2020 International Symposium on Nonlinear Theory and Its Applications,
NOLTA2020, Virtual, November 16-19, 2020

Reinforcement learning based on electro-optic delay-based reservoir computing

Kazutaka Kanno† and Atsushi Uchida†

†Department of Information and Computer Sciences, Saitama University,
255 Shimo-Okubo, Sakura-ku, Saitama City, Saitama, 338-8570, Japan
Emails: kkanno@mail.saitama-u.ac.jp, auchida@mail.saitama-u.ac.jp.

Abstract—Machine learning based on photonic implementation has been paid increasing attention for fast information processing and energy efficiency. In this study, we present numerical demonstration of reinforcement learning based on photonic reservoir computing using an electro-optic delay system. Reinforcement learning is one of machine learning schemes to do so as to maximize a reward from environment. Photonic reservoir computing receives states in environment and decides actions, where photonic reservoir computing is trained based on Q-learning to maximize the total reward. Cart-pole problem, which is a famous benchmark task in reinforcement learning, is demonstrated to evaluate the performance of our scheme.

1. Introduction

Reinforcement learning is one of machine learning schemes and concerned with the problem of training an action policy to maximize the total reward [1]. Many complex tasks and games have been solved using reinforcement learning models [2, 3]. Deep neural networks have been used in many of their works. However, computational costs are often very expensive in deep neural networks based reinforcement learning schemes because networks weights are repeatedly trained using vast playing data. Some techniques have been proposed to reduce computational costs, such as the prioritized experienced replay [4].

Another scheme to reduce computational costs uses reservoir computing [5]. Reservoir computing is an information processing framework based on recurrent neural networks [6]. The main characteristic of reservoir computing is that input and connection weights are randomly fixed whereas only output weights are trained. The characteristic makes it possible to reduce the computational cost of learning because only the output weights are trained.

Recently, physical implementations of reservoir computing has been intensively studied [7]. Photonic implementation of reservoir computing is one of them, where a laser diode and a delayed feedback loop are used to implement a reservoir [8]. A motivation for photonic implementation of reservoirs is to realize fast information processing devices with low learning cost. It has been reported that speech recognition at 1.1 Gbyte/s is achieved by photonic reservoir computing [9]. The reduction of computational costs and fast processing speed are expected if reinforcement learning is implemented using delay-based reservoir computing.

In this study, reinforcement learning based on delay-based reservoir computing is numerically demonstrated. Delay-based reservoir computing is implemented using an electro-optic delay system, which has been studied for physical reservoir computing [8]. Delay-based reservoir computing is used for selecting an agent's action. Cart-pole problem, which is a famous benchmark task in reinforcement learning, is demonstrated to evaluate our scheme.

2. Reinforcement learning based on reservoir computing

Figure 1 shows a schematic diagram of reinforcement learning based on reservoir computing. In reinforcement learning, an active decision-making agent and its environments are considered. The agent interacts with its environment, where the agent's action affects the future state of the environment and a reward is given by the environment to the agent as the result of the action. The agent's objective is to maximize the total reward. However, the agent has no information about a good action policy. Here, a value $Q(\mathbf{s}_n, a_n)$ for the state $\mathbf{s}_n$ and the action a_n at the n-th time step is considered. The value $Q(\mathbf{s}_n, a_n)$ is called the action-value function. The agent selects an action with the highest value of Q in a given state. If the agent has the knowledge of Q, the total reward can be enhanced. In many researches, the action-value function is replaced to deep neural networks and the networks are trained using Q-Learning [2,3]. In this study, the action-value function is replaced to delay-based reservoir computing. In the following subsessions,

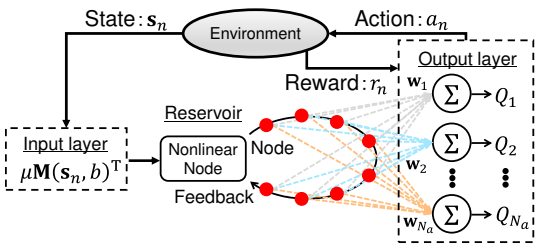

Figure 1: Schematic diagram of reinforcement learning based on delay-based reservoir computing.

we describe a scheme of delay-based reservoir computing, a training method for reservoir computing, and an electro-optic delay system for reservoir computing.

