Neural Computing and Applications (2021) 33:9391–9402
https://doi.org/10.1007/s00521-021-05699-z

ORIGINAL ARTICLE

On-chip trainable hardware-based deep Q-networks approximating a backpropagation algorithm

Jangsaeng Kim[1] · Dongseok Kwon[1] · Sung Yun Woo[1] · Won-Mook Kang[1] · Soochang Lee[1] · Seongbin Oh[1] · Chul-Heung Kim[1] · Jong-Ho Bae[2] · Byung-Gook Park[1] · Jong-Ho Lee[1]

Received: 27 July 2020 / Accepted: 5 January 2021 / Published online: 10 February 2021

Abstract
Reinforcement learning (RL) using deep Q-networks (DQNs) has shown performance beyond the human level in a number of complex problems. In addition, many studies have focused on bio-inspired hardware-based spiking neural networks (SNNs) given the capabilities of these technologies to realize both parallel operation and low power consumption. Here, we propose an on-chip training method for DQNs applicable to hardware-based SNNs. Because the conventional backpropagation (BP) algorithm is approximated, a performance evaluation based on two simple games shows that the proposed system achieves performance similar to that of a software-based system. The proposed training method can minimize memory usage and reduce power consumption and area occupation levels. In particular, for simple problems, the memory dependency can be significantly reduced given that high performance is achieved without using replay memory. Furthermore, we investigate the effect of the nonlinearity characteristics and two types of variation of non-ideal synaptic devices on the performance outcomes. In this work, thin-film transistor (TFT)-type flash memory cells are used as synaptic devices. A simulation is also conducted using fully connected neural network with non-leaky integrated-and-fire (I&F) neurons. The proposed system shows strong immunity to device variations because an on-chip training scheme is adopted.

Keywords Reinforcement learning (RL) · Hardware-based deep Q-networks (DQNs) · On-chip training · Synaptic devices

1 Introduction

Recently, neuromorphic computing inspired by the human brain has emerged as one of the most promising types of computing architectures. It overcomes the limitations of the conventional von Neumann architecture, which is associated with a bottleneck between the memory and the processor, and offers advantages in terms of time and power consumption [1–3]. Two training methods are most commonly used to training a neural network: the backpropagation (BP) algorithm and the spike-timing-dependent plasticity (STDP) learning rule [4, 5]. The BP algorithm propagates error values obtained from the output layer in the backward direction and updates the synaptic weights through these error values. It is suitable for processing labeled data and is mainly used for offline supervised learning. Software-based deep neural networks (DNNs) using the BP algorithm have shown high performance in many fields [6–8]. However, this training algorithm requires considerable amounts of time and power to determine the error values [4]. The STDP learning rule is inspired by the weight changes of biological synapses and is mainly used for online unsupervised learning. Unlike the BP algorithm, the synaptic weights are updated according to the time difference between the presynaptic and postsynaptic spikes [9]. The STDP learning rule has an advantage in that the neural network consumes less power by enabling event-driven operations and on-chip training. However, it still lacks performance compared to the BP algorithm [10–13].

✉ Jong-Ho Lee
jhl@snu.ac.kr

[1] Department of Electrical and Computer Engineering, Inter-University Semiconductor Research Center (ISRC), Seoul National University, Gwanak-ro, Gwanak-gu, Seoul 08826, Republic of Korea

[2] Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, USA

For low power consumption and high speed, hardware-based spiking neural networks (SNNs) using the conductance of electronic synaptic devices as synaptic weights have been actively studied [14–17]. Hardware-based neural networks can perform massively parallel computations with low levels of power. When training a hardware-based neural network using the BP algorithm, two methods are most commonly used: an off-chip training method that simply transfers the weight values trained in software using the BP algorithm to the synaptic devices, and an on-chip training method that continuously updates the synaptic weights during the training process [4]. Off-chip training uses more power because the training process for weight updating occurs in the software. Furthermore, the performance can be degraded due to variations of non-ideal synaptic devices [5, 18]. On the other hand, on-chip training is immune to variations of non-ideal synaptic devices [10, 19–22]. This approach is also advantageous given its low power consumption and high-speed training capabilities, as both weighted sum and weight updating occur in the hardware-based neural network [5, 23, 24]. Recently, several studies have investigated on-chip training for low power consumption and good performance outcomes [25–27]. On-chip training in a hardware-based neural network showed performance fairly similar to that by the conventional software-based neural network.

Thus, we focused on the on-chip training method using the BP algorithm in a hardware-based neural network. This is advantageous given its relatively high performance compared to the STDP learning rule, low power consumption, high-speed training capabilities, and strong immunity to variations of non-ideal synaptic devices compared to the off-chip training method. While several studies related to on-chip training have been conducted on various networks, no studies have been reported on an on-chip reinforcement learning (RL) to the best of our knowledge. RL achieves human-level performance, which is difficult to achieve with simple DNNs [28, 29]. In RL with complex problems, training using deep Q-networks (DQNs) is preferred, as performance levels when using this method have already surpassed the human levels in several fields [28–32].

In this work, we propose an on-chip training method for RL using vanilla DQNs while minimizing memory usage. While several training methods applicable to RL have been reported, vanilla DQNs were used as a simple training method suitable for implementing hardware-based RL [31–38]. In addition, nonlinearity characteristics and variations of non-ideal synaptic devices are considered. Two types of variation of non-ideal synaptic devices were considered: the pulse-to-pulse variation and the device-to-device variation. The entire training process is divided into four phases: two forward phases, one backward phase, and one update phase. Moreover, for simple problems, the network can be trained without using replay memory, thereby significantly reducing the memory dependency. We use the previously proposed neuron circuits [39] and the synaptic devices [40] to evaluate the proposed training method. The performance of the training method is evaluated through two example games: a 'Fruit Catching' game and a 'Rush Hour' game.

This paper is organized as follows. Section 2 presents the characteristics of the synaptic device used in this work and the proposed training method for hardware-based DQNs. Section 3 provides the system-level simulation results of the proposed hardware-based neural network based on two simple games and a discussion about the results. Section 4 provides a summary of the overall paper and the future work.

2 Device characteristics and training method

2.1 Synaptic device

In a hardware-based neural network, synaptic devices representing weight values are very important. In this work, we use the thin-film transistor (TFT)-type flash memory cells fabricated using a method published in an earlier report by the authors as synaptic devices [40]. A schematic 3D view of the proposed synaptic device is shown in Fig. 1a. TFT-type flash memory cells are fabricated on a six-inch Si wafer with conventional CMOS process technology. Between the word line (WL) and the source line (SL), a half-covered n^+ poly-Si floating gate (FG) is formed as a charge storage layer. Because the FG covers only half of the poly-Si channel, the threshold voltage does not fall below 0 in the full erase state. This prevents leakage current and reduces the standby power during the training process. The thicknesses of the poly-Si active layer, tunneling SiO_2 layer, blocking SiO_2 are 20 nm, 7 nm, and 15 nm, respectively. The distance between the source and drain is 0.5 μm, and the width of the control gate is 2 μm. If the width of the control gate is scaled to the minimum feature size (F), one synaptic device can be scaled down to 8 F^2. Figure 1b shows the measured long-term potentiation (LTP) and long-term depression (LTD) characteristics of the proposed synaptic device as a parameter of the number of the pulses applied to the WL and SL. Fifty repeated erase pulses (V_{WL} = − 3 V, V_{SL} = 5 V) and 300 repeated program pulses (V_{WL} = 0 V, V_{SL} = − 4.8 V) are applied in this measurement.

