Photonic Reinforcement Learning Based on Optoelectronic Reservoir Computing

Kazutaka Kanno (kkanno@mail.saitama-u.ac.jp)
Saitama University
Atsushi Uchida
Saitama University

Research Article

Keywords: Reinforcement learning, artificial intelligence, CartPole-v0 and MountanCar-v0 tasks, photonic reservoir computing.

Posted Date: October 28th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-988124/v1

Version of Record: A version of this preprint was published at Scientific Reports on March 8th, 2022. See the published version at https://doi.org/10.1038/s41598-022-07404-z.

Abstract

Reinforcement learning has been intensively investigated and developed in artificial intelligence in the absence of training data, such as autonomous driving vehicles, robot control, and internet advertising. However, the computational cost of reinforcement learning with deep neural networks is extremely high, and reducing the learning cost is a challenging issue. We propose a photonic on-line implementation of reinforcement learning using optoelectronic delay-based reservoir computing, both experimentally and numerically. In the proposed scheme, we accelerate reinforcement learning at a rate of several megahertz because there is no required learning process for the internal connection weights in reservoir computing. We perform two benchmark tasks, CartPole-v0 and MountanCar-v0 tasks, to evaluate the proposed scheme. Our results represent the first hardware implementation of reinforcement learning based on photonic reservoir computing and paves the way for fast and efficient reinforcement learning as a novel photonic accelerator.

Introduction

Machine learning in artificial intelligence has been the primary automation tool used in processing large amounts of data in communications and information technologies [1]. In particular, reinforcement learning is a machine learning scheme that is involved in training an action policy to maximize the total reward in a particular situation or environment. Reinforcement learning has the ability to learn in the absence of the training data, such as autonomous driving vehicles, robot control, internet advertising, and elastic optical networks [2–5]. Reinforcement learning involves training an action policy to maximize the total reward [4]. Deep neural networks [6] have often been used for reinforcement learning based on Q-learning, known as the deep Q network [3], which learns the value of the action in a particular state. However, learning the connection weights of the network using reinforcement learning with deep neural networks entails high computation costs because of the repeated training of network weights from vast playing data [7, 8], indicating the need for a large number of parameters used for learning to improve the performance of deep neural networks known as overparameterization [8–10]. Large-scale overparameterization has several hundred million parameters for deep learning [8], and the training time is required for days or weeks using the deep Q network on GPU [8]. Several techniques have been proposed to reduce learning costs, such as prioritized experienced replay [11], however, the prioritized experienced replay speeds up only by a factor of two. A more efficient implementation than deep neural networks is required for reinforcement learning.

Reservoir computing has attracted significant attention in various fields of research because it is capable of fast learning that resulted in reduced computational/training costs compared to other recurrent neural networks [12, 13]. It is a computation framework used for information processing and was originally used for supervised learning [14, 15]. The main advantage of reservoir computing is having randomly fixed connection weights between the input signal and network and between network nodes. Only the output weights (readout weights) are trained using a simple learning rule, realizing a fast learning process and enabling a reduction in its computational cost.

Recently, physical implementations of reservoir computing and its hardware implementations have been intensively studied [16–25]. Specifically, the photonic implementation of reservoir computing based on the concept of photonic accelerators [26], which consisted of a semiconductor laser and a time-delayed feedback loop, can realize fast information processing with low learning costs [27–32]. A previous study reported the realization of speech recognition at 1.1 GB/s using photonic reservoir computing [33], which can also result in the reduction of computational cost and fast processing speed. However, reservoir computing requires supervised learning to determine the readout weights, and no hardware implementation of reservoir computing for reinforcement learning has been reported yet.

In this study, we demonstrate the photonic on-line implementation of reinforcement learning based on optoelectronic delay-based reservoir computing, both experimentally and numerically. The photonic reservoir computing is implemented based on an optoelectronic time-delayed system [27, 34, 34] and is used to select an agent's action to evaluate the action-value function. The output weights in reservoir computing are trained based on the reward obtained from the reinforcement learning environment, where Q-learning is used to update the output weights in reservoir computing. We perform two benchmark tasks, CartPole-v0 and MountainCar-v0, for the evaluation of our proposed scheme. Our demonstration is a novel on-line hardware implementation of reinforcement learning based on photonic reservoir computing.

Results

Reinforcement learning based on reservoir computing

Figure 1 shows a schematic of reinforcement learning based on reservoir computing, incorporating the decision-making agent, which affects the future state of the environment by its actions, and its environment, which provides rewards to every action of the agent [4]. The objective of the agent is to maximize the total reward. However, the agent has no information regarding a good action policy. Here, we consider the action-value function $Q\left(\mathbf{s}_n, a_n\right)$ for state $\mathbf{s}_n$ and action a_n at the n-th time step [4]. The agent selects an action with the highest Q value in each state, and the total reward can be increased if the agent initially knows the value of Q. In various previous studies, the action-value function was replaced by deep neural networks, which were trained using some methods, including Q-learning [2, 3]. In this study, the action-value function is replaced by photonic delay-based reservoir computing to reduce the learning cost and realize fast processing.

Delay-based Reservoir Computing

The reservoir of delay-based reservoir computing systems consists of a nonlinear element and a feedback loop that realizes a network with many connected nodes [36]. In this scheme, the nodes in the network are virtually implemented by dividing the temporal output into short node intervals θ, resulting in

an easier implementation because it does not require the preparation of a large number of spatial nodes to construct a network.

The n-th input data into the reservoir is the state vector given by the environment $\mathbf{s}_n^T = \left(s_{1,n}, s_{2,n}, \cdots, s_{N_s,n}\right)$, where N_s is the number of state elements, and the superscript T represents the transpose operation. The state is initially preprocessed via the masking procedure, which is defined in Eq. (1), before injecting into the reservoir [36, 37], where the state is multiplied by a mask matrix $\mathbf{M}$. The value of the mask is randomly obtained from a uniform distribution of $[-1, 1]$.

$$\mathbf{u}_n^T = \left(\mu s_{1,n}, \mu s_{2,n}, \ldots, \mu s_{N_s,n}, b\right)\mathbf{M} = \left(\mu \mathbf{s}_n^T, b\right)\mathbf{M}, \quad (1)$$

where $\mathbf{M}$ is the mask matrix, which is an $N \times \left(N_s + 1\right)$ matrix, and μ is the scaling factor for the input state $\mathbf{s}_n$, and N is the number of nodes in the reservoir. The elements of the preprocessed input vector $\mathbf{u}_n$ are given by $u_{i,n}$, which correspond to the input data into the i-th virtual node in the photonic reservoir computing. Moreover, a fixed bias b is added to Eq. (1), which prevents the signal $\mathbf{u}_n$ from being equal to zero when the elements of state $\mathbf{s}_n$ are close to zero and can lead to different nonlinearities for each virtual node.

