nature electronics

ARTICLES

https://doi.org/10.1038/s41928-019-0221-6

Reinforcement learning with analogue memristor arrays

Zhongrui Wang[1,8], Can Li[1,8], Wenhao Song[1], Mingyi Rao[1], Daniel Belkin[1], Yunning Li[1], Peng Yan[1], Hao Jiang[1], Peng Lin[1], Miao Hu[2], John Paul Strachan[3], Ning Ge[3], Mark Barnell[4], Qing Wu[4], Andrew G. Barto[5], Qinru Qiu[6], R. Stanley Williams[7], Qiangfei Xia[1]* and J. Joshua Yang[1]*

Reinforcement learning algorithms that use deep neural networks are a promising approach for the development of machines that can acquire knowledge and solve problems without human input or supervision. At present, however, these algorithms are implemented in software running on relatively standard complementary metal–oxide–semiconductor digital platforms, where performance will be constrained by the limits of Moore's law and von Neumann architecture. Here, we report an experimental demonstration of reinforcement learning on a three-layer 1-transistor 1-memristor (1T1R) network using a modified learning algorithm tailored for our hybrid analogue–digital platform. To illustrate the capabilities of our approach in robust in situ training without the need for a model, we performed two classic control problems: the cart–pole and mountain car simulations. We also show that, compared with conventional digital systems in real-world reinforcement learning tasks, our hybrid analogue–digital computing system has the potential to achieve a significant boost in speed and energy efficiency.

A primary goal of machine learning is to equip machines with behaviours that optimize their control over different environments. Unlike supervised or unsupervised learning, reinforcement learning, which is inspired by cognitive neuroscience, provides a way to formulate the decision-making process that is learned without a supervisor providing labelled training examples. It instead uses less informative evaluations in the form of 'rewards', and learning is directed towards maximizing the amount of rewards received over time[1].

Developments in deep neural networks have advanced reinforcement learning[2,3], exemplified with the recent achievements of AlphaGo[4,5]. However, the first generation of AlphaGo (or AlphaGo Fan) ran on 1,920 central processing units (CPUs) and 280 graphics processing units (GPUs), consuming a peak power of half a megawatt. Application-specific integrated circuits (ASIC) like DaDianNao[6], tensor processing unit (TPU)[7] and Eyeriss[8] offer potential enhancements in speed and reductions in power consumption. However, since the majority of neural network parameters (for example, weights) are still stored in dynamic random-access memory (DRAM), moving data back-and-forth between the DRAM and the caches (for example, static random-access memory; SRAM) of processing units increases both the latency and power consumption. The growing challenge of this communication bottleneck, together with the saturation of Moore's law, limits the speed and energy efficiency of complementary metal–oxide–semiconductor (CMOS)-based reinforcement learning in the era of big data.

A processing-in-memory architecture could provide a highly parallel and energy-efficient approach to address these challenges, relying on dense, power-efficient, fast, and scalable building blocks such as ionic transistors[9], phase change memory[10–13] and redox memristors[14–26]. A key advantage of networks based on these emerging devices is 'compute by physics', where vector-matrix multiplications are performed intrinsically via Ohm's law (for multiplication) and Kirchhoff's current law (for summation)[27,28]. Such a network computes exactly where the data are stored and thus avoids the communication bottleneck. Furthermore, it is able to compute in parallel and in the analogue domain. Applications, including signal processing[29,30], scientific computing[31,32], hardware security[33] and neuromorphic computing[22,28,34–41], have been recently demonstrated (see Supplementary Table 1).

Memristor arrays have shown potential speed and energy enhancements for in situ supervised learning (for spatial/temporal pattern classification)[22,34–37,39,41,42] and unsupervised learning (for data clustering)[40,43,44]. Although the memristor crossbar implementation of reinforcement learning may significantly benefit the reward predictions based on the forward passes of the deep-Q network, using historical observations repeatedly replayed from the 'experience' to optimize the decision-making in unknown environments[3], it has yet to be demonstrated due to lack of the hardware and its corresponding algorithm.

In this Article, we report an experimental demonstration of reinforcement learning in analogue memristor arrays. The parallel and energy-efficient in situ reinforcement learning with a three-layer fully connected memristive deep-Q network is implemented on a 128×64 1-transistor 1-memristor (1T1R) array. We show that the learning can be generally applied to classic reinforcement learning environments, including cart–pole[45] and mountain car[46] problems. Our results indicate that in-memristor reinforcement learning can achieve a 4 to 5 bit representation capability per weight using a two-pulse write-without-verification scheme to program the 1T1R array, with potential improvements in computing speed and energy efficiency (see Supplementary Note 1).

[1]Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA. [2]Binghamton University, Binghamton, NY, USA. [3]Hewlett Packard Labs, Hewlett Packard Enterprise, Palo Alto, CA, USA. [4]Air Force Research Laboratory, Information Directorate, Rome, NY, USA. [5]College of Information and Computer Sciences, University of Massachusetts, Amherst, MA, USA. [6]Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA. [7]Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. [8]These authors contributed equally: Zhongrui Wang, Can Li. *e-mail: qxia@umass.edu; jjyang@umass.edu

