Learning to Control Random Boolean Networks: A Deep Reinforcement Learning Approach

Georgios Papagiannis(✉) and Sotiris Moschoyiannis

Department of Computer Science, Faculty of Engineering and Physical Sciences,
University of Surrey, Guildford, Surrey GU2 7XH, UK
{g.papagiannis,s.moschoyiannis}@surrey.ac.uk

Abstract. In this paper we describe the application of a Deep Reinforcement Learning agent to the problem of control of Gene Regulatory Networks (GRNs). The proposed approach is applied to Random Boolean Networks (RBNs) which have extensively been used as a computational model for GRNs. The ability to control GRNs is central to therapeutic interventions for diseases such as cancer. That is, learning to make such interventions as to direct the GRN from some initial state towards a desired attractor, by allowing at most one intervention per time step. Our agent interacts directly with the environment; being an RBN, without any knowledge of the underlying dynamics, structure or connectivity of the network. We have implemented a Deep Q Network with Double Q Learning that is trained by sampling experiences from the environment using Prioritized Experience Replay. We show that the proposed novel approach develops a policy that successfully learns how to control RBNs significantly larger than previous learning implementations. We also discuss why learning to control an RBN with zero knowledge of its underlying dynamics is important and argue that the agent is encouraged to discover and perform optimal control interventions in regard to cost and number of interventions.

Keywords: Controllability · Complex networks · Q learning · Prioritized Experience Replay · Interventions

1 Introduction

Random Boolean Networks (RBNs) were introduced [11] as a computational model to simulate the dynamics of Gene Regulatory Networks (GRNs) and is the model that has received the most attention. In the context of GRNs the state of the cell at a specific time step is characterized by the expression of its genes (*gene expression*). RBNs are networks comprised of nodes corresponding to the genes of a cell where *gene expression* is quantized as binary values $\{1, 0\}$, namely active or inactive gene. At each time step, the expression of a specific gene is governed by a Boolean function assigned

This research was partly funded by EIT Digital IVZW, under the Real-Time Flow project, activity 18387-SGA2018, and partly by the EPSRC IAA project AGELink (EP/R511791/1).

H. Cherifi et al. (Eds.): COMPLEX NETWORKS 2019, SCI 881, pp. 721–734, 2020.
https://doi.org/10.1007/978-3-030-36687-2_60

at that gene, as well as other genes interacting with it. The number of states an RBN can have are 2^n where n corresponds to the number of nodes. Regardless of their simplicity, RBNs have provided valuable insights to the dynamics of GRNs [21, 23].

Perturbations can be caused to the state of the cell as the result of environmental causes such as the climate an organism may reside in or malfunctions of other cells of that organism. Such perturbations can cause the state of the cell to transition from some current state towards some other state where the cell may exhibit undesirable (*collective*) behavior. GRNs have been observed to exhibit sudden emergence of ordered collective behavior, which in the context of RBNs is manifested as *attractors* [10]. Such collective behavior has been found for example in cancerous cells in the form of Proliferation and Apoptosis [9, 10].

Thus, *controllability* can be viewed as the process of developing a strategy aiming to cause such perturbations by targeted interventions to specific genes of the cell, in order to drive it towards a desirable attractor, referred to as the *target attractor*. That is, a state that no longer exhibits undesirable behavior, such as uncontrollable cell proliferation. Such interventions are of significant importance as they can have a positive therapeutic effect in various diseases such as cancer.

In this paper we apply a Deep Q Network (DQN) [17] with Double Q Learning (Double DQN) which we refer to as the *agent* [7, 8] that is trained by sampling experiences from the environment; being an RBN, using Prioritized Experience Replay (PER) [20] in order to determine a control policy that drives the RBN from any state towards the *target* attractor. The proposed approach makes no assumptions and requires no knowledge about the underlying structure, connectivity, dynamics or Boolean functions of the RBN. The agent learns how to control the network merely by direct interaction and observation of the RBN's state transitions. In order to ensure that an intervention is as less intrusive to the network as possible we limit the number of interventions allowed per time step to a maximum of one and attempt to control the RBN in a *restricted specified* time frame.

