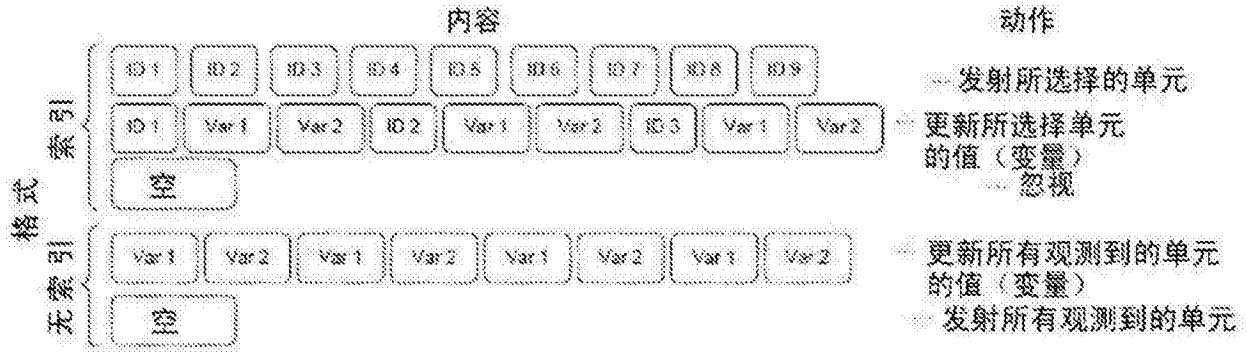

内容
动作
格式
索引
无索引
ID 1
ID 2
ID 3
ID 4
ID 5
ID 6
ID 7
ID 8
ID 9
Var 1
Var 2
空
发射所选择的单元
更新所选择单元的值（变量）
忽视
更新所有观测到的单元的值（变量）
发射所有观测到的单元