The behavioral model of the nonlinear synaptic device is generally expressed using the following equations [10]:

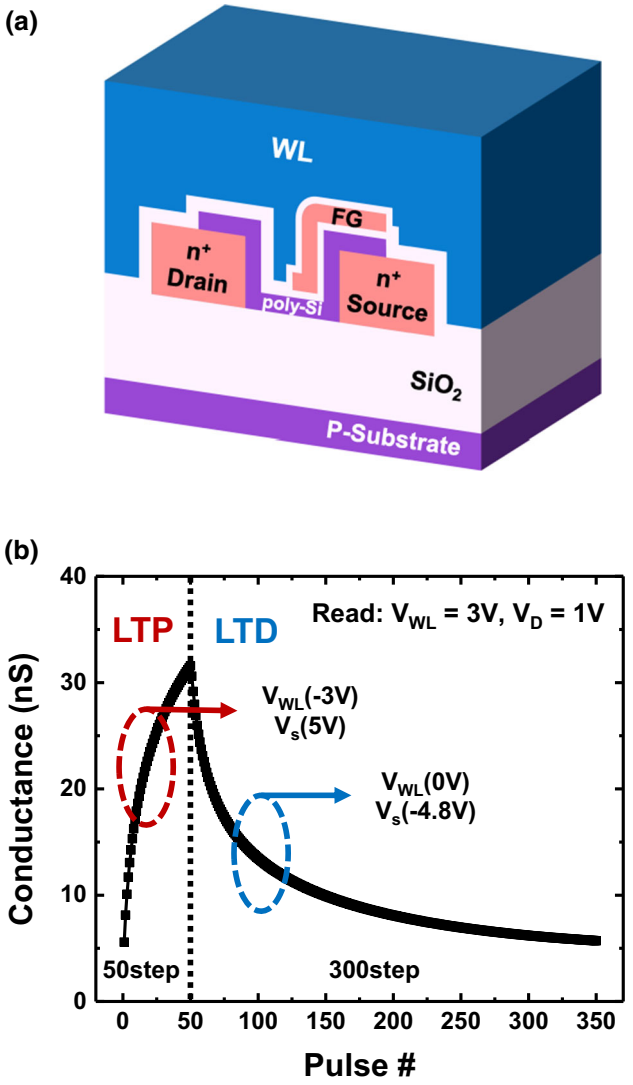

Fig. 1 **a** 3D view of TFT-type flash memory cells used as synaptic devices. **b** Measured LTP/LTD characteristics of the proposed synaptic device

$$G(n+1) = G(n) + \alpha_p \exp\left(-\beta_p \frac{G(n) - G_{\min}}{G_{\max} - G_{\min}}\right) \quad (1)$$

$$G(n+1) = G(n) - \alpha_d \exp\left(-\beta_d \frac{G_{\max} - G(n)}{G_{\max} - G_{\min}}\right). \quad (2)$$

Equations (1) and (2) represent the LTP and LTD behaviors of synaptic devices, respectively. $G(n)$ denotes the conductance of the synaptic device when potentiation or depression pulses are applied n times, and $G_{\max}$ and $G_{\min}$ are the maximum and minimum conductance values, respectively. α_p and β_p represent the fitting parameters of the potentiation characteristics. Similarly, α_d and β_d represent the fitting parameters of the depression characteristics. The nonlinearity of the synaptic device is determined by β_p and β_d in these equations. The LTP and LTD characteristics of the proposed TFT-type synaptic device are fitted with β_p equal to 2.5 and β_d equal to 5. In addition, the synaptic device consumes energy of $\sim$100 fJ/spike on average (approximately a 10 nA current at a pulse with a 1 V amplitude and a 10 μs width).

2.2 Training method

Before introducing the training method, we describe the behavior of the integrate-and-fire (I&F) neuron model used in the neural network. The spikes from presynaptic neurons are integrated into the membrane capacitor of the postsynaptic I&F neurons. When the membrane potential exceeds the threshold voltage, postsynaptic spikes are generated from the I&F neuron and are transmitted to the next layer. The behavior of the I&F neurons in the SNNs can approximate the rectified linear unit (ReLU) activation function of conventional DNNs, as the number of the postsynaptic spikes is proportional to the membrane potential of the I&F neuron. Here, the membrane potentials are initialized to zero, and the lower limit is set to zero.

Details of the behavior of the membrane potential (V_j^l) of the j-th I&F neuron in the l layer are expressed as shown below.

$$V_j^l(t) = V_j^l(t-1) + \sum_i^{N^{l-1}} \frac{S_i^{l-1}(t)}{C_{\text{mem}}} w_{ij}^{l-1} \quad (3)$$

$$if\ V_j^l(t) > V_{\text{th}} : \begin{cases} V_j^l(t) = V_j^l(t) - V_{\text{th}} \\ S_j^l(t) = 1 \end{cases} \quad (4)$$

$$\text{else}: \quad S_j^l(t) = 0 \quad (5)$$

Here, $S_j^l(t)$ represents the spikes generated in the j-th neuron in the l layer at time t in the form of a voltage pulse. w_{ij}^{l-1} indicates the synaptic weight between the i-th neuron in the l-1 layer and the j-th neuron in the l layer. C_{mem} and N^{l-1} correspondingly represent the membrane capacitance of the I&F neuron and the total number of neurons in the l-1 layer. Equation (4) represents the behavior of the j-th neuron in the l layer when the membrane potential exceeds the threshold voltage (V_{th}). At this time, the membrane potential drops by V_{th} and the neuron generates a postsynaptic spike ($S_j^l(t)$).

By accumulating the teaching signal ($Z_k(t)$) and the output spike, we can obtain the error value (δ_k^L) in the output layer ($l = L$), as follows,

$$\delta_k^L = \sum_t^T \left(Z_k(t) - S_k^L(t)\right), \quad (6)$$

where T is the total number of time steps taken to train one set of input pixels. In this paper, images were used as input

data, and the input pixels are presented as binary data with a value of 0 or 1. The total number of times a neuron fires in each layer during the training process ($\sum_t^T S_j^l(t)$) is limited to a maximum of T times. The teaching signal supervises the training direction and is obtained through methods described in Sect. 2.3.

The error values in the previous layers ($l \in \{1, 2, \ldots, L-1\}$) are obtained through the backward weighted sum, as follows,

$$\delta_j^l = \begin{cases} \sum_k^{N^{l+1}} \delta_k^{l+1} w_{jk}^l, & \text{if } \sum_t^T S_j^l \geq 1 \\ 0, & \text{otherwise} \end{cases}. \tag{7}$$

When the j-th neuron in the l layer fires more than once during time T ($\sum_t^T S_j^l \geq 1$), the error value is obtained through the backward weighted sum. On the other hand, the error value is 0 when the j-th neuron in the l layer does not fire during time T ($\sum_t^T S_j^l = 0$). This reflects the derivative value of the ReLU activation function.