We consider the input data $u_{i,n}$ for the i-th virtual node defined as $\mu\left(m_{1,i}s_{1,n} + m_{2,i}s_{2,n} + \cdots + m_{N_s,i}s_{N_s,n}\right) + bm_{N_{s+1},i}$, where $m_{p,q}$ is an element of the mask matrix $\mathbf{M}$ in row p and column q. The representation of $u_{i,n}$ indicates that the input data for the i-th node oscillates with the center on the bias $bm_{N_{s+1},i}$. The center point of the oscillation in the input data is different for each node because the element $m_{N_{s+1},i}$ of the mask matrix is different for each node. A different part of the nonlinear function that represent the relationship of the input and output in the reservoir is used for each node because of the bias $bm_{N_{s+1},i}$, leading to different nonlinearities for each node. Therefore, adding an input bias can enhance the approximation of the reservoir.

An input signal injected into the reservoir is generated by temporally stretching the elements of $\mathbf{u}_n$ to the node interval θ as follows:

$$u(t + (n-1)T_m) = u_{i,n} \quad \left((i-1)\theta \le t < i\theta\right), \quad (2)$$

where T_m is the signal period of each input data and is called the mask period, which corresponds to the product of the number of nodes N and the node interval θ ($T_m = N\theta$). The input signal $u(t)$ is injected into the reservoir to generate a response signal, where the virtual nodes are extracted by dividing them by a small interval θ. The number of virtual nodes N corresponds to the number of elements in $\mathbf{u}_n$.

The output of reservoir computing is calculated from the weighted linear combination of virtual node states, which are extracted from the temporal output of the reservoir by dividing the output by the node interval θ, and is considered as $Q\left(\mathbf{s}_n, a\right)$ for reinforcement learning. The output related to action a is represented as $Q\left(\mathbf{s}_n, a\right)$, which is defined as:

$$Q\left(\mathbf{s}_n, a\right) = \sum_{j=1}^{N} w_{a,j} v_{j,n} = \mathbf{w}_a^T \mathbf{v}_n, \quad (3)$$

where $\mathbf{v}_n$ is the vector of the node states for the n-th input and $\mathbf{w}_a$ is the weight vector for action a. The number of reservoir outputs corresponds to the number of agents' actions during reinforcement learning. In reinforcement learning, the action with the highest Q value is selected. In the following subsection, $\mathbf{w}_a$, which is trained based on the Q-learning in reinforcement learning, is explained and discussed.

Training For Reservoir Weights Using Reinforcement Learning

Q-learning is a well-known training algorithm for reinforcement learning [4]. In this study, we use this training algorithm to train the reservoir weights. Moreover, the update rule based on Q-learning is called off-policy learning [4], which shows that the action used for training differs from the agent's actual action. In the Q-learning method, the maximum Q value $\max_a Q\left(\mathbf{s}_{n+1}, a\right)$ for action a at the next state $\mathbf{s}_{n+1}$ is used, and the actual action is not always used for training. In our scheme, $Q\left(\mathbf{s}_n, a\right)$ is approximated using reservoir computing by considering a one-step temporal difference error $\delta_n = r_{n+1} + \gamma \max_a Q\left(\mathbf{s}_{n+1}, a\right) - Q\left(\mathbf{s}_n, a_n\right)$ and the square of the temporal difference error as the loss function. Then, the update rule for the reservoir weights is described as:

$$\mathbf{w}_{a_n} \leftarrow \mathbf{w}_{a_n} + \alpha \left[r_{n+1} + \gamma \max_a \mathbf{w}_a^T \mathbf{v}_n - \mathbf{w}_{a_n}^T \mathbf{v}_n \right] \mathbf{v}_n, \quad (4)$$

where α is the constant step-size parameter and γ is the discount rate for a future expected reward. These hyperparameters should be appropriately selected for a successful computation. We set α as a small positive value and is related to the training speed. Moreover, γ is set to a positive value of less than 1.

Furthermore, we employ the experienced replay method to train the reservoir weights [38]. In this method, the observed data (state, action, and reward) are preserved in the memory, sampled randomly, and used for training. The randomly sampled data is referred to as the mini-batch. The size of the mini-batch and the number of preserved data are hyperparameters. Using the randomly sampled and preserved data for

training may reduce the correlation between the data used for training and exhibits easier convergence of the Q-learning.

Moreover, we use the ϵ-greedy method for the action selection in which the actions are randomly sampled using the probability of ϵ, which is initially set to 1 to indicate that the agent first takes a random action. Then, the value is reduced as the number of episodes for the reinforcement-learning task increased. The value of ϵ is updated by $\epsilon = \epsilon_0 + \left(1 - \epsilon_0\right)\exp\left(-k_\epsilon n_{ep}\right)$, where n_{ep} is the episode index of the reinforcement learning task and k_ϵ is the attenuation coefficient. The value of ϵ converges to the value ϵ_0, which is fixed at 0.01, as the number of episodes increases.

Scheme For Optoelectronic Reservoir Computing

In this study, we use an optoelectronic delay system shown in Fig. 2(a), which is commonly applied to explore complex phenomena such as dynamical bifurcation, chaos, and chimera states [39–41], as a reservoir. Moreover, the application of this system in physical reservoir computing has also been studied [42, 43]. The system is composed of a laser diode (LD), a Mach-Zehnder modulator (MZM), and an optical fiber for delayed feedback. In particular, the modulator provides a nonlinear transfer function $\cos^2(\cdot)$ from the electrical inputs to the optical outputs. The optical signal is transmitted through the optical fiber with a delay time of τ and is transformed into an electric signal using a photodetector (PD). The electric signal is fed back to the MZM after the signal passes through an electric amplifier (AMP). An input signal for reservoir computing is injected into the reservoir by coupling with the feedback signal. The temporal dynamics of the system are described using simple delay differential equations [41]. We use delay differential equations for the numerical verifications of the proposed scheme, which are described in the Methods section.

In our experiment, we employ a system similar to the scheme shown in Fig. 2(a), except for the absence of the delayed feedback, as shown in Fig. 2(b). Thus, the proposed system is considered as an extreme learning machine, which has been studied as a machine-learning scheme [44]. The digital oscilloscope (OSC) and arbitrary waveform generator (AWG) were controlled by a personal computer (PC). Initially, the state of the reinforcement learning task was calculated using the PC. Then, an input signal was generated from the state by applying a masking procedure for reservoir computing. The input signal was transferred to the AWG, which produced the signal, from the PC. The signal was amplified by AMP and injected into the MZM. The optical output of the MZM was modulated based on the injected signal. The optical signal was transformed into an electric signal at the PD. The electric signal was acquired by the OSC and was then transferred to the PC. The node states of the reservoir were extracted from the signal. The output of reservoir computing was calculated from the weighted sum of the node states, which corresponded to a Q value for each action in a reinforcement learning task. An action was selected based on the Q values, and the state of the reinforcement learning task was updated based on the selected

action. In addition, the reservoir weights were updated based on Q-learning. The above procedure was repeated until the reinforcement learning task was terminated. This procedure for reinforcement learning was executed under the control of OSC, AWG, and PC in an on-line manner.