图5

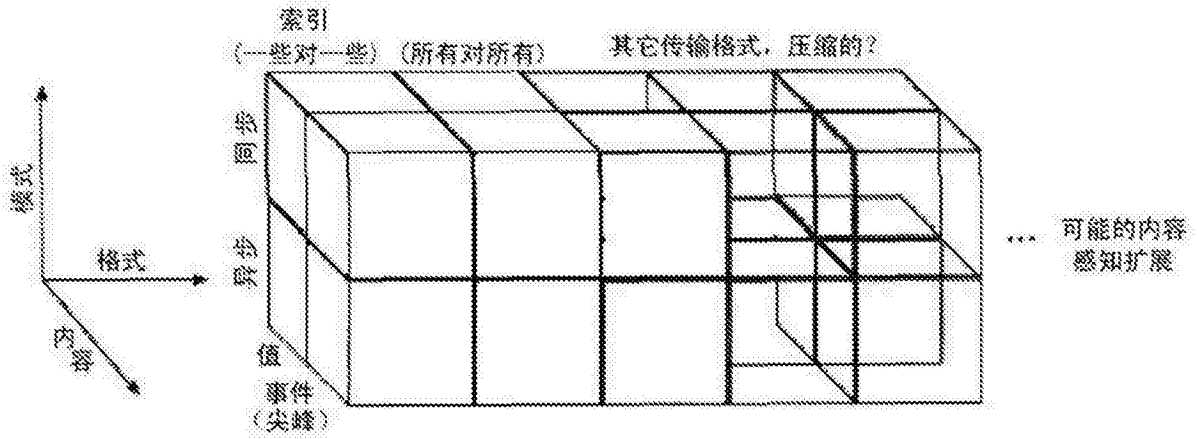

索引
（一些对一些）（所有对所有）
其它传输格式，压缩的？
模式
格式
内容
空间
时间
值
事件
（尖峰）
可能的内容感知扩展

图6

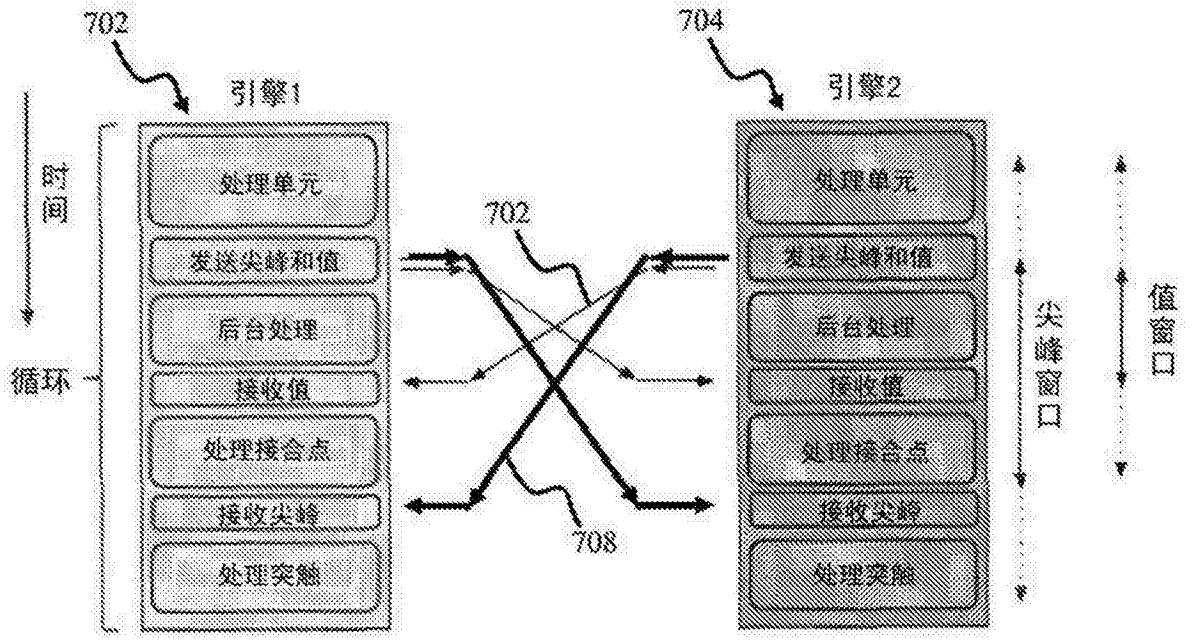

702
引擎1
704
引擎2
时间
循环
处理单元
发送尖峰和值
后台处理
接收值
处理接合点
接收尖峰
处理突触
702
708
尖峰窗口
值窗口

图7A

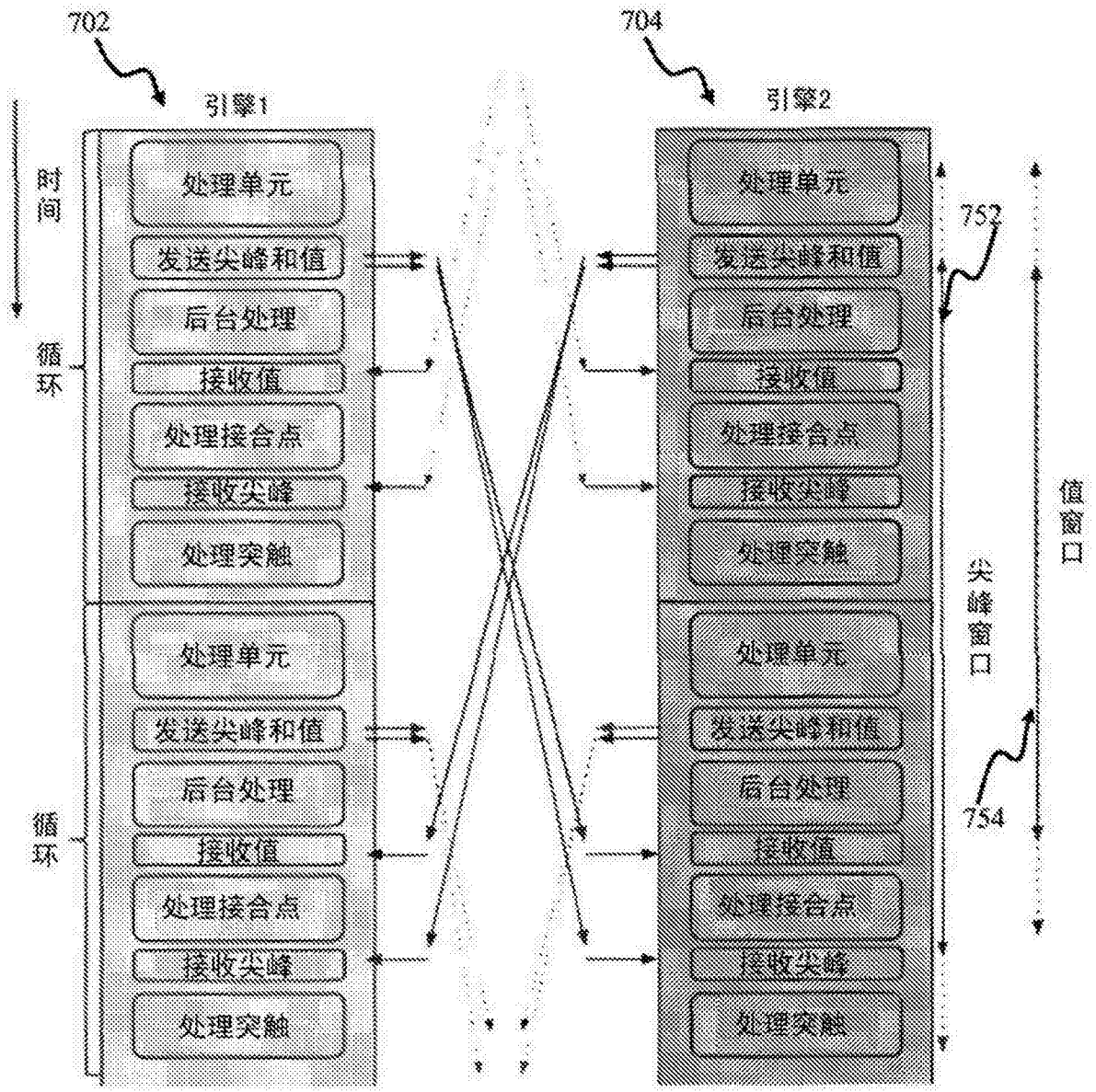

702
704
引擎1
引擎2
时间
循环
循环
处理单元
发送尖峰和值
后台处理
接收值
处理接合点
接收尖峰
处理突触
752
754
值窗口
尖峰窗口

图7B

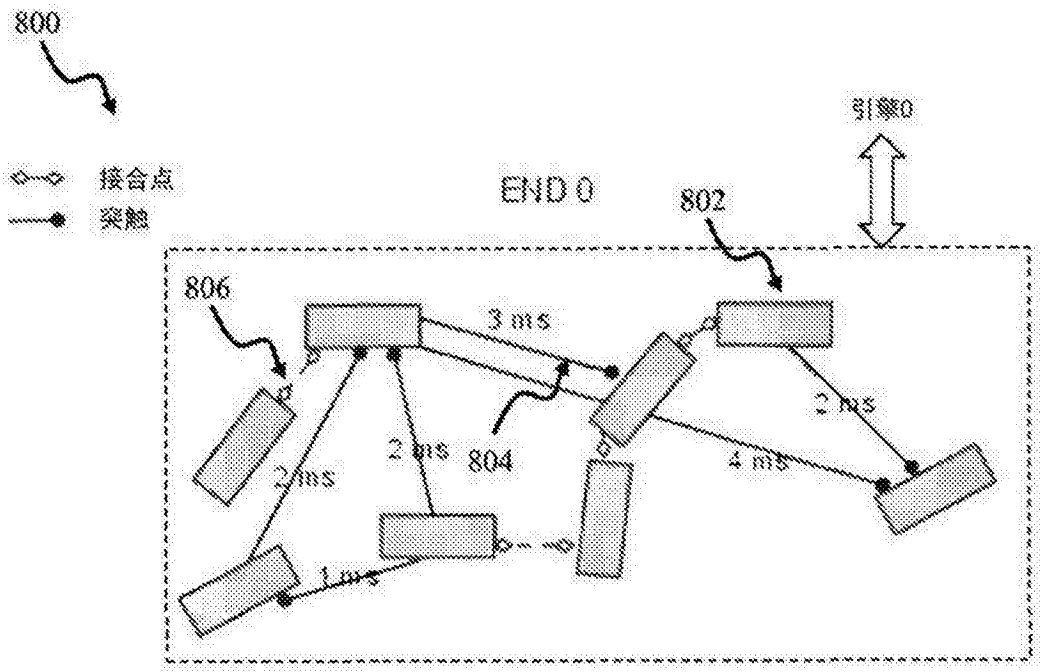

800
接合点
突触
END 0
引擎0
802
806
3 ms
2 ms
2 ms
804
2 ms
4 ms
1 ms

图8

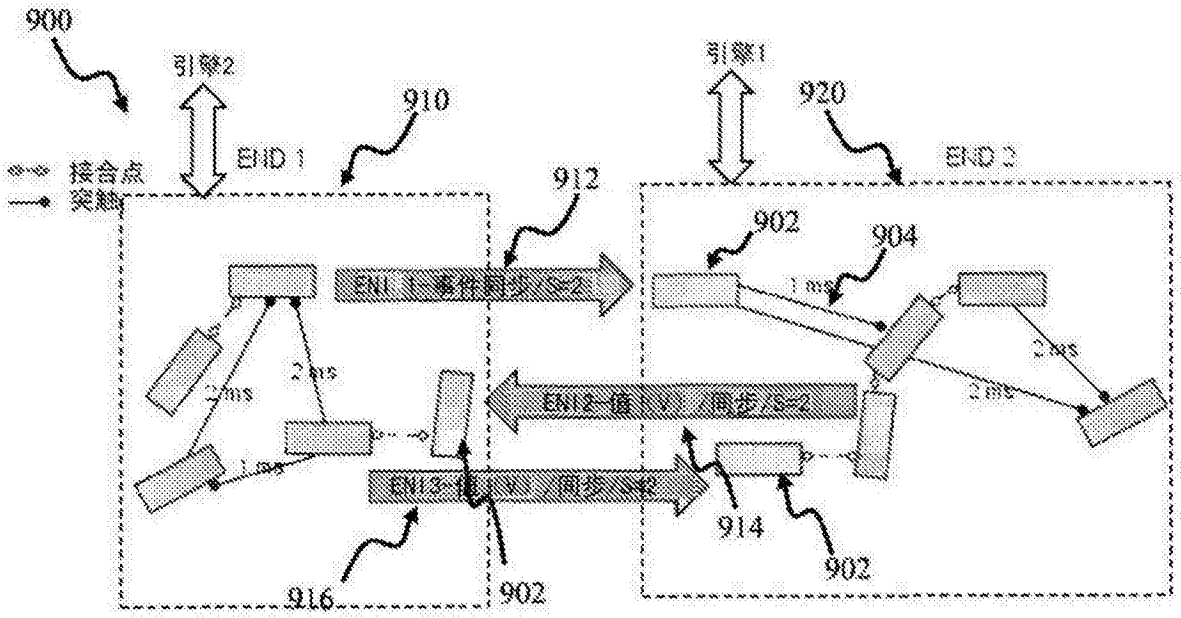

900
引擎2
END 1
910
接合点
突触
912
引擎1
920
END 2
902
904
1 ms
2 ms
2 ms
2 ms
914
902
916
902

图9

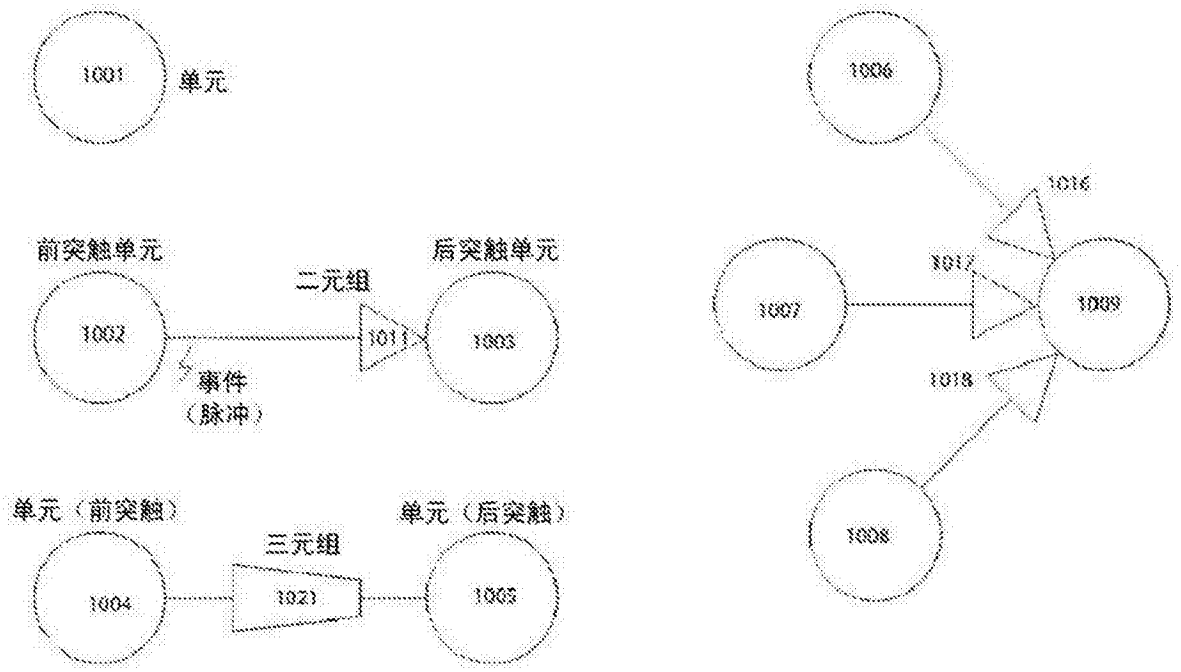

1001
单元
前突触单元
二元组
后突触单元
1002
1011
1003
事件
（脉冲）
单元（前突触）
三元组
单元（后突触）
1004
1021
1005
1006
1016
1017
1007
1009
1018
1008

