SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2021, Vol. 64 132202:1–132202:13
https://doi.org/10.1007/s11432-019-2988-1

Hybrid neural state machine for neural network

Lei TIAN[1†], Zhenzhi WU[2†], Shuang WU[1] & Luping SHI[1*]

[1]*Department of Precision Instrument, Center for Brain-Inspired Computing Research (CBICR), Beijing Innovation Center for Future Chip, Tsinghua University, Beijing* 100084*, China;*
[2]*Lynxi Technologies Co., Ltd., Beijing* 100080*, China*

Received 5 November 2019/Revised 15 May 2020/Accepted 29 June 2020/Published online 22 January 2021

Abstract The integration of computer-science-oriented and neuroscience-oriented approaches is believed to be a promising way for the development of artificial general intelligence (AGI). Recently, a hybrid Tianjic chip that integrates both approaches has been reported, providing a general platform to facilitate the research of AGI. The control algorithm for handling various neural networks is the key to this platform; however, it is still primitive. In this work, we propose a hybrid neural state machine (H-NSM) framework that can efficiently cooperate with artificial neural networks and spiking neural networks and control the workflows to accomplish complex tasks. The H-NSM receives input from different types of networks, makes decisions according to the fusing of various information, and sends control signals to the sub-network or actuator. The H-NSM can be trained to adapt to context-aware tasks or sequential tasks, thereby improving system robustness. The training algorithm works correctly even if only 50% of the forced state information is provided. It achieved performance comparable to the optimum algorithm on the Tower of Hanoi task and achieved multiple tasks control on a self-driving bicycle. After only 50 training epochs, the transfer accuracy reaches 100% in the test case. It proves that H-NSM has the potential to advance control logic for hybrid systems, paving the way for designing complex intelligent systems and facilitating the research towards AGI.

Keywords hybrid neural state machine, ANNs, SNNs, supervised learning, context-aware task, sequential task

Citation Tian L, Wu Z Z, Wu S, et al. Hybrid neural state machine for neural network. Sci China Inf Sci, 2021, 64(3): 132202, https://doi.org/10.1007/s11432-019-2988-1

1 Introduction

There are currently two general approaches to the development of artificial general intelligence (AGI): computer-science-oriented and neuroscience-oriented approaches [1]. Recently, a hybrid platform that supports both prevailing artificial neural networks (ANNs) and neuroscience-inspired models and algorithms was reported, demonstrating simultaneously processing of versatile models in a single platform that facilitates the research of AGI [2]. To make this platform work efficiently, a framework for handling different types of neural networks and controlling workflows is critical and highly desired.

Current artificial intelligence has made remarkable achievements in some specific tasks, such as image capturing and classification, automatic piloting, and language translation. However, it still lacks the ability to control multiple tasks, remember long sequences, and accomplish context-based tasks. In such cases, a state machine is required for remembering the current state and implementing complicated control tasks. Existing ANNs, such as recurrent neural networks (RNNs) and the neural Turing machine [3] are difficult to realize a complete state machine with multiple states and transfer conditions because a differentiable network and global gradient descent training are required but hard to be satisfied. Although there are some non-differentiable methods such as reinforcement learning [4] or recursive tree search [5], the state control part is deeply task-specific and too complex to transfer to other tasks. Spiking neural network (SNN), on the contrary, is an expert in realizing a trainable state machine and dealing with sequential problems because of spatiotemporal dynamics and rich coding schemes. Recently, there are

* Corresponding author (email: lpshi@tsinghua.edu.cn)
† Tian L and Wu Z Z have the same contribution to this work.

© Science China Press and Springer-Verlag GmbH Germany, part of Springer Nature 2021

some studies on state-dependent computations [6–9]. It shows that SNN architecture is able to coordinate the computation of local winner-take-all and solve constraint satisfaction problems [6, 7].

A general way to control neural networks is the key to maximize the potential of current artificial intelligence. There are several paths in the literature regarding neural network control. Differential neural computer [10] and neural Turing machine [3] introduce a long short-term memory network for the external memory read and write control. Google multi-modal training [11] introduces a trainable gating network for selecting a sparse number of simple neural networks (experts) from the expert pool to work on a variety of tasks. There were reports on the relationship between finite state machines and RNNs [12–14]. A second-order RNN was demonstrated to learn and represent a state machine fairly well [12] and some RNN approaches were applied for realizing a trainable state machine [15–17]. However, these RNN solutions suffer from instability when the sequence is long [18] and are hard to train when the task has a dependence on long-term state, because the gradient descent algorithm is difficult to affect the long state transfers task. On the other hand, a McCulloch-Pitts net can be used to represent a finite state machine that works on binary signals. This mathematically proves that a finite-state model can be constructed in biologically plausible neural networks [18]; however, because the network is not differentiable and gradient descent is hard to perform, training methods are limited.

Despite the importance of developing an efficient state machine for multitask cooperation, this has been neglected for a long time and its development is stalled. Instead, most of the recent efforts are directed on applying the traditional state machine in some specific domains, such as deriving tests [19, 20], automatically generating Ethereum contracts [21], scalable state machine replication [22] and road network extraction [23]. In addition, these studies only deal with one kind of neural networks, which limits their application scenarios. Only a few alternative methods of building a state machine were reported, including extraction from code by user interaction [24], and integrating neural network with the state machine, such as liquid state machine (LSM) [25–27]. LSM was proved to be trainable to accomplish speech recognition [26], but it is difficult to extract the specific state information of a complex LSM. Some researchers have tried to build a hybrid system to deal with complex tasks, such as building a hybrid artificial neural network for predicting stock price [28], using graph network to improve the visual reasoning task [29], using the portfolio theory in supply chain management [30], implementing self-assessment technique for parallel network systems [31]. However, these studies mainly focus on single tasks. The methods are highly coupled with the tasks, which makes it difficult to transfer to other tasks. Moreover, at a high level of abstraction, the control logic is similar among some tasks. To promote the development of the intelligence system, an independent and general network framework is a pressing need.

In general, ANNs and SNNs show different advantages [32]: (i) ANNs directly transmit the high-precision activation (analog value) and process information continuously. They have demonstrated powerful capability in scenarios that require high-precision and dense computation. (ii) SNNs memorize the historical temporal information via intrinsic neuronal dynamics and code information into digital spike train, enabling event-driven computation. They are naturally suitable for scenarios with rich temporal information and sparse dataflow. In order to fully make use of both advantages of ANN and SNN, in this work, we propose a hybrid neural state machine (H-NSM) framework that can be built on ANN, SNN or a hybrid ANN/SNN to achieve efficient cooperation between different networks. Normally, a state machine consists of six parts, including the input alphabet, the output alphabet, the transition function, the output function, the states set, and the start state. Each part of H-NSM can be formed by different types of neural networks, either ANNs or SNNs. Here, we designed an H-NSM based on an SNN state controller (SNN-based H-NSM) to validate the framework and a training algorithm was developed for weight updating. As the training algorithm only requires local information, it can be performed by a segment of a long transfer sequence. Hence, it enables a long sequence to be learned and expressed stably, exhibiting training ability for different tasks. In addition, the SNN-based H-NSM also allows the states to be represented by binary values (the one-hot coding scheme of state neurons), showing increased robustness of the system compared to representing states with analog values, as in RNN networks. Furthermore, through the experiments of the Tower of Hanoi and a self-driving bicycle, we demonstrate the adaptability of H-NSM to different state transfer rules. It can accomplish sequential tasks consisting of different types of networks or actuators with performance comparable to that with the optimum algorithm, and achieve the corresponding activation of different branch networks according to the actual situation. The proposed H-NSM aids the design of advanced control logic for hybrid systems, paving the way for designing complex intelligent systems and facilitating the research towards AGI.