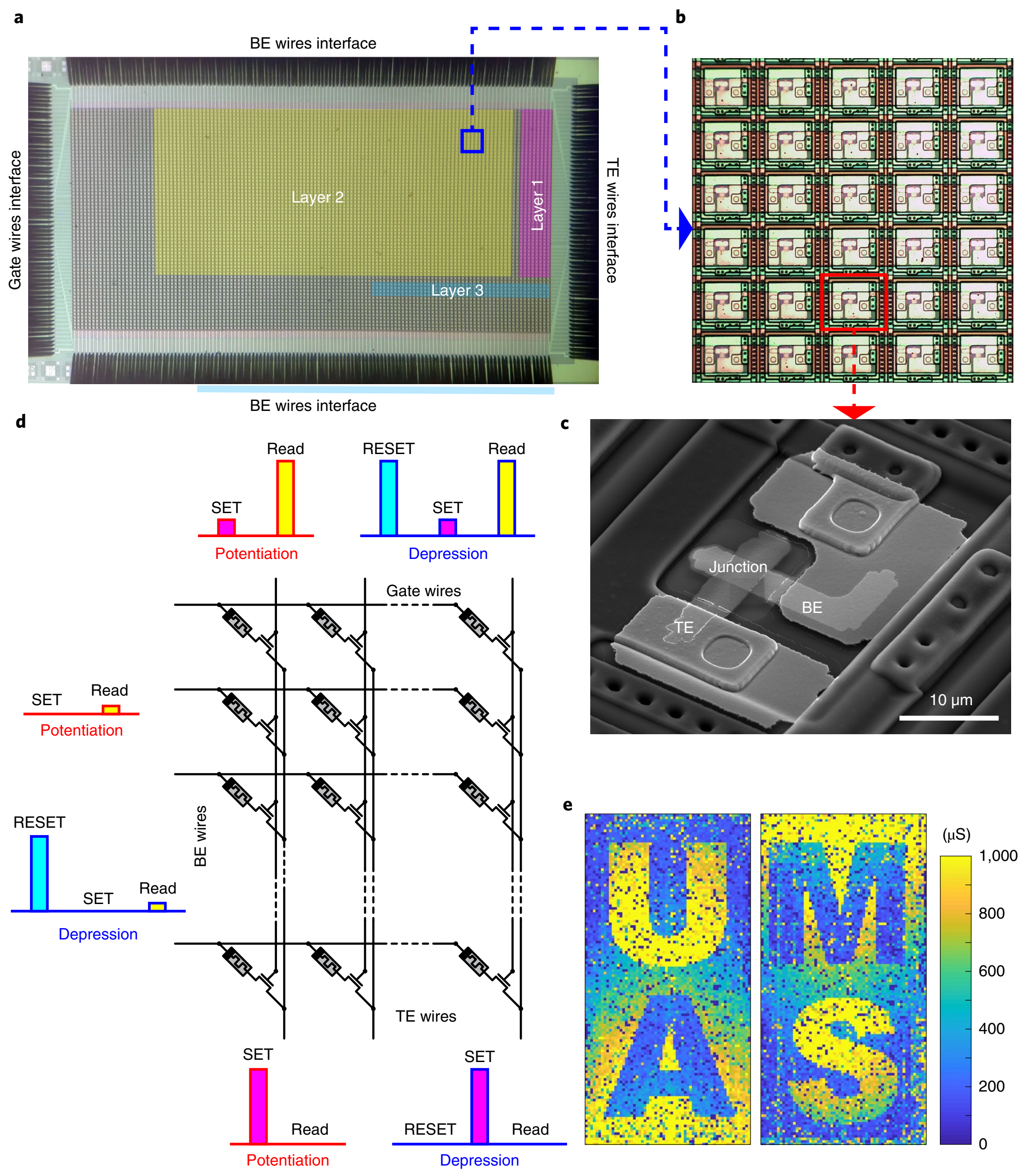

Fig. 1 | Memristor synapse array and programming scheme. a, Optical micrograph of the 128 × 64 memristor synaptic network with the probe card in place. The colour blocks illustrate how the array is partitioned to form the three-layer neural network in Fig. 3, with 48 neurons in each of the hidden layers. The dimensions of the layers are 8 × 48, 96 × 48 and 48 × 4, respectively. Differential pairs are formed between adjacent bottom (top) electrode wires in the first two (last) layers (layer), using a total of 5,184 memristors. **b**, Magnified view of the selected subarray in **a**. Memristive synapses of the same row share bottom electrode (BE) wires while those of the same column share top electrode (TE) and transistor gate wires. **c**, Scanning electron micrograph of a single 1T1R cell in **b**. The crossbar junction of the Pd/HfO_2/Ta memristor sits on top of a foundry-built transistor. **d**, Schematic illustration of the transistor-assisted synaptic weight update scheme. Positive voltage pulses were applied to the TE of the memristor and the gate of the associated transistor to potentiate the synapse. Positive voltage pulses were applied to the memristor BE and the gate of the transistor to fully depress the synapse before applying another potentiation to achieve a targeted conductance. The memristors form a crossbar array if all transistors are switched ON. **e**, Conductance map of the memristor synaptic array after a single round of programming to write the letters 'UMAS'.

1T1R memristor array platform and synapse programming

The optical image of the 1T1R memristor array we used to store the synaptic weights and perform parallel computing is shown in Fig. 1a. Such a chip provides a platform to build an assembly of smaller layers to constitute a functional multilayer network, as to be discussed in the coming text. Memristive synapses of the same row share bottom electrode (BE) wires while those of the same column possess common top electrode (TE) and transistor gate wires (see Fig. 1b,c). The current compliance produced by the series transistors provides an efficient control over synaptic weight tuning at sufficient accuracy. All programming and computational signals are generated off-chip (see Supplementary Fig. 1 for the photo the measurement system).

Figure 1d schematically illustrates the protocol of synapse programming using the current compliance provided by the on-chip transistors, which has been successfully used in our early work on pattern classification[35]. To potentiate a memristor synapse, positive voltage spikes are applied to the top electrode of the memristor and the gate of the associated transistor concurrently. The targeted conductance is linearly proportional to the amplitude of the gate pulse. To produce a depression to a target value, the synapse receives a positive pulse on its bottom electrode to fully depress the synapse before undergoing a potentiation to achieve the desired conductance. Such a synaptic programming scheme works well with our analogue Pd/HfO_2/Ta memristors[47], yielding fast programming speed (see Supplementary Fig. 2), and linear and symmetric conductance ramping at reasonable accuracy[28,35] (see Fig. 1e and Supplementary Fig. 3a). Alternatively, an amplitude-modulated memristor RESET pulse could be used to induce gradients of memristor weights. However, this method may suffer practically from less satisfactory results and rely on empirical techniques to suppress the nonlinearity and asymmetry of programming (see Supplementary Fig. 3b).

In-memory reinforcement learning algorithm

To implement deep-Q learning on memristors, we consider tasks in which an agent interacts with an environment through a sequence of experience replay, action decision, and observation, as schematically illustrated in the flowchart of Fig. 2a, at discrete time steps[1]. The agent is physically implemented with our hybrid analogue–digital computing system, which combines the advantages of the speed–energy efficiency of the analogue memristors in performing the computationally expensive vector-matrix multiplications and transistor circuitry in implementing the digital logic for the remaining part of the reinforcement learning algorithm (that is, draw/add samples from the experience, calculate loss using the Bellman equation, and plan for weight changes with the RMSprop optimizer).

The Q-learning is practiced by the agent, which randomly samples a minibatch from the fixed size pool of stored transitions (s_j,a_j,r_j,s_{j+1}) at each time step, where s_j, a_j and r_j are the state, action and reward at time $j \leq t$ (t is the current time step), respectively (see Fig. 2a). These stored observations, also termed as 'experience', are used to train the agent for the decision-making in the future, which benefits the learning by minimizing correlations between samples.

Since the memristor Q-network serves as the Q-function approximator[2,3], the agent begins learning the strategy from the selected experience by physically performing forward passes on the memristor Q-network with all states s_m of the minibatch, where m refers to the index of any state or future state of the minibatch, as indicated in Fig. 2a. The results of the forward pass are used by the recursive relationship or the Bellman equation[1] to evaluate the mean squared error (MSE) in the digital domain with a general-purpose computer. The error is backpropagated through the three-layer fully connected Q-network using the physically acquired memristor conductance of the Q-network. Note the backpropagation could be completely performed on memristors with bi-directional hardware neurons, which is beyond the scope of this proof-of-concept demonstration. The weight gradients are then used to calculate how the memristor-based Q-network will be updated by the RMSprop, a variant of stochastic gradient descent (SGD) with weight and gradient history dependence[48], in the digital domain using the computer. Here we use the two-pulse write-without-verification programming scheme to potentiate or depress the memristors[35], which has a speed advantage over the write-and-verify programming scheme at a reasonable accuracy (see Supplementary Fig. 3a).

The agent then applies what the memristor Q-network has learned in reacting to the environment, as shown in the blue colour block in Fig. 2a. It is responsible for selecting an action a_t with respect to the current state s_t of the game. The agent estimates action-values, that is, $Q(s_t,a_t)$, defined as the expected sum of discounted future rewards starting in state s_t given that action a_t is taken, by physically performing another forward pass with the state s_t on the memristor Q-network, and thereafter decides the action following the ε-greedy policy[3]. The agent first employs a trial-and-error strategy to randomly explore the environment modulated by the initially large ε. With more accumulated feedback, which evaluates the problem-solving consequences of the previous actions, the agent gradually switches to the greedy policy, which picks action a_t with the largest estimated Q-value, that is, $\max_{a_t \in \{a\}} Q(s_t, a_t)$, among all allowed actions $\{a\}$ to maximize the cumulative future reward. The agent also observes the environment state transition (for example, the subsequent state s_{t+1}) and the reward r_t by adding both to the experience after applying the selected action a_t to the learning environment, which is done in the digital computer.