图10

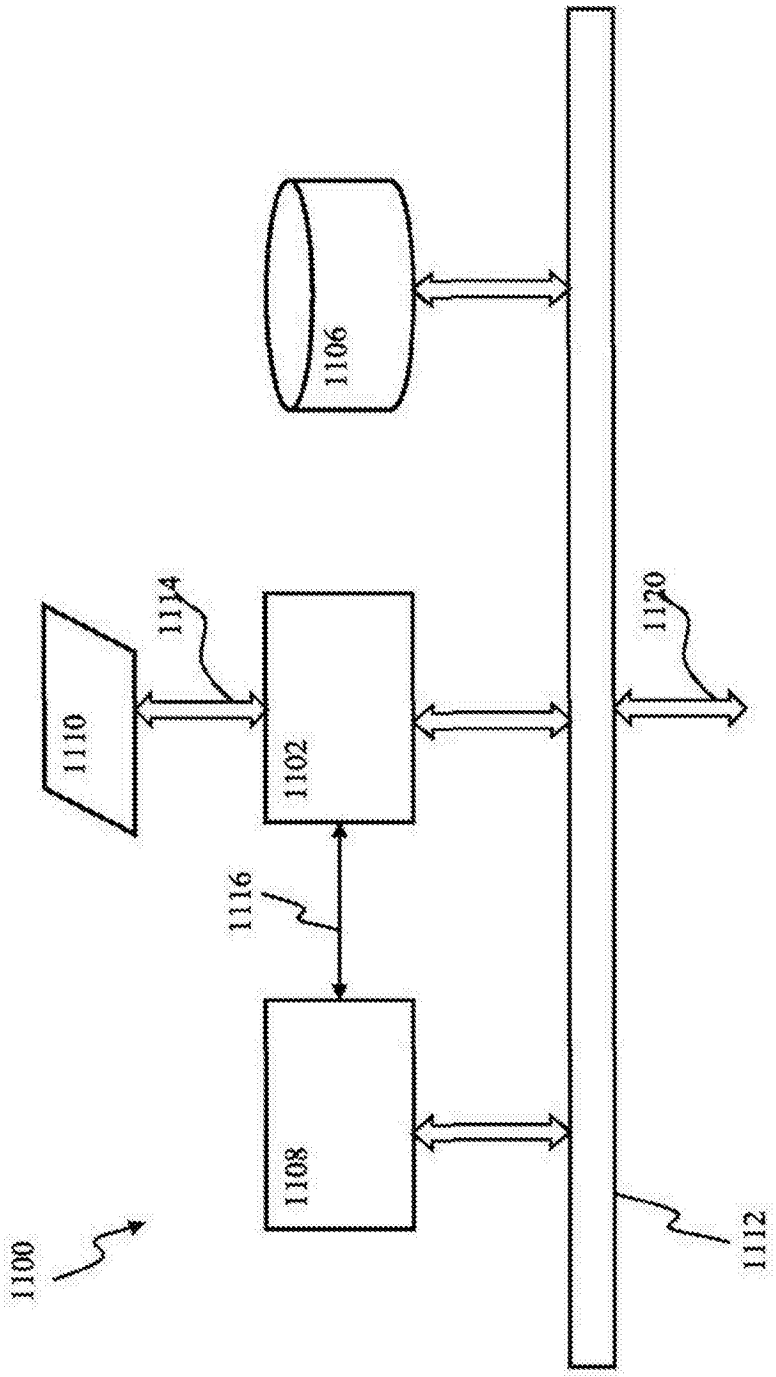


图11A

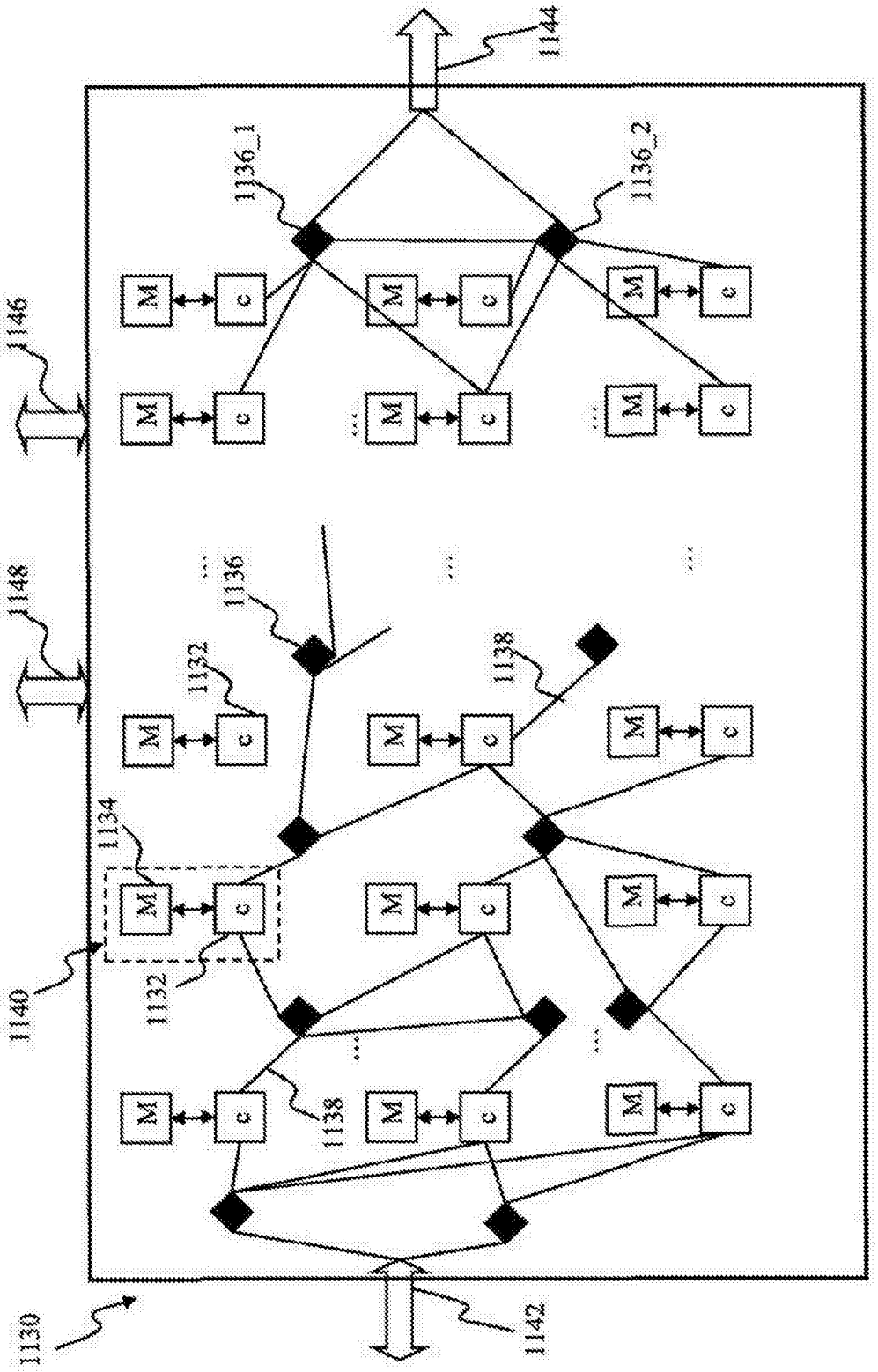

1144
1136_1
1136_2
1146
1148
1136
1132
1138
1134
1140
1132
1138
1130
1142
M
c

图11B

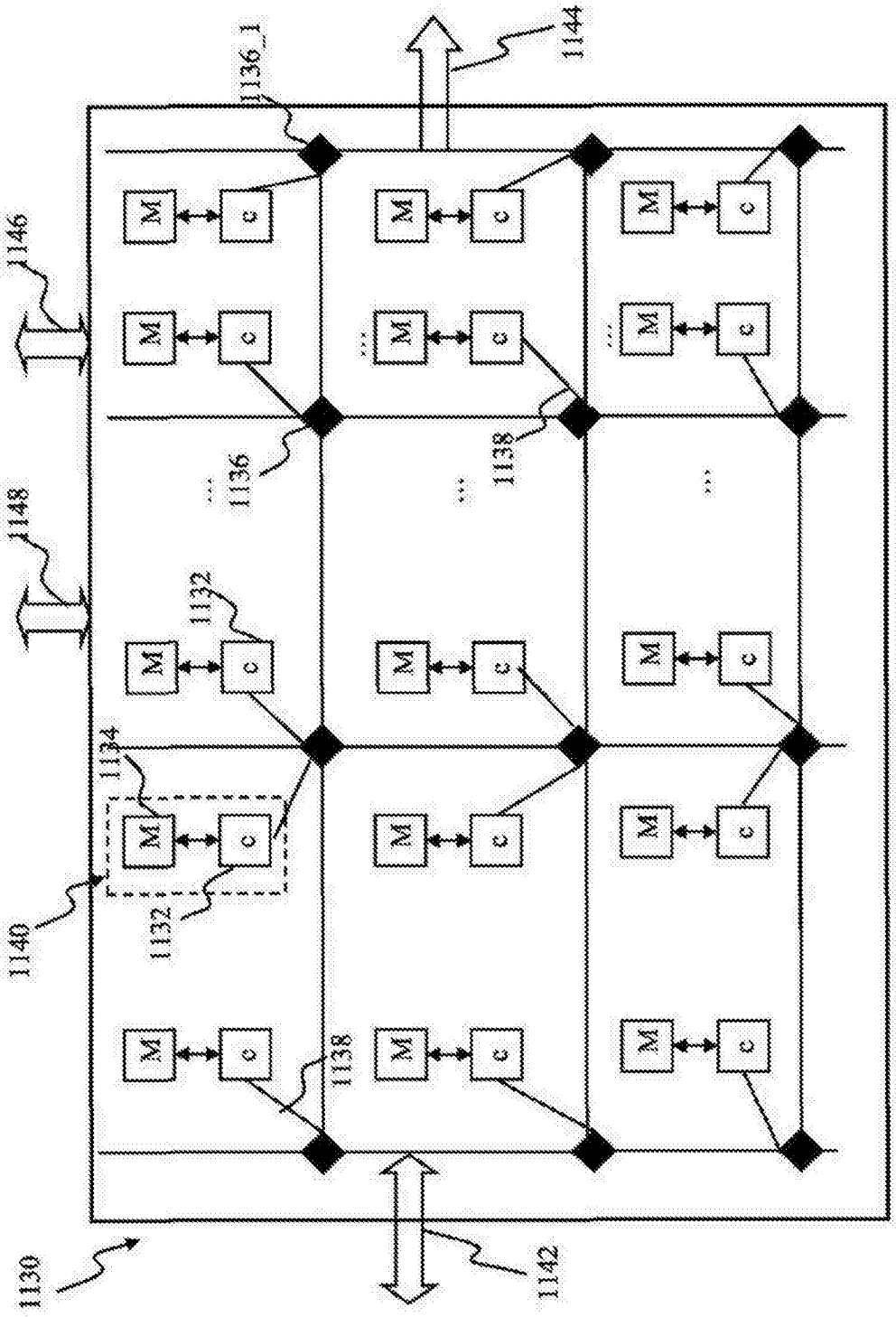


图11C

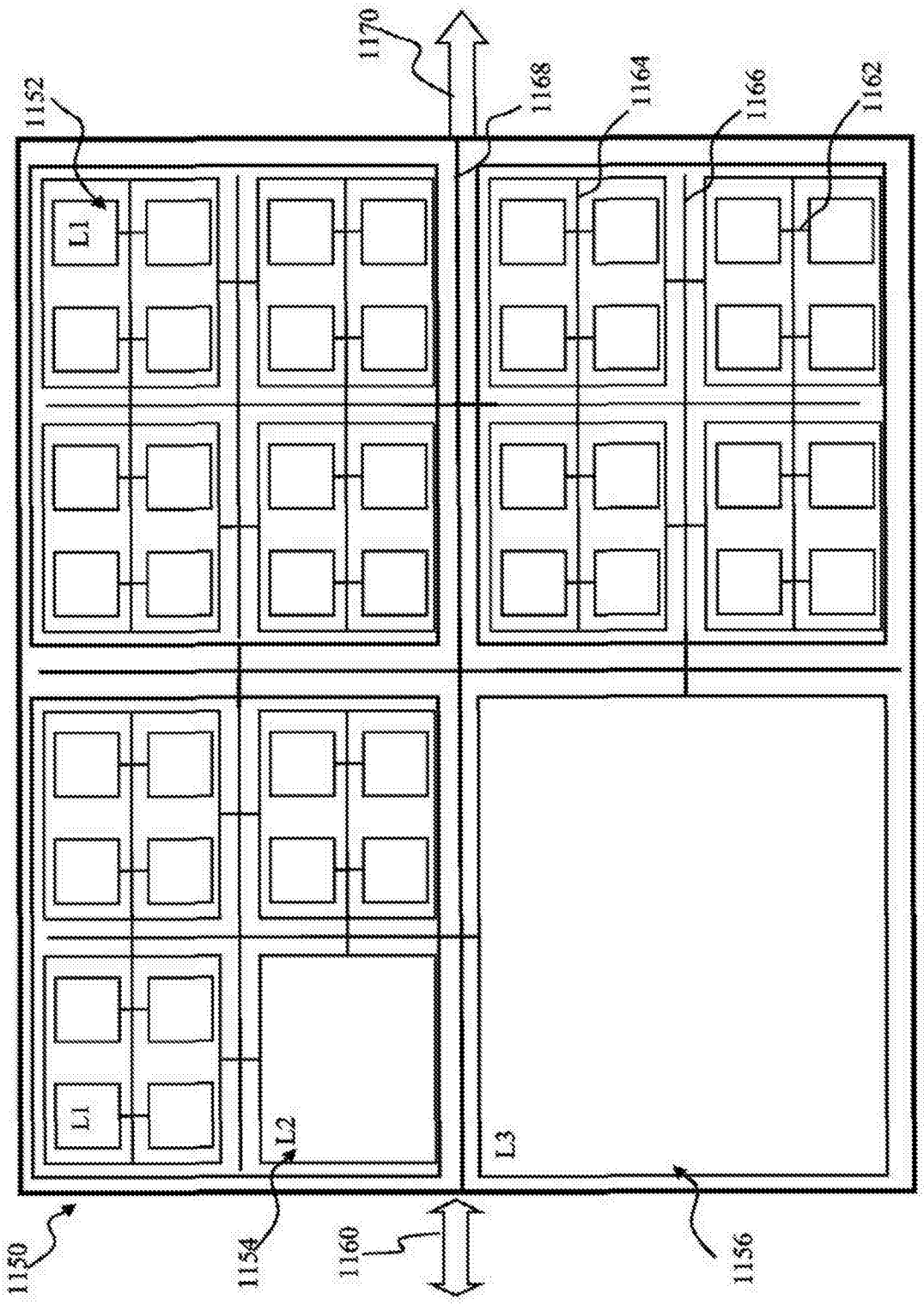


图11D

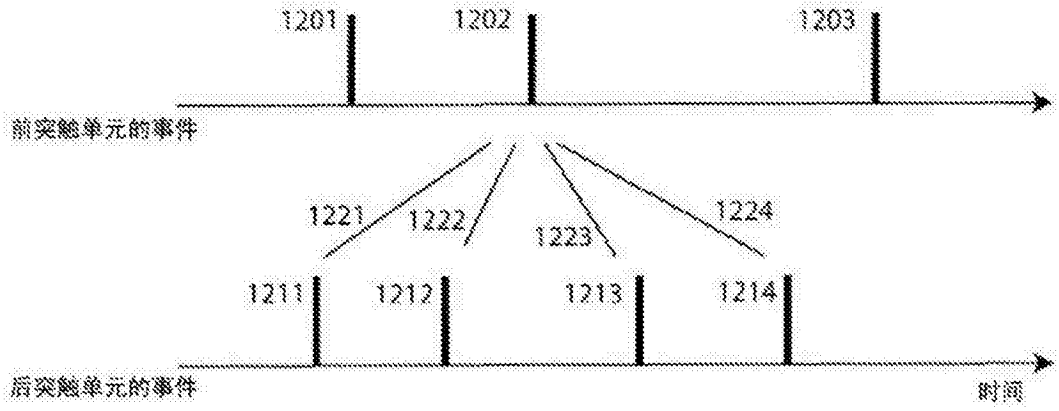

1201
1202
1203
前突触单元的事件
1221