The remaining part of this paper is organized as follows. Section 2 outlines the framework of the

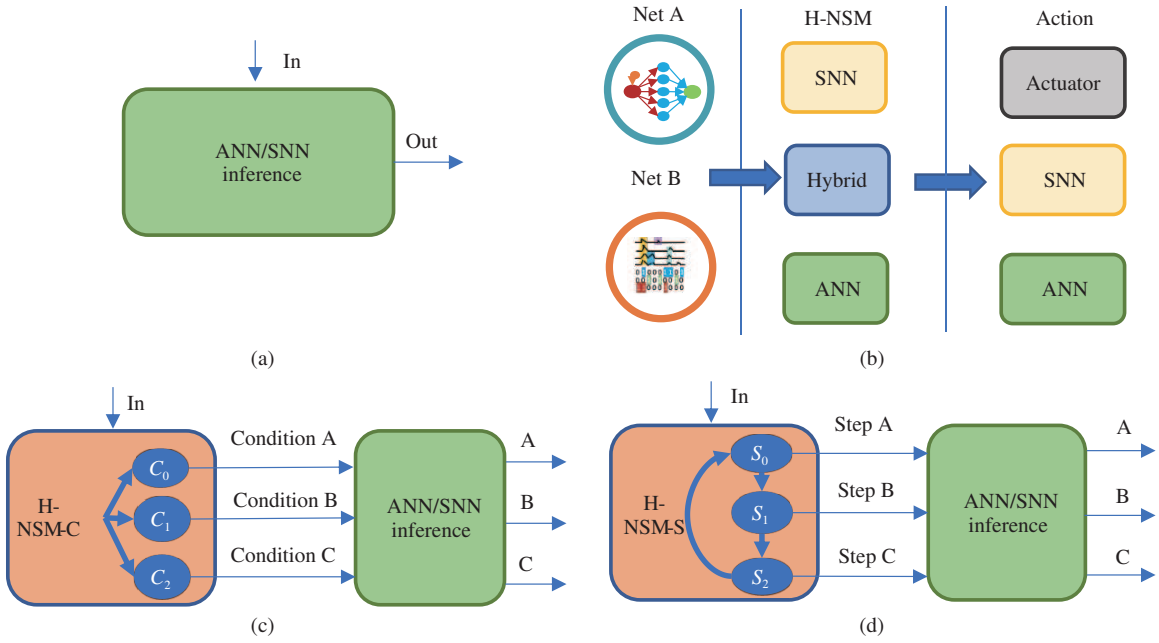


Figure 1 (Color online) (a) Traditional neural network workflow; (b) H-NSM takes input from ANNs and/or SNNs, and controls the working flow according to diverse tasks, and then sends control signals to the inference networks or actuators; (c) H-NSM-C makes decision based on different conditions and activates the desired branches to accomplish different tasks; (d) H-NSM-S accomplishes sequential tasks and sends control signals according to the current step.

H-NSM. Section 3 describes an H-NSM based on SNN. Section 4 introduces the training rules. Section 5 presents the results of the experiments. Section 6 presents the discussion, and finally Section 7 is the conclusion.

2 Framework design

Our proposed H-NSM framework that integrates different types of networks is depicted in Figure 1. The difference between the current neural networks and H-NSM can be clearly seen from Figures 1(a) and (b). As a state machine, H-NSM takes input from an ANN or an SNN, performs a state transfer based on the current state and input, and then sends control signals to the inference networks or actuators. Here, the 'Hybrid' is reflected in two aspects as shown in Figure 1(b): the controller can be based on different types of networks; the control part communicates with different types of networks.

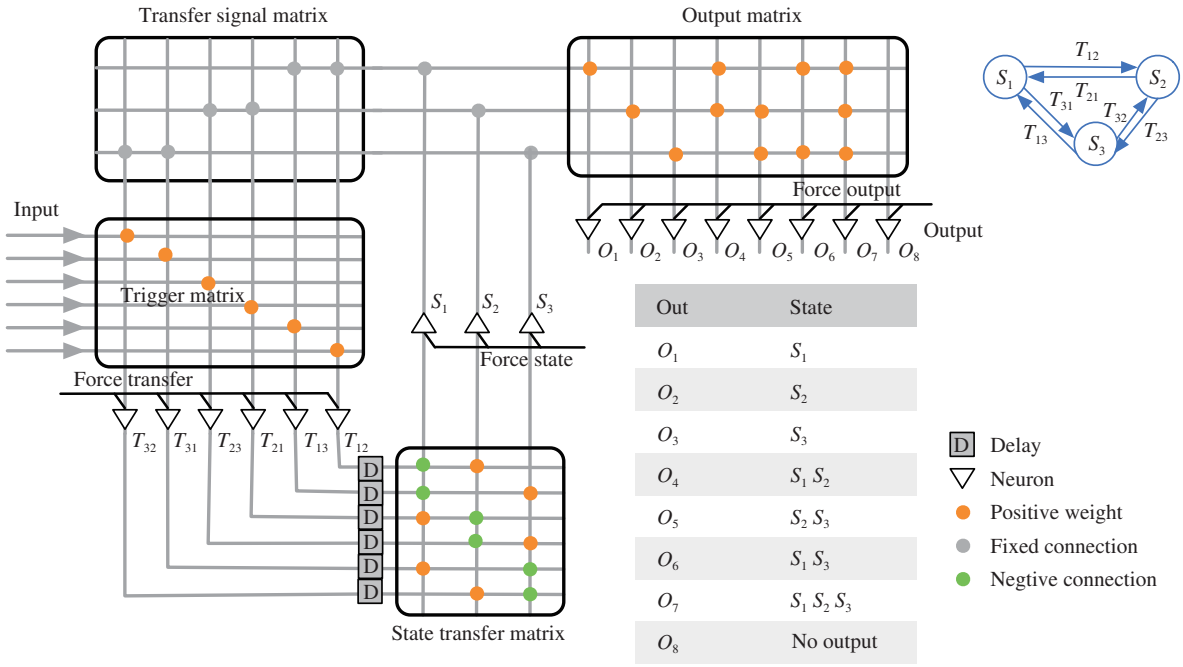
Two kinds of H-NSM were designed to handle condition-based tasks and sequential tasks, as shown in Figures 1(c) and (d), respectively. H-NSM-C is based on control logic and makes decisions according to different conditions, and H-NSM-S accomplishes sequential tasks.

As a controller, H-NSM can be easily integrated with different kinds of networks, such as ANNs and SNNs. The data path may contain many branches, especially in the execution part. A branch is activated according to the input. The control signals are applied to trigger the events so that the desired branches accomplish different tasks or directly control the actuator. H-NSM is also able to monitor the workflow and accomplish sequential tasks.