2.1. Delay-based reservoir computing

In delay-based reservoir computing systems, a reservoir consists of a nonlinear element and a feedback loop, which emulates a network with a large number of connected nodes [5]. The n-th input data into the reservoir is the state vector given from the environment $\mathbf{s}_n = (s_{n,1}, s_{n,2}, \cdots, s_{n,N_s})$, where N_s is the number of states. The state is injected into the reservoir after preprocessing is applied. The preprocessing is called masking procedure, where the state is multiplied by a mask matrix M. The value of the mask is randomly taken from a uniform distribution of [-1, 1]. The masking procedure is given by the following equation,

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

where $\mathbf{M}$ is the mask matrix which is a $N_s + 1 \times N$ matrix, μ is the scaling factor, and T represents the transposition. N is the number of nodes in the reservoir. The elements of the vector $\mathbf{u}_n$ is given by $u_{n,i}$, which corresponds to an input data into the i-th virtual node. It should be noted that a fixed bias b is added to Eq. (1). The role of this bias is to prevent that the signal $\mathbf{u}_n$ becomes nearly equal to zero when the elements of the state $\mathbf{s}_n$ are nearly equal to zero. A signal injected into the reservoir is generated by stretching the elements of $\mathbf{u}_n$ to the node interval θ as following,

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

where T_m is called the mask period. The parameters μ and b are fixed at $\mu = 0.8$ and $b = 1.0$, respectively.

The output of reservoir computing is the action-value function $Q(\mathbf{s}_n, a)$ for reinforcement learning and calculated from the weighed linear combination of virtual node states. The virtual node states are extracted from the temporal output of the reservoir by temporally dividing the output with the node interval θ. The output for an action a is represented as $Q(\mathbf{s}_n, a)$ and given by the following equation,

$$Q(\mathbf{s}_n, a) = \sum_{j=1}^{N} w_j x_{j,n} = \mathbf{w}_a \mathbf{x}_n^T, \quad (3)$$

where $\mathbf{x}_n$ is the vector of node states for the n-th input and $\mathbf{w}_a$ is the weight vector for the action a. The number of reservoir outputs corresponds to the number of agent's actions in reinforcement learning. In reinforcement learning, an action with the highest value of Q is selected. The weight w_a is trained based on Q-learning in reinforcement learning.

2.2. Training for reservoir weights in reinforcement learning

One of popular training algorithms in reinforcement learning is Q-learning [1]. In our study, Q-learning is used for training reservoir weights. In Q-learning, the action-value function $Q(\mathbf{s}_n, a_n)$ is updated in accordance with the following rule [1],

$$Q(\mathbf{s}_n, a_n) \leftarrow Q(\mathbf{s}_n, a_n) + \alpha\left[r_{n+1} + \gamma \max_a Q(\mathbf{s}_{n+1}, a) - Q(\mathbf{s}_n, a_n)\right] \quad (4)$$

where α is the constant step-size parameter and γ is the parameter called the discount rate. The action-value function $Q(\mathbf{s}_n, a_n)$ is replaced using Eq. (3).

$$\mathbf{w}_{a_n}\mathbf{x}_n^T \leftarrow \mathbf{w}_{a_n}\mathbf{x}_n^T + \alpha\left[r_{n+1} + \gamma \max_a \mathbf{w}_a\mathbf{x}_{n+1}^T - \mathbf{w}_{a_n}\mathbf{x}_n^T\right] \quad (5)$$

The node state vector is multiplied to the both sides of Eq. (5) from the right, and the both sides are divided by the square of the absolute state vector.

$$\mathbf{w}_{a_n} \leftarrow \mathbf{w}_{a_n} + \frac{\alpha}{|\mathbf{x}_n|^2}\left[r_{n+1} + \gamma \max_a \mathbf{w}_a\mathbf{x}_{n+1}^T - \mathbf{w}_{a_n}\mathbf{x}_n^T\right]\mathbf{x}_n \quad (6)$$

Equation (6) is the update rule of the reservoir weights. In this study, $\alpha/|\mathbf{x}_n|^2$ is replaced to α for simplicity. The parameters α and γ are fixed at $\alpha = 0.00001$ and $\gamma = 0.995$, respectively.

For training the reservoir weights, a technique known as experience replay [10] is applied. For experience replay, the size of minibatch is 64 and the memory size is 2000.

2.3. Numerical model of an electro-optic delay system

Electro-optic delay systems have been studied for delay-based reservoir computing [8]. The schematic diagram of an electro-optic system for reservoir computing is shown in Fig. 2. The main components of the system are a laser diode (LD), a modulator (Mach-Zehnder modulator, MZM), and an optical fiber. The modulator provides a nonlinear transfer function $\cos^2(\cdot)$. The optical fiber implements the delayed feedback loop with the delay time τ. The optical output of the MZM is delayed with the optical fiber and transformed to an electric signal by a photodetector. The electric signal is fed back to the MZM after the signal passed through an electric amplifier. An input signal for reservoir computing is injected by coupling with the feedback signal.