The synaptic weights are updated as follows using the error value obtained in Eq. (7),

$$\Delta w_{jk}^l = \eta \delta_k^{l+1} \text{sign}\left(\sum_t^T S_j^l(t) \right), \tag{8}$$

where η denotes the ratio used when converting the magnitude of the error value to the pulse width. The conversion ratio η has a meaning similar to that of learning rate in software-based networks. In addition, the weight update value (Δw_{jk}^l) becomes 0 when the number of presynaptic spikes generated during time T ($\sum_t^T S_j^l(t)$) is 0. Then, the synaptic weight between the j-th neuron in the l layer and the k-th neuron in the $l+1$ layer is updated using the following equation:

$$w_{jk}^l = w_{jk}^l + \Delta w_{jk}^l. \tag{9}$$

The derivative value of the activation function is not employed in the overall training process, as the derivative value of the ReLU function is 1 or 0. When $\sum_t^T S_j^l(t)$ is 0, δ_j^l becomes 0 in Eq. (7) and Δw_{ij}^{l-1} becomes 0 in Eq. (8), outcomes identical to reflecting the derivative value 0 of the ReLU function. When $\sum_t^T S_j^l(t)$ exceeds 1, δ_j^l is obtained as the derivative value 1 of the ReLU function is reflected and Δw_{ij}^{l-1} is obtained through Eq. (8).

In a hardware-based neural network, the weight value between the i-th presynaptic neuron and the j-th postsynaptic neuron is represented by the difference in the conductance of two synaptic devices, as follows,

$$w_{ij} = G_{ij}^+ - G_{ij}^-, \tag{10}$$

where G_{ij}^+ and G_{ij}^- represent the positive and negative weight values, respectively. Two synaptic devices are required for each weight value to express a negative synaptic weight because the conductance of a synaptic device only has a positive value. The update and reset methods of G^+ and G^- follow the method proposed by Lim [41]. G^+ is increased when weight potentiation is required, and G^- is increased when weight depression is necessary. If G^+ reaches $G_{\max}$ and weight potentiation is necessary, G^- is initialized and then increased to a conductance level one step lower than the previous value. When both G^+ and G^- reach $G_{\max}$, they are initialized to $G_{\min}$.

2.3 Hardware-based deep Q-network

Figure 2 represents the overall training process of the hardware-based DQN. It consists of three elements: the environment in which the game progresses, the DQN where training takes place and the appropriate action is selected, and replay memory for an experience replay.

First, the current state (s) of the environment is applied to the input of the DQN and stored in the replay memory (① of Fig. 2). In the network, forward propagation occurs and the first fired output neuron is selected as an action (a) according to the learning rule. This action is applied to the environment and stored in the replay memory (② of Fig. 2). Next, the reward (r) and the next state (s') that appear when the given action is performed are stored in the replay memory (③ of Fig. 2). Through this process, one set of (s, a, r, s') data is stored in the replay memory. Finally, using the data stored in the replay memory, the DQN is trained using the method described in the previous Sect. 2.2 (④ of Fig. 2).

In this hardware-based network, the entire training process is divided into four phases. Each phase is split into five time steps (T = 5) with a total length of 150 µs. Only one spike can be generated during each time step, and the input pixel is presented as binary data with a value of 0 or 1. When the input data from a pixel is 0, no input spike is generated, and if it is 1, input spikes are generated at every time step. Figure 3a shows a schematic illustration of a fully connected network with one hidden layer as an example. The case of the second phase where the total number of the time step per phase is 5 and the input data from a pixel is 1 (equivalent to 5 pulses with a 3 V amplitude) is shown in Fig. 3b as an example. Figure 3c presents the pulse scheme for weight updating during the fourth phase.

The first phase is the forward phase that receives state s' as an input and obtains the maximum value of the output value (Q). This output value Q represents the long-term expected return of executing action a' from given state s'.

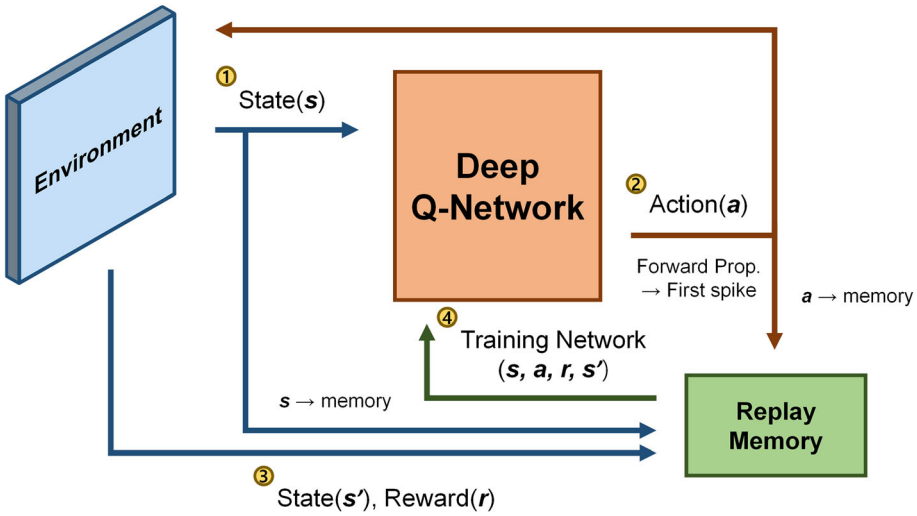

Fig. 2 Overall training process of the hardware-based DQN consists of three elements and four training steps

With a higher the Q-value, better long-term results in state s' can be obtained when the corresponding action a' is performed. In order to obtain the maximum result, the agent selects an action that will lead to the highest Q-value in each state. When the weighted sum from the presynaptic neurons is stored in the forward direction membrane capacitor of the postsynaptic neuron and the membrane potential exceeds the threshold voltage, a postsynaptic spike is generated through the I&F neuron circuit. When the first spike is generated in the k-th output neuron, the membrane potentials of the other output neurons are set to 0. The generated spikes of the k-th output neuron then charge the connected capacitor. The amount of charge stored in the capacitor connected to the k-th output neuron represents the maximum Q-value, which is used as a teaching signal in the next phase.

The second phase is also the forward phase that receives state s as an input and obtains the error value for back-propagation. A process identical to that during the first phase occurs, except that the input data are different. In the neurons of all layers except for the last layer ($l \in \{1, 2, \ldots, L-1\}$), whether or not each neuron fired during time T ($\text{sign}\left(\sum_t^T S_j^l(t)\right)$) is stored as a single bit. In other words, neurons that fired more than once during time T are stored with a value of 1, and neurons that did not fire during time T are stored as a value of 0. In this phase, a teaching signal is generated through a pulse-width modulation (PWM) circuit and the error value is obtained from the difference between the output spike and the teaching signal. The teaching signal Z_j is obtained as shown below using the maximum Q-value obtained in the first phase and the reward r obtained when action a is taken in state s.

$$Z_j = r + \gamma \max_{a'} Q(s', a'), \tag{11}$$

where γ represents a discount factor that decreases the value of future rewards over time. γ is between 0 and 1, and usually has a value of 0.9. This equation is well known as the Bellman equation. The teaching signal is applied only to the m-th output neuron and only the m-th error value is calculated. This m-th output neuron is a neuron corresponding to action a taken when changing from state s to state s'. The error values of other output neurons are set to 0.

The third phase is the backward phase, which propagates the error values obtained in the second phase. In this phase, the weighted sum of the error values obtained in the next layer is stored in the backward direction membrane capacitor. The hidden layer has two membrane capacitors: the forward direction membrane capacitor to store the weighted sum and the backward direction membrane capacitor to store the weighted sum of the error values obtained in the next layer.