In our numerical simulation and experiment, the number of nodes N is 600, and the node interval θ is 0.4 ns. Then, the mask interval T_m is given as $T_m = N\theta = 240$ ns. The feedback delay time is set similar as the mask interval in various studies on delay-based reservoir computing [36]. Moreover, it has been reported that the slight mismatch between the delay time and the mask interval enhances the performance of information processing [27, 45]. Therefore, we set the feedback delay time to $\tau = 236.6$ ns, which is related to the mask interval and the node interval by $T_m = \tau + \theta$.

Experimental And Numerical Results On Benchmark Tasks

We evaluate our reinforcement learning scheme based on delay-based reservoir computing using the reinforcement learning task, CartPole-v0 [46] in OpenAI Gym [47]. An un-actuated joint attaches a pole to a cart that moves along a frictionless track. The goal of the task is to keep the pole upright during an episode, which has a length of 200 time steps. A reward of $+1$ is provided for every time steps in which the pole remains upright. The task is solved when the pole remained upright for 100 consecutive episodes. The details of the task are described in the Methods section. The hyperparameters for reinforcement learning are fixed at $\alpha = 0.000400$, $\gamma = 0.995$, and $k_\epsilon = 0.04$, respectively. The number of memories and the size of the mini-batch for experience replay were 4000 and 64, respectively.

Figure 3(a) shows the numerical results of the total reward for each episode, indicating that if the total reward obtained is 200, the pole remains upright over an episode. Meanwhile, a total reward of less than 200 indicates that the pole falls in the middle of an episode. As shown in Fig. 3, the input bias is applied ($b = 0.80$) as represented by the black curve. The pole cannot be kept upright when the episode index was small. On the other hand, the pole become upright over an episode as the number of episodes increases. The CartPole-v0 task is successfully solved since a total reward of 200 is obtained from each 100 consecutive episodes. Moreover, the total reward does not reach 200, and the pole cannot keep upright for all episodes when the input bias is not applied ($b = 0.0$, the red curve), indicating the necessity of introducing the input bias to solve the task. When the input bias is not introduced, we observe that a selected action over an episode is determined based on the initial state. The push to the right (left) is only selected over an episode if the pole angle is positive (negative), which results in the immediate fall of the pole. When the input bias is introduced, an action that prevents the pole from falling is selected after several episodes, enhancing the ability to identify an input state. The reservoir can be trained to prevent the pole from falling because it can identify the inputs of different states when the input bias is introduced.

Figure 3(b) shows the experimental results in which the total reward reaches 200 at the 110th episode and does not vary until the 300th episode, indicating that the task is successfully solved in the experiment. Moreover, we found that reaching the total reward to 200 is slower in the experimental result than that in the numerical result. The discrepancy between the numerical and experimental results is caused by the existence of the measurement noise in the experiment, which may perturb the estimated Q value of the reservoir. Moreover, an incorrect action may be selected in the experiment because of the influence of noise when the difference between the Q values of the two actions is too small. Therefore, it is necessary to learn the Q values until the difference in the Q values becomes sufficiently large to ensure the proper selection of the correct action in a noisy environment.

In addition, the system had no time-delayed feedback in the experiment. If the reservoir has time-delayed feedback, it would have a memory effect that would become advantageous since the reservoir with memory can learn the state-action value function including the past and current states. This is equivalent to expanding the dimension of the input state space, making it possible to approximate a more complex state-action value function and achieve a higher total reward by adding time-delayed feedback. Nevertheless, the time evolution of the state is uniquely determined only from the current state for the benchmark task used in this study, and this task can still be solved using a reservoir without time-delayed feedback in the experiment.

We emphasize that one action (episode) of reinforcement learning can be potentially determined by the processing rate of reservoir computing at a frequency of 4.2 MHz in our scheme, where one virtual network is constructed from a time series with $N\theta = 240$ ns (N is 600 and θ is 0.4 ns). This is much faster than conventional reinforcement learning, and we can further increase the processing speed by decreasing the node interval θ with a faster photonic dynamical system. In addition, the size of the trained parameters (600) is smaller than that for deep neural networks (e.g., 480 M parameters for ImageNet [8–10]). The hardware implementation of photonic reservoir computing is promising for realizing fast and efficient reinforcement learning.

We demonstrated another benchmark task called the MountainCar-v0 task, which is provided by OpenAI Gym [46]. This task aims to make a car reach the top of the mountain by accelerating the car to the right or left. A reward of -1 is given for every step until an episode ends, consisting of 200 steps. An episode is terminated if the cart reaches the top of the mountain. Therefore, a higher value of the total reward is obtained if the cart reaches the top of the mountain faster. Solving this task is defined as obtaining an average reward of -110.0 for 100 consecutive trials [44]. The hyperparameters for reinforcement learning are fixed at $\alpha = 0.000010$, $\gamma = 0.995$, and $k_\epsilon = 0.04$, respectively. The number of memories and the size of the mini-batch for experience replay are 4000 and 256, respectively.

Figure 4(a) shows the numerical results of the total rewards for each episode. The black curve represents the total reward for each episode while the red curve represents the moving average of the total reward calculated from the past 100 episodes. The total reward is -200 in the first several episodes, indicating that the car does not reach the top of the mountain at all. However, as the number of episodes increases,

the car is able to reach the top of the mountain. The average reward increases as the number of episodes increases and exceeds -110 during the 267th episode, indicating that the task is solved using our scheme.

Moreover, the experimental results are shown in Fig. 4(b). The moving average (red curve) increases as the number of episodes increases. However, the moving average does not reach the blue dashed line (total reward of -110). In this task, a larger value of the total reward ranges from -120 to -80, which depends on the initial condition of the state in the task. In particular, the moving average becomes larger if the length of consecutive episodes where a large total reward is obtained is long. The length in the experimental result (the episode from 170th to 192nd) is shorter than the numerical result (the episode from 197th to 249th). Thus, the moving average in the experiment does not increase as much as that in the numerical simulation.

We conduct another experiment using the MoutainCar-v0 task. We use a reservoir weight, which is not updated to solve the task, trained in the experiment, as shown in Fig. 4(b). Figure 4(c) shows the total reward for each episode, where the weight trained until the 180th episode in Fig. 4(b) is used. The moving average (red curve) exceeds -110 at the 141st episode. Moreover, the average total reward of consecutive 100 episodes exceeds -110, and the task is solved. The results indicate that the trained weight properly works even if the conditions on the setup are slightly changed, such as the detected power at the PD. Therefore, the trained weight is robust against perturbations in the system parameters.