To summarize our contributions are as follows:

- We propose a novel approach to the problem of RBN control and show that it can scale to significantly larger RBNs compared to previous *learning* algorithms.
- We discuss the significance of *learning* how to control an RBN without making any assumptions of the underlying dynamics or structure of the network.
- Finally, we show that the proposed approach is trained in such way as to discover *near* optimal interventions in terms of the cost and number of interventions; dependent to some problem related assumptions [22, 24, 25].

The remainder of this paper is structured as follows: In Sect. 2 we briefly review previous approaches to the problem of RBN control. In Sect. 3 we introduce RBNs. Section 4 formally introduces the problems of control in the context of RBNs. Then Sect. 5 adumbrates the main concept behind Q-Learning and Deep Reinforcement Learning and discusses why learning to control RBNs with no knowledge or assumptions of the network's dynamics, structure or connectivity is of significant importance. In Sect. 6, we present and discuss our Experiments and Results. Finally, Sect. 6.4 includes concluding remarks and suggestions for future work.

2 Related Work

In recent years a number of solutions have been suggested to the problem of control of both synchronous and asynchronous RBNs [1, 3, 4, 6, 12, 15, 16, 19]. Gao et al. [5] suggested a *k-walk* method for directed tree networks which was used to show how one node can control a set of target nodes. A greedy algorithm was also proposed that approximated the number of *driver* nodes needed in order to control such target nodes. Other approaches include those proposed by Chen et al. [3] where the authors introduce a control scheme that freezes part of the network according to the average sensitivity of Boolean functions for a specific time frame. Paul et al. [19] recently suggested a decomposition based approach to control asynchronous RBNs. The authors apply control only for a single time step aiming to make the minimum number of interventions required. The proposed approach exploits the underlying structure of the RBN in order to decompose it into Blocks which are used to determine the strong basin of attraction of the RBN; that is determine the states that will inevitably lead the RBN to a *target* attractor. However, to our knowledge, the only previous approach that *learns* to control RBNs was proposed by Karlsen et al. [12] who apply a variant of a Learning Classifier System (LCS) called XCS that uses a genetic algorithm to evolve control rules in order to drive an RBN from a state towards a specified *target* attractor. The authors also explore XCS control rules compared to optimal control rules [13].

In the context of Reinforcement Learning (RL), Deep Reinforcement Learning (DRL) has received a surge of interest recently. Mnih et al. [17] proposed a Deep Q Network (DQN) that was trained to play Atari games by sampling experiences from the environment with the use of Experience Replay (ER) [14], having as input only the raw pixels of the game. The DQN was successful in exceeding human performance in 3 of the ATARI games and outperforming any other previous approaches. In this paper we have implemented two advancements in DQNs and ER, that is Double DQN [7, 8] trained using Prioritized Experience Replay (PER) [20]. The state-space of ATARI is significantly large and DRL has shown very satisfying results, thus it is reasonable to consider DRL as a potential approach to the problem of *learning* how to control RBNs in order to exceed the number of nodes previous *learning* approaches have achieved.