The fourth phase is the update phase, which updates the synaptic weights using the error values obtained in the third phase. When the error values are positive or negative, error spikes with corresponding values of 5.0 V or −1.8 V are generated. The pulse width of the error spike is proportional to the magnitude of the error value using the PWM circuit. These error spikes are applied to the source of the synaptic devices twice during this phase, as shown in Fig. 3c. The second error spike is applied after 10 µs. When a single bit value per neuron ($\text{sign}\left(\sum_t^T S_j^l(t)\right)$) stored in the second phase is 1, two 10 µs width spikes having magnitudes of 3 V and −3 V are applied to the gate of the synaptic device in turn. If the error spike is positive, an erase pulse is applied to the synaptic device by overlapping the error spike applied to the source line and the negative part of the spike applied to the gate of the synaptic device, which potentiated the synaptic weight, as shown in

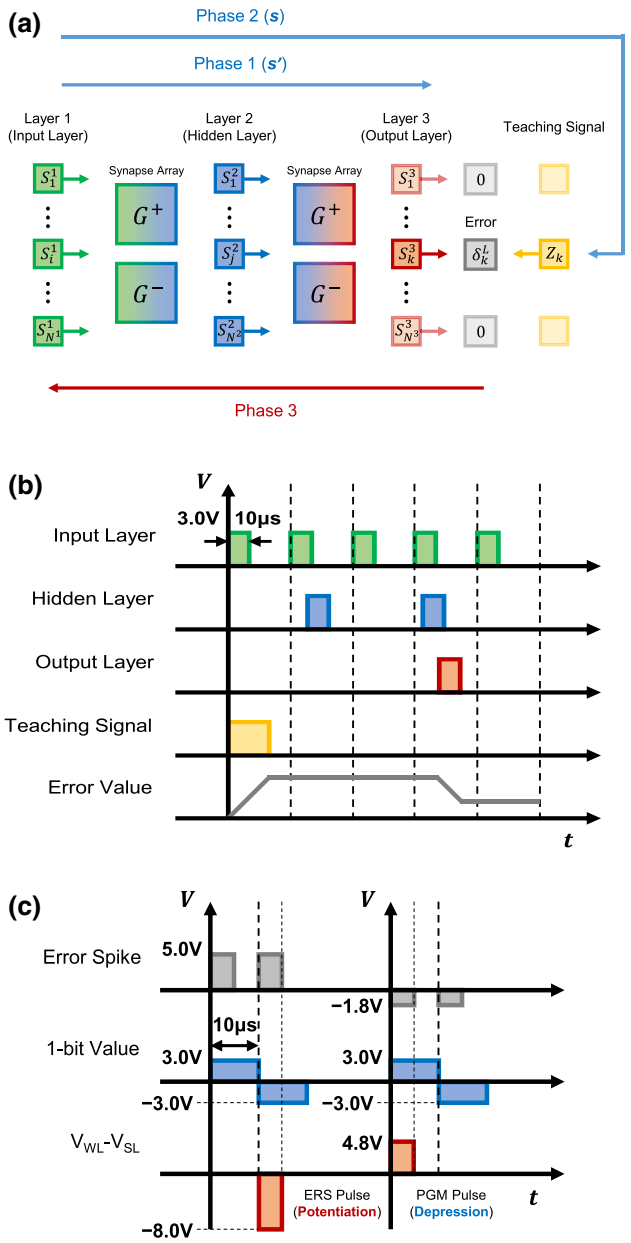

Fig. 3 **a** Schematic illustration of a fully connected network with 1 hidden layer. **b** Pulses generated at each layer during the second phase of the training process when $T = 5$. **c** Proposed pulse scheme for weight update during the fourth phase of the training process

Fig. 3c. In the opposite case, when the error spike is negative, a program pulse is applied to the synaptic device by overlapping the error spike applied to the source line and the positive part of the spike applied to the gate of the synaptic device, which depressed the synaptic weight. However, when a single bit value is 0, no spike is applied to the gate of the synaptic device, and a weight update does not occur.

3 Results and discussion

Two system-level simulations are conducted using Python, a programming language, along with the PyTorch library to evaluate the proposed training method and hardware-based network architecture during the Fruit Catching game and the Rush Hour game. The parameters used in this simulation are shown in Table 1. The synaptic weights for all the simulations are initialized using the initialization method proposed by He [42].

3.1 Fruit catching game

Figure 4 shows an example of how the Fruit Catching game proceeds. In a 10 × 10 grid world, the fruit is 1 × 1 in size and the basket is 1 × 3 in size. When a new game starts ($t_{\text{game}} = 1$), a fruit is created at a random position among ten columns in the first row, and as each time step passes, the fruit falls one row. For each time step, the basket can take three actions at the bottom row: stop or moving to the left or right by one column. The bottom of Fig. 4 presents an example of output spikes generated in the output neurons when $t_{\text{game}} = 1$. In this case, because the firing rate of the output neuron representing the moving left action is the highest, the agent takes an action that moves the basket to the left. When the fruit reaches the ninth row ($t_{\text{game}} = 9$), as shown in the rightmost part of Fig. 4, if the basket exists under the fruit, the agent receives a reward of 1. On the other hand, if the basket does not exist under the fruit, the agent receives a reward of −1. In all other situations, the reward received by the agent is 0. One episode ends with this process, and the new game begins again.

Figures 5a and b represent the catching rate of the proposed hardware-based neural network with different epsilon values (ε) and discount factors (γ). Figure 5a is an ideal case with an epsilon value of 1 and a discount factor of 0.9, and Fig. 5b is a simplified case with an epsilon value of 0 and a discount factor of 1.0. The catching rate was obtained through 1000 test games. The network size used in the simulation is 100-100-100-3. Replay memory with a size of 500 is used in this work, and the network is trained using 50 randomly selected datasets for each action. Here, for example, replay memory with a size of 1 has the number of bits required to store one (s, a, r, s') dataset. As

Table 1 Model parameters

Parameter	Value	Parameter	Value	Parameter	Value
$G_{\max}$	30 (nS)	V_{th}	0.2 (V)	η	10 (μs/V)
$G_{\min}$	5 (nS)	C_{mem}	45 (fF)		

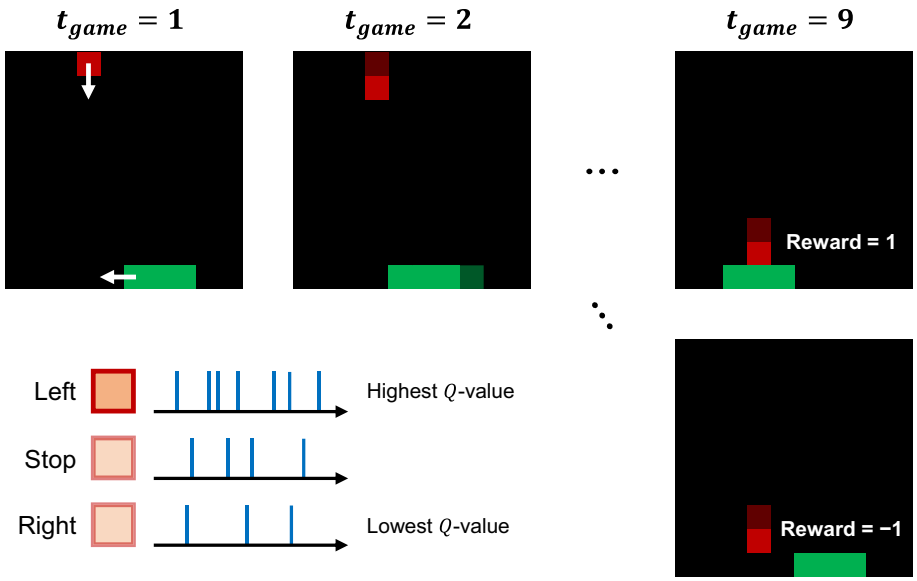

Fig. 4 The process of one episode of the Fruit Catching game. Examples of the output spikes when $t_{\text{game}} = 1$ are shown below

training progresses, errors decrease and the catching rate increase, as shown in both Figs. 5a and b. The insets in Figs. 5a and b show the catching rates for the last 200 episodes. Figure 5c shows the average value of the catching rate during the last 200 episodes for the ideal case ($\varepsilon = 1, \gamma = 0.9$) and the simplified case ($\varepsilon = 0, \gamma = 1.0$). As the networks with ideal cases and simplified cases are well trained without a significant difference, it is clear that the network is trained well even if exploration is not employed and the discount factor is set to 1 in the Fruit Catching game, which is a relatively simple game.