The weights of the three-layer deep-Q network are physically mapped onto the conductances of three memristor subarrays, as schematically shown in Fig. 2b. Each weight uses a differential pair of memristors for convenient representation of negative weights. In the case of a forward pass, all transistors of the 1T1R array, which are useful only during synaptic weight updating, are fully turned ON so the memristors equivalently form a simple resistor array to implement the physics-based vector-matrix multiplication. The subarrays are interfaced to each other via the hidden neurons, comprising trans-impedance amplifiers (TIAs), analogue-to-digital converters (ADCs) and digital-to-analogue converters (DACs), applying a rectified-linear (ReLU) activation function in the digital domain (see Methods) to the weighted sums physically computed by the memristor crossbar.

For a given sample of the experience replay, the computational complexity for a forward pass is traditionally $O(n_i \times n_{h1})$ + $O(n_{h1} \times n_{h2})$ + $O(n_{h2} \times n_{no})$ where n_i, n_{h1}, n_{h2} and n_{no} are the number of input, first hidden layer, second hidden layer and output layer neurons, respectively. However, the time complexity reduces to $O(1)$ + $O(1)$ + $O(1)$ once the weights are physically mapped to the 1T1R array, because all multiply–accumulate operations (MACs) within the same layer are performed in a single step, illustrating the remarkable energy–speed efficiency associated with the memristor neural network (see Supplementary Note 1). All other computations shown in the computational graph in Fig. 2a are of much less computational complexity. The Bellman equation involves $O(1)$ computational complexity per sample of the minibatch, while the RMSprop optimizer, consisting of elementwise operations, is performed once per minibatch. The time and energy consumption are dominated by the trainable parameter update. Since the memristor crossbar not only serves as the processing unit, but also replaces DRAM as the main memory for storing neural network parameters, suitable on-chip memristor programming could be more efficient in both speed and energy than off-chip DRAM sequential writing (see Supplementary Note 1).

Cart–pole game

We use the cart–pole problem[45] as a testbed for our processing-in-memory reinforcement learning system. A single 128×64 1T1R array was partitioned to construct the three-layer Q-network with 4 input neurons, 48 neurons in each hidden layer, and 2 output neurons. The network consisted of 2,592 weights or differential pairs, implemented on 5,184 memristors of the 1T1R network, as schematically illustrated in Fig. 1a. Compared to the state-of-the-art reinforcement learning on conventional platforms[2,3], the dimension of the memristor Q-network in this proof-of-concept demonstration is constrained by the physical array size, using the low-dimensional state vector of the environment rather than the visualized image frames as the input.

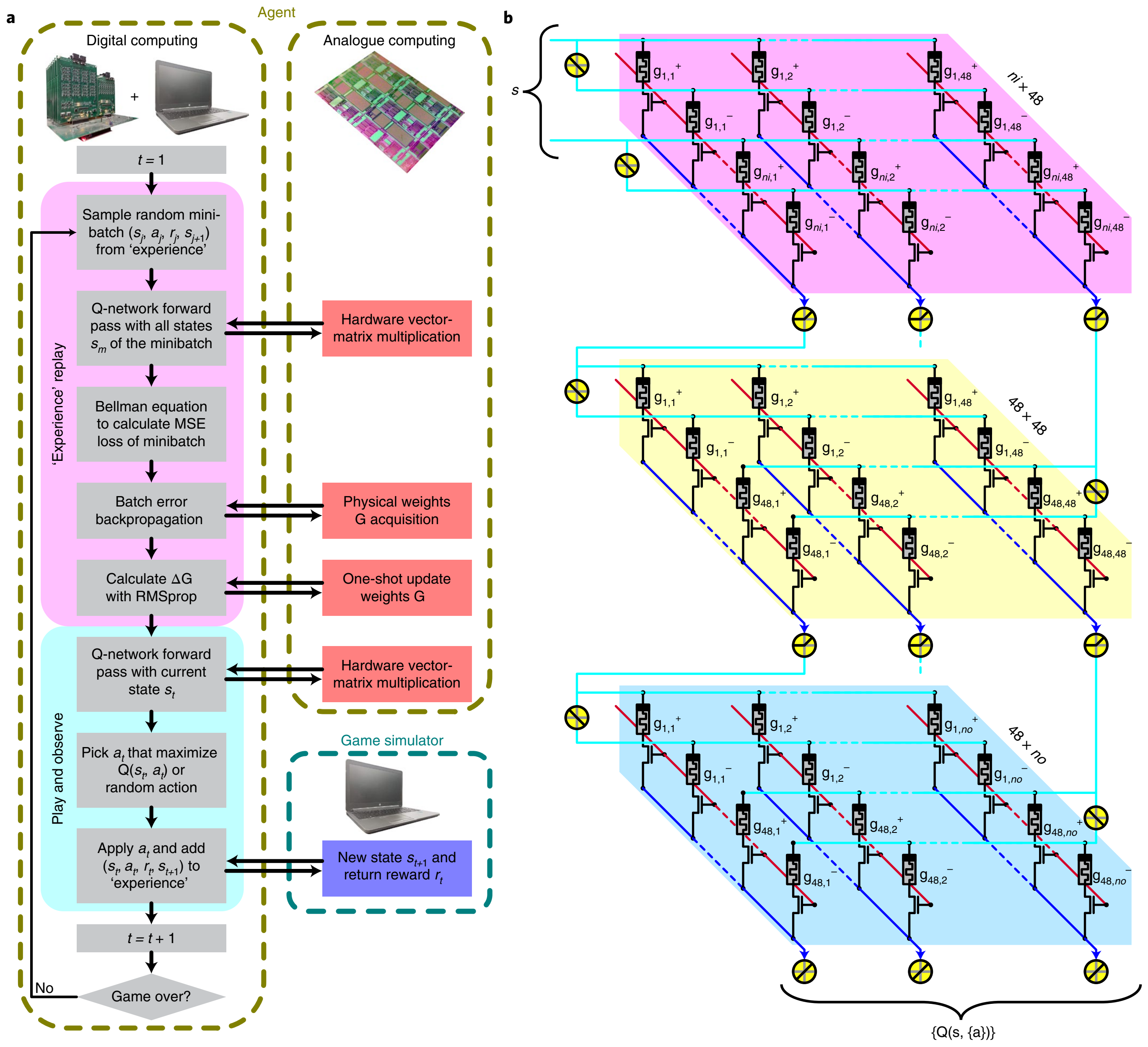
a
Agent
Digital computing
Analogue computing
t = 1
Sample random mini-batch (s_j, a_j, r_j, s_j+1) from 'experience'
Q-network forward pass with all states s_m of the minibatch
Bellman equation to calculate MSE loss of minibatch
Batch error backpropagation
Calculate ΔG with RMSprop
'Experience' replay
Q-network forward pass with current state s_t
Pick a_t that maximize Q(s_t, a_t) or random action
Apply a_t and add (s_t, a_t, r_t, s_t+1) to 'experience'
Play and observe
t = t + 1
Game over?
No
Hardware vector-matrix multiplication
Physical weights G acquisition
One-shot update weights G
Hardware vector-matrix multiplication
Game simulator
New state s_t+1 and return reward r_t
b
s
ni × 48
48 × 48
48 × no
{Q(s, {a})}

Fig. 2 | Scheme of the hybrid analogue–digital reinforcement learning. a, Flowchart of the deep-Q reinforcement learning algorithm. The agent is physically implemented by the hybrid analogue–digital computing platform. The analogue 1T1R memristor array physically performs the computationally expensive vector-matrix multiplications, while the digital components draw/add samples from the experience, calculate loss using the Bellman equation, and plan for weight changes with the RMSprop optimizer. **b**, The three-layer fully connected Q-network, where the input is the state *s* and each output neuron represents the approximate sum of discounted future rewards, or Q-value of the state *s*, by taking a particular allowed action *a*. The subarrays are of dimensions $ni \times 48$, 48×48 and $48 \times no$, where *ni* and *no* denote the numbers of input and output neurons. Hidden layer and output layer neurons implement rectified linear unit (ReLU) function and linear activation on the physically read currents, respectively. Neurons of the input layer or hidden layers physically source paired voltage signals (with equal amplitude but different polarities) to memristor differential pairs. The conductance difference between two memristors of a differential pair maps to the positive or negative weight of the corresponding synapse (see Methods).