1222
1223
1224
1211
1212
1213
1214
后突触单元的事件
时间

图12

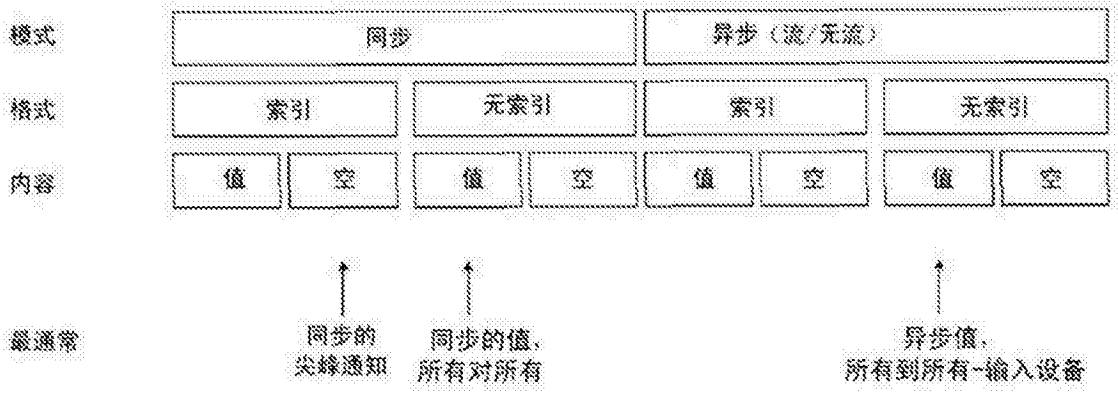

模式
同步
异步（流/无流）
格式
索引
无索引
索引
无索引
内容
值
空
值
空
值
空
值
空
最通常
同步的
尖峰通知
同步的值、
所有对所有
异步值、
所有到所有-输入设备

图13

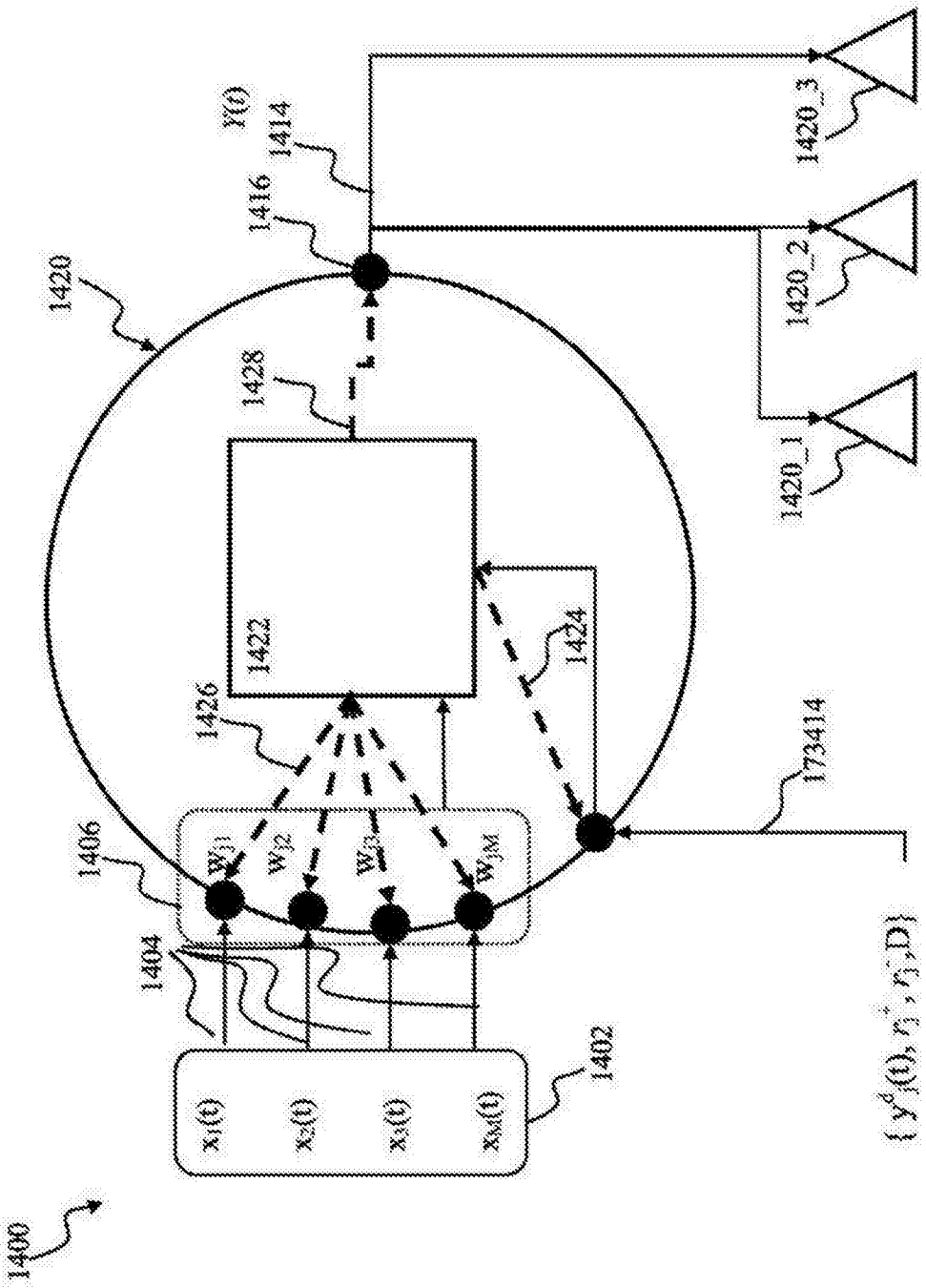

1400
1402
1404
1406
1414
1416
1420
1420_1
1420_2
1420_3
1422
1424
1426
1428
1434
Y(t)
{ y^d_j(t), r_j^+, r_j^-, D }