The catching rate of the network as a parameter of the nonlinearity factor (β) is also investigated, as indicated in Fig. 6a. Training was conducted under conditions identical to those in Fig. 5b, with the result showing the average catching rate for the last 200 episodes. As the nonlinearity factor increases, the catching rate decreases slightly (about 5% when β = 5).

Figure 6b shows the average catching rate for the last 200 episodes versus the variation of the synaptic weights. The pulse-to-pulse variation and the device-to-device variation are considered. The pulse-to-pulse variation is modeled as follows:

$$\Delta G_{\text{real}} = \Delta G_{\text{expected}} \times N\left(1, \sigma^2\right). \tag{12}$$

The device-to-device variation is modeled as follows:

$$G_{\text{real}} = G_{\text{expected}} \times N\left(1, \sigma^2\right). \tag{13}$$

The x-axis in Fig. 6b represents the standard deviation (σ). For the two variation cases, the catching rate scarcely drops and remains nearly constant even if σ increases to 0.5, as the on-chip training scheme is employed.

3.2 Rush hour game

The second example used to verify the proposed training method is a simple Rush Hour game. In a 6 × 6 position, several cars of length 2 or 3 are placed horizontally or vertically. Only one car can be moved by one position in one move. The goal of the game is to move the target car (red car) to the exit with the fewest number of moves. If the road between the target car and the exit is not blocked, the agent receives a reward of 1 and one episode ends. In all other situations, the reward received by the agent is 0.

Figure 7a and d show two examples of the Rush Hour game. Figure 7a is an example of a relatively easy game, where average adults will only require less than a few minutes to solve the problem. On the other hand, Fig. 7d is a relatively complicated example, and it is difficult to know which car to move first. The neural network used in the simulation has 288 input neurons and 16 output neurons with no hidden layers. Here, 288 (6 × 6 × 8) input neurons are used because each car can have 36 (6 × 6) positions, and 16 (8 × 2) output neurons are used because each car can move in two directions, i.e., up/left or down/right.

Figure 7b and e represent the number of moves required to move the target car to the exit. Both Figs. 7b and e are ideal cases with an epsilon value of 1 and a discount factor of 0.9. As above, replay memory with a size of 500 is used, and the network is trained using 50 randomly selected datasets for each action. As training progresses, the number of moves required to move the target car to the exit decreases to the optimal value (9 in Fig. 7b and 14 in Fig. 7e). Figure 7c and f are simplified cases in which the agent only takes random actions in the first episode ($\varepsilon = 1$, exploration only). In subsequent episodes, the agent does not take random actions and only exploitation occurs

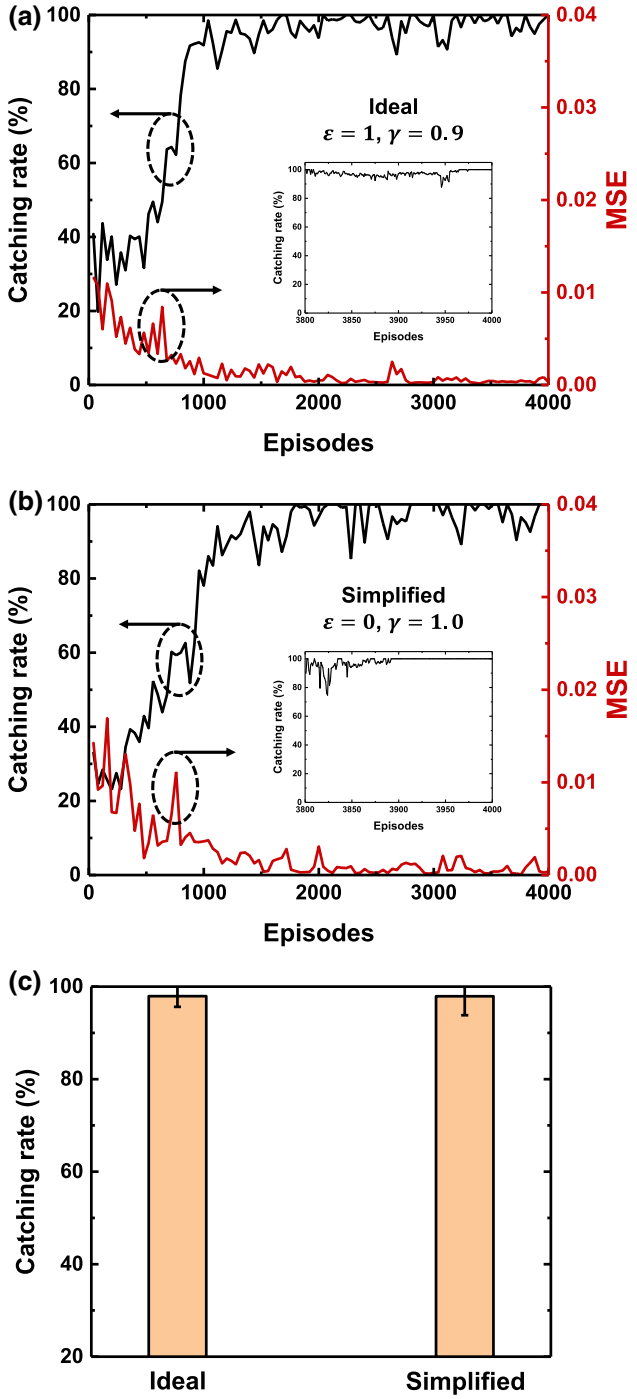

Fig. 5 **a** The catching rate of the proposed hardware-based network with $\varepsilon = 1, \gamma = 0.9$, **b** $\varepsilon = 0, \gamma = 1.0$. **c** The average catching rate during the last 200 episodes for the ideal case and the simplified case

($\varepsilon = 0$, exploitation only). In the simplified cases, the number of moves required to move the target car to the exit converges to the optimal value, as in the ideal cases. This means that training can be conducted well in a simpler manner in that exploration is conducted only in the first episode. The subsequent simulations for training are performed with this simplified method.