We numerically investigate the dependence of the performance on the input bias b in the MountainCar-v0 task. In Fig. 5(a), the red solid curve represents the maximum of the moving average of the total reward in 1000 episodes. The total reward is averaged for 10 trials, with each trial consisting of 1000 episodes. The total reward is nearly equal to zero for a small input bias value ($b \leq 0.5$). A large value of the total reward is obtained for a large input bias value ($b > 0.5$). This result indicates that an input bias is necessary for solving the task. An input bias with a value close to 1 is suitable for increasing the total reward, which can be related to the normalized half-wave voltage (V_{π}) of the MZM, where the normalized voltage is equal to 1 in our numerical simulation. An input bias nearly equal to 1 can produce the nonlinearity of the MZM ($\cos^2(\cdot)$). It is considered that nonlinearity can assist in identifying different input states.

We also investigate the effect of the time-delayed feedback in the numerical simulation. In Fig. 5(a), the blue dashed curve represents the case in which the reservoir has no delayed feedback. At $b = 0.85$, a total reward of -130.29 is obtained. Thus, the reservoir can be successfully trained for the car to reach the top of the mountain, even if the reservoir has no delayed feedback. However, the performance is lower than in the case with delayed feedback (the red solid curve), which indicates that the delayed feedback can enhance the performance of the task.

For a more detailed investigation, we show the dependence of the total reward on the feedback strength, as shown in Fig. 5(b). For the three curves in Fig. 5(b), different input bias values are used. In the black ($b = 0.90$) and red ($b = 0.70$) curves, a large value of the total reward is obtained at a feedback strength

of approximately 1. However, the total reward decreases as the feedback strength increases to more than 1. When the feedback strength becomes larger than 1, the temporal dynamics of the optoelectronic system change from a steady state to a periodic oscillation, even though the system has no input signal. If the reservoir dynamics is not in a steady state, the temporal responses of the reservoir can be different when the same signal is repeatedly injected, which is called the consistency property of dynamical systems [48]. If the reservoir has no consistency, the performance deteriorates because the same state cannot be identified. In a slightly small value of the input bias (blue curve, $b = 0.50$), the total reward is almost equal to -200 for different feedback strengths. This result indicates that the adjustment of the feedback strength cannot enhance the performance if the input bias is too small.

Discussion

We experimentally and numerically demonstrated the on-line implementation of reinforcement learning based on photonic reservoir computing, which consists of a laser diode, a Mach-Zehnder modulator, and a fiber delay line. We demonstrated two benchmark tasks, including CartPole-v0 and MoutainCar-v0 tasks, using our proposed scheme. The results show that the state-action value function in reinforcement learning can be trained, and their tasks can be solved using photonic reservoir computing. In the experiment, the photonic reservoir computing without feedback (i.e., an extreme learning machine) was implemented. To the best of our knowledge, this is the first on-line hardware implementation of reinforcement learning based on photonic reservoir computing. In particular, reservoir computing is used to approximate the Q-function, and the output weights of the reservoir are trained by Q-learning. The high-dimensional mapping between the states and Q-values for reinforcement learning can be easily trained by reservoir computing. The speed of one action is determined by the processing rate of reservoir computing at 4.2 MHz (240 ns) in our experiment.

We introduced an input bias for preprocessing the input state in reinforcement learning. Our results show that the input bias, which has the same role as the bias introduced in the general neuron models that controls the firing frequency, is necessary for solving the reinforcement tasks in our scheme. In this study, the activation function of the virtual node of the reservoir is $\cos^2(\cdot)$, and an input bias is used to control the initial position of $\cos^2(\cdot)$ function. For example, if the input bias is set near the extreme value of $\cos^2 n\theta$ ($\theta = 0, \pm\pi/2$, and n is an integer), the reservoir does not respond well with respect to the change in the input signal. In contrast, when the input bias is set to a quadrature point ($\theta = \pm\pi/4$), the reservoir shows a large response with respect to the change in the input signal. Therefore, it is possible to adjust the sensitivity of the virtual nodes to the input signal by changing the input bias. In the presence of an input bias, different input states can be distinguished well, which can enhance the performance of reinforcement learning based on reservoir computing. We show that input bias has a significant effect on reinforcement learning in our scheme.

The hardware implementation of reinforcement learning based on photonic reservoir computing is promising for fast and efficient reinforcement learning as a novel photonic accelerator, and can be

applied for edge computing in real-time distributed control and communication path selection in optical communication.

Methods

CartPole-v0 task

The CartPole-v0 task is a benchmark task for reinforcement learning given by the OpenAI Gym [46]. In this task, we consider a pole attached by an un-actuated joint to a cart that moves along a frictionless track with four states: cart position, cart velocity, pole angle, and pole velocity at the tip. These states are initialized to uniform random values. The agent's action is to push the cart to the right (+1) and left (-1). The goal of the task is to keep the pole upright during an episode with a length of 200 timesteps, and the task is considered solved when the pole remains upright for 100 consecutive episodes. A reward of +1 is provided for every timesteps in which the pole remains upright. The episode ends when the pole is more than 12 ° from the vertical or the cart position moves more than 2.4 units or less than -2.4 units from the center. The magnitudes of the cart position and pole velocity were normalized to the range of [-1.0 to 1.0] before injecting into the reservoir.

Mountaincar-v0 Task

The MountainCar-v0 task is also provided by OpenAI Gym [47]. The goal of this task is for a car to reach the top of the mountain by accelerating the car to the right or left. The observable states of the task are the cart position and cart velocity. In the initial state, the cart position is randomly set from a uniform distribution [-0.6 to -0.4], and the cart velocity is fixed at zero. The agent's action is to push the cart to the left (0), neutral (1), and push the cart to the right (2). A reward of -1 is given for every step until an episode, which consists of 200 steps, ended. An episode is terminated if the cart reaches the top of the mountain.

Numerical Method

Optoelectronic delay systems [39] have been studied for delay-based reservoir computing [27, 34, 35], using the following delay differential equations [40]:

$$\tau_L \frac{dx(t)}{dt} = -\left(1+\frac{\tau_L}{\tau_H}\right)x(t) - y(t) + \beta\cos^2\left[\kappa x(t-\tau) + \frac{\pi}{4}u(t) + \varphi_0\right] + \xi(t), \tag{5}$$

$$\tau_H \frac{dy(t)}{dt} = x(t), \tag{6}$$

where x is the normalized output of MZM, τ_L and τ_H are the time constants describing the low-pass and high-pass filters related to the frequency bandwidths of the system components, relatively, β is the feedback strength (dimensionless), φ_0 is the offset phase of MZM, $u(t)$ is the input signal injected into the reservoir, and $\xi(t)$ is the white Gaussian noise with properties $\langle \xi(t) \rangle = 0$ and $\langle \xi(t)\, \xi\left(t_0\right) \rangle = \delta\left(t - t_0\right)$, where $\langle \cdot \rangle$ denotes the ensemble average and $\delta(t)$ is Dirac's delta function. Parameter values used in this study are shown in Table 1.

Table 1
Parameter values used in numerical simulations.