3 Building an SNN-based H-NSM

3.1 Simplified leaky integrate and fire (LIF) model

In this study, an SNN is adopted to design the state controller of the H-NSM, which is referred as an SNN-based H-NSM. Its input and output can be from ANNs or SNNs. This system is discrete in the time domain with a minimum time unit called a 'step'. In each step, the signal transmitted between neurons is represented by a binary value, where '1' denotes that there is a spike in the current time step and '0' denotes that there is no spike. In the SNN, all neurons are modeled by a simplified LIF



Out	State
O_1	S_1
O_2	S_2
O_3	S_3
O_4	$S_1\ S_2$
O_5	$S_2\ S_3$
O_6	$S_1\ S_3$
O_7	$S_1\ S_2\ S_3$
O_8	No output

Figure 2 (Color online) Demonstrate a three-state complete Moore state machine using SNN-based H-NSM.

model. It integrates the input with synapse weight ($V_{t+1} = V_t + \sum \text{Input} \times \text{Weight}$). If $V >$ threshold, the neuron fires and performs a reset procedure ($V = V_{\text{reset}}$). The spike is directly transmitted to other neurons, or delayed for one step and then transmitted to itself or other neurons. The leakage procedure ($V = V - \text{leakage}$) is performed at last.

Although in some cases the spiking triggers from external networks can be read by the SNN-based H-NSM directly; however, analog triggers consist of more information and can be obtained from ANN networks directly. In such cases the encoder acts as an information aggregator, which collects the information from ANNs and transfers to simple spiking triggers which serve as the input of transfer neurons. In another aspect, the spike output from the SNN-based H-NSM may contain sparsity which is hard to be incorporate with ANN executors directly. Therefore, an ANN-based decoder converts the spike output codes into a more condensed analog code, which is similar to the word embedding extensively applied in ANN-based natural language processing. In the decoder, several neurons' spikes may convert into a single analog output in order to connect to ANNs. The encoder and decoder can be trained in a supervised manner. For the encoder, the analog triggers are derived from ANNs, and the spike triggers are pre-defined labels. For the decoder, the state and the spike output are the known controlling signals for the downstream ANNs. These ANNs that accept such decoded codes will generate the final results, which are compared with the labels to calculate the loss, thus making supervised learning applicable in our proposed H-NSM.

3.2 Complete state machine model

A Moore machine can be represented as a 6-tuple $(S, S_1, \sum, \bigwedge, T, G)$. Here, S is the finite set of states with a start state of S_1, $\sum$ is the input (trigger) alphabet, and $\bigwedge$ is the output alphabet. A transfer function $T : S \times \sum \longrightarrow S$ maps a state and trigger to the next state, and an output function $G : S \longrightarrow \bigwedge$ maps each state to the output alphabet. The number of states in S is N. We first consider a complete state machine, wherein a state has a transfer connection to any other state with trigger condition, and all possible outputs for these states are provided. The desired state machine usually is a subset of the given complete model.

The complete Moore state machine realized by SNN-based H-NSM is shown in Figure 2 (for $N_{\text{state}} = 3$).

The state machine is composed of three subgroups of neurons, i.e., state neurons, transfer neurons, and output neurons.

The states are represented by the state neurons in one-hot coding, and each neuron represents a state. When the state machine stays in a state, the corresponding state neuron is activated (fired at all steps).

There is only one active neuron among all the state neurons. If no transfer signal is activated, the state is kept; hence, the current state activation is fed back to the state neuron network with a step delay to make the current neuron fire repeatedly.

The transfer signals are necessary for transition to a new state. When the corresponding transfer signal is turned on, the activated neuron representing the current state will be depressed and the neuron for the target state will be activated.

The transfer signals are formed by another group of neurons. The transfer signal matrix specifies which state the system is currently in, and the trigger matrix collects the trigger signal for the current state. Both the state signal and the trigger signal are in charge of activating the transfer signal: these signals can be combined using logical OR or logical AND to generate the transfer signal.

The output matrix is necessary for defining the output signal according to the states. The output signals are then used for controlling the neural network data path or actuators.

Given N states, there are $N(N-1)$ possible transfers, and 2^N output possibilities. Hence, the state transfer matrix is of size $N(N-1)$ by N, which means that there are $N(N-1)$ transfer signals, and the system outputs N states. The transfer signal matrix is pre-defined (fixed). The trigger matrix associates the input trigger with the transfer signal, and is of size N by $N(N-1)$. One transfer signal may respond for multiple input triggers, and one input trigger may serve several transfer signals. The number of triggers is not limited.

The output matrix can output all possibilities depending on the state conditions. Note that for a desired state machine, only a part of the outputs and a part of the state transfer signals are needed.

The force state input and force transfer input are only for training, and not necessary for the inference. In this section, we assume that the state machine is pre-trained. The training procedure is given in Section 4.

3.3 Scalability of the model

Our framework is aimed to handle multiple tasks in dynamic environments. It can be easily trained according to the task. The scalability of the model can be divided into two aspects:

(1) The parameter matrix can be trained and overwritten across different tasks. If it is a totally new case, we need to update the whole H-NSM. Firstly, we should set the state number, input number and output number, which is relative to the number of three kinds of neurons and the size of the weight matrix. Then we can train this new H-NSM with the method given in Section 4.

(2) The original state and weight matrix could be transferred directly as a subset of a new weight matrix in new tasks. If it is a new task based on the old one, we should modify the H-NSM accordingly. It means that we can only train the growing part. For example, When the representation size grows, the number of neurons and the size of the weight matrix will increase. We can gradually modify the model parameters to meet the need of the new task: the original state and weight matrix can be transferred directly as a subset of the new weight matrix in the new task. New training data needs to be generated for the training procedure. Then the growing part of the weight matrix can be trained with the new data.

4 Training an SNN-based H-NSM

This work investigates supervised learning as an example, wherein the states, transfer signals, and output signals are provided by the supervisor or obtained from another neural network. For accurate training, only one-shot training is required. When the supervised information is inaccurate or even occasionally wrong, for example, if the transfer state is unclear, the SNN-based H-NSM can still learn the correct result after accumulating enough training times.

4.1 SNN-based H-NSM training algorithm

In this training scheme, the supervisor provides the 'force state' and 'force transfer' and the state transfer network is learned by a spike-timing-dependent plasticity (STDP)-like rule. This is suitable for training the state transfer matrix, output matrix, and transfer signal matrix. The rules are as follows.