3 Random Boolean Networks

3.1 Structure and Dynamics

In order to show the direct correlation of RBNs with GRNs we take the case of a GRN with n genes. Each gene regulates the activity of some other gene(s); it is in other words one of the factors determining whether some other gene in the cell is active or inactive. Thus, we assume that the state (*gene expression*) of gene g_i at some time step t is expressed as $g_i(t)$. Hence, $g_i(t) = 1$ indicates that a gene is active and $g_i(t) = 0$ inactive. We now construct a vector $G^h(t) = [g_1(t), g_2(t), \ldots, g_n(t)]$ where $G^h(t)$ corresponds to state $h \in N$ of the RBN at time t, also known as Gene Activity Profile (GAP) [9] in the context of GRNs, where N is the set of all 2^n possible RBN states. Each node g_i of the RBN is assigned a Boolean function f^i and input nodes inp^i. Thus, the Boolean functions of RBN nodes are expressed as a vector of functions $F = [f^1, f^2, \ldots, f^n]$. For example, if node g_k has inputs $inp^k = [g_x, g_m]$ and Boolean function f^k then the value of $g_k(t+1)$ would be $g_k(t+1) = f^k([g_x(t), g_m(t)]) = f^k(inp^k)$ To be more exact, assume at time step t node $g_k(t) = 1$, showing that gene k is active. Also, assume that $g_x(t) = 1$, $g_m(t) = 0$ and $f^k = OR$. At the next time step the value of $g_k(t+1)$ will be $g_k(t+1) = f^k([1, 0]) = 1\, OR\, 0 = 1$. We refer to this process as a *natural* state transition of the RBN. In this paper we consider *synchronous* state transition updates where at each time step the values of all nodes are updated.

3.2 Attractors

Attractors of an RBN are sets of states, subsets of N, that in the context of Markov Chains and stochastic processes are *irreducible* and *periodic* or are *absorbed* states. *Irreducible* and *periodic* means that every state in the *attractor* is reachable from any other state with period equal to the number of states of the attractor set. That is if an *attractor* is comprised of m states and at time step t the state of the RBN is s the next time the RBN will transition to s will be at $t+m$. An *absorbed* state is a state that once visited the RBN cannot transition to any other state. In both cases the RBN is considered to be '*trapped*' in that set of states or single state and thus cannot escape without external interventions.

3.3 Example

We present here (*Fig.* 1) an example of an RBN to demonstrate what was introduced at Subsects. 3.1 and 3.2. The RBN has $n = 5$ and $F = [XOR, XOR, AND, OR, XOR]$. The connectivity of the RBN is shown at Fig. 1. As shown, regardless of the state an RBN is in, it will eventually transition into one of the three attractors $\mathcal{A}_1 = \{10010, 11011, 01110\}$, $\mathcal{A}_2 = \{11010\}$, $\mathcal{A}_3 = \{00000\}$. This can easily be verified by looking at the *State Transition Graph* (STG) in Fig. 1. (*right* graph) that shows every state transition that occurs starting from every possible state in the set N of RBN states. Attractor $\mathcal{A}_1$ corresponds to an *irreducible* set of states, while $\mathcal{A}_2$ and $\mathcal{A}_3$ to *absorbed* states.