图14

```
DEFINE UNIT TeacherNeuron
{
    // 这可以是常规神经元，此处不详述
}

DEFINE UNIT InputNeuron
{
    // 这可以是常规神经元，此处不详述
}

DEFINE UNIT Neuron
{
    INIT
    {
        float I = 0;
        float v = 0;
        float u = 0;
        //初始化一些更多的变量，如果需要的话

        bool teaching_signal = false;
    }

    UPDATE_RULE
    {
        CODE
        {
            // 更新神经动态，例如：
            v += -v*0.99;
        }
    }

    EVENT_RULE
    {
        EXEC_CONDITION { v >= 1}
        CODE
        {
            //  重置神经动态
            //  更新局部变量，例如：
            v = 0;

            //  自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
            //  并且传播该尖峰
            //

        }
```

```
41      }

42      DENDRITIC_EVENT_RULE
43      {
44          EXEC_CONDITION { teaching_signal }
45          CODE {
46              // 重置树突状值，例如
47              teaching_signal = false;

48              // 自动地触发定义在突触中的DENTRITIC_EVENT_RULE
49              //
50              // 并且不传播尖峰
51          }
52      }
53 }

54 DEFINE SYNAPSE FeedForwardSynapse
55 {
56      INIT
57      {
58          float w = 0.0;
59          int t_in = 0;
60          int t_out = 0;
61      }
62      PRESYNAPTIC_EVENT_RULE
63      {
64          // 尖峰传送
65          Unit.I += w;

66          // 在尖峰前的记录
67          t_in = Sys.Now;

68          // 更新突触效力-LTD
69          w += LTD( t_in, t_out ); // 此处未详述

70      }
71      POSTSYNAPTIC_EVENT_RULE
72      {
73          // 在尖峰后的记录
74          t_out = Sys.Now;
75          // 更新突触效力-LTP
76          w += LTP( t_in, t_out ); // 此处未详述
77      }
78
79      DENRITIC_EVENT_RULE
80      {
```

```
81              // 更新突触效力-调制的STDP
82              w += MSTDP(t_in, t_out, Sys.Now)
83              // 基于电压的MSTDP计算的例子
84              //0.1*exp(-(t_out - t_in))*exp(-(Sys.Now -
85              ///t_out))*(Unit.v-1);
86      }
87 }

88 DEFINE SYNAPSE TeacherSynapse
89 {
90      PRESYNAPTIC_EVENT_RULE
91      {
92          Unit.teaching_signal = true;
93 }
94 }

95 CREATE 1 InputNeuron;
96 CREATE 1 Neuron AS SpecialNeuron;
97 CREATE 1 TeacherNeuron;

98 CONNECT FROM InputNeuron TO SpecialNeuron WITH
   FeedForwardSynapse;
99 CONNECT FROM TeacherNeuron TO SpecialNeuron WITH
   TeacherSynapse;
100     // 特殊神经元可以连接到剩余的网络
```

图15A

```
DEFINE UNIT InputNeuron
{
    // 这可以是常规神经元，此处不详述
}

DEFINE UNIT MyNeuron
{
    INIT
    {
        float v = 0;
        float threshold = 1;
        // 初始化一些更多的变量，如果需要的话

        int spike_counter = 0;
    }

    UPDATE_RULE
    {
        CODE
        {
            // 更新神经动态，例如：
            v += -v*0.99;
        }
    }

    EVENT_RULE
    {
        EXEC_CONDITION { v >= threshold }
        CODE {
            //  重置神经动态
            //  更新局部变量，例如：
            v = 0;
            spike_counter +=1;

            //  自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
            //
            //  并且传播该尖峰
        }
    }

    DENDRITIC_EVENT_RULE
    {
        EXEC_CONDITION { spike_counter >= 100}
        CODE {
```

```
39                  // 重置树突状值，
40                  //  执行固有的可塑性更新，例如
41                  threshold *=0.9;
42                  spike_counter = 0;

43                  //  自动地触发定义在突触中的DENRITIC_EVENT_RULE
44                  //
45                  //  并且不传播尖峰
46              }
47      }
48 }

49 DEFINE SYNAPSE FeedForwardSynapse
50 {
51      INIT
52      {
53          float w = 0.0;
54      }

55      PRESYNAPTIC_EVENT_RULE
56      {
57          //尖峰传送
58          Unit.v += w;
59      }

60      POSTSYNAPTIC_EVENT_RULE
61      {
62
63      }

64      DENDRITIC_EVENT_RULE
65      {
66          // 衰减突触效力
67          w *=0.99;
68      }
69 }

70 CREATE 1 InputNeuron;
71 CREATE 1 MyNeuron AS OutputNeuron;

72 CONNECT FROM InputNeuron TO OutputNeuron WITH
   FeedForwardSynapse;
```

图15B

```
DEFINE UNIT InputNeuron
{
    //  这可以是常规神经元，此处不详述
}

DEFINE UNIT TeacherNeuron
{
    //  这可以是常规神经元，此处不详述
}

DEFINE UNIT MyNeuron
{
    INIT
    {
        float v = 0;
        //  初始化一些更多的变量，如果需要的话
        bool generate_dendritic_event = false;
    }

    UPDATE_RULE
    {
        CODE
        {
            //  更新神经动态，例如：
            v += -v*0.99;
        }
    }

    EVENT_RULE AS AxonicEvent
    {
        EXEC_CONDITION { v >= 1}
        CODE {
            //  重置神经动态
            //  更新局部变量，例如：
            v = 0;
            //  触发定义在树突中的轴突事件，
            //
            //  并且人工地传播该尖峰
            spike();
        }
    }

    EVENT_RULE AS DendriticEvent
    {
        EXEC_CONDITION { generate_dendritic_event }
```

```
41          CODE {
42              // 重置树突状值，例如：
43              generate_dendritic_event = false;
44
45              // 自动地触发定义在突触中的DENRITIC_EVENT_RULE
46              //
47              //  并且不传播尖峰

48          }
49      }
50 }

51 DEFINE SYNAPSE FeedForwardSynapse
52 {
53      INIT
54      {
55          float w = 0.0;
56          int t_in = 0;
57          int t_out = 0;
58      }

59      PRESYNAPTIC_EVENT_RULE
60      {
61          // 尖峰传送
62          Unit.v += w;
63          // 在尖峰前的记录
64          t_in = Sys.Now;
65      }
66
67      EVENT_RULE AS AxonicEvent
68      {
69          // 在尖峰后的记录
70          t_out = Sys.Now;
71      }

72      EVENT_RULE AS DendriticEvent
73      {
73           // 更新突触效力-调制的STDP
74          w += MSTDP(t_in, t_out, Sys.Now)
75          // 基于电压的MSTDP计算的例子
76          //0.1*exp(-(t_out - t_in))*exp(-(Sys.Now -
77          ///t_out))*(Unit.v-1);
74      }
75 }
76 DEFINE SYNAPSE TeacherSynapse
77 {
```

```
78      PRESYNAPTIC_EVENT_RULE
79      {
80           Unit.generate_dendritic_event = true;
81      }
82 }
83 CREATE 1 InputNeuron;
84 CREATE 1 MyNeuron AS OutputNeuron;
85 CREATE 1 TeacherNeuron;

86 CONNECT FROM InputNeuron TO OutputNeuron WITH
   FeedForwardSynapse;
87 CONNECT FROM TeacherNeuron TO OutputNeuron WITH
   TeacherSynapse;
```

图15C

```
DEFINE UNIT InputNeuron
{
    // 这可以是常规神经元，此处不详述
}

DEFINE UNIT TeacherNeuron
{
    // 这可以是常规神经元，此处不详述
}

DEFINE UNIT MyNeuron
{
        INIT
        {
            float v = 0;
            // 初始化一些更多的变量，如果需要的话
        }

        UPDATE_RULE
        {
            CODE
        {
            // 更新神经动态，例如：
            v += -v*0.99;
        }
    }

    EVENT_RULE
    {
        EXEC_CONDITION ( v >= 1)
        CODE {
            // 重置神经动态
            // 更新局部变量，例如：
            v = 0;
            // 自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
            // 并且传播该尖峰
            //

        }
    }

    EVENT_RULE AS TeacherEvent
    {
        // 定义产生教导事件的能力
```

```
39.          // 其自动地调用在突触中定义的教导事件规则
40.          //
41.
42.     }

43. }

44. DEFINE SYNAPSE FeedForwardSynapse
45. {
46.     INIT
47.     {
48.          float w = 0.0;
49.          int t_in = 0;
50.          int t_out = 0;
51.     }

52.     PRESYNAPTIC_EVENT_RULE
53.     {
54.          // 尖峰传送
55.          Unit.v += w;
56.          // 在尖峰前的记录
57.          t_in = Sys.Now;
58.     }

59.     POSTSYNAPTIC_EVENT_RULE
60.     {
61.          // 在尖峰后的记录
62.          t_out = Sys.Now;
63.     }

64.     EVENT_RULE AS TeacherEvent
65.     {
66.          // 更新突触效力-调制的STDP
67.          w += MSTDP(t_in, t_out, Sys.Now)
68.          // 基于电压的MSTDP计算的例子
69.          //0.1*exp(-(t_out - t_in))*exp(-(Sys.Now -
70.          ///t_out))*(Unit.v-1);

71.     }
72. }

73. DEFINE SYNAPSE TeacherSynapse
74. {
75.     PRESYNAPTIC_EVENT_RULE
76.     {
78          //  触发定义在单元中的教导事件规则
```

```
77.          Unit.TeacherEvent
78.     }
79. }

80. CREATE 1 InputNeuron;
81. CREATE 1 MyNeuron AS OutputNeuron;
82. CREATE 1 TeacherNeuron;

83. CONNECT FROM InputNeuron TO OutputNeuron WITH
    FeedForwardSynapse;
84. CONNECT FROM TeacherNeuron TO OutputNeuron WITH
    TeacherSynapse;
```

图15D

```
1. DEFINE UNIT InputNeuron
2. {
3. // 这可以是常规神经元，此处不详述
4. }

5. DEFINE UNIT MyNeuron
6. {
7.       INIT
8.       {
9.       float v = 0;
10.           float threshold = 1;
11.           //   初始化一些更多的变量，如果需要的话
12.           int spike_counter = 0;
13.      }
14.
15.      UPDATE_RULE
16.      {
17.           CODE
18.           {
19.                //   更新神经动态，例如：
20.                v += -v*0.99;
21.           }
22.      }

23.      EVENT_RULE
24.      {
25.           EXEC_CONDITION { v >= threshold }
26.           CODE {
27.                // 重置神经动态
28.                // 更新局部变量，例如：
29.                v = 0;
30.                spike_counter +=1;
79                 // 自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
80                 //
81                 //   并且传播该尖峰
31.           }
32.      }

33.      EVENT_RULE AS OverexcitabilityEvent
34.      {
35.           EXEC_CONDITION { spike_counter >= 100 }
36.           CODE {
37.                threshold *=0.9;
38.                spike_counter = 0;
39.                // 自动地触发在突触中定义的过兴奋度事件
```

```
40.                 // 并且不传播尖峰
41.                 //

42.             }
43.     }
44. }

45. DEFINE SYNAPSE FeedForwardSynapse
46. {
47.     INIT
48.     {
49.         float w = 0.0;
50.     }

51.     PRESYNAPTIC_EVENT_RULE
52.     {
53.         // 尖峰传送
54.         Unit.v += w;
55.     }

56.     POSTSYNAPTIC_EVENT_RULE
57.     {
58.     }

59.     EVENT_RULE AS OverexcitabilityEvent
60.     {
61.         // 衰减突触效力
62.         w *=0.99;
63.     }
64. }

65. CREATE 1 InputNeuron;
66. CREATE 1 MyNeuron AS OutputNeuron;

67. CONNECT FROM InputNeuron TO OutputNeuron WITH
  FeedForwardSynapse;
```