Symbol	Parameter	Value
$\left(2\pi\tau_L\right)^{-1}$	Low-pass cutoff frequency	12.5×10^9 Hz
$\left(2\pi\tau_H\right)^{-1}$	High-pass cutoff frequency	0.625×10^6 Hz
τ	Feedback delay time	239.6×10^{-9} s
β	Dimensionless gain	1.0
κ	Dimensionless feedback strength	0.9
φ	Bias point for MZM	-0.25π rad
μ	Scaling coefficient for input state	0.6

Experimental Setup

The experimental setup used in the experiments is shown in Fig. 2(b). The system has no delayed feedback for simple implementation. Thus, the system corresponds to an extreme learning machine, which has been studied as a machine-learning scheme [44]. A distributed-feedback laser diode (LD, NTT electronics, NLK1C5GAAA) with an optical wavelength of 1547 nm was used as the optical source. The lasing threshold of the LD was 11.6 mA, and the driving current was 30.00 mA. The optical output of the LD was injected into a Mach-Zehnder modulator (MZM, EO Space, AX-0MKS-20-PFA-PFA-LV-UL), where a bias controller (BC, IXBlue, MBC-AN-LAB) was inserted to stabilize the operation bias of the MZM, which was fixed at the quadrature point. Moreover, a modulation signal was generated from an arbitrary waveform generator (AWG, Tektronix, AWG70002A, 25 GSample/s, 10 bit vertical resolution) and transferred to the MZM after amplification by an electric amplifier (AMP, IXBlue, DR-AN-10-HO). A photodetector (PD, Newport, 1554-B, 12-GHz bandwidth) was used to detect the optical signal of the MZM, and the detected power was 0.280 mW when there was no modulation input. The detected signal at the PD was transferred to a digital oscilloscope (OSC, Tektronix, DPO72304SX, 23 GHz bandwidth) and sampled at 50 GSample/s.

The signal amplitude injected into the MZM and the half-wave voltage V_π of the MZM are important for successful computation. The signal amplitude is determined from the input scaling μ and the bias scaling b, the output amplitude of the AWG, and the amplification gain of the AMP. The output amplitude of the AWG is 0.30 V at peak-to-peak. The amplification gain of the AMP is typically 30 dB under small-signal conditions. The half-wave voltage of the MZM was $V_\pi = 4$ V. The input signal was preprocessed using Eq. (1) in PC, where the input scaling μ and the bias scaling b are fixed at $\mu = 0.50$ and $b = 0.40$, respectively. These parameter values are different from those used for the numerical simulation because the signal amplitude in the experiments depends on these parameter values and the condition of the output amplitude of the AWG. In our experiments, the values of $\mu = 0.50$ and $b = 0.40$ produce an electric signal with an amplitude nearly equal to the half-wave voltage of the MZM. The condition of the bias scaling b is consistent with the value for successful computation in our numerical simulation, as shown in Fig. 5.

Declarations

Acknowledgments

This study was supported in part by JSPS KAKENHI (JP19H00868 and JP20K15185), JST CREST JP-MJCR17N2, and the Telecommunications Advancement Foundation.

Author contributions

All authors have contributed to development and/or implementation of the concept.

K. K. performed the numerical simulations and analyzed the data. K. K. and A. U. contributed to the discussion of the results. K. K. and A. U. contributed to writing the manuscript.

Competing Interests

The authors declare that they have no competing interests.

References

1. Andrae, A. & Edler, T. On Global Electricity Usage of Communication Technology: Trends to 2030. Challenges 6, 117-157 (2015).
2. Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484-489 (2016).
3. Mnih, V. et al. Human-level control through deep reinforcement learning. Nature 518, 529-533 (2015).
4. Sutton, Richard S. & Barto, Andrew G., Reinforcement Learning: An Introduction (The MIT Press, Cambridge, 2018), 2nd ed.
5. Chen, X. et al. DeepRMSA: a deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks. Journal of Lightwave Technology, 37, 4155-4163

(2019).

6. Steven, K. E. et al. Convolutional networks for fast, energy-efficient neuromorphic computing. Proc. Natl. Acad. Sci. 113, 11441-11446 (2016).
7. Graves, A. et al. Hybrid computing using a neural network with dynamic external memory. Nature 538, 471-476 (2016).
8. Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F., The computational limits of deep learning. Preprint at https://arxiv.org/abs/2007.05558v1 (2020).
9. Soltanolkotabi, M., Javanmard, A., & D Lee, J., Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. IEEE Transactions on Information Theory 65, 742–769 (2019).
10. Xie, Q., Minh-Thang, L., Eduard, H., & Quoc V., L., Self-Training With Noisy Student Improves ImageNet Classification. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10687-10698 (2020).
11. Schaul, T., Quan, J., Antonoglou, I., & Silver, D., Prioritized experience replay. Preprint at https://arxiv.org/abs/1511.05952 (2016).
12. Chang, H. & Futagami, K., Reinforcement learning with convolutional reservoir computing. Appl. Intell. 50, 2400-2410 (2020).
13. Szita, I., Gyenes, V., & Lőrincz, A., Reinforcement Learning with Echo State Networks. ICANN2006 4131, 830-839 (2006).
14. Jaeger, H. & Haas, H., Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304, 78-80 (2004).
15. Lukoševičius, M., & Jaeger H., Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. 3, 127-149 (2009).
16. Tanaka, G. et al. Recent advances in physical reservoir computing: A review. Neural Networks 115, 100-123 (2019).
17. Snyder, D., Goudarzi, A., & Teuscher, C., Computational capabilities of random automata networks for reservoircomputing. Phys. Rev. E 87, 042808 (2013).
18. Coulombe, J. C., York, M. C. A., & Sylvestre, J., Computing with networks of nonlinear mechanical oscillators. PLoS ONE 12, e0178663 (2017).
19. Katayama, Y., Yamane, T., Nakano, D., Nakane, R., & Tanaka, G., Wave-based neuromorphic computing framework for brain-like energy efficiency and integration. IEEE Transactions on Nanotechnol. 15, 762-769 (2016).
20. Torrejon, J. et al. Neuromorphic computing with nanoscale spintronic oscillators. Nature 547, 428-431 (2017).
21. Nakajima, K., Hauser, H., Li, T., & Pfeifer, R., Information processing via physical soft body. Sci. Rep. 5, 10487 (2015).