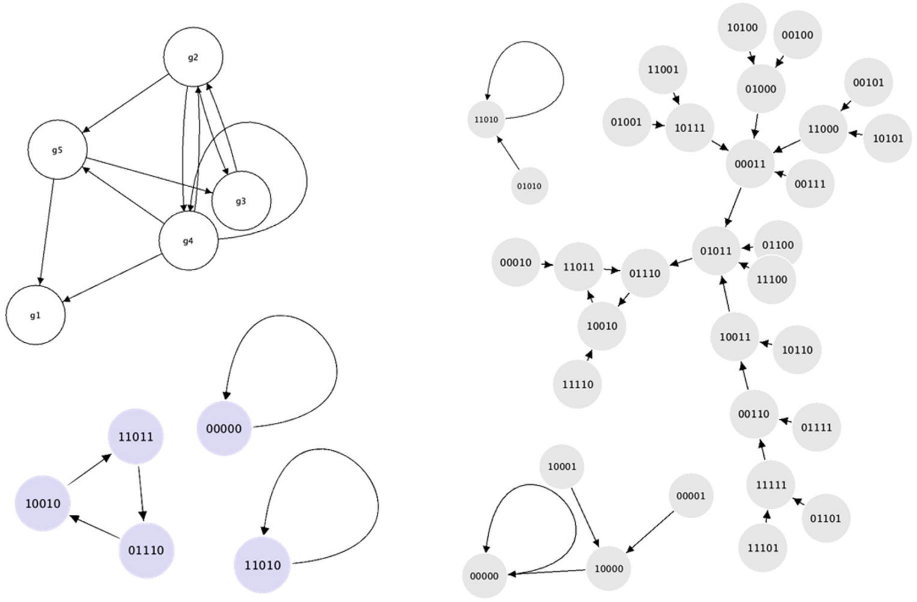

Fig. 1. The top left network (*white*) shows an RBN n = 5. The right graph (*light gray*) corresponds to the *State Transition Graph* (STG) of the RBN, showing the natural state transitions that occur starting from every possible of the *N* RBN states. The bottom left graphs (*light blue*) show the attractors of the RBN.

4 Control of RBNs: Problem Formulation

As explained and shown in Sect. 3 an RBN regardless of its initial state it will eventually transition into an *attractor*. Transitioning to a specific *attractor* is dependent on the initial state of the RBN. If we take the example of the RBN in Sect. 3.3 and observe Fig. 1 we notice that if the RBN is initialized at state 10100 or 00110 for example, the RBN will always eventually transition to attractor $\mathcal{A}_1$. On the other hand, the only way to *naturally* transition to $\mathcal{A}_2$ is to either have an RBN with initial state 11011 or 01010. Similarly, for $\mathcal{A}_3$ only four possible states (Fig. 1) can *naturally* transition to that *attractor*. However, as discussed in Sect. 1, some *attractors* may be undesired, as they may exhibit some *undesirable* functionality.

Hence, *controllability* takes the form of making *targeted interventions* at each time step of *natural* state transitions in order to drive the RBN from some state or *undesirable* attractor towards a *desirable* one, in a *restricted*, *specified* time frame.

4.1 Intervention

We refer to *intervention* to the state of an RBN as the process of perturbing the value of a node of the network; hence, changing its value from 1 to 0 or vice versa. In order to make interventions to the network as less invasive as possible we restrict the number of possible *interventions* per time step to a maximum of 1; that is, either *no* intervention occurs and the RBN *naturally* transitions to the next state or the value of one node is switched. After each intervention a *natural* state transition occurs for one time step. Thus, we define *intervention* at time step t as follows:

$$In\left(G^{h}(t), v\right) \tag{1}$$

where $G^h(t)$ is the state of the RBN at time step t and v indicates the position of the n nodes whose value will be perturbed; where v is a non-negative integer $v \in [0, n]$. A value of $v = 0$ indicates that *no* intervention occurs and the RBN *naturally* transitions to the next state, whereas $v = i$ and $0 < i \leq n$ indicates perturbation to node g_i.

For example, assume that the RBN introduced at Sect. 3.3 is in state 00110 at time step t and we choose to intervene at the third node i.e. $In(G^{00110}(t), 3)$. The state of the RBN at the next time step will be as follows: The third node is perturbed, thus state 0 0110 becomes 00010 and $G^{00010}(t+1) = 11011$. So, one intervention occurred at the third node of the RBN which was then followed by a *natural* state transition. It is important to note that if no *natural* transition was allowed to occur after each *intervention*, then the problem of control would be trivial as we could just calculate the Hamming distance of the current state to one of the *target* attractor states and switch the value of the different nodes one per time step. As previously noted, we also *restrict* the number of time steps interventions can occur, which makes the problem of control challenging. By making random interventions to the state of the RBN indefinitely, the network would eventually be controlled. However, by limiting the number of time steps interventions are allowed, a strategy needs to be discovered in order to drive the RBN from its current state towards the *target* attractor. As we will show in Sect. 6 the number of time steps of allowed interventions are set to be significantly smaller than average random interventions needed to control each state. In our case the use of *time steps* is arbitrary, however in a real-world case it could correspond for example to the duration of application of a specific therapy applied to the gene and would be determined by a field practitioner such as a doctor or a biologist.