The electro-optic system can be modeled by the follow-

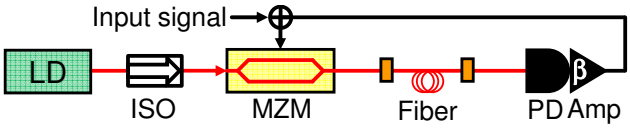

Figure 2: Model of an electro-optic delay system for reservoir computing.

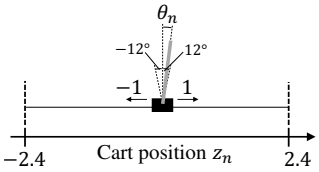

Figure 3: Schematic diagram of the CartPole-v0 task [12].

ing delay differential equations [11],

$$\tau_L \frac{dx(t)}{dt} = -\left(1 + \frac{\tau_L}{\tau_H}\right) x(t) - y(t) + \xi(t) + \beta \cos^2\left[\kappa x(t-\tau) + \pi u(t)/4 + \phi\right] \quad (7)$$

$$\tau_H \frac{dy_1(t)}{dt} = x_1(t) \quad (8)$$

where x is the normalized output of the MZM. τ_L and τ_H are the time constants describing the low-pass and high-pass filters which are related to frequency bandwidths of the components. β is the dimensionless constant that describes the feedback strength. ϕ represent the offset phase of the MZM. $u(t)$ is the input signal injected into the reservoir. $\xi(t)$ is white Gaussian noise with the properties $\langle \xi(t) \rangle = 0$ and $\langle \xi(t)\xi(t_0) \rangle = \delta(t - t_0)$, where $\langle \cdot \rangle$ denotes the ensemble average and $\delta(t)$ is the Dirac's delta function.

In our numerical simulation, the number of virtual nodes N is 600 and the node interval θ is 0.05 μs. Then, the delay time τ is given by $\tau = (N-1)\theta = 29.95$ μs. The offset phases ϕ is $-\pi/4$. $1/(2\pi\tau_L) = 100$ MHz, $1/(2\pi\tau_H) = 5$ kHz, $\beta = 1$, and $\kappa = 0.9$ are fixed.

3. Numerical Results

We evaluate our scheme of reinforcement learning based on delay-based reservoir computing in a famous reinforcement learning task: CartPole-v0 [12] in OpenAI Gym [13]. The schematic diagram of the CartPole-v0 task is shown in Fig. 3. A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The number of states of the task is 4: cart position z_n, cart velocity, pole angle θ_n, and pole velocity at tip. The agent's action is to push the cart to right (+1) and left (-1). The goal of the task is to keep the pole upright over an episode. A length of the episode is 200 time step. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends at the pole velocity $|\theta_n| \geq 12$ degrees (about 0.209 radian) or the cart position $|z_n| \geq 2.4$. It is considered the task is solved when the pole remains upright over an episode for 100 consecutive episodes.

The magnitudes of the cart position and the pole velocity are normalized to become one at their maximum values before injected into the reservoir.

For action selection, an ε-greedy method is used, where random selection is taken with the probability of ε. A value of ε is updated by $\varepsilon = 0.01 + (1.0 - 0.01)\exp(-0.01 n_{ep})$, where n_{ep} is the episode index.

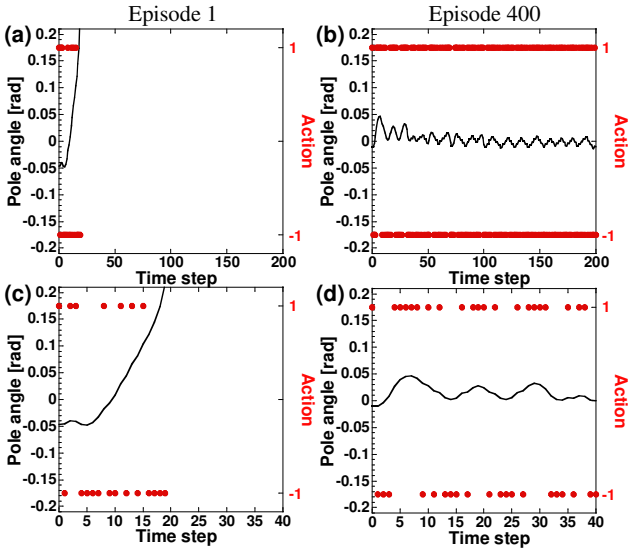

Figure 4: Temporal evolution of the pole angle and the selected action in the CartPole-v0 task. (a) and (b) are the results of the episode 1 and 400, respectively. The black curve is the pole angle and the red circle is the selected action. (c) and (d) are the enlarged views of (a) and (b), respectively.