Figure 3a shows a schematic representation of a cart–pole scenario[45]. A rigid pole is hinged to a cart which is free to move within the bounds of a one-dimensional track. The pole rotates in the vertical plane above the track due to both the gravitational force and the motion of the cart. The learning agent can either apply an impulsive 'left' or 'right' push of fixed force magnitude to the cart at discrete time intervals. The dynamics of the cart–pole environment were simulated in software (see Methods). The environment was abstracted as a 4-dimensional Markov state vector $\left(x, \frac{\mathrm{d}x}{\mathrm{d}t}, \theta, \frac{\mathrm{d}\theta}{\mathrm{d}t}\right)$, where x and θ are the cart position and pole angle, respectively, which also serves as the input to the memristive Q-network.

The learning was model-free. Therefore, the agent was unaware of the equations of motion of the cart–pole system. The only feedback for evaluating performance was a binary reward signal, which was 0 if the pole fell past a certain angle from the vertical or the cart reached an end of the track, or 1 otherwise. If a '0' reward signal is seen by the agent, it ends the current game epoch and resets the cart–pole environment to start another game. This forms an

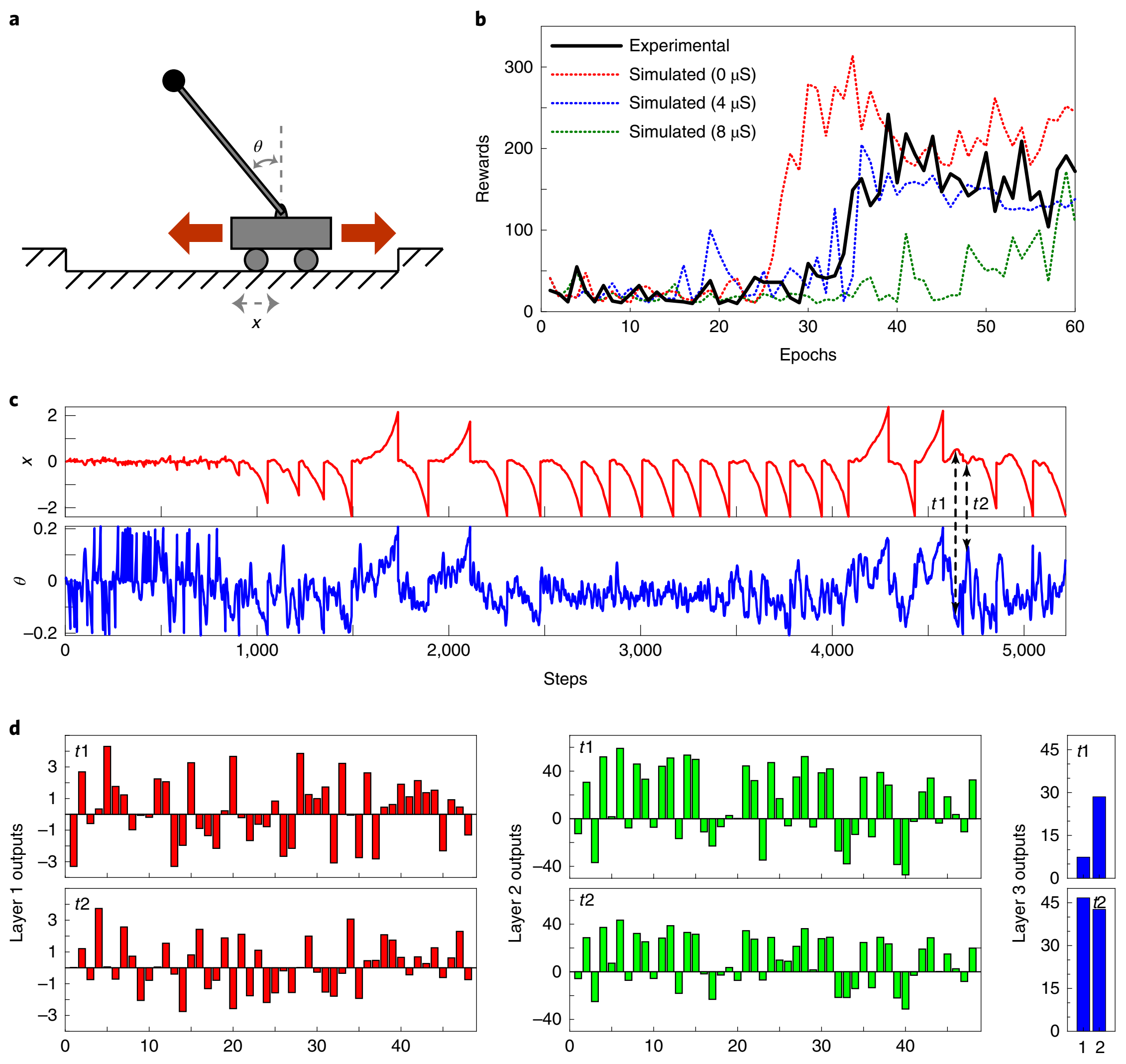

Fig. 3 | In-memristor reinforcement learning in the cart–pole environment. a, Schematic illustration of the cart–pole environment. The cart is free to move along the track while supporting a hinged pole. The learning agent can make a left or right push at each discrete time step to avoid the pole falling or the cart driving beyond the bounds. **b**, Experimental curve of the hybrid analogue–digital system and digitally simulated curves with 0, 4 and 8 μS programming noise tracking the number of rewards per epoch. The experimental memristor performance is similar to the simulation with 4 μS programming noise, following the same trend as the noise-free double-precision floating-point simulation. The simulated curve with 8 μS programming noise had a relatively poor performance with a slow reward rise. **c**, The time evolution of the cart position *x* and pole angle *θ*. The failures of the early game epochs are mainly due to the pole falling. The pole was kept upright (that is, *θ* rarely hit the upper/lower bounds) for many more steps in the later phases of the game. **d**, Output of all layers of the memristor Q-network at time *t*1 and *t*2 specified in **c**. At time *t*1, the pole was tilted anticlockwise. The output of the second neuron of Layer 3, representing the left push (push from the right), was larger. In contrast, at time *t*2, the pole was tilted clockwise. The output of the first neuron of Layer 3, representing the right push (push from the left), was larger.

'avoidance control problem', so the reward '1' makes the agent try to avoid failure for as long as possible.