4.2 Markov Decision Process

Markov Decision Process (MDP) is a framework for modelling stochastic processes and is also used to describe an environment in the context of Reinforcement Learning. We briefly discuss in this section how the problem of RBN control can be expressed in terms of an MDP. RBNs while deterministic, can me modelled as a special case of MDPs where the probability of transitioning from one state to the next is 1. Modelling the RBN as an MDP, the next state of the RBN depends only on the current state; that is, it satisfies the Markov property. Then the tuple $(S, A, P, \mathcal{R}, \gamma)$ describes the full problem of RBN control where S is the set of RBN states, A is the set of actions i.e. possible RBN interventions, $P: S \times A \times S \rightarrow \mathbb{Z}$ is the transition matrix which is deterministic. $\mathcal{R}: S \rightarrow \mathbb{Z}$[1] is the reward function where:

[1] The set of integers is used as it applies to this work since the processes are deterministic with the state transition probabilities 1 and rewards are integer values.

$$\mathcal{R} = \begin{cases} 10, & \textit{if } s \in S \textit{ is the target attractor} \\ -1, & \textit{if } s \in S \textit{ is a non} - \textit{attractor state} \\ -2, & \textit{if } s \in S \textit{ is a non} - \textit{target attactor state} \end{cases}$$

and $\gamma \in [0,\ 1]$ is the discount factor. The problem can be expressed as determining a control strategy to maximize the future cumulative reward as follows:

$$\mathcal{R}^{Total} = \gamma^0\,\mathcal{R}_1 + \gamma^2\,\mathcal{R}_2 + \gamma^3\,\mathcal{R}_3 + \ldots + \gamma^{T-1}\,\mathcal{R}_T = \sum_{t=1}^{T} \gamma^{t-1}\,\mathcal{R}_t$$

where T corresponds to some terminal state which in the context of RBNs corresponds to a target attractor or reaching the specified intervention limit as discussed in the next section.

5 Double Deep Q Network with Prioritized Experience Replay

The goal of a Reinforcement Learning (RL) agent is to determine the actions to perform to the states of an *environment* in order to maximize some notion of cumulative reward in an *episode.* In the context of RBNs, we define an *episode* as one attempt to control an RBN; that is, we either make the appropriate interventions, successfully control the RBN and the attempt terminates, or we reach the number of allowed step interventions and the attempt terminates. Thus, the RBN can be seen as an *environment*, the possible interventions can be seen as the potential *actions*, and maximization of cumulative reward is equivalent to the RL agent discovering a strategy which in the context of RL we refer to as an *policy*, that given a state of the RBN, the agent performs at each time step the intervention that will maximize the future cumulative reward.

5.1 Q Learning

Q Learning [25] is a model-free, off-policy RL algorithm under the umbrella of Temporal-Difference (TD) Learning [22] that is used to determine $Q^*(s,\ a)$ where $s \in N$ is the current state of the *environment* (RBN), $a \in A$ is an action where A is the set of all possible actions (interventions, including no intervention) and the result of $Q^*(s,\ a)$ is the *true* value of the expected reward of taking action a at state s and following some *optimal* policy (control strategy). Thus, it becomes clear that if Q^* is known then an *optimal policy* can be constructed where at each state of the RBN we *greedily* select to perform the intervention that would yield the highest expected reward. Q Learning is defined as follows:

$$Q(s_t, A_t) = Q(s_t, A_t) + \mathbb{a}[\mathcal{R}_{t+1} + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, A_t)] \qquad (2)$$

where $s_t \in N$ is the state of the environment at time step t, A_t is the set of possible actions that can be performed at time step t, $\mathcal{R}_{t+1}$ is the reward received by performing an action $a \in A$ and transitioning to state s_{t+1}. Also, γ is a discount factor that is used to determine the weight of immediate rewards compared to rewards received later in the episode and $\mathbb{a}$ is a constant $0 < \mathbb{a} \leq 1$, that determines how fast the agent forgets past interactions with the environment.