H-NSM training rules. For each synapse in the axon-neuronal matrix:

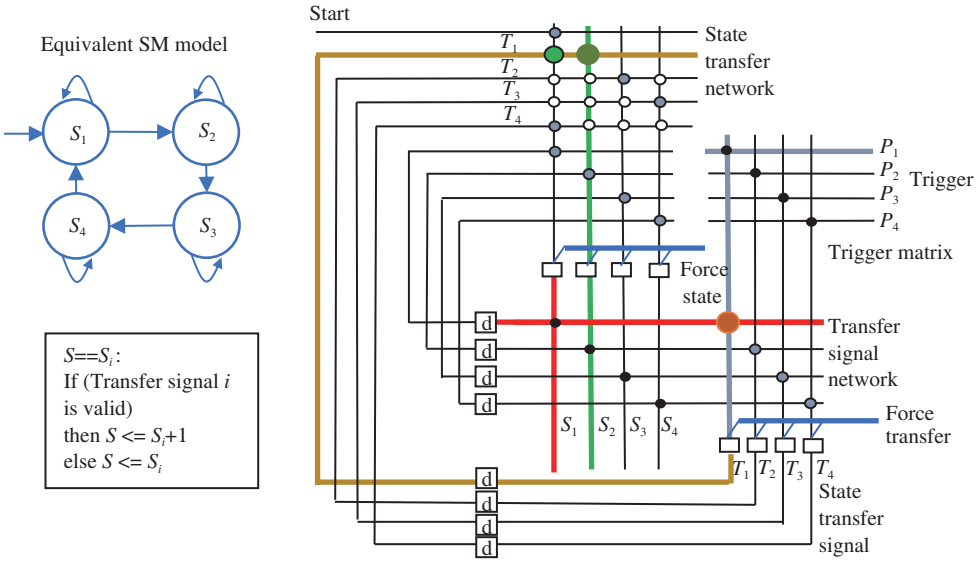


Figure 3 (Color online) Training procedure of an SNN-based H-NSM.

(1) When there is a 'force transfer' spike in transfer neurons' axon in time step $t-1$, and the state neuron is not fired with the LIF rule but forced to fire at time step t, the synapse weight between them is increased by a constant δ;

(2) When there is a 'force transfer' spike in transfer neurons' axon in time step $t-1$, and the state neuron is fired with the LIF rule but forced to not fire at time step t, the synapse weight between them is decreased by a constant δ;

(3) There is a saturation threshold in the positive weight and negative weight;

(4) The training is required to be performed by several epochs to get a stable synapse.

Because whether the state neuron is fired or not is controlled by the 'force fire' supervise signal, the connection is defined by the training sequence, which includes the current state, state transfer indication, and trigger signal.

A sequential task training process on the SNN-based H-NSM is depicted in Figure 3, which illustrates how the T_1-S_1, T_1-S_2, and S_1-T_1 synapse weights are trained. When the H-NSM is in state S_1, the red lines are active. When the force-transfer signal for T_1 is on, the purple line is active. Then, in the next time step, the force state forces S_1 to become inactive and S_2 to become active. Because there exists a connection (colored in dark green as shown at the top area of Figure 3) between T_1 and S_2, T_1 was activated at the previous time step, and S_2 is inactive but forced to be active at the current step, the weight of the orange cross-point is increased. Similarly, because T_1's activation inactivates S_1, the green weight cross-points are decreased. The state transfer network can be trained similarly.

The next step is to train the transfer signal network. When the trigger for activating T_1 is on and T_1 is forced on by the force-transfer signal, the brown cross-point weight is increased. The training of the output matrix follows a similar procedure.

4.2 Inference and training procedure

The inference and training can be represented in the same framework, as shown in Proc 1. In this procedure, the state transfer is fixed over the discrete time step t.

In this work, we set the state fire threshold ST = 1 and transfer signal fire threshold STT = 2. In addition, δ is an experience constant and is set to 0.1. Parameter P_{Ths} is unlimited, N_{Ths} is set to -1, P_{Tht} is set to 1.0, and N_{Tht} is set to 0. CM is the concat of state transfer matrix (SM) and an identity matrix. The initial $\mathrm{CM}_{(r,j)}$ is $[I(S,S);O(T,S)]$ (the semicolon indicates the concat of two matrices), and the initial $\mathrm{TM}_{(r,j)}$ is $[O(S,T);I(T,T)]$, where S is the number of states, and T is the number of transfer signals.

The training of the output matrix is similar to that of SubProc4, or it can be directly assigned according to the fixed relationship between the current state and the desired output signal.

Proc 1 Inference and training procedure

for $t = 1$ **to** Tmax **do**
SubProc 1. Integrate and fire of state neurons;
SubProc 2. If training, learning the state transfer matrix (SM);
SubProc 3. Integrate and fire of transfer neurons;
SubProc 4. If training, learning the trigger matrix (TM);
end for
SubProc 1 is applied for deciding current state according to previous states and transfer signals:
$\mathrm{SIn} = [\mathrm{SOut}, \mathrm{TOut}]$;
$V_S = \mathrm{SIn} \cdot \mathrm{CM}$;
$\mathrm{SOut}_j = 1$ if $(V_{S_j} > \mathrm{ST})$ else $0, j \in [1, S]$;
SubProc 2 is the STDP-like training of state transfer matrix:
If $(\mathrm{SOut}_j == 0$ & $\mathrm{SForce}_j(t) == 1$ & $\mathrm{SIn}_{(S+r)} == 1$ & $\mathrm{CM}_{(r,j)} < P_{\mathrm{Ths}})$ then
$\mathrm{CM}_{(r,j)} = \mathrm{CM}_{(r,j)} + \delta$;
If $(\mathrm{SOut}_j == 1$ & $\mathrm{SForce}_j(t) == 0$ & $\mathrm{SIn}_{(S+r)} == 1$ & $\mathrm{CM}_{(r,j)} > N_{\mathrm{Ths}})$ then
$\mathrm{CM}_{(r,j)} = \mathrm{CM}_{(r,j)} - \delta$;
$r \in [1, T],\ j \in [1, S]$;
SubProc 3:
$\mathrm{TIn} = [\mathrm{Trigger}, \mathrm{SOut}]$;
$V_T = \mathrm{TIn} \cdot \mathrm{TM}$;
$\mathrm{TOut}_j = 1$ if $(V_{T_j} > \mathrm{STT})$ else $0, j \in [1, T]$;
SubProc 4:
If $(\mathrm{TOut}_j == 0$ & $\mathrm{TForce}_j(t) == 1$ & $\mathrm{TIn}_r == 1$ & $\mathrm{TM}_{(r,j)} < P_{\mathrm{Tht}})$ then
$\mathrm{TM}_{(r,j)} = \mathrm{TM}_{(r,j)} + \delta$;
If $(\mathrm{TOut}_j == 1$ & $\mathrm{TForce}_j(t) == 0$ & $\mathrm{TIn}_r == 1$ & $\mathrm{TM}_{(r,j)} > N_{\mathrm{Tht}})$ then
$\mathrm{TM}_{(r,j)} = \mathrm{TM}_{(r,j)} - \delta$;
$r \in [1, S], j \in [1, T]$.

We test the training algorithm with inaccurate supervise signals and compare it with that training with accurate state information. Inevitably, the sensing network is not able to provide accurate information all the time. For example, the current state may not be clearly set during training. In this example, the states are transferred from S_1 to S_{10}. We expect to obtain a similar result to that of Figure 4(a). However, in this time, only 60% of the forced state information is clear (with the trigger information clear), and in the remaining 40% of the time, the force state is set to 0, which indicates that no state in the state machine is active. In such a case, after enough training times, it can still 'learn' the correct transfer rules. Figure 4(b) shows the state transfer matrix distribution after 32 times of inaccurate supervised training, indicating that the training algorithm works correctly even if only 60% of the forced state information is provided. Then we reduce the proportion of correct supervised signals to 50%. Figures 4(c) and (d) show the state transfer matrix distribution after 32 training epochs and 100 training epochs. It takes more training epochs to 'learn' the correct transfer rules when the proportion of correct signals decreases. These results demonstrate the robustness of the training algorithm when supervised signals are partially wrong.