As shown in Fig. 3b, the memristor backend learning agent scored poorly in the first 30 epochs of the game, as it had not yet learned a good Q-function. Figure 3c shows the time evolution of cart position and pole angle, which reveals that most failures in this phase were triggered by the fact that the pole was no longer upright while the cart was still within the legal range. The failure signal usually occurred after a long sequence of individual actions, which is why the agent needed a thousand steps to figure out what was responsible for the failure. In the second half of the learning course, the agent gradually constructed associations between the input and output based on the reinforcement feedback. The pole was almost upright in this phase, as depicted in Fig. 3c, which reveals that the memristor Q-network had gained the capability to balance the pole (see the full evolution of memristor conductance maps, weights, and gate voltages in Supplementary Video 1). To benchmark the performance of the hybrid analogue–digital system, we implemented the exact same learning digitally with weights of varying precision (or equivalently simulated memristors of varying programming noise). Note that the Pd/HfO_2/Ta memristor used here is resilient to conductance drift[47]. The current reading is acquired by 16-bit ADCs, as detailed in the Methods, with negligible reading noise. To capture the cycle-to-cycle and device-to-device variation in memristor programming, we assumed the conductance update noise following a normal distribution with standard deviations of 0, 4 and 8 μS. The experimental curve in Fig. 3b is almost indistinguishable from the simulated curves with 4 μS programming noise. On the other hand, we observed a consistent trend with the training of noise-free weight update (or equivalently double-precision

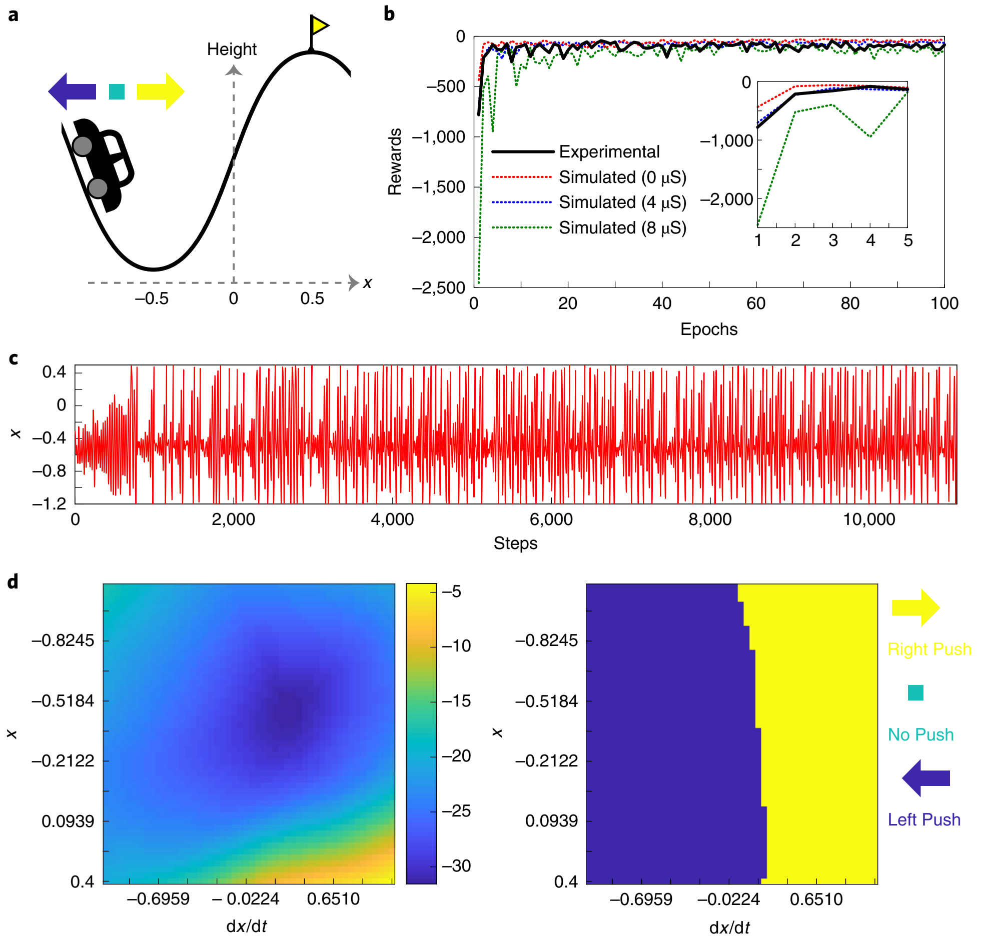

Fig. 4 | In-memristor reinforcement learning in the mountain car environment. a, Schematic illustration of the mountain car environment. The car is originally situated in the bottom of the valley. Its engine is too weak to overcome the gravitational force to reach the goal marked by the flag. The learning agent can make a left push, right push, or no push at each discrete time step to make the car reach the target in the shortest time. **b**, Experimental curve of the hybrid analogue–digital system and digitally simulated curves with 0, 4 and 8 µS programming noise tracking the number of rewards per epoch. The experimental memristor performance is similar to the simulated one with 4 µS programming noise, following the same trend as the noise-free double-precision floating-point simulation. The simulated curve with 8 µS programming noise has a relatively poor performance with more negative rewards in the early epochs, as illustrated in the zoomed inset. **c**, The time evolution of the car position *x*. The agent quickly discovered the technique to drive the car back and forth, as indicated by the oscillation patterns. **d**, Left: value map of all possible input states over the two-dimensional input domain, given by the maximum Q-value associated with the allowed actions of a particular input state. Notably, the right bottom corner is the highest action value since the positive position and velocity has guaranteed success. Similarly, the left upper corner is a local maximum because the altitude equips the car with a large potential energy to accelerate in the subsequent rightward motion. Right: learned map of actions of all possible input states over the two-dimensional input domain. The agent exerts a left (right) push if the car moves left (right), which helps the car to quickly build up momentum.

floating-point simulation). The simulated curve with 8 µS programming noise shows a fairly poor performance, with a slower rise in rewards. Given the simulated memristance in the range [109 µS, 273 µS], the 4 µS noise yields a 4 to 5 bit representation capability per weight with the two-pulse write-without-verification programming scheme. The noise-induced performance degradation is further confirmed in Supplementary Fig. 4a,b, where increasing programming noise clearly increases the number of epochs needed to achieve two consecutive rewards that both are larger than 100, simulated with different uniformly and normally distributed initial weights. The stochasticity of memristors could be suppressed by engineering the memristor materials such as to employ crystalline switching media with controlled dislocations to restrict the trajectories of the ions[49].

Sample responses of the network in different scenarios are illustrated in Fig. 3d. At time *t*1, the angle of the pole was negative, indicating the pole tilted towards the left end of the track. The Q-network suggested the action of a left push, which could physically make the cart accelerate leftwards to restore the balance of the pole. In contrast, at time *t*2, the pole experienced a clockwise rotation. A right push was made by the learning agent to avoid further tilting.

Mountain car game

To show that the same system can be applied to another type of control problem, we applied the same memristor Q-network to the mountain car game[46] while keeping the hyperparameters and learning procedures the same (see Supplementary Table 3).

As shown in Fig. 4a, the under-powered car must drive up a steep hill to reach the flag position[46]. The car is initially situated in a valley. Since gravity is strong whereas the engine of the car is weak, the car cannot simply accelerate up the steep slope. Therefore, the agent must learn to leverage its potential energy by repeatedly backing up the hill to the left and then going forwards up the hill on the right towards the goal. The mountain car environment could be abstracted to a vector $\left(x, \frac{dx}{dt}\right)$, where x is the car position.

Similar to the cart–pole problem, the training consisted of a series of epochs, each starting from a randomly selected non-goal state (see Methods) and continuing until the goal region was reached. In contrast to the cart–pole problem, on each time step a penalty (negative reward) of −1 was incurred here. The mountain car problem needs to minimize penalties, or negative rewards, by reaching the goal region and ending the game as soon as possible. To facilitate training, a pre-training memory of 20,000 random walk samples (see Supplementary Fig. 5) was included in the stored experience in advance.