图15E

```
1
2  DEFINE UNIT InputNeuron
3  {
4       // 这可以是常规神经元，此处不详述
5  }

6  DEFINE UNIT TeacherNeuron
7  {
8       //  这可以是常规神经元，此处不详述
9  }

10 DEFINE UNIT MyNeuron
11 {
12      INIT
13      {
14           float v = 0;
15           //  初始化一些更多的变量，如果需要的话
16      }

17      UPDATE_RULE
18      {
19           CODE
20           {
21                // 更新神经动态，例如：
22                v += -v*0.99;
23           }
24      }

25      EVENT_RULE
26      {
27           EXEC_CONDITION { v >= 1}
28           CODE {
29                // 重置神经动态
30                // 更新局部变量，例如：
31                v = 0;
32                //
33                // 自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
34                // 并且传播该尖峰
35           }
36      }

37      EVENT_RULE AS OverexcitabilityEvent
38      {
39           EXEC_CONDITION { spike_counter >= 100}
40           CODE {
41                threshold *=0.9;
```

```
42                  spike_counter = 0;
43                  // 自动地触发定义在突触中的过兴奋度事件
                    //
44                  //  并且不传播尖峰
45              }
46          }
47
48          EVENT_RULE AS TeacherEvent
49          {
50              //   定义产生教导事件的能力
51              //
52              //   其自动地调用定义在突触中的教导事件规则
53          }
54  }
55  DEFINE SYNAPSE FeedForwardSynapse
56  {
57          INIT
58          {
59              float w = 0.0;
60              int t_in = 0;
61              int t_out = 0;
62          }
63          PRESYNAPTIC_EVENT_RULE
64          {
65              // 尖峰传送
66              Unit.v += w;
67              // 在尖峰前的记录
68              t_in = Sys.Now;
69          }
70          POSTSYNAPTIC_EVENT_RULE
71          {
72              //  在尖峰后的记录
73              t_out = Sys.Now;
74          }
75          EVENT_RULE AS TeacherEvent
76          {
77              // 更新突触效力-调制的STDP
78              w += MSTDP(t in, t out, Sys.Now)
79              // 基于电压的MSTDP计算的例子
80              //0.1*exp(-(t_out - t_in))*exp(-(Sys.Now -
```

```
81            ///t_out))*(Unit.v-1);

82        }

83        EVENT_RULE AS OverexcitabilityEvent
84        {
82            // 衰减突触效力
85            w *=0.99;
86        }
87 }

88 DEFINE SYNAPSE TeacherSynapse
89 {
90        PRESYNAPTIC_EVENT_RULE
91        {
92            // 触发定义在单元中的教导事件规则
93            Unit.TeacherEvent
94        }
95 }

96 CREATE 1 InputNeuron;
97 CREATE 1 MyNeuron AS OutputNeuron;
98 CREATE 1 TeacherNeuron;

99 CONNECT FROM InputNeuron TO OutputNeuron WITH
   FeedForwardSynapse;
100     CONNECT FROM TeacherNeuron TO OutputNeuron WITH
   TeacherSynapse;
```

图15F

```
83 DEFINE UNIT TeacherNeuron
84 {
85      // 这可以是常规神经元，此处不详述
86 }
87
88 DEFINE UNIT InputNeuron
89 {
90      //   这可以是常规神经元，此处不详述
91 }

92 DEFINE UNIT Neuron
93 {
94      INIT
95      {
96           float I = 0;
97           float v = 0;
98           float u = 0;
99           // 初始化一些更多的变量，如果需要的话
100               bool teaching_signal = false;
101          }

102          UPDATE_RULE
103          {
104               CODE
105               {
106                    //   更新神经动态，例如：
107               }
108          }

109          EVENT_RULE
110          {
111               EXEC_CONDITION { v > 30 }
112
113               CODE
114               {
115                    // 重置神经动态
116                    // 更新局部变量，例如：
117                    // 自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
118                    //
119                    //   并且传播该尖峰
120               }
121          }
```

```
122         DENDRITIC_EVENT_RULE
123         {
124             EXEC_CONDITION { teaching_signal }
125
126             CODE
127             {
128                 // 重置树突状值
129                 teaching_signal = false;
130
131                 // 自动地触发定义在突触中的DENRITIC_EVENT_RULE
                    //
132                 // 并且不传播尖峰
133
134             }
135         }
136     }

137     DEFINE SYNAPSE FeedForwardSynapse
138     {
139         INIT
140         {
141             float w = 0.0;
142             int t_in = 0;
143             int t_out = 0;
144         }

145         PRESYNAPTIC_EVENT_RULE
146         {
147             // 尖峰传送
148             Unit.I += w;
149
150             // 在尖峰前的记录
151             t_in = Sys.Now;
152
153             // 更新突触效力-LTD
154             w += LTD( t_in, t_out ); // 此处未详述

155         }

156         POSTSYNAPTIC_EVENT_RULE
157         {
158             // 在尖峰后的记录
159             t_out = Sys.Now;
160
161             // 更新突触效力-LTP
```

```
162                 w += LTP( t_in, t_out ); // 此处未详述

163           }

164           DENRITIC_EVENT_RULE
165           {
166                 // 更新突触效力-LTP
167     `213456         w += LTP( t_in, t_out ); // 此处未详述

168           }
169     }

170     DEFINE SYNAPSE TeacherSynapse
171     {
172           PRESYNAPTIC_EVENT_RULE
173           {
174                 Unit.generate_dendritic_event = true;
175           }
176     }

177     CREATE 1 InputNeuron AS InputNeuron;
178     CREATE 1 TeacherNeuron AS TeacherNeuron;
179     CREATE 1 Neuron AS SpecialNeuron;

180     CONNECT FROM InputNeuron TO SpecialNeuron WITH
  FeedForwardSynapse;
181     CONNECT FROM TeacherNeuron TO OutputNeuron WITH
  TeacherSynapse;

182     //  特殊神经元可以连接到剩余的网络
```

图15G

```
DEFINE UNIT TeacherNeuron
{
    // 这可以是常规神经元，此处不详述
}

DEFINE UNIT InputNeuron
{
    //  这可以是常规神经元，此处不详述
}

DEFINE UNIT Neuron
{
    INIT
    {
        float I = 0;
        float v = 0;
        float u = 0;
        //  初始化一些更多的变量

        bool teaching_signal = false;
    }

    UPDATE_RULE
    {
        CODE
        {
            //  更新神经动态
        }
    }

    // This is a regular/traditional event_rule
    EVENT_RULE
    {
        EXEC_CONDITION { v > 30 }

        CODE
        {
            //  重置神经动态
            //  更新局部变量

            //  自动地触发定义在突触中的POTSYNAPTIC_EVENT_RULE
            //
            //  并且传播该尖峰
        }
    }
```

```
40      //    定制事件规则CUSTUM_EVENT_RULE
41
42      {
43          EXEC_CONDITION { teaching_signal }
44
45          CODE
46          {
47              // 重置树突状值
48              teaching_signal = false;
49
50              //
51              // 显式地调用定义在突触中的规则
52              Syn.DendriticRule;
53          }
54      }
55 }

56 DEFINE SYNAPSE FeedForwardSynapse
57 {
58      INIT
59      {
60          float w = 0.0;
61          int t_in = 0;
62          int t_out = 0;
63      }

64      PRESYNAPTIC_EVENT_RULE
65      {
66          // 尖峰传送
67          Unit.I += w;

68          // 在尖峰前的记录
69          t_in = Sys.Now;
70
71          // 更新突触效力-LTD
72          w += LTD( t_in, t_out ); // 此处未详述
73      }

74      POSTSYNAPTIC_EVENT_RULE
75      {
76          // 在尖峰后的记录
77          t_out = Sys.Now;
78
79          // 更新突触效力-LTP
80          w += LTP( t_in, t_out ); // 此处未详述
81      }
```

```
82      EVENT_RULE AS DendriticRule
83      {
84          // 更新突触效力-LTP
85          w += LTP( t_in, t_out ); // 此处未详述
86      }
87 }

88 DEFINE SYNAPSE TeacherSynapse
89 {
90      PRESYNAPTIC_EVENT_RULE
91      {
92          Unit.generate_dendritic_event = true;
93      }
94 }

95 CREATE 1 InputNeuron AS InputNeuron;
96 CREATE 1 TeacherNeuron AS TeacherNeuron;
97 CREATE 1 Neuron AS SpecialNeuron;

98 CONNECT FROM InputNeuron TO SpecialNeuron WITH
   FeedForwardSynapse;
99 CONNECT FROM TeacherNeuron TO OutputNeuron WITH
   TeacherSynapse;

100     //  特殊神经元可以连接到剩余的网络
```