5.2 Double Deep Q Network (Double DQN)

The idea of a Double Deep Q Network was proposed by Hasselt et al. [8]. It is an improvement of the original DQN implementation proposed by Mnih et al. [17]. As the state-space increases it becomes impossible to store every state-action Q value pair [2] and hence classical Q-Learning becomes impractical. In order to address this issue, a DQN and hence a Double DQN uses a non-linear function approximator to parametrize the Q function as $Q(s,\ a;\ \theta_t)$ where θ_t are the parameters of the approximator at time step t. It is comprised of a *policy* network used to estimate the expected reward of an action and a *target* network that is used to update its parameters [8, 17]. The *policy* network parameters are copied to *target* network parameters every k episodes. However, a Double DQN [8] uses a *double* estimator [7] to update θ instead of a single estimator. The goal of a Double DQN is to minimize a loss function as follows:

$$L_i(\theta_i) = \left(\mathcal{R}_{t+1} + \gamma Q\left(s_{t+1}, \arg max_{a'} Q(s_{t+1},\ a';\ \theta_i);\ \theta_i^-\right) - Q(s_t,\ a_t;\ \theta_i)\right)^2 \tag{3}$$

where θ_i^- are the parameters of the *target* network at some time step i and $\theta_i^- = \theta_i$ every k episodes. Differentiating Eq. (3) we obtain:

$$\begin{aligned}\nabla_{\theta_t} L_i(\theta_i) = (\mathcal{R}_{t+1} + \gamma Q\left(s_{t+1},\ \arg max_{a'} Q(s_{t+1},\ a';\ \theta_i);\ \theta_i^-\right) \\ - Q(s_t,\ a_t;\ \theta_i))\nabla_{\theta_t} Q(s_t,\ a_t;\ \theta_t)\end{aligned} \tag{4}$$

which can be optimized using Stochastic Gradient Descent. Thus, in the context of RBNs at each time step we select an action a_t to perform to state s_t of the RBN following some policy π, observe some reward $\mathcal{R}_{t+1}$, state transition s_{t+1} and according to these, update our *target* network in order to minimize Eq. (4). For a more detailed technical explanation the reader is referred to [7, 8, 17].

5.3 Prioritized Experience Replay

Prioritized Experience Replay (PER) was introduced by Schaul et al. [20]. PER breaks the strong correlated parameter updates caused by training on consecutive data samples and addresses the issue of *rapid* forgetting. During interaction with the environment, instead of updating the Double DQN's values immediately after each interaction and transition observation, we store that observation as an experience $e_t = (s_t, a_t, R_{t+1}, s_{t+1})$ which is then sampled from a Replay Buffer with some probability: $\mathcal{P}(i) = \frac{p_i^\omega}{\sum_k p_k^\omega}$, where $p_i = |\delta| + c$ is the *priority* of sample i, $\delta = \mathcal{R}_{t+1} + \gamma Q\left(s_{t+1}, \arg max_{a'} Q(s_{t+1}, a'; \theta_i); \theta_i^-\right) - Q(s_t, a_t; \theta_i)$ and c is a small constant to prevent experiences with $\delta = 0$ from never being sampled. That way PER replays some experiences more than others according to how big the actual versus the predicted Q value was, with the assumption that the function approximator has '*more*' to learn from experiences whose best action expected reward it predicted wrongly with greater margin. In order to address some issues arising with prioritising experience sampling, such as induced bias towards some experiences, Schaul et al. introduced Importance Sampling weights which are folded in Eq. (4) when optimising the Double DQN's parameters [19]. For a more detailed explanation the reader is referred to [6, 7, 13, 16, 19, 22, 25].

5.4 Importance of *Learning* to Control with Model Free RL

As discussed in Subsect. 5.1, Q Learning, the foundation behind Double DQN, is a *model-free* RL algorithm. In the context of RBNs, it takes the form of constructing a *policy* that successfully drives the RBN to a *target* attractor where no knowledge of the underlying dynamics, structure, connectivity, Boolean functions or the state transition graph of the RBN is required. Other non-*learning* approaches such as those mentioned in Sect. 2 do make use of the structure and connectivity of the RBN. However, it is important to consider *control* cases where nothing is known about the underlying dynamics of the system.