5 Experiments

We designed two experiments to test our SNN-based H-NSM model. (1) The Tower of Hanoi is to demonstrate the ability to solve sequential tasks, and (2) an autopilot bicycle with multitasks is to demonstrate the ability to solve context-aware tasks.

5.1 Tower of Hanoi

We extended the H-NSM-S to play the Tower of Hanoi, a game consisting of moving many disks among pegs, and thus many states and many transfer rules need to be applied. The Tower of Hanoi is a typical test for executive functions in cognitive neuroscience [33]. The objective of this task is to move disks from one stack to another obeying the following rules: only one disk can be moved at a time; each move consists of taking the top disk from one stack and placing it on the top of another one; no disk can be placed on the top of a smaller one.

We used the algorithm shown in Algorithm 1 [34] as the tutor.

Algorithm 1. Optimum Tower of Hanoi algorithm.

Number the disks 1 to n (largest to smallest).

- If n is odd, the first move is from peg A to peg C.
- If n is even, the first move is from peg A to peg B. Now, add these constraints:

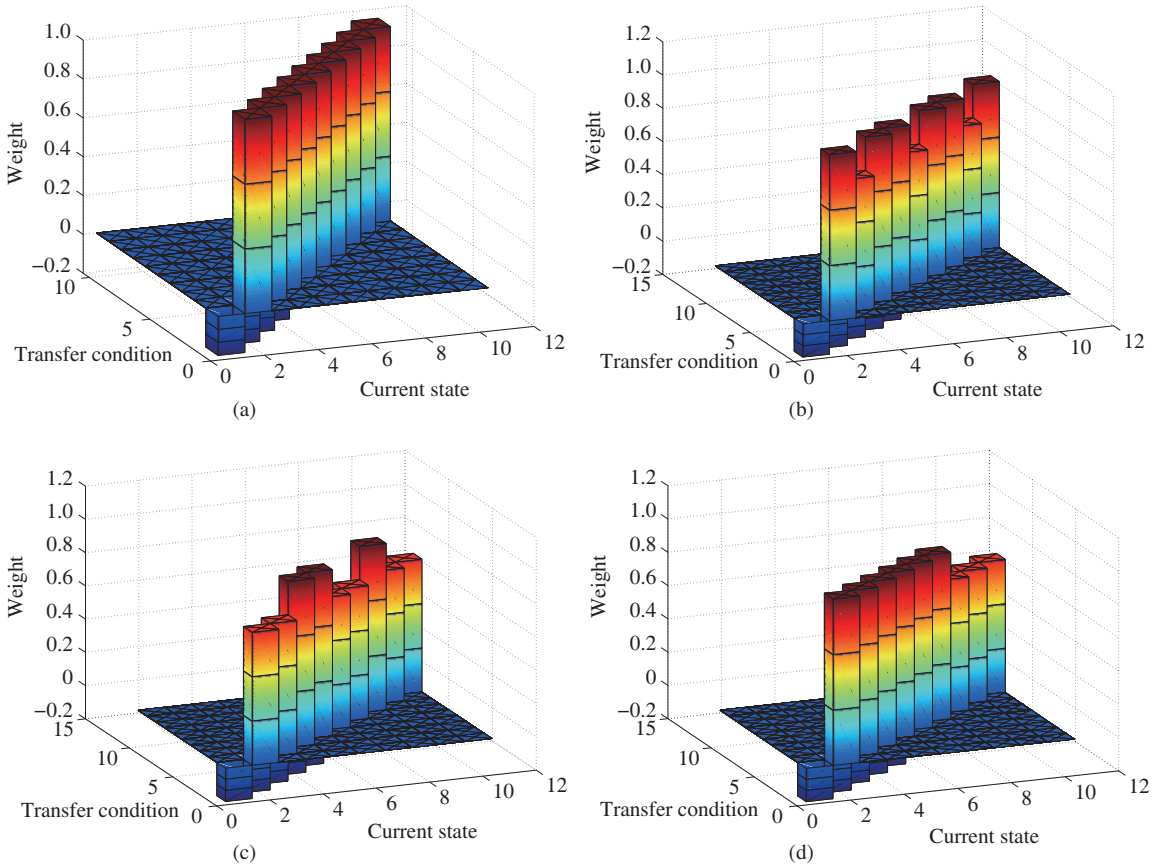


Figure 4 (Color online) (a) State transfer matrix training with accurate supervised signals; (b) state transfer matrix training with 60% correct supervised signals after 32 training epochs; (c) state transfer matrix training with 50% correct supervised signals after 32 training epochs; (d) state transfer matrix training with 50% correct supervised signals after 100 training epochs.

• No odd disk may be placed directly on an odd disk.

• No even disk may be placed directly on an even disk.

• There will sometimes be two possible pegs: one will have disks, and the other will be empty. Place the disk on the non-empty peg.

• Never move a disk twice in succession.

Considering those constraints after the first move, there is only one legal move at every subsequent turn.

We defined a function to test the constraints. Assume we wish to move the top disk from s to t. Then, $f(s,t) =$ yes when $((s = \text{odd}, t = \text{even})$ or $(s = \text{even}, t = \text{odd}))$ and (t is not empty).

We assume that there is an empty peg detector that outputs whether that peg is empty, and an odd-disk detector, which outputs whether the top plate on the peg is odd (if it is odd, then the detector outputs 1; otherwise, it outputs -1). There is also a motor that moves the detectors to peg s first and t second. Accordingly the odd-disk detector's results are sent to neuron N_1 first and neuron N_2 second. The odd-disk detector also serves as the trigger for the trigger matrix. The output of function f is also the trigger for the trigger matrix.

The H-NSM-S is shown in Figure 5(a). The state definition is shown in Table 1. The H-NSM-S consists of five states. S_0–S_4 represents 'start state', 'select state', 'verifying state', 'moving state', and 'finish state', respectively. The transfer condition is shown in Table 2. The transfer condition T_0 is always true, because state S_0 always transfers to S_3 at the next time step. Similarly, T_1 is always true. At the start state S_0, we set the first move for this task. Then the H-NSM-S runs the search and verifying loop (S_1–S_2) until the correct candidate path is found (S_3). It will finish when all the disks are on peg C (S_4).

Function $f(s,t)$ can be realized by a simple LIF neural network, as depicted in Figure 5(b), where a motor is in charge of moving the camera to peg s at time t_1 and peg t at time t_2. Sensors infer the number of disks from the image. Here, sp denotes the source pet. tp denotes the target pet. gt denotes 'greater than'; lastmv denotes 'last move'. p denotes 'candidate path'. At each step, it searches from

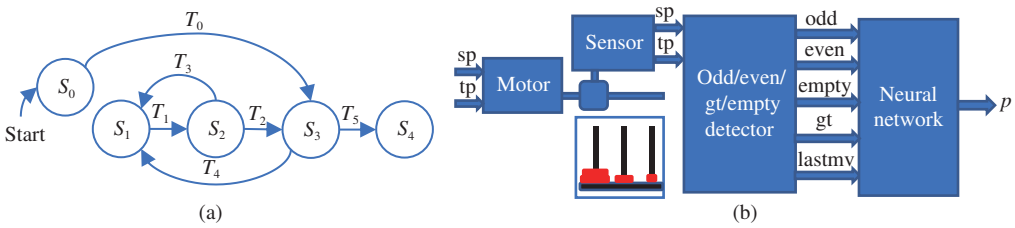


Figure 5 (Color online) (a) The state transfer rules for the Tower of Hanoi; (b) the process flow for function $f(s,t)$ using LIF neural network.