Figure 4(a) shows the temporal evolution of the angle and the selected action at the first episode. The enlarged view of Fig. 4(a) is shown in Fig. 4(c). In the first episode, the agent takes a random action because ε is 1. The pole angle monotonically increases for more than the time step of 6. The pole leans to the right for a positive value of the pole angle and the agent needs to select the action of +1 to prevent for the pole to fall. However, the action is randomly selected and the pole falls (the angle becomes larger than 0.209 radian) at the time step of 19. Figure 4(b) shows the result of the episode over which the pole remains upright. The enlarged view is shown in Fig. 4(d). The angle repeatedly increases and decreases around zero . When the angle increases (for example, the time step is from 5 to 8 in Fig. 4(d)), the agent selects the action of +1, which corresponds to pushing the cart to the right to prevent for the pole to fall to the right. It is found that the reservoir is trained to prevent for the pole to fall.

Figure 5 shows the total score over an episode, where the pole remains upright over the episode if the total reward is 200. When the episode index is small, the pole cannot keep upright. However, the pole becomes to be able to keep upright over an episode as the episode index increases. In Fig. 5, the CartPole-v0 task is solved since 100 consecutive episodes with the total reward of 200 is found.

Finally, we investigate the effect of the input bias b for preprocessing an input signal in reservoir computing. Figure 6 shows the total reward over an episode when the input bias b is fixed at zero. The total reward does not reach at 200 and the pole cannot keep upright for all episodes. The

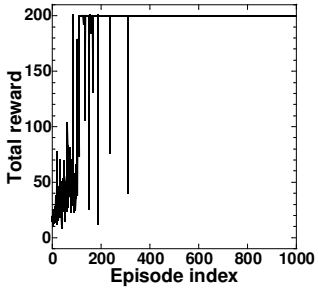

Figure 5: Total reward in each episode. The pole keeps upright over an episode if the total reward is 200.

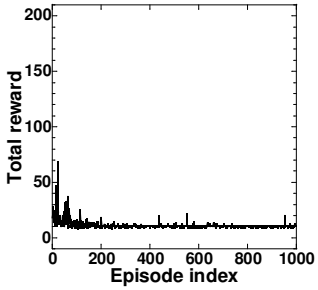

Figure 6: Total reward in each episode. The input bias b for preprocessing in reservoir computing is zero.

reason why the task cannot be solved when the input bias b is not included in the reservoir preprocessing is the following: there is no input information when the state of the task is zero or a small value.

4. Conclusions

In this study, reinforcement learning based on delay-based reservoir computing was numerically demonstrated. Main components of delay-based reservoir computing were a nonlinear node and a delayed feedback loop. Delay-based reservoir computing was implemented using an electro-optic delay system. In reinforcement learning, a state given from environment was injected into the reservoir and it produced the action-value function. The weights of the reservoir were trained using Q-Learning. To evaluate our system, a CartPole-v0 task was demonstrated. Our system showed successful computation for the CartPole-v0 task. We also investigated an input bias used for preprocessing input data in reservoir computing. Without the bias, the CartPole-v0 task could not be achieved. Therefore, the input bias was required for solving the reinforcement learning task.

Acknowledgments

This work was supported in part by JSPS KAKENHI (JP19H00868 and JP20K15185), JST CREST JPMJCR17N2, and The Telecommunications Advancement Foundation.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, second edition, The MIT Press, (2018).

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Nature, vol. 529, pp. 484–489 (2016).

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Nature, vol. 518, pp. 529–533 (2015).

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, arXiv:1511.05952 (2016).

[5] H. Chang and K. Futagami, Applied Intelligence, vol. 50, pp. 2400–2410 (2020).

[6] H. Jaeger and H. Haas, Science, vol. 304, pp. 78–80 (2004).

[7] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, Neural Networks, vol. 115, pp. 100–123 (2019).

[8] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, Scientific Report, vol. 2, Article number: 287 (2012).

[9] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, Nat. Commun., vol. 4, Article number: 1364 (2013).

[10] J. O' Neill, B. Pleydell-Bouverie, D. Dupret, J. Csicsvari, Trends in Neurosciences, vol. 33, pp. 220–229 (2010).

[11] T. E. Murphy, A. B. Cohen, B. Ravoori, K. R. B. Schmitt, A. V. Setty, F. Sorrentino, C. R. S. Williams, E. Ott, and R. Roy, Phil. Trans. R. Soc. A, vol. 368, pp. 343–366 (2010).

[12] A. G. Barto, R. S. Sutton, and C. W. Anderson, IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, pp. 834–846 (1983).

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, and W.Z. Jie Tang, arXiv:1606.01540 (2016).