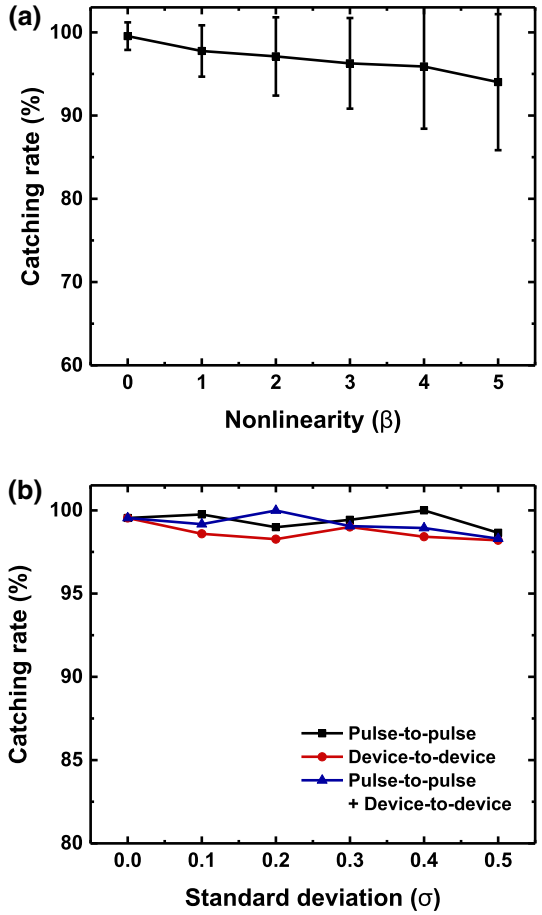

Fig. 6 **a** The average catching rate of the network as a function of the nonlinearity factor (β). **b** The average catching rate of the network as a function of the standard deviation (σ) of the device variations

In addition to the two examples discussed above, 18 random examples were trained under identical conditions. When the optimal number of moves is reached for each example, it is considered that the training is done well. Otherwise, it is considered that the network needs more training. Figure 8 presents how many of the 20 examples reached the optimal number of moves. The inset in Fig. 8 shows the accuracy of the last 50 episodes. As training progresses, the accuracy converges to 100%. This indicates that the network can be trained well for various game examples.

3.3 Network without replay memory

Thus far, we have trained the network using replay memory with a size of 500 in all simulations. However, for a relatively simple problem such as a Fruit Catching game, by not using the replay memory, various advantages, such as

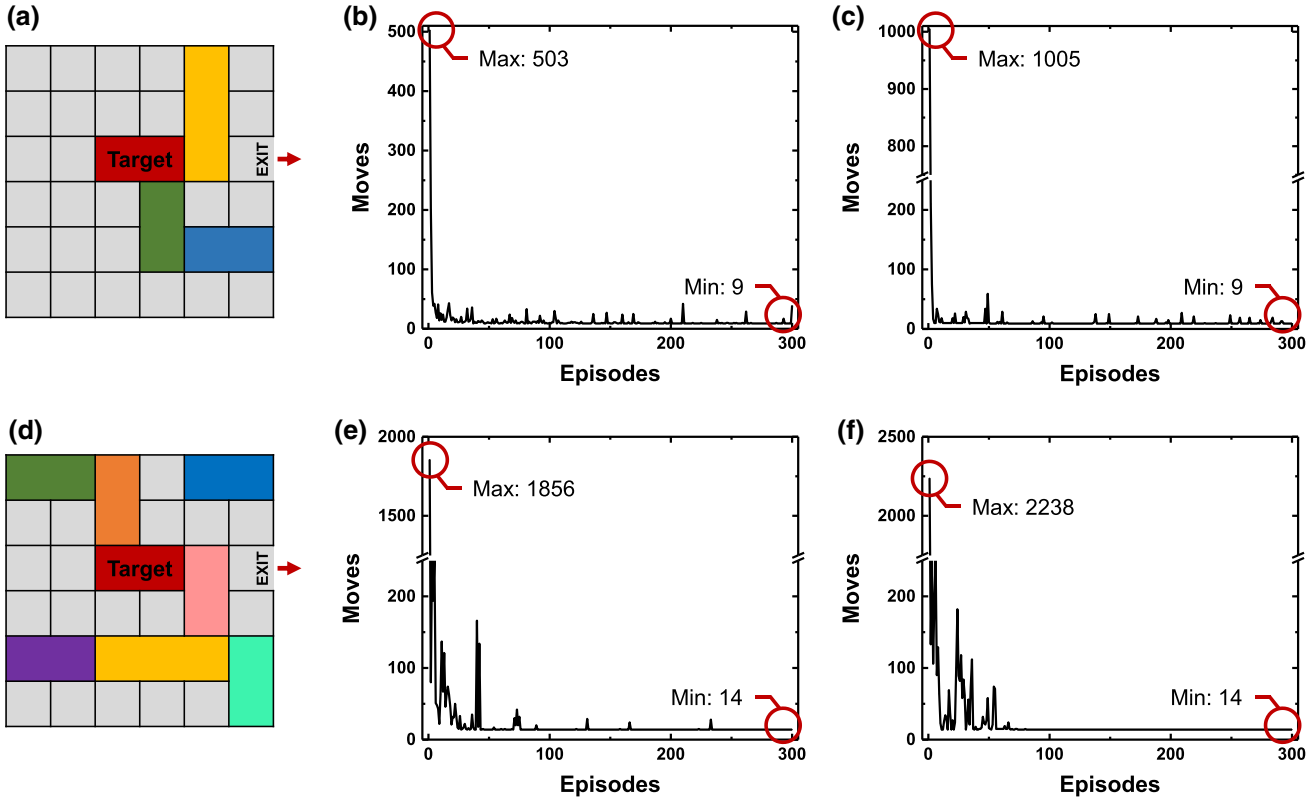

Fig. 7 **a** Relatively easy example of the Rush Hour game. **b** The number of moves required to move the target car **a** to the exit as a function of the episodes when $\varepsilon = 1, \gamma = 0.9$, **c** only exploration occurs in the first episode. **d** Relatively complicated example of the Rush Hour game. **e** The number of moves required to move the target car **d** to the exit as a function of the episodes when $\varepsilon = 1, \gamma = 0.9$, **f** only exploration occurs in the first episode

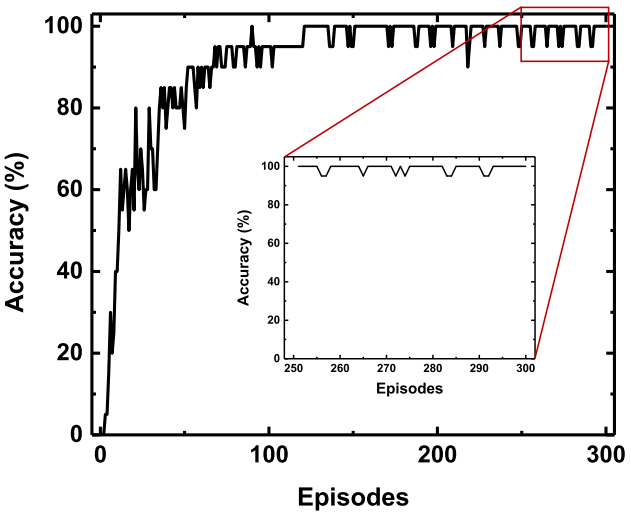

Fig. 8 Ratio of reaching optimal move among the 20 random example games as a function of the episodes

better power consumption, occupied area, and fast learning speed outcomes, can be obtained. Figure 9 shows the overall process of training when the replay memory is not used. Only one set of (s, a, r, s') data is stored for each moment and is used for network training. As shown in ③ of Fig. 2, the network with replay memory receives s' and stores it in the replay memory first, after which the network is trained (④ of Fig. 2). However, in the network without replay memory, s' is not stored and is immediately applied to the input of the DQN (③ of Fig. 9), which becomes the first phase of the four-phase training process. Therefore, in ④ of Fig. 9, it is sufficient to proceed with training from the second phase, which increases the overall training speed.