Table 1 State configuration

State	Context	Action
S_0	First move	Select the source and target peg for the first move
S_1	Select	Random select a source peg and a target peg
S_2	Verify	Perform function f
S_3	Move	Perform move
S_4	Finish	–

Table 2 Transfer condition

Transfer	Condition	Action
T_0	S_0–S_3	Always true
T_1	S_1–S_2	Always true
T_2	S_2–S_3	Function f returns true
T_3	S_2–S_1	Function f returns false
T_4	S_3–S_1	Not finish
T_5	S_3–S_4	Finish (all the disks are on peg C)

Table 3 Time (ms)/number of steps cost using different methods

Method	Number of pegs				
	4	5	6	7	8
Optimum	0.6/16	1.3/32	3.4/64	6.9/128	11.8/256
Random	25.1/315	681.5/8328	3068.8/39930	5927.0/78599	86107.6/1273359
H-NSM	0.9/16	2.3/32	3.9/64	9.8/128	16.9/256

the possible path. The network takes the detectors' results as parameters, and then tests whether the constraints are satisfied. If they are satisfied, the network will output this path as a candidate path. If not, it will continue the search process until a correct path is found.

We recorded the time cost and step cost of different algorithms under the same simulation environment (python 2.7, CPU: 2.7 GHz Intel Core i5, software: PyCharm Community Edition 2016.2.3). As we can see from Table 3, the H-NSM-S performs the same steps as the optimum algorithm, while a random search algorithm performs many more steps than the other approaches. Moreover, we have found that the step cost using the random method varies over a wide range in repeated tests. It is because the path searched in each test is different using the random search algorithm. The H-NSM takes longer to complete the same task compared with the optimum algorithm. The per-step time cost using H-NSM increases along with the number of disks. Because it takes more time to perform the function $f(s,t)$ to search the correct path. These results prove that H-NSM-S is capable of learning a long sequence task and achieving the performance comparable to that of an optimum algorithm. Besides, the H-NSM-S can be easily transferred to other similar tasks as long as we change the verification function.

5.2 Autopilot bicycle

Recently, a demonstration platform [2] based on autonomous bicycle was reported. In this experiment, we extended the H-NSM-C to control a multitask bicycle. It is equipped with a series of sensors and actuators. The video information is detected by a convolutional neural network (CNN) and the voice information is processed by an SNN. The bicycle controller is an SNN/ANN mixed neural network.

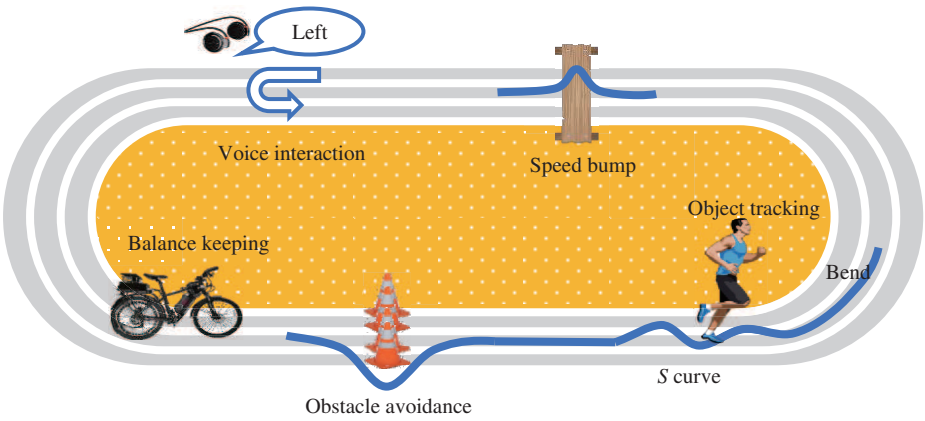


Figure 6 (Color online) Multitask bicycle platform, which is able to take command from the environment to accomplish different tasks such as following a target person and avoiding the obstacles.

It receives the camera video streams and microphone voice streams and controls a steering motor for following a target person, executing voice command or avoiding the obstacles, as shown in Figure 6. The center of the controller is the state machine constructed by the SNN-based H-NSM, which receives the input from an ANN or an SNN, and sends enable signals to the ANN or action signals to output controllers. There is a multi-layer perception network which connects the detection part to action-making part.

On this basis, here, we further detailed and demonstrated the capability of H-NSM-C as a controller to enable different types of networks to work together in the autopilot bicycle platform. The H-NSM-C consists of six states, as shown in Figure 7(a). The default state is S_0, where the bicycle goes straight. S_1–S_5 represents 'detection state', 'speed change state', 'turning state', 'avoiding obstacle state', and 'target searching state', respectively. When the state transfer condition is satisfied, the H-NSM-C will transfer to the corresponding new state. For example, in the state S_0, it waits for a voice command. If no voice command is detected, it stays in the state S_0. If the 'follow me' command is detected, it will transfer to the state S_1.

There are three different modes for the autopilot bicycle as shown in Figure 7(b). The H-NSM chooses the correct working mode according to the input of voice streams and video streams. The S_2 and the S_3 work in the 'voice mode', when the bicycle performs the voice command. The S_1 works in the 'following mode', when the bicycle follows the target. The S_4 and the S_5 work in the 'obstacle mode', when the bicycle acts to avoid the obstacle.

The SNN outputs speech recognition results. We use one-hot coding to represent the different voice commands. It serves as the input signal for the H-NSM-C and can be directly connected to our SNN-based H-NSM.

The CNN outputs analog activations with multi-level values indicating the confidence (probability) of a person or obstacle. When these signals communicate with the H-NSM-C, the multi-level activations are converted to a spike train before being injected to the H-NSM-C FCores according to the rate coding scheme in each conversion FCore (ANN axon or SNN soma). In other words, the accumulated activation value stored as the membrane potential in each conversion FCore is compared with a fixed threshold. When it exceeds the threshold, a spike is fired at time step t. For this example, a higher threshold can improve the stability of the detection results, but increases the response time. After firing, the membrane potential resets to a constant value (following the membrane potential fire-reset rule). Therefore, a higher input activation value usually results in more spikes. In this way, the ANN activations are converted into spike coding. Then a smoothing process (in logical terms) is used to obtain a stable 'trigger' signal according to these spikes, which is implemented as a sliding-window LIF model. First, a sliding window is used in the axon of each FCore, which accumulates the spikes in the temporal domain. Second, the accumulated membrane potential is compared with a pre-set threshold. Once the membrane potential exceeds the threshold, a trigger signal (spike) is fired to the H-NSM-C.