图15H

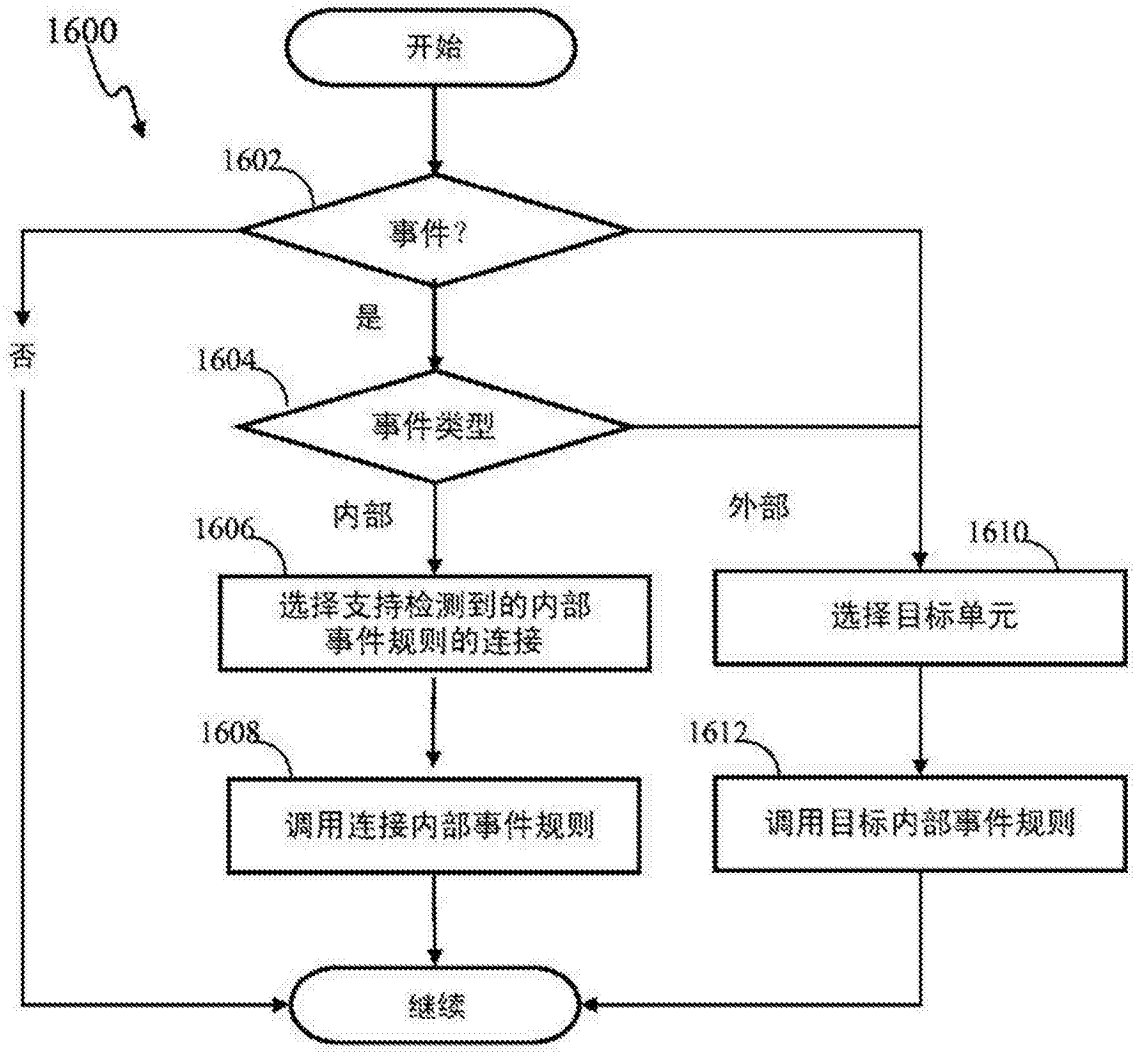

1600
开始
1602
事件?
是
否
1604
事件类型
内部
外部
1606
选择支持检测到的内部事件规则的连接
1610
选择目标单元
1608
调用连接内部事件规则
1612
调用目标内部事件规则
继续

图16A

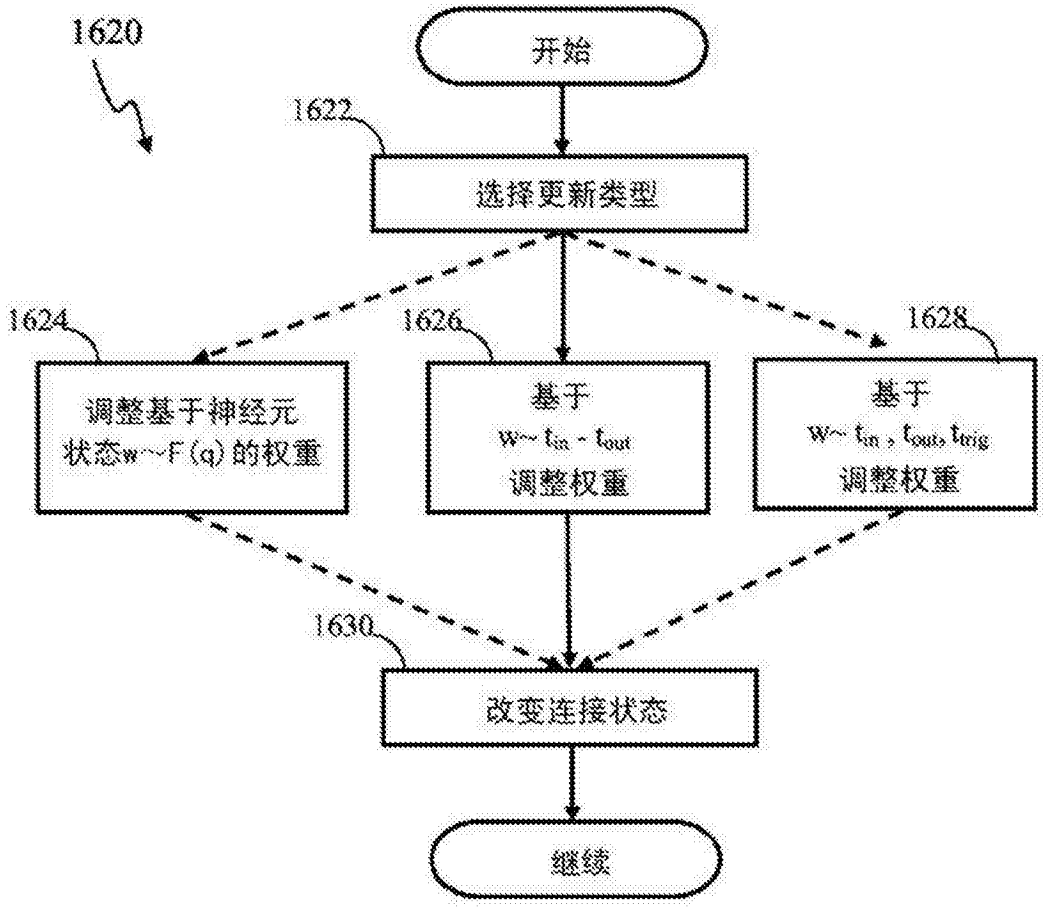

1620
开始
1622
选择更新类型
1624
调整基于神经元状态w~F(q)的权重
1626
基于 w~ t_{in} - t_{out} 调整权重
1628
基于 w~ t_{in} , t_{out}, t_{trig} 调整权重
1630
改变连接状态
继续

图16B

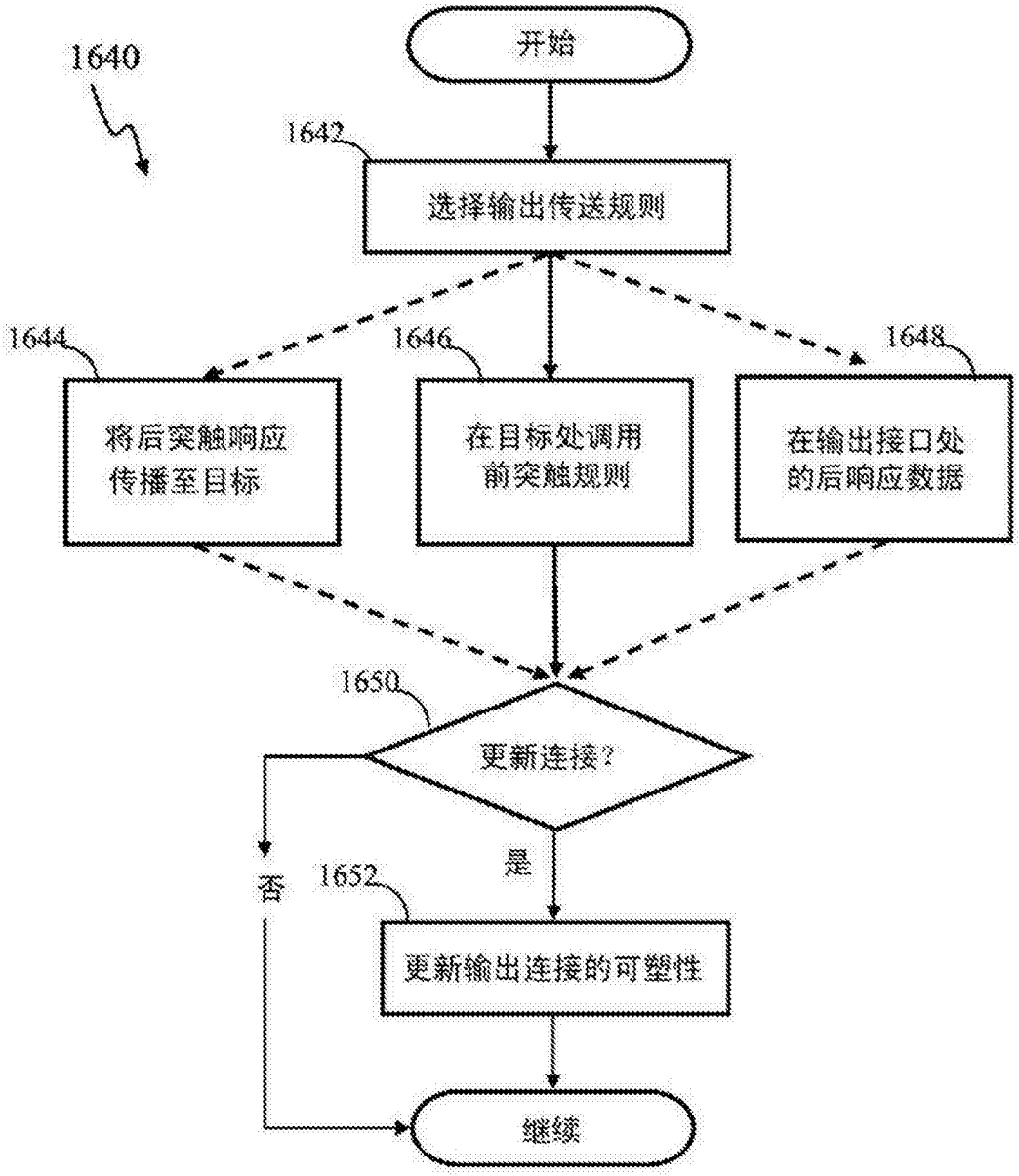

1640
开始
1642
选择输出传送规则
1644
1646
1648
将后突触响应传播至目标
在目标处调用前突触规则
在输出接口处的后响应数据
1650
更新连接？
否
是
1652
更新输出连接的可塑性
继续