The proposed approach in this paper can be used to interact directly with an RBN and construct a successful control policy simply by making interventions and observing state transitions. It is important to note that while theoretically the suggested approach can control RBNs of any size, practically training can be very time consuming, as training a Double DQN to find a control strategy for RBNs can require a large number of experiences to be sampled. Also, while *classical* Q Learning [25] has been shown to converge to an *optimal* policy with probability 1, the use of non-linear function approximator (Double DQN) does not have that guarantee [22, 24, 25] and as shown in the following section tends to determine *near* optimal control interventions with no guarantee that it does so for every state. However, applying *classical* Q Learning would be impractical as it would not scale further than a few nodes [16].

6 Experiments and Results

6.1 Random Boolean Network

In our experiments we applied a Double DQN that was trained using Prioritized Experience Replay to an RBN with 25 nodes. As follows we describe the connectivity and Boolean functions of the RBN in order to make our results reproducible; however, training the Double DQN to determine a control policy is agnostic to those details. The connectivity of the RBN is described as follows: $inp^1 = [g_{10}, g_2]$, $inp^2 = [g_{12}, g_{25}]$, $inp^3 = [g_1, g_8]$, $inp^4 = [g_9, g_8]$, $inp^5 = [g_{15}, g_6]$, $inp^6 = [g_8, g_{13}]$, $inp^7 = [g_{24}, g_{16}]$, $inp^8 = [g_1, g_{20}]$, $inp^9 = [g_2, g_3]$, $inp^{10} = [g_9, g_{16}]$, $inp^{11} = [g_7, g_{15}]$, $inp^{12} = [g_7, g_{25}]$, $inp^{13} = [g_6, g_{15}]$, $inp^{14} = [g_{17}, g_8]$, $inp^{15} = [g_{17}, g_{20}]$, $inp^{16} = [g_3, g_{15}]$, $inp^{17} = [g_4, g_{21}]$, $inp^{18} = [g_5, g_{15}]$, $inp^{19} = [g_5, g_{18}]$, $inp^{20} = [g_{20}, g_9]$, $inp^{21} = [g_{23}, g_1]$, $inp^{22} = [g_{10}, g_{15}]$, $inp^{23} = [g_{12}, g_{23}]$, $inp^{24} = [g_{24}, g_{14}]$, $inp^{25} = [g_{14}, g_4]$. The function vector of the RBN is $F = [XOR, XOR, XOR, XOR, AND, AND, AND, AND, OR, AND, AND, AND, OR, AND, OR, OR, XOR, OR, AND, OR, XOR, AND, AND, AND, AND]$ and the RBN attractors are

$\mathcal{A}_1 = \{'0000000000001011010100000\}$, $\mathcal{A}_2 = \{0000000000001011010100000\}$, $\mathcal{A}_3 = \{0000000000000000000000000\}$ $\mathcal{A}_4 = \{1001010100001111110111000$, $10000101010010111110111000$, $1000010111001011010111000$, 101100011100101101010000, $0001010100001111010110001$, $1100000010001011110100000$, $0111010000001011010100001\}$, $\mathcal{A}_5 = \{0000010111001011110110000$, $1010010001001111110101000$, $1010000110001011010111000\}$. In order to select our target attractor, we randomly pick a sample of 10000 states and let them naturally evolve to an attractor. We note that the attractor that would *naturally* occur the least, is attractor $\mathcal{A}_3$ with probability 0.0035. Hence, we set $\mathcal{A}_3$ as the target attractor. We then attempt to control a sample of 10000 randomly selected states to $\mathcal{A}_3$ by random interventions. We note that on average 9612 random interventions are needed for successful control. Thus, in order to demonstrate the robustness and strength of the proposed approach we select the allowed interventions to be only 0.15% of the interventions that would randomly achieve control; that is, we allow interventions for only 14 time steps.