The H-NSM-C outputs enable signals for the CNN and two hybrid-mode cores that contain trajectory patterns for left/right turns from voice commands and obstacle avoidance, respectively. The hybrid-mode cores were implemented by several somas working in the ANN mode (or axons in SNN mode), which

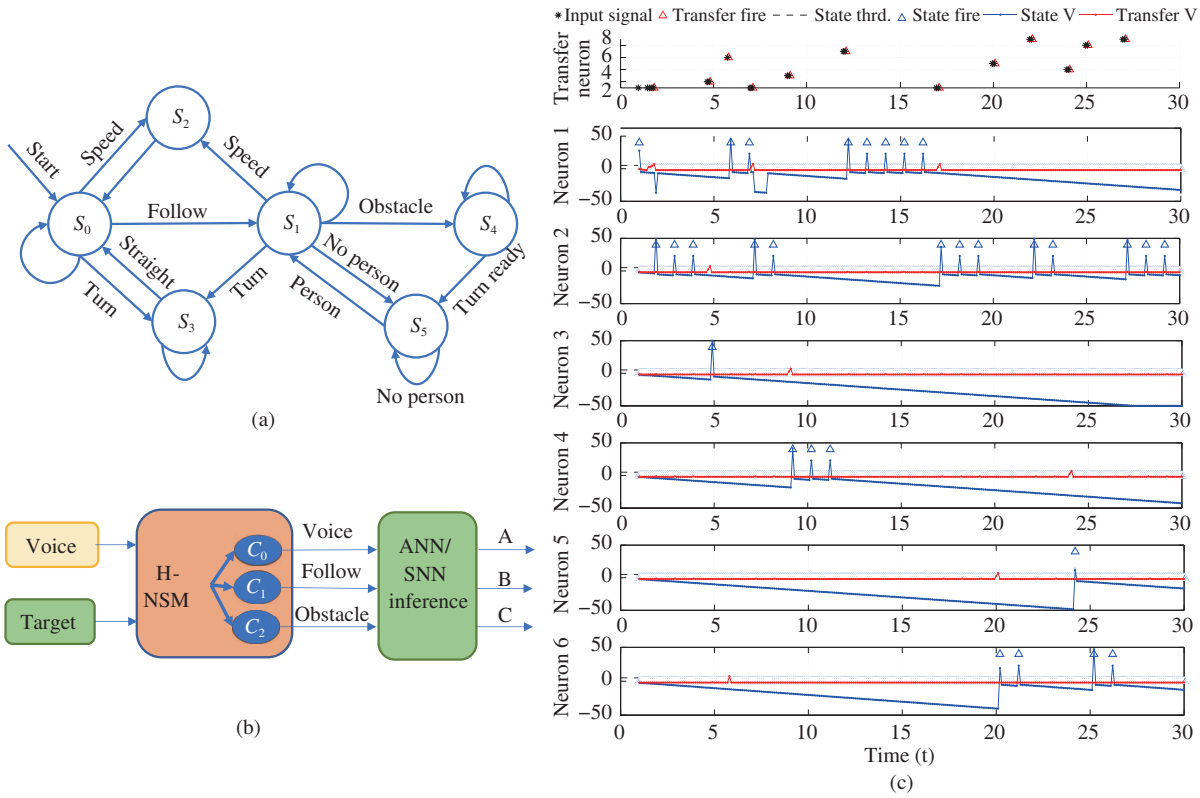


Figure 7 (Color online) The 6-state state machine for autopilot bicycle demo. (a) The state transfer rules; (b) the H-NSM-C receives the camera video streams and microphone voice streams, and controls a steering motor for following a target person, executing voice command or avoiding the obstacles; (c) neuron state and event signal recorded during the testing.

achieves the conversion between digital spikes and analog activations. The H-NSM-C part is a typical SNN model that is fully compatible with and easily implemented in Tianjic FCores. Specifically, the state transfer, trigger, and output matrices were implemented by a synaptic 'crossbar', the integration process was realized using a dendrite, and the LIF dynamics were implemented in the soma module. The weights and threshold for the H-NSM-C model were trained following the offline STDP-like rule and then programmed into Tianjic.

As depicted in Figure 7(c), we used a test sequence S_0-S_1-S_2-S_0-S_1-S_3-S_0-S_1-S_5-S_1-S_4-S_5-S_1. The bicycle executes the following instructions in sequence, which are 'follow me', 'speed up', 'follow me', 'turn left', 'go straight', 'follow me', and 'avoid obstacle'. The first row of Figure 7(c) records the input signals and transfer neurons fire events. The rest rows record the membrane potential of state neurons and some transfer neurons as shown in the legend. Only one of the state neurons fires at each time step. If the transfer neuron that connects to the state neuron fires, the state neuron that represents the new state will fire at this time step. For example, the transfer neuron 1 fires at time steps 2, 7 and 12. The state transfers from state S_0 to S_1 as shown in Figure 7(c). The membrane potential decreases after firing due to the leakage procedure. Because the spike generated by state neuron 3 (state S_2) will be delayed for one step and then transmitted to the transfer neuron, the state S_2 will always transfer to S_0 at the next time step. The bicycle only executes the acceleration command once. Then it will return to normal speed. After 50 training epochs, the transfer accuracy of the test case reaches 100%. The state transfer rules are accomplished. It will transfer from S_0 to different states according to the input information, which shows its ability to finish context-aware tasks.

6 Discussion

For a hybrid system, our H-NSM model provides the potential to develop more complex applications, which can interact with various types of neural networks and offer substantial flexibility. H-NSM can combine the advantages of ANNs (high precision, model compactness, etc.) and SNNs (temporal associ-

ation, sparse activity, energy-efficient processing, low input latency, etc.), resulting in better performance in complex tasks, in terms of versatile, high accuracy, fast response, and high efficiency. Thus, with the continuous development of new models and algorithms of ANNs and SNNs, new configurations and signal route schemes, H-NSM will become more and more powerful and may accomplish the challenging tasks that currently plague us.

Currently, the research on the theoretical framework of a hybrid neural network (HNN) is in its infancy. We believe that more researches about other kinds of H-NSM and the way to communicate with different networks will make full use of our proposed framework. Take an ANN-based H-NSM as an example. The control part is built on ANN. It may receive input from ANNs or SNNs. We need to design an SNN-to-ANN encoder and an ANN-to-SNN decoder for the controller to connect with the SNNs, which are different from those used in SNN-based H-NSM. The state representation and the state transfer rules also need to be modified accordingly. Therefore, when dealing with different tasks, the ability to integrate different networks and the simplicity of training methods are two important aspects that need to be considered for designing a tailored H-NSM.

There are also a few open issues of the H-NSM worthy to be further explored. (1) More state representation methods. In this work, we represent the state using one single neuron to build a basic model. However, there is evidence that neuronal representations in networks of neurons with synchronized activity are more stable even after lack of training or following damage [35]. (2) Coding mechanism to make the H-NSM model more powerful. In this work, we use rate coding to communicate with the ANNs as an example. There are other efficient coding schemes such as rank coding, which are worthy to be explored further. (3) Training method for online learning. In this work, the supervised learning algorithm still requires labeled information to learn the transfer rules. It will be more exciting if we can realize online learning without requiring supervised information. (4) Combination with existing methods. We hope to promote the development of relevant fields, such as model compression and hardware acceleration [36].