As shown in Fig. 4b, the learning agent took more than 800 steps in its first game epoch before the car reached the goal region. This success made the agent discover a relatively long-term strategy, which is reflected by the clearly improved performances in the subsequent epochs (see Fig. 4b). The agent had mastered the technique to swing the car back and forth, which is illustrated by the oscillation pattern of the car position over time steps in Fig. 4c (see the full evolution of memristor conductance maps, weights, and gate voltages in Supplementary Video 2). Like the cart–pole problem, the performance of the hybrid analogue–digital system in Fig. 4b tightly follows the simulated curve with the 4 μS programming noise, verifying the 4 to 5 bit representation capability of the memristive synapse with the two-pulse write-without-verification programming scheme. On the other hand, a consistent trend in reward evolution is observed with the noise-free weight update (or equivalently double-precision floating-point simulation). The simulated curve with 8 μS programming noise shows more negative rewards in the first few epochs depicted in the inset of Fig. 4b, which is also confirmed by Supplementary Fig. 4c,d, where the negative rewards of the first episode ramp up quickly with the increasing noise, simulated with different uniformly and normally distributed initial weights.

We examined the representations learned by the memristor Q-network that underpinned the successful performance of the agent in the context of the game in Fig. 4d. The Q-network has identified the action value of possible inputs without knowledge of the game (see left panel of Fig. 4d). Notably, the right bottom corner is the highest action value, since the position is close to the goal region with a sufficiently large positive car speed that could make the car easily hit the flag. Interestingly, the left upper corner, corresponding to the state with a position at the left end of the track and a large negative velocity, is associated with a much larger action value than those in the centre of the map. If the car reaches the left end of the track, its altitude has secured a large potential energy that can be released in the subsequent right movement towards the flag. The right panel of Fig. 4d shows the learned map of actions. Basically, the push direction is speed-dependent. If the car moves left, the agent will exert a left push, otherwise a right push, which is exactly the strategy for a quick build-up of momentum.

Conclusions

We have experimentally demonstrated deep-Q reinforcement learning in emerging memory devices. Our hybrid analogue–digital architecture successfully learned strategies for different classical control environments (the cart–pole and mountain car) with minimal prior knowledge, exhibiting 4 to 5 bit representation capability per weight with the two-pulse write-without-verification programming scheme. Since the performance of the reinforcement learning is largely dependent on the intrinsic stochasticity of the memristor, material system engineering or circuit-level mitigation methods to suppress the programming stochasticity of the memristors could further boost the performance. In addition, we showed that, compared with conventional digital systems in real-world reinforcement learning tasks, our approach has the potential to achieve a nearly one order of magnitude boost in speed and energy efficiency (see Supplementary Note 1). Our hybrid analogue–digital computing could lead to fast and low-power autonomous learning systems.

Methods

Fabrication of 1T1R memristors. The 1T1R arrays are composed of Pd/HfO$_2$/Ta memristors. The front-end and part of the back-end processes to build the transistor arrays were completed in a commercial foundry. To make good electrical connections between the top Al layers and the memristors, an argon plasma treatment was used to remove the native Al$_2$O$_3$ layers, which was followed sequentially by the deposition of 8 nm Ag, 2 nm Ti and 200 nm Pd by sputtering, lift-off to define features, and annealing at 300 °C for 0.6 hours. A 5 nm Ta adhesive layer and 60 nm Pd bottom electrodes were subsequently deposited by sputtering and followed by lift-off. The HfO$_2$ switching layer was deposited by atomic layer deposition at 250 °C. The patterning of the switching layer was performed by photolithography and reactive ion etching in an ambient of CHF$_3$ and O$_2$. The top electrodes of 50 nm Ta and 20 nm Pd were processed by sputtering and lift-off, forming 4 × 4 μm^2 cross-junctions.

Measurement set-up. An in-house measurement system has been built to electrically read and write the 1T1R chip[28,30]. The system features 128 + 64 + 64-way concurrent analogue voltage inputs (up to ±10 V) with a minimum pulse width of ~100 ns and parallel current sensing capability (see Supplementary Fig. 1 for the photo of the system and Supplementary Fig. 6 for the functional block diagram).

Each row wire or gate wire of the 1T1R memristor array could be independently configured for voltage biasing, ground, or high-impedance. For voltage biasing, each wire is driven by a channel of a DAC (LTC2668, Analog Devices). The DAC communicates with the micro-controller unit (MCU) via digital I/O ports. A fast trigger pulse controlled by the MCU allows a minimum ~100 ns DAC pulse generation.

Each column wire of the 1T1R memristor array could be independently configured for voltage biasing, ground, high-impedance, or current sensing. The voltage biasing is implemented in the same manner with the row or gate wires. For current sensing, each column wire connects to one of the four TIAs (LTC6268, Analog Devices) with different gains. The voltage outputs of the TIAs are read by the ADCs (MAX11046, Maxim Integrated) and fetched by the MCU via its digital I/O before being sent back to the computer.

Basic electrical array operations. The basic electrical operations of the 1T1R memristor array include potentiation programming, weight readout and vector-matrix multiplication. All operations are performed by the measurement system with the aid of the on-chip transistors (see Supplementary Table 2 for transistor operation schemes).

For potentiation programming, the memristor array is programmed row-by-row (see the flowchart of Supplementary Fig. 7a). To program a selected row, all row wires are floated, except the selected one, which is grounded. Each gate wire is assigned a different voltage based on the targeted conductance of the associated memristor. All column wires (TEs of memristors) are biased with the same SET voltages. Therefore, this scheme programs all memristors of the same row simultaneously.

For weight readout of all 128 × 64 memristors, the row wires receive a scaled identity voltage matrix in 128 time steps (that is, the only nonzero input V_{Read} at the ith time step is with the ith row). Within each time step, the currents of all column wires are read by the MCU. The detailed weight readout process is illustrated in the flowchart of Supplementary Fig. 7b.

For batch vector-matrix multiplication, the row wires receive the ith input vector $\mathbf{V}_i$ at the ith time step. The MCU reads the ADCs of all columns in each time step and sends the reading back to the computer. The detailed batch vector-matrix multiplication process is illustrated in the flowchart of Supplementary Fig. 7c.

More advanced functions such as hidden neurons are implemented with the aid of the computer. The MATLAB environment of the computer receives the vector-matrix multiplication readings, applies activation functions, and scales results back to voltages. The MATLAB environment then sends the calculated DAC voltages to the MCU as the inputs of the subsequent vector-matrix multiplication.

Reinforcement learning algorithm. The deep-Q learning algorithm was a modified version of that reported by Mnih et al[3].

```
Algorithm 1: in-memory deep-Q learning with experience replay
Initialize replay memory
Initialize weights and biases
Set $t$ to 1
for epoch = 1: maximum epoch number
  Initialize $s_t$ to a random legal state
  while $r_t$ is not the epoch ending reward, or the first step of the epoch
  if $t$ is not 1
  Sample a minibatch $(s_j, a_j, r_j, s_{j+1})$ from replay memory
  Physical inference on all $s_j$ and $s_{j+1}$ in minibatch together with $s_t$ on 1T1R array
```

$$\text{Set } y_j = \begin{cases} r_j, \text{ if the sample of the minibatch is the last step of an epoch} \\ r_j + \gamma \cdot \max_{a_{j+1} \in \{a\}} (Q(s_{j+1}, a_{j+1})), \text{otherwise} \end{cases}$$

```
            Calculate gradients by minimizing $\sum_{j\in minibatch}(y_j - Q(s_j,a_j))^2$ using RMSprop
            Physical update of conductance of 1T1R array with the calculated weight gradients
        end
        Generate a random number in the range of (0,1)
        if the random number $<\varepsilon$, or $t=1$
            Select a random action $a_t$
        else
            Select the action with $\max_{a_t\in\{a\}} Q(s_t,a_t)$
        end
        Execute action $a_t$ on emulator, receive reward $r_t$ and next state $s_{t+1}$
        Store $(s_t,a_t,r_t,s_{t+1})$ to replay memory
        Set $\varepsilon = \varepsilon_{\min} + (\varepsilon_{\max} - \varepsilon_{\min})\exp(-\lambda t)$
        Increase $t$ by 1
    end
end
```

Three modifications are made to the original algorithm to better serve the learning with our in-house 1T1R memristor platform (see Supplementary Fig. 8 for comparison of the pseudo-codes).