Figure 10a presents the catching rate of the network without the replay memory. The size of the network used in the simulation and all other parameters are identical to those in Fig. 5b, except that the replay memory is not used and the network is trained with only one dataset for each action. Therefore, more episodes are needed for training compared to the network with replay memory. However, because the memory access required for training is significantly reduced for each episode, the total time required for training is decreased. The catching rate was obtained through 1000 test games. The inset in Fig. 10a shows the catching rate for the last 200 episodes. Figure 10b shows the average value of the catching rate during the last 200

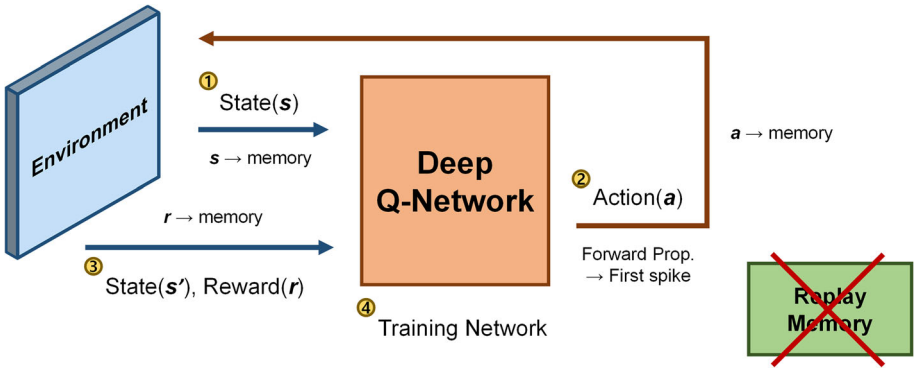

Fig. 9 Overall training process of hardware-based DQN without replay memory consisting of two elements and four training steps

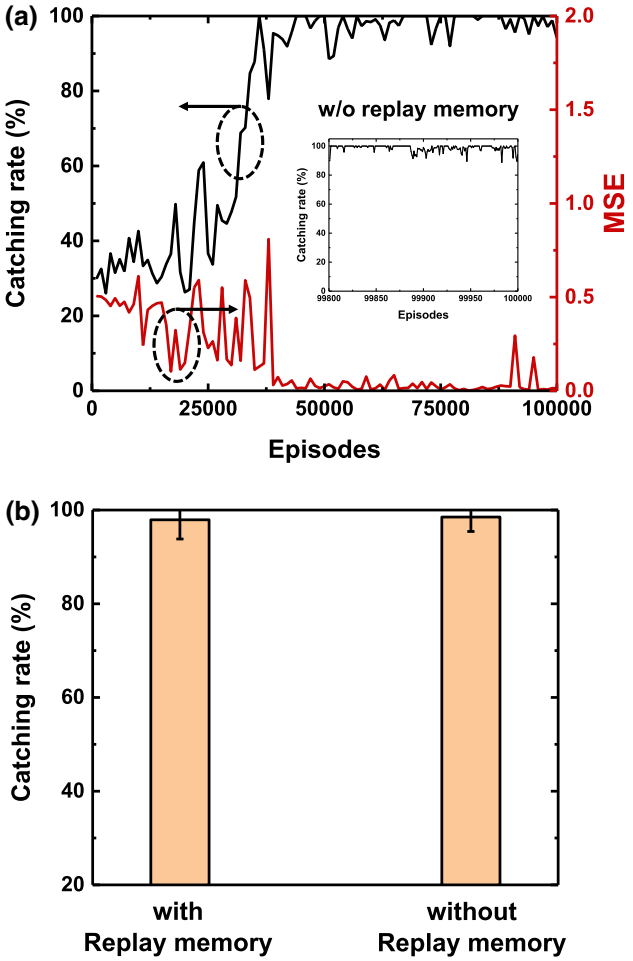

Fig. 10 **a** The catching rate of the proposed hardware-based network without replay memory ($\varepsilon = 0, \gamma = 1.0$). **b** The average catching rate during the last 200 episodes for the network with and without replay memory

episodes for the network with and without the replay memory. For a simple problem, the network can be trained well regardless of whether or not the replay memory is used.

4 Conclusion

In this paper, we proposed a training method for on-chip trainable hardware-based DQNs. The entire training process is divided into four phases: two forward phases, one backward phase, and one update phase. In each forward phase, two values are stored in each case: the value for the target spike ($\max_{a'} Q(s', a')$) and the generation of the spike ($\text{sign}\left(\sum_t^T S_j^l(t)\right)$) for weight update. In the backward phase and update phase, a training method approximating the conventional backpropagation algorithm is used. To implement on-chip training, only a single bit of memory per neuron is used, and the dependency of memory is low.

The performance of the proposed training method is evaluated through two example games: a Fruit Catching game and a Rush Hour game. Evaluation results show that the network is trained well without significant performance differences relative to the outcomes from a software-based training method in both cases. In particular, for one of the simple games here, specifically the Fruit Catching game, high performance in the form of a catching rate of approximately 98% was achieved despite the fact that the replay memory was not used. This means that the network can be suitably trained while significantly reducing the use of memory, thus reducing the power consumption and area occupation that comes with memory usage. In addition, further performance improvements can be achieved through optimization of the parameters used in the simulation. Dealing with large input image data is a challenging future study. It might require a large amount of replay memory and additional convolutional neural networks (CNNs). This issue will be addressed more thoroughly in future work.

In this work, TFT-type flash memory cells are used as synaptic devices. Because the FG covers only half of the channel, the threshold voltage does not fall below 0 in the full erase state, and the standby power consumption is reduced by preventing leakage current. In addition, the bidirectional conductance update characteristic makes this

device suitable for use as a synaptic device in which conductance updates frequently occur.

The effects of non-ideal properties of the synaptic devices are also investigated. Nonlinear characteristics and two variations of the synaptic devices are considered. The performance of the proposed training method is evaluated while increasing β, a factor indicating the nonlinearity of the synaptic device, to 5. There is a slight decrease in the performance as β is increased, but overall the outcome indicates good performance nonetheless. In addition, because the on-chip training scheme is employed, the proposed system shows strong immunity to device variations.