7 Conclusion

In summary, we have proposed a hybrid state machine framework for integrating different types of networks and controlling workflows on a hybrid intelligence system, and outlined a general rule for building the framework. An SNN-based H-NSM was demonstrated to have the ability to train different tasks and improve the robustness of the system. The training algorithm can work correctly even with 50% of the correct supervised signals. We have also demonstrated that H-NSM can accomplish sequential tasks consisting of different types of networks or actuators, and activate different branch networks according to the condition. The H-NSM-S achieves the performance comparable to that of an optimum algorithm. Besides, the H-NSM-S can be easily transferred to other similar tasks as long as we change the verification function. The H-NSM-C could be trained with only 50 epochs. It shows good convergence and stability. This work advances the development of the control logic for a hybrid hardware system that will facilitate the research and development of AGI tasks, and aid the control logic design for hybrid systems.

Acknowledgements This work was partly supported by National Natural Science Foundation of China (Grant No. 61836004), Brain-Science Special Program of Beijing (Grant No. Z181100001518006), and CETC Haikang Group-Brain Inspired Computing Joint Research Center, the Suzhou-Tsinghua Innovation Leading Program (Grant No. 2016SZ0102).

References

1 Goertzel B. Artificial general intelligence: concept, state of the art, and future prospects. J Artif General Intell, 2014, 5: 1–48
2 Pei J, Deng L, Song S, et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. Nature, 2019, 572: 106–111
3 Graves A, Wayne G, Danihelka I. Neural turing machines. 2014. ArXiv:1410.5401
4 Sutton R S, Barto A G. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 2018
5 Pierrot T, Ligner G, Reed S, et al. Learning compositional neural programs with recursive tree search and planning. 2019. ArXiv:1905.12941
6 Liang D C, Indiveri G. A neuromorphic computational primitive for robust context-dependent decision making and context-dependent stochastic computation. IEEE Trans Circ Syst II, 2019, 66: 843–847
7 Liang D C, Indiveri G. Robust state-dependent computation in neuromorphic electronic systems. In: Proceedings of IEEE Biomedical Circuits and Systems Conference (BioCAS), 2017
8 Rutishauser U, Douglas R J. State-dependent computation using coupled recurrent networks. Neural Comput, 2009, 21: 478–509
9 Neftci E, Binas J, Rutishauser U, et al. Synthesizing cognition in neuromorphic electronic systems. Proc Natl Acad Sci USA, 2013, 110: 3468–3476
10 Graves A, Wayne G, Reynolds M, et al. Hybrid computing using a neural network with dynamic external memory. Nature, 2016, 538: 471–476

11 Kaiser L, Gomez A N, Shazeer N, et al. One model to learn them all. 2017. ArXiv:1706.05137
12 Giles C L, Miller C B, Chen D, et al. Learning and extracting finite state automata with second-order recurrent neural networks. Neural Comput, 1992, 4: 393–405
13 Arai K, Nakano R. Stable behavior in a recurrent neural network for a finite state machine. Neural Netw, 2000, 13: 667–680
14 Wennekers T. Synfire graphs: from spike patterns to automata of spiking neurons. 2013. https://oparu.uni-ulm.de/xmlui/handle/123456789/2524
15 Wennekers T, Ay N. Finite state automata resulting from temporal information maximization and a temporal learning rule. Neural Comput, 2005, 17: 2258–2290
16 Clarke D A, Minsky M L. Computation: finite and infinite machines. Am Math Mon, 1968, 75: 428
17 Horne B G, Hush D R. Bounds on the complexity of recurrent neural network implementations of finite state machines. Neural Netw, 1996, 9: 243–252
18 Forcada M L, Carrasco R C. Finite-state computation in analog neural networks: steps towards biologically plausible models? In: Emergent Neural Computational Architectures, LNAI 2036, 2001. 480–493
19 Tvardovskii A S, Vinarskii E M, Yevtushenko N V. Experimental evaluation of timed finite state machine based test derivation. In: Proceedings of the 20th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices (EDM), 2019. 102–107
20 Laputenko A V. Logic circuit based test derivation for microcontrollers. In: Proceedings of the 20th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices (EDM), 2019. 70–73
21 Mavridou A, Laszka A. Designing secure ethereum smart contracts: a finite state machine based approach. In: Proceedings of International Conference on Financial Cryptography and Data Security, 2018. 523–540
22 Le L H, Bezerra C E, Pedone F. Dynamic scalable state machine replication. In: Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2016. 13–24
23 Wang J, Song J W, Chen M Q, et al. Road network extraction: a neural-dynamic framework based on deep learning and a finite state machine. Int J Remote Sens, 2015, 36: 3144–3169
24 Said W, Quante J, Koschke R. Towards interactive mining of understandable state machine models from embedded software. In: Proceedings of International Conference on Model-driven Engineering & Software Development, 2018. 117–128
25 Chen M Z, Saad W, Yin C C. Liquid state machine learning for resource and cache management in LTE-U unmanned aerial vehicle (UAV) networks. IEEE Trans Wirel Commun, 2019, 18: 1504–1517
26 Zhang Y, Li P, Jin Y, et al. A digital liquid state machine with biologically inspired learning and its application to speech recognition. IEEE Trans Neural Netw Learn Syst, 2015, 26: 2635–2649
27 Smith M R, Hill A J, Carlson K D, et al. A novel digital neuromorphic architecture efficiently facilitating complex synaptic response functions applied to liquid state machines. In: Proceedings of International Joint Conference on Neural Networks (IJCNN), 2017. 2421–2428
28 Ghasemiyeh R, Moghdani R, Sana S S. A hybrid artificial neural network with metaheuristic algorithms for predicting stock price. Cybern Syst, 2017, 48: 365–392
29 Hudson D, Manning C D. Learning by abstraction: the neural state machine. 2019. ArXiv:1907.03950
30 Takami M A, Sheikh R, Sana S S. Product portfolio optimisation using teaching-learning-based optimisation algorithm: a new approach in supply chain management. Int J Syst Sci, 2016, 3: 236–246
31 Ameri Z, Sana S S, Sheikh R. Self-assessment of parallel network systems with intuitionistic fuzzy data: a case study. Soft Comput, 2019, 23: 12821–12832
32 Deng L, Wu Y J, Hu X, et al. Rethinking the performance comparison between SNNS and ANNS. Neural Netw, 2020, 121: 294–307
33 Beers S R, Rosenberg D R, Dick E L, et al. Neuropsychological study of frontal lobe function in psychotropic-naive children with obsessive-compulsive disorder. Am J Psychiat, 1999, 156: 777–779
34 Mayer H, Perkins D. Towers of Hanoi revisited a nonrecursive surprise. Sigplan Not, 1984, 19: 80–84
35 Gonzalez W G, Zhang H, Harutyunyan A, et al. Persistence of neuronal representations through time and damage in the hippocampus. Science, 2019, 365: 821–825
36 Deng B L, Li G, Han S, et al. Model compression and hardware acceleration for neural networks: a comprehensive survey. Proc IEEE, 2020, 108: 485–532