(1) Difference in the times of calling batch vector-matrix multiplication. In the original algorithm, the forward pass of the neural networks is called at two different places. The first call is used for determining the action a_t while the other is used for minibatch update (see Supplementary Fig. 8). The customized algorithm performs a single hardware call with the combined input ($\{s_j\}$, $\{s_{j+1}\}$, and s_t). Doing so could reduce the instrumental delay of our measurement system.

(2) Removal of the target $\hat{Q}$-network. The target $\hat{Q}$-network used to set y_j in the work of Mnih et al. could help reduce the correlations present in the sequence of observations[3] (see Supplementary Fig. 8). However, the $\hat{Q}$-network requires extra 1T1R memristors. Given the fixed size of the memristor chip, only the Q-network is used in this work.

(3) Difference in the input pre-processing. Minh et al. employed a preprocessing module $\varphi(\{x_i\})$ to reduce the dimension of the fixed size frame sequence $\{x_i\}$[3]. This module is not necessary for the classical control problems studied here.

Fully connected neural network. The weights of the three-layer fully connected neural network are physically embodied in the conductance of the memristors. The biases were implemented in the software. The weights and biases were randomly initialized with uniform probability in the range [−0.5,0.5] (see Supplementary Fig. 4 for the performance comparison between the uniformly distributed initial weights and normally distributed initial weights). The optimization of the network followed the standard backpropagation of errors[50,51], based on the physically read currents and weights. The deltas of the output layer (linear activation) are given by the following partial derivative:

$$\frac{\partial Err}{\partial z_j^{(L=4)}} = \frac{\partial Err}{\partial a_j^{(L=4)}}\frac{\partial a_j^{(L=4)}}{\partial z_j^{(L=4)}} = \frac{\partial Err}{\partial a_j^{(L=4)}}$$

where $z_j^{(L=4)}$ and $a_j^{(L=4)}$ denote the signal before and after activation of the jth neuron of the layer $L=4$, respectively. The deltas for the two hidden layers (where $1<L<4$) with ReLU activations were calculated backwards.

$$\begin{aligned}\frac{\partial Err}{\partial z_j^{(1<L<4)}} &= \frac{\partial Err}{\partial a_j^{(1<L<4)}}\frac{\partial a_j^{(1<L<4)}}{\partial z_j^{(1<L<4)}}\\ &= \begin{cases} 0, \forall z_j^{(1<L<4)} \le 0 \\ \sum_i \frac{\partial Err}{\partial z_i^{(L+1)}} w_{i,j}^{(L+1)}, \forall z_j^{(1<L<4)} > 0\end{cases}\end{aligned}$$

where $w_{i,j}^{(L+1)}$ denotes the weight of the synapse interfacing between neuron i in layer $L+1$ and neuron j in layer L. The weight and bias gradients were calculated by the chain rule:

$$\frac{\partial Err}{\partial b_j^{(L)}} = \frac{\partial Err}{\partial z_j^{(L)}}\frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = \frac{\partial Err}{\partial z_j^{(L)}}$$

$$\frac{\partial Err}{\partial w_{j,k}^{(L)}} = \frac{\partial Err}{\partial z_j^{(L)}}\frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} = \frac{\partial Err}{\partial z_j^{(L)}} a_k^{(L-1)}$$

The calculated gradients of the weights and biases were subtracted by their mean of each layer, which could help to keep the weights following a distribution with zero mean to avoid potential divergence at the onset of the training at the expense of a slower convergence rate.

Table 1 | List of parameters used for environment simulation

Quantity	Symbol	Value
Cart-pole		
Gravitational acceleration	g	9.8
Mass of cart	M_{cart}	1
Mass of pole	M_{pole}	0.1
Total mass	M_{total}	1.1
Half of the pole length	L_{pole}	0.5
Magnitude of the force	$\lvert F\rvert$	10
Simulation time step	Δt	0.02
Allowed range of cart position		(−2.4,2.4)
Allowed range of pole angle		$\left(-\frac{\pi}{15},\frac{\pi}{15}\right)$
Mountain car		
Gravitational acceleration	g	9.8
Mass of car	M_{car}	0.2
Magnitude of the force	$\lvert F\rvert$	0.2 (left/right push) or 0 (no push)
Coefficient of friction	μ	0.5
Simulation time step	Δt	0.1
Allowed range of car position		(−1.2,0.6)
Goal position		0.5

Hardware interfacing. The Markov states of the cart–pole and mountain car were mapped to voltage vectors sent to the 1T1R array. For the cart-pole game, θ was multiplied by a factor of 10 to yield a similar mean and variance as x, $\frac{dx}{dt}$ and $\frac{d\theta}{dt}$. The voltage scaling factor was $0.2/\max\left(\text{abs}\left(x,\frac{dx}{dt},10\theta,\frac{d\theta}{dt}\right)\right)$ for the cart–pole game in each time step, so the peak voltage applied to the first layer was always approximately 0.2 V. The same voltage scaling applied to the mountain car game.

The synaptic weight was linearly mapped to the conductance difference between memristors in differential pairs (see Supplementary Table 3). The conductance was then linearly mapped to the transistor gate voltage based on the extracted slopes of the individual chips (see Supplementary Table 3 and Supplementary Fig. 3).

Simulation of the cart–pole environment. The cart–pole environment was simulated in software, based on the reported model[45]. The parameters of the physical system are listed in Table 1.

Assuming $\tilde{F} = F + M_{\text{pole}} L_{\text{pole}} \left(\frac{d\theta(t)}{dt}\right)^2 \sin\theta(t)\frac{1}{M_{\text{total}}}$, the angular acceleration of the pole is given by $\frac{d^2\theta}{dt^2} = \frac{(g\sin\theta(t) - \tilde{F}\cos\theta)}{L_{\text{pole}}\left(\frac{4}{3} - \frac{M_{\text{pole}}\cos^2\theta(t)}{M_{\text{total}}}\right)}$. The acceleration of the cart is given by $\frac{d^2x}{dt^2} = \tilde{F} - M_{\text{pole}} L_{\text{pole}} \frac{d^2\theta(t)}{dt^2}\cos\theta(t)\frac{1}{M_{\text{total}}}$. The state evolution follows the equations below:

$$x(t+\Delta t) = x(t) + \Delta t\frac{dx}{dt}$$

$$\theta(t+\Delta t) = \theta(t) + \Delta t\frac{d\theta}{dt}$$

$$\frac{dx(t+\Delta t)}{dt} = \frac{dx(t)}{dt} + \Delta t\frac{d^2x}{dt^2}$$

$$\frac{d\theta(t+\Delta t)}{dt} = \frac{d\theta(t)}{dt} + \Delta t\frac{d^2\theta}{dt^2}$$

Simulation of the mountain car environment. The mountain car environment was simulated in software, based on the reported model[46]. The parameters of the physical system are listed in Table 1.

The state evolution follows the equations below:

$$x(t+\Delta t) = x(t) + \Delta t\frac{dx}{dt}$$

$$\frac{dx(t+\Delta t)}{dt} = \frac{dx(t)}{dt} + \Delta t\left(-gM_{\mathrm{car}}\cos 3x + \frac{F}{M_{\mathrm{car}}} - \mu\frac{dx(t)}{dt}\right)$$

Once the car hits the left bound, which is assumed to be totally inelastic, its speed is set to zero.