Acknowledgements This work was supported by the Brain Korea 21 Plus Project in 2020, and National Research Foundation of Korea (NRF-2016M3A7B4909604).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Kuzum D, Yu S, Wong H-S (2013) Synaptic electronics: materials, devices and applications. Nanotechnology 24(38):382001. https://doi.org/10.1088/0957-4484/24/38/382001
2. Indiveri G, Liu S-C (2015) Memory and information processing in neuromorphic systems. Proc IEEE 103(8):1379–1397. https://doi.org/10.1109/JPROC.2015.2444094
3. McKee SA (2004) Reflections on the memory wall. Proc Conf Comput Front. https://doi.org/10.1145/977091.977115
4. Kim C-H, Lim S, Woo SY, Kang W-M, Seo Y-T, Lee S-T, Lee S, Kwon D, Oh S, Noh Y, Kim H, Kim J, Bae J-H, Lee J-H (2018) Emerging memory technologies for neuromorphic computing. Nanotechnology 30(3):1–33. https://doi.org/10.1088/1361-6528/aae975
5. Yu S (2018) Neuro-inspired computing with emerging nonvolatile memory. Proc IEEE 106(2):260–285. https://doi.org/10.1109/JPROC.2018.2790840
6. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536. https://doi.org/10.1038/323533a0
7. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444. https://doi.org/10.1038/nature14539
8. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. Adv Neural Inform Process Syst (NIPS), pp 1097–1105 http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks
9. Burr GW, Shelby RM, Sebastian A, Kim S, Kim S, Sidler S, Virwani K, Ishii M, Narayanan P, Fumarola A, Sanches LL, Boybat I, Gallo ML, Moon K, Woo J, Hwang H, Leblebici Y (2016) Neuromorphic computing using non-volatile memory. Adv Phys X 2(1):89–124. https://doi.org/10.1080/23746149.2016.1259585
10. Querlioz D, Bichler O, Dollfus P, Gamrat C (2013) Immunity to device variations in a spiking neural network with memristive nanodevices. IEEE Trans Nanotechnol 12(3):288–295. https://doi.org/10.1109/TNANO.2013.2250995
11. Bichler O, Querlioz D, Thorpe SJ, Bourgoin J-P, Gamrat C (2012) Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. Neural Netw 32:339–348. https://doi.org/10.1016/j.neunet.2012.02.022
12. Diehl PU, Cook M (2015) Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front Comput Neurosci 9(99):1–9. https://doi.org/10.3389/fncom.2015.00099
13. Ambrogio S, Ciocchini N, Laudato M, Milo V, Pirovano A, Fantini P, Ielmini D (2016) Unsupervised learning by spike timing dependent plasticity in phase change memory (PCM) synapses. Front Neurosci 10(56):1–12. https://doi.org/10.3389/fnins.2016.00056
14. Masquelier T, Thorpe SJ (2010) Learning to recognize objects using waves of spikes and Spike Timing-Dependent Plasticity. Int Joint Conf Neural Netw (IJCNN). https://doi.org/10.1109/IJCNN.2010.5596934
15. Suri M, Bichler O, Querlioz D, Cueto O, Perniola L, Sousa V, Vuillaume D, Gamrat C, DeSalvo B (2011) Phase change memory as synapse for ultra-dense neuromorphic systems: Application to complex visual pattern extraction. IEDM Tech Dig. https://doi.org/10.1109/IEDM.2011.6131488
16. Yu S, Gao B, Fang Z, Yu H, Kang J, Wong H-SP (2012) A neuromorphic visual system using RRAM synaptic devices with Sub-pJ energy and tolerance to variability: experimental characterization and large-scale modelling. IEDM Tech Dig. https://doi.org/10.1109/IEDM.2012.6479018
17. Sidler S, Pantazi A, Wozniak S, Leblebici Y, Eleftheriou E (2017) Unsupervised learning using phase-change synapses and complementary patterns. Int Conf Artif Neural Netw (ICANN). doi: https://doi.org/10.1007/978-3-319-68600-4-33
18. Kim H, Hwang S, Park J, Yun S, Lee J-H, Park B-G (2018) Spiking neural network using synaptic transistors and neuron circuits for pattern recognition with noisy images. IEEE Electron Device Lett 39(4):630–633. https://doi.org/10.1109/LED.2018.2809661
19. Burr GW, Shelby RM, Sidler S, Nolfo C, Jang J, Boybat I, Shenoy RS, Narayanan P, Virwani K, Giacometti EU, Kurdi BN, Hwang H (2015) Experimental demonstration and tolerancing of a large-scale neural network (165000 synapses) using phase-change memory as the synaptic weight element. IEEE Trans Electron Dev (TED) 62(11):3498–3507. https://doi.org/10.1109/TED.2015.2439635
20. Liu B, Li H, Chen Y, Li X, Wu Q, Huang T (2015) Vortex: variation-aware training for memristor X-bar. In: Proceedings of the 52nd annual design automation conference (DAC), pp 1–6. https://doi.org/10.1145/2744769.2744930
21. Prezioso M, Merrikh-Bayat F, Hoskins BD, Adam GC, Likharev KK, Strukov DB (2015) Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature 521:61–64. https://doi.org/10.1038/nature14441

22. Hu M, Li H, Chen Y, Wu Q, Rose GS, Linderman RW (2014) Memristor crossbar-based neuromorphic computing system: a case study. IEEE Trans Neural Netw Learn Syst 25(10):1864–1878. https://doi.org/10.1109/TNNLS.2013.2296777
23. Yu S (2017) Neuro-inspired computing using resistive synaptic devices. Springer, New York. https://doi.org/10.1007/978-3-319-54313-0
24. Hasan R, Taha TM, Yakopcic C (2017) On-chip training of memristor crossbar based multi-layer neural networks. Microelectron J 66:31–40. https://doi.org/10.1016/j.mejo.2017.05.005
25. Pfeiffer M, Pfeil T (2018) Deep learning with spiking neurons: opportunities and challenges. Front Neurosci 12(774):1–18. https://doi.org/10.3389/fnins.2018.00774
26. Tavanaei A, Maida A (2019) BP-STDP: approximating backpropagation using spike timing dependent plasticity. Neurocomputing 330(22):39–47. https://doi.org/10.1016/j.neucom.2018.11.014
27. Zhang Q, Wu H, Yao P, Zhang W, Gao B, Deng N, Qian H (2018) Sign backpropagation: an on-chip learning algorithm for analog RRAM neuromorphic computing systems. Neural Netw 108:217–223. https://doi.org/10.1016/j.neunet.2018.08.012
28. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518:529–533. https://doi.org/10.1038/nature14236
29. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lilicrap T, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of Go without human knowledge. Nature 550:354–359. https://doi.org/10.1038/nature24270
30. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint, arXiv: 1312.5602, pp 1–9.
31. Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning. In: Proceedings of the AAAI, pp 1–7.
32. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N (2016) Dueling network architectures for deep reinforcement learning. arXiv preprint, arXiv: 1511.06581, pp 1–15
33. Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. Adv Neural Inform Process Syst (NIPS), pp 1–12
34. Xu Z, van Hasselt HP, Silver D (2018) Meta-gradient reinforcement learning. Adv Neural Inform Process Syst (NIPS), pp 1–12.
35. Srinivasan S, Lanctot M, Zambaldi V, Perolat J, Tuyls K, Munos R, Bowling M (2018) Actor-critic policy optimization in partially observable multiagent environments. Adv Neural Inform Process Syst (NIPS), pp 1–14.
36. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint, arXiv: 1707.06347, pp 1–12
37. Mnih V, Badia AP, Mirza M, Graves A, Harley T, Lillicrap TP, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd international conference on machine learning, vol 48, pp 1928–1937.
38. Hessel M, Modayil J, van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M, Silver D (2017) Rainbow: combining improvements in deep reinforcement learning. arXiv preprint, arXiv: 1710.02298, pp 1–14.
39. Kang W-M, Kim C-H, Lee S, Woo SY, Bae J-H, Park B-G, Lee J-H (2019) A spiking neural network with a global self-controller for unsupervised learning based on spike-timing-dependent plasticity using flash memory synaptic devices. Int Joint Conf Neural Netw (IJCNN). https://doi.org/10.1109/IJCNN.2019.8851744
40. Kim C-H, Lee S, Woo SY, Kang W-M, Lim S, Bae J-H, Kim J, Lee J-H (2018) Demonstration of unsupervised learning with spike-timing-dependent plasticity using a TFT-type NOR flash memory array. IEEE Trans Electron Devices (TED) 65(5):1774–1780. https://doi.org/10.1109/TED.2018.2817266
41. Lim S, Bae J-H, Eum J-H, Lee S, Kim C-H, Kwon D, Park B-G, Lee J-H (2019) Adaptive learning rule for hardware-based deep neural networks using electronic synapse devices. Neural Comput Appl 31(11):8101–8116. https://doi.org/10.1007/s00521-018-3659-y
42. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. IEEE Int Conf Comput Vis (ICCV), pp 1026–1034. https://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html