图16C

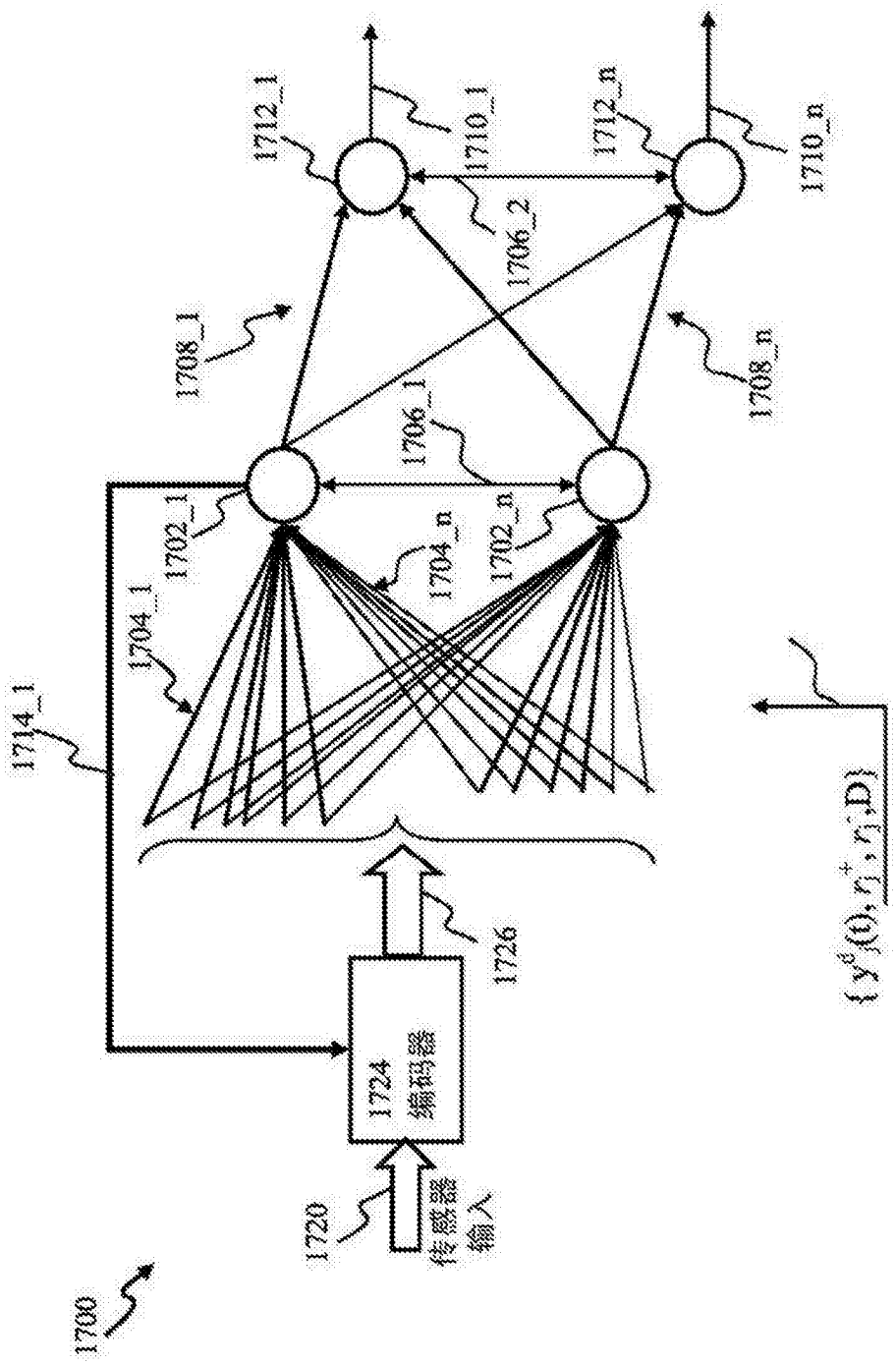

1700
1720
传感器输入
1724
编码器
1726
1714_1
1704_1
1704_n
1702_1
1702_n
1706_1
1706_2
1708_1
1708_n
1712_1
1712_n
1710_1
1710_n
$\{y^d(0), r_1^+, r_1^-, D\}$

图17

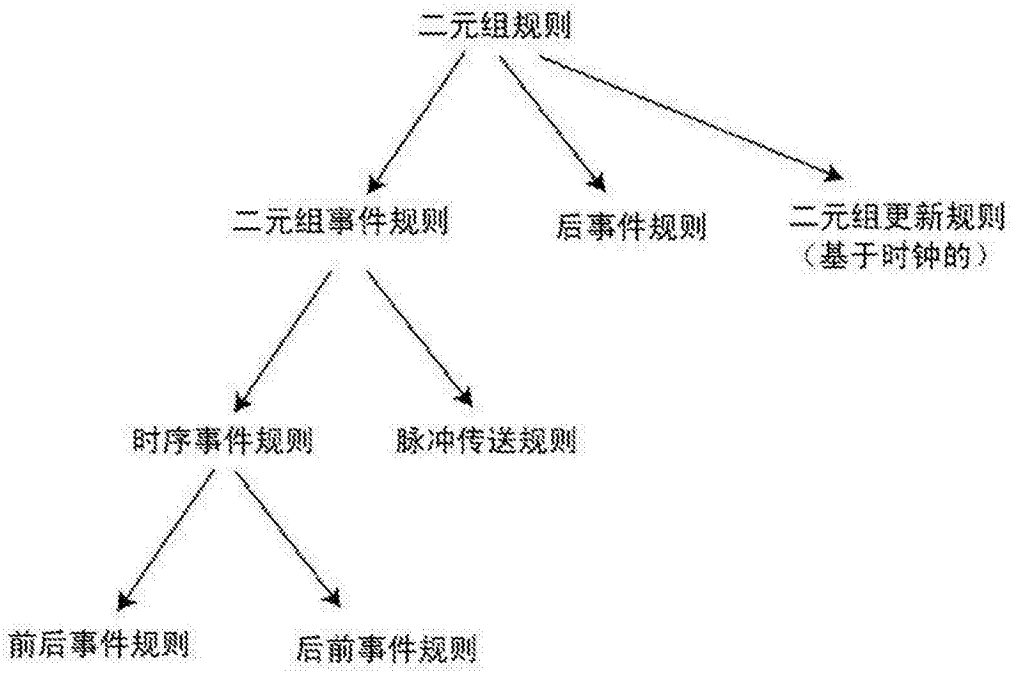

二元组规则
二元组事件规则
后事件规则
二元组更新规则
（基于时钟的）
时序事件规则
脉冲传送规则
前后事件规则
后前事件规则

图18

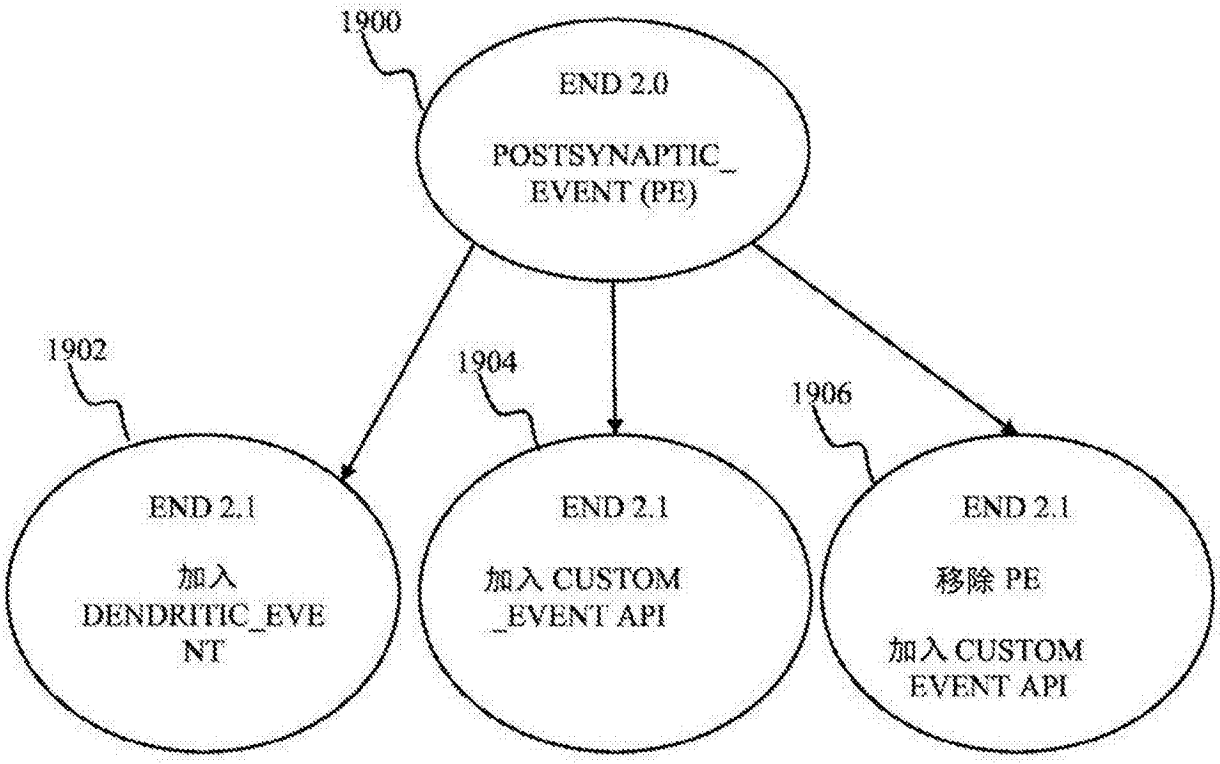

1900
END 2.0
POSTSYNAPTIC_
EVENT (PE)
1902
1904
1906
END 2.1
加入
DENDRITIC_EVE
NT
END 2.1
加入 CUSTOM
_EVENT API
END 2.1
移除 PE
加入 CUSTOM
EVENT API

图19