Code availability

The computer code used in this study can be found at https://github.com/zhongruiwang/memristor_RL.

Data availability

The data that support the plots within this paper and other findings of this study are available from the corresponding authors upon request.

Received: 28 September 2018; Accepted: 20 February 2019;
Published online: 15 March 2019

References

1. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction.* Second Edition (MIT Press, Cambridge, 2018).
2. Lillicrap, T. P. et al. Continuous control with deep reinforcement learning. Preprint at https://arXiv.org/abs/1509.02971 (2015).
3. Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature.* **518**, 529–533 (2015).
4. Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
5. Silver, D. et al. Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017).
6. Chen, Y. et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* 609–622 (IEEE Computer Society, 2014).
7. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* 1–12 (ACM, 2011).
8. Chen, Y.-H., Krishna, T., Emer, J. S. & Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE. J. Solid-St. Circ.* **52**, 127–138 (2017).
9. van de Burgt, Y. et al. A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nat. Mater.* **16**, 414–418 (2017).
10. Suri, M. et al. Phase change memory as synapse for ultra-dense neuromorphic systems: application to complex visual pattern extraction. In *2011 International Electron Devices Meeting (IEDM)* 4.4.1–4.4.4 (IEEE, 2011).
11. Eryilmaz, S. B. et al. Brain-like associative learning using a nanoscale non-volatile phase change synaptic device array. *Front. Neurosci.* **8**, 205 (2014).
12. Burr, G. W. et al. Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Elect. Dev.* **62**, 3498–3507 (2015).
13. Ambrogio, S. et al. Unsupervised learning by spike timing dependent plasticity in phase change memory (PCM) synapses. *Front. Neurosci.* **10**, 56 (2016).
14. Strukov, D. B., Snider, G. S., Stewart, D. R. & Williams, R. S. The missing memristor found. *Nature* **453**, 80–83 (2008).
15. Jo, S. H. et al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* **10**, 1297–1301 (2010).
16. Yu, S., Wu, Y., Jeyasingh, R., Kuzum, D. & Wong, H. S. P. An electronic synapse device based on metal oxide resistive switching memory for neuromorphic computation. *IEEE Trans. Elect. Dev* **58**, 2729–2737 (2011).
17. Ohno, T. et al. Short-term plasticity and long-term potentiation mimicked in single inorganic synapses. *Nat. Mater.* **10**, 591–595 (2011).
18. Pershin, Y. V. & Di Ventra, M. Neuromorphic, digital, and quantum computation with memory circuit elements. *Proc. IEEE* **100**, 2071–2080 (2012).
19. Lim, H., Kim, I., Kim, J. S., Hwang, C. S. & Jeong, D. S. Short-term memory of TiO_2-based electrochemical capacitors: empirical analysis with adoption of a sliding threshold. *Nanotechnology* **24**, 384005 (2013).
20. Sheridan, P., Ma, W. & Lu, W. Pattern recognition with memristor networks. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)* 1078–1081 (IEEE, 2014).
21. La Barbera, S., Vuillaume, D. & Alibart, F. Filamentary switching: synaptic plasticity through device volatility. *ACS Nano* **9**, 941–949 (2015).
22. Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal–oxide memristors. *Nature* **521**, 61–64 (2015).
23. Hu, S. G. et al. Associative memory realized by a reconfigurable memristive Hopfield neural network. *Nat. Commun.* **6**, 7522 (2015).
24. Serb, A. et al. Unsupervised learning in probabilistic neural networks with multi-state metal–oxide memristive synapses. *Nat. Commun.* **7**, 12611 (2016).
25. Park, J. et al. TiO_x-based RRAM synapse with 64-levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing. *IEEE Elect. Dev. Lett.* **37**, 1559–1562 (2016).
26. Shulaker, M. M. et al. Three-dimensional integration of nanotechnologies for computing and data storage on a single chip. *Nature* **547**, 74–78 (2017).
27. Hu, M. et al. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC)* 1–6 (IEEE, 2016).
28. Hu, M. et al. Memristor-based analog computation and neural network classification with a dot product engine. *Adv. Mater.* **30**, 1705914 (2018).
29. Sheridan, P. M. et al. Sparse coding with memristor networks. *Nat. Nanotechnol.* **12**, 784–789 (2017).
30. Li, C. et al. Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* **1**, 52–59 (2018).
31. Le Gallo, M. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
32. Zidan, M. A. et al. A general memristor-based partial differential equation solver. *Nat. Electron.* **1**, 411–420 (2018).
33. Nili, H. et al. Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors. *Nat. Electron.* **1**, 197–202 (2018).
34. Yao, P. et al. Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017).
35. Li, C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
36. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).
37. Bayat, F. M. et al. Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nat. Commun.* **9**, 2331 (2018).
38. Boybat, I. et al. Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* **9**, 2514 (2018).
39. Chen, W.-H. et al. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *2018 IEEE International Solid-State Circuits Conference (ISSCC)* 494–496 (IEEE, 2018).
40. Jeong, Y., Lee, J., Moon, J., Shin, J. H. & Lu, W. D. *K*-means data clustering with memristor networks. *Nano Lett.* **18**, 4447–4453 (2018).
41. Li, C. et al. Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intel.* **1**, 49–57 (2019).
42. Nandakumar, S. et al. Mixed-precision architecture based on computational memory for training deep neural networks. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2018).
43. Wang, Z. et al. Fully memristive neural networks for pattern classification with unsupervised learning. *Nat. Electron.* **1**, 137–145 (2018).
44. Choi, S., Shin, J. H., Lee, J., Sheridan, P. & Lu, W. D. Experimental demonstration of feature extraction and dimensionality reduction using memristor networks. *Nano Lett.* **17**, 3113–3118 (2017).
45. Barto, A. G., Sutton, R. S. & Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.* **SMC-13**, 834–846, (1983).
46. Sutton, R. S. Generalization in reinforcement learning: successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8* 1038–1044 (NIPS, 1996).
47. Jiang, H. et al. Sub-10 nm Ta channel responsible for superior performance of a HfO_2 memristor. *Sci. Rep.* **6**, 28525 (2016).
48. Tieleman, T. & Hinton, G. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* **4**, 26–31 (2012).
49. Choi, S. et al. SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations. *Nat. Mater.* **17**, 335–340 (2018).
50. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).
51. LeCun, Y., Touresky, D., Hinton, G. & Sejnowski, T. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School* 21–28 (CMU, Pittsburgh, PA, Morgan Kaufmann, 1988).

Acknowledgements

This work was supported in part by the US Air Force Research Laboratory (AFRL) (Grant No. FA8750-15-2-0044), the Defense Advanced Research Projects Agency (DARPA) (Contract No. D17PC00304), the Intelligence Advanced Research Projects Activity (IARPA) (Contract 2014-14080800008) and the National Science Foundation (NSF) (ECCS-1253073). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL. Part of the device fabrication was conducted in the clean room of the Centre for Hierarchical Manufacturing (CHM), an NSF Nanoscale Science and Engineering Centre (NSEC) located at the University of Massachusetts Amherst.

Author contributions

J.J.Y. conceived the concept. J.J.Y., Q.X., Z.W. and C.L. designed the experiments. M.R. and P.Y. fabricated the devices. Z.W. and C.L. performed electrical measurements. W.S., D.B., Y.L., H.J., P.L., M.H., J.P.S., N.G., M.B., Q.W., A.G.B., Q.Q. and R.S.W. helped with experiments and data analysis. J.J.Y., Q.X., Z.W. and C.L. wrote the paper. All authors discussed the results and implications and commented on the manuscript at all stages.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41928-019-0221-6.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to Q.X. or J.J.Y.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.