22. Shastri, B. J. et al. Photonics for artificial intelligence and neuromorphic computing. Nat. Photon. 15, 102-114 (2021).
23. Genty, G. et al. Machine learning and applications in ultrafast photonics. Nat. Photon. 15, 91-101 (2021).
24. Moughames, J. et al. Three-dimensional waveguide interconnects for scalable integration of photonic neural networks. Optica 7, 640-646 (2020).
25. Estébanez, I., Schwind, J., Fischer, I., & Argyris, A., Accelerating photonic computing by bandwidth enhancement ofa time-delay reservoir. Nanophotonics 9 20200184 (2020).
26. Kitayama, K. et al. Novel frontier of photonics for data processing—photonic accelerator. APL Photonics 4, 090901 (2019).
27. Paquot, Y. et al. Optoelectronic reservoir computing. Sci. Rep. 2, 287 (2012).
28. Martinenghi, R., Rybalko, S., Jacquot, M., Chembo, Y. K., & Larger, L., Photonic nonlinear transient computing with multiple-delay wavelength dynamics. Phys. Rev. Lett. 108, 244101 (2012).
29. Bueno, J., Brunner, D., Soriano, M. C., & Fischer, I., Conditions for reservoir computing performance using semiconductor lasers with delayed optical feedback. Opt. Express 25, 2401-2412 (2017).
30. Duport, F., Schneider, B., Smerieri, A., Haelterman, M. & Massar, S., All-optical reservoir computing. Opt. Express 20, 22783-22795 (2012).
31. Sugano, C., Kanno, K., & Uchida, A., Reservoir computing using multiple lasers with feedback on a photonic integrated circuit. IEEE J. Sel. Top. Quantum Electron. 26, 1500409 (2020).
32. Antonik, P., Marsal, N., Brunner, D., & Rontani, D., Human action recognition with a large-scale brain-inspired photonic computer. Nat. Mach. Intell. 1, 530-537 (2019).
33. Brunner, D., Soriano, M. C., Mirasso, C. R., & Fischer, I., Parallel photonic information processing at gigabyte per second data rates using transient states. Nat. Commun. 4, 1364 (2013).
34. Larger, L. et al. Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. Opt. Express 20, 3241-3249 (2012).
35. Larger, L. et al. High-speed photonic reservoir computing using a time-delay-based architecture: Million words per second classification. Phys. Rev. X 7, 011015 (2017).
36. Appeltant, L. et al. Information processing using a single dynamical node as a complex system. Nat. Commun. 2, 468 (2011).
37. Soriano, M. C. et al. Optoelectronic reservoir computing: tackling noise-induced performance degradation. Opt. Express 21, 12-20 (2013).
38. O'Neill, J., Pleydell-Bouverie, B., Dupret, D., & Csicsvari, J., Play it again: reactivation of waking experience and memory. Trends Neurosci. 33, 220-229 (2010).
39. Larger, L. & Dudley, J. M., Nonlinear dynamics: Optoelectronic chaos. Nature 465, 41-42 (2010).
40. Chembo, Y. K., Brunner, D., Jacquot, M. & Larger, L. Optoelectronic oscillators with time-delayed feedback. Rev. Mod. Phys. 91, 035006 (2019).

41. Murphy, T. E. et al. Complex dynamics and synchronization of delayed-feedback nonlinear oscillators. Phil. Trans. R. Soc. A 368, 343-366 (2010).
42. Antonik, P. et al. Online training of an opto-electronic reservoir computer applied to real-time channel equalization. IEEE Transactions on Neural Networks Learn. Syst. 28, 2686-2698 (2017).
43. Duport, F., Smerieri, A., Akrout, A. Haelterman, M., & Massar, S., Fully analogue photonic reservoir computer. Sci. Rep. 6, 22381 (2016).
44. Ortín, S. et al. Aunified framework for reservoir computing and extreme learning machines based on a single time-delayed neuron. Sci. Rep. 5, 14945 (2015).
45. Stelzer, F., Röhm, A., Lüdge, K., & Yanchuk, S., Performance boost of time-delay reservoir computing by non-resonant clock cycle. Neural Networks 124, 158-169 (2020).
46. Barto, A. G., Sutton, R. S., & Anderson, C. W., Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Syst. Man, Cybern. SMC-13, 834-846 (1983).
47. Brockman, G. et al. OpenAI Gym. Preprint at https://arxiv.org/abs/1606.01540 (2016).
48. Uchida, A., McAllister, R., & Roy, R. Consistency of nonlinear system response to complex drive signals. Phys. Rev. Lett. 93 244102 (2004).
49. Antonik, P. et al. Random Pattern and Frequency Generation Using a Photonic Reservoir Computer with Output Feedback. Neural Processing Letters 47, 1041-1054 (2018).

Figures

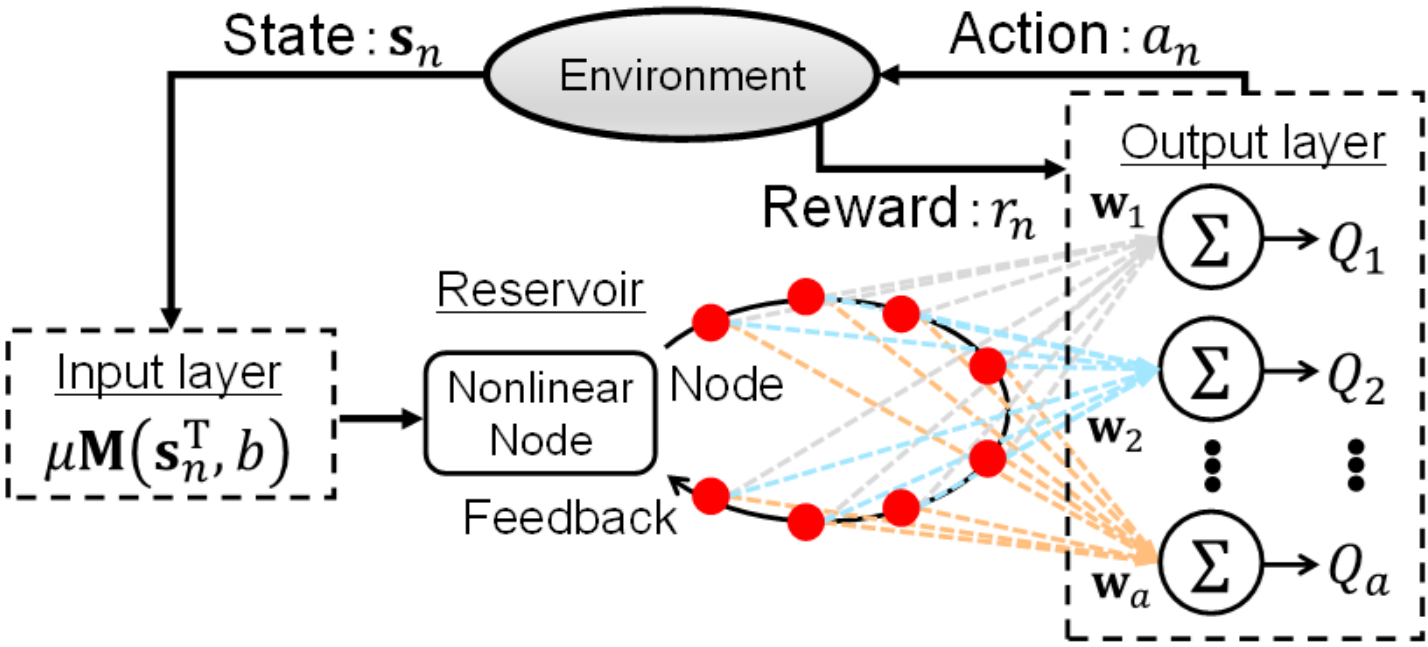

Figure 1

chematic diagram of reinforcement learning based on delay-based reservoir computing.