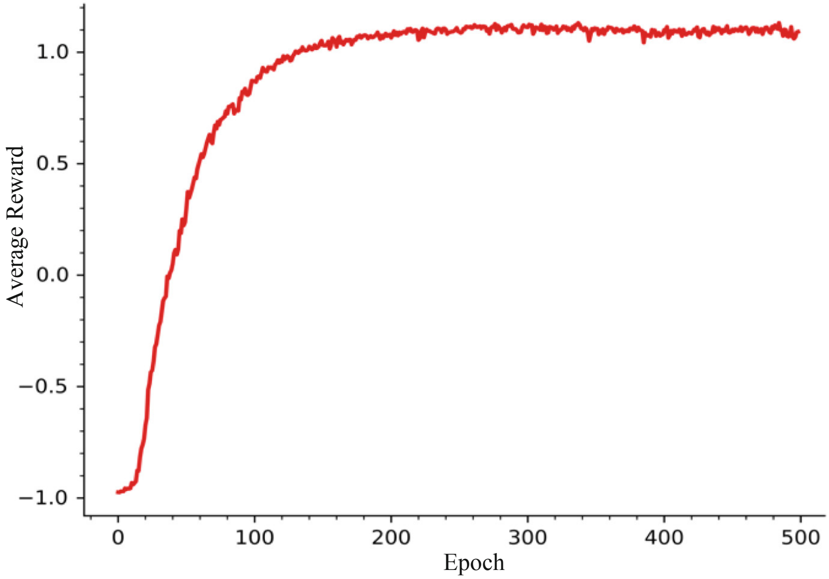

Fig. 2. The average episodic reward per epoch (epoch = 5000 episodes) increases after training starts, implying that the network discovers ways to control the RBN in less than 14 time steps.

6.2 Double DQN and Prioritized Experience Replay

Our Double DQN is comprised of an input layer of size n corresponding to the observed state of the RBN, two hidden layers each of 100 rectifier units and a linear output layer of size $n+1$, where n corresponds to the Q values of possible interventions (*actions*) and the extra unit corresponds to the Q value of no intervention. We set our target network update parameter $k = 5000$ and discount factor $\gamma = 0.98$ to strongly weight future rewards. For PER we set $\omega = 0.6$ and linearly anneal β from 0.4 to 1 in 70% of training. The size of the Replay Buffer is set to 100000 and during training 512 experiences are sampled per episode used to minimize Eq. (3) for which we use Huber Loss to calculate the loss in order to avoid exploding gradients by error clipping [16]. The parameters of the network are optimized using RMSProp optimizer, we train the network for 2500000 episodes and the policy we use during training is ε-greedy where ε is linearly annealed from 1 to 0.05. Finally, we assign reward $\mathcal{R}_{t+1}$ of −2 if an intervention directs the network in the next time step to a non-*target* attractor, −1 to a non-attractor state and a reward of 10 for interventions that direct the network to the *target* attractor.

6.3 Results

After training has finished, we attempt to control the RBN starting from every possible state towards $\mathcal{A}_3$ using a greedy policy; that is, from the $n+1$ Q values outputted by the trained network we select the one with the highest expected reward. We note that the agent achieves 100% controllability. After plotting the average reward per episode per epoch received during training (Fig. 2; we assume 1 epoch = 5000 episodes) we note that the reward begins to increase shortly after training has begun reaching a peak value after about 1250000 episodes. However, this result would not be possible if the agent had not discovered a policy that achieves successful control in less time steps than the allowed number of 14. This can easily be deduced by observing Eq. (2) and noting that if control was achieved strictly in 14 time steps the average reward should be negative and in the best-case scenario (assuming that no non-target attractors are visited) it would be $(-1 \cdot 13 + 10) \div 14 = -0.75$. However, looking at Fig. 2 we can see that the average reward peaks at approximately ≈1.2. Hence, we can deduce that on average the maximum number of time steps required by the developed policy is just above 4, which in fact shows that during training our network tends to find optimal