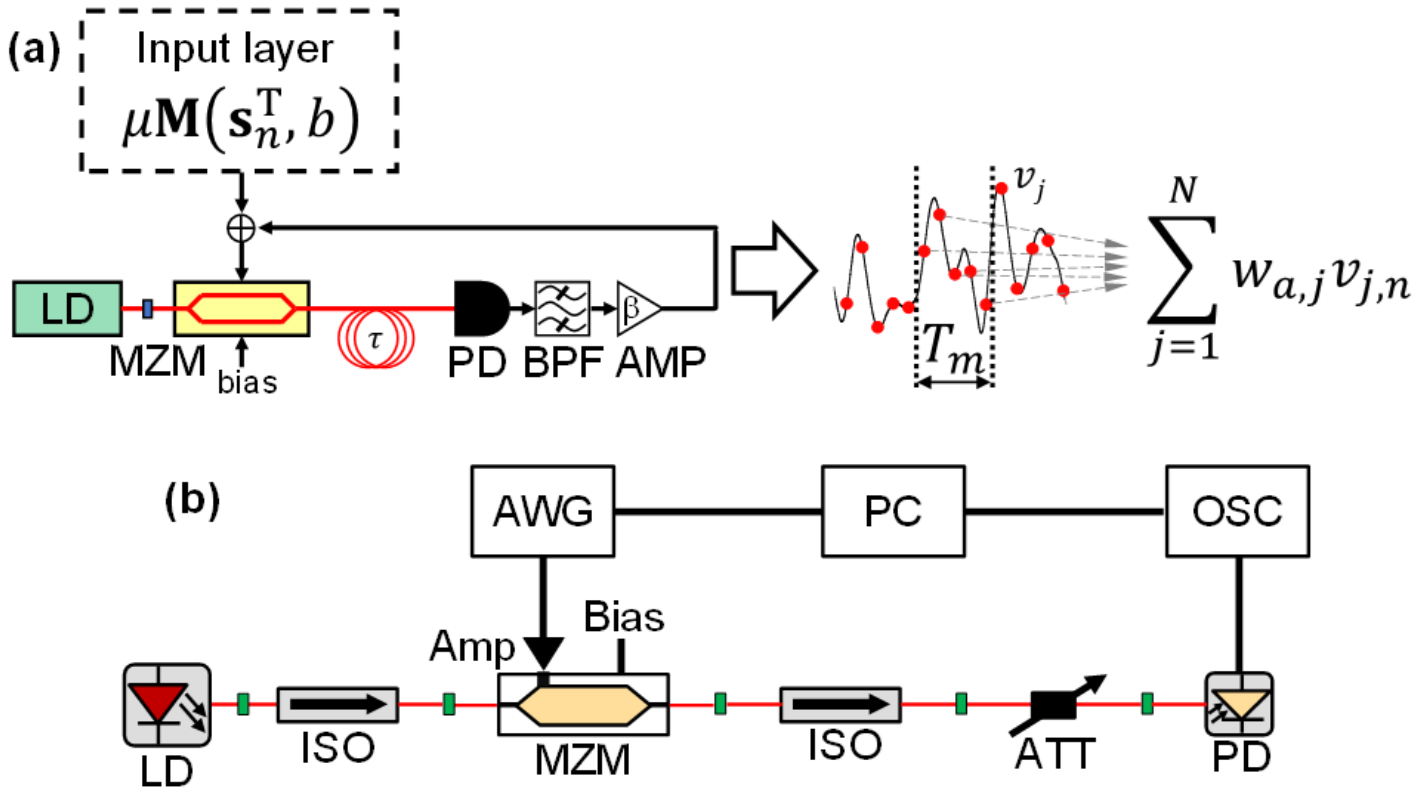

Figure 2

(a) Schematic diagram of the optoelectronic delay system for reservoir computing. An input signal is preprocessed before injecting into the reservoir and added to a feedback signal. Reservoir node states are extracted from the temporal response of the reservoir and are shown as red circles. In the schematic diagram, MZM is the Mach-Zehnder modulator, PD is the photodetector, BPF is the bandpass filter, and AMP is the electric amplifier. (b) Experimental setup for reinforcement learning. The system has no delayed feedback, and the detected signal at the PD is not fed back to the MZM. In the personal computer (PC), environmental states in reinforcement learning tasks are calculated and the masking procedure is applied. The input data preprocessed in the PC is transferred to the arbitrary waveform generator (AWG). The optical signal from the MZM is detected at the PD, and the detected power is adjusted using the optical attenuator (ATT). The detected signal at the PD is measured at the digital oscilloscope (OSC). The AWG and the OSC are controlled by the PC in an on-line manner.

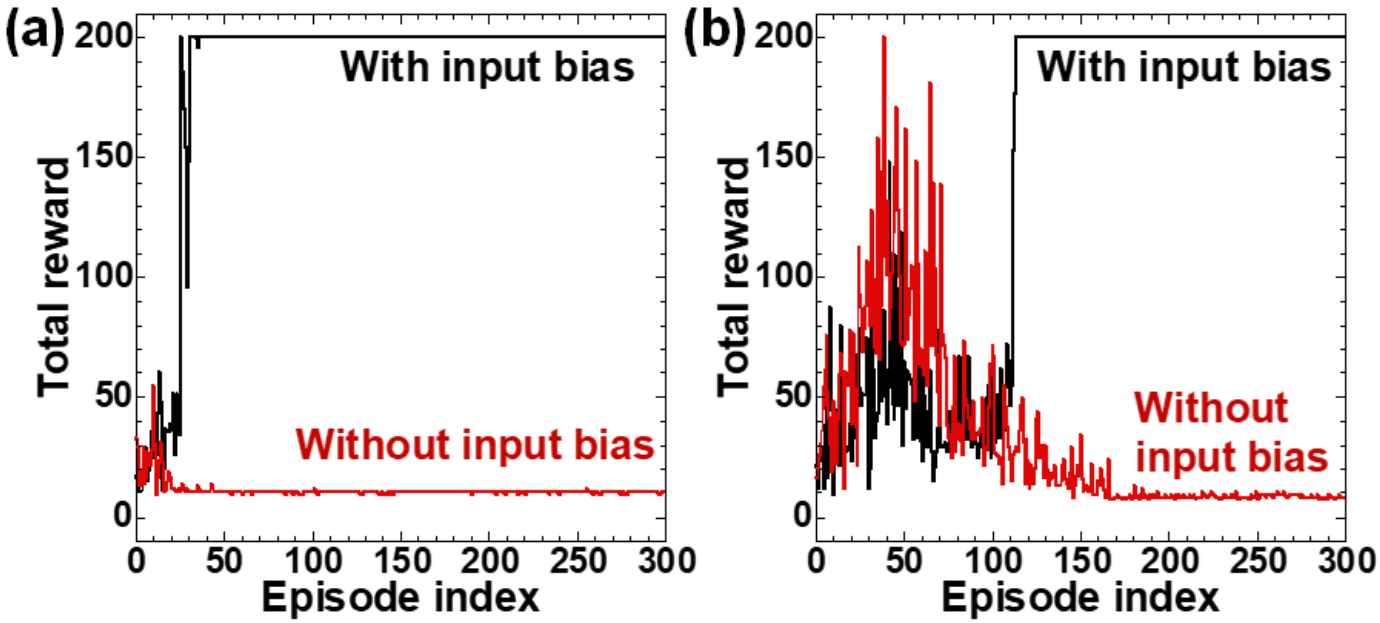

Figure 3

(a) Numerical and (b) experimental results of the CartPole-v0 task. The result shows the total reward for each episode. The total reward of 200 indicates that the pole keeps upright over an episode. The black and red curves show the case with and without the input bias (b=0.8 and b=0.0), respectively.

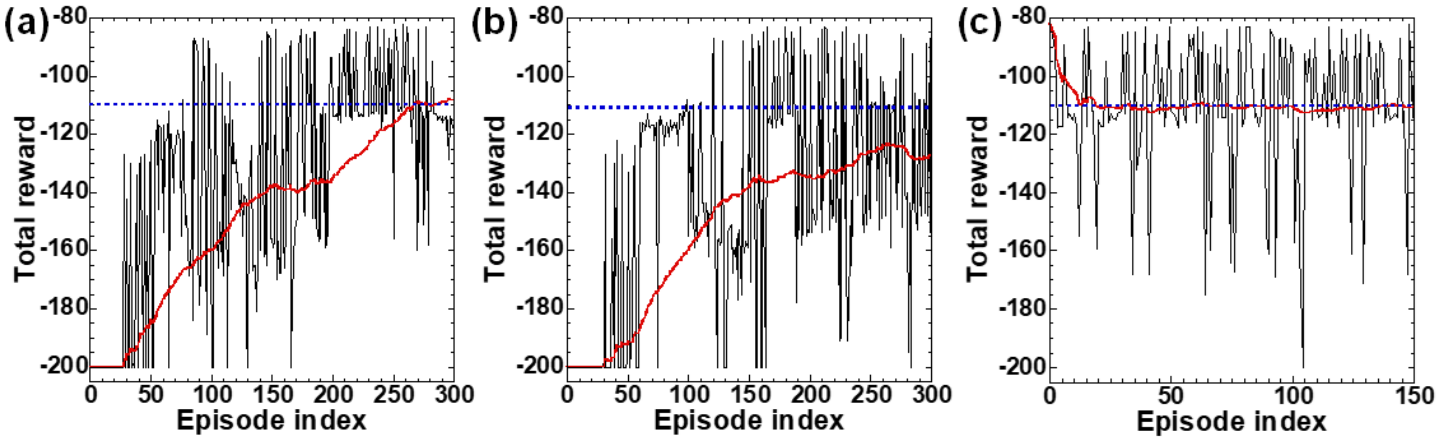

Figure 4

(a) Numerical and (b) experimental results of the MountainCarutida-v0 task. The black curve represents the total reward for each episode. The moving average of the total reward is represented as the red curve. The average is calculated from the past 100 episodes. The total reward of -200 indicates that the car does not reach the top of the mountain. A larger value of the total reward indicates that the car reaches the top of the mountain at a smaller number of steps. (c) The total reward for each episode, where the reservoir weight at the 180th episode in (b) is used in experiment. The weight is not updated in (c). The

blue dashed line corresponds to the total reward of -110 that indicates that the task is successfully solved.

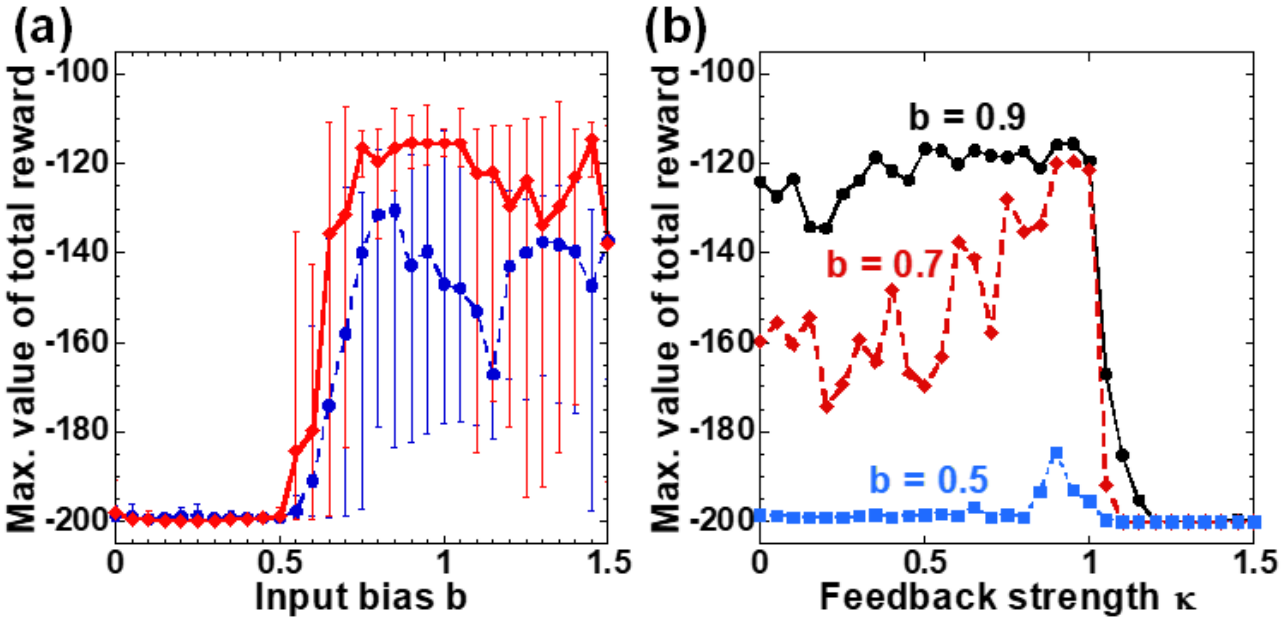

Figure 5

(a) Maximum of the average total reward as a function of the input bias b. The red solid (with diamonds) and blue dashed (circles) curves represent the case with (κ=0.9) and without (κ=0) delayed feedback. The average total reward is calculated using a moving window from the past 100 episodes. The error bar represents the maximum and minimum values for 10 trials. (b) Maximum of the average total reward as a function of the feedback strength κ. The input bias is set to be b=0.9, 0.7, and 0.5 for the black solid (circles), red dashed (diamonds), and blue dotted (squares) curves, respectively. The plotted value is the maximum of the average total reward in 100 consecutive episodes. The error bar represents the maximum and minimum values for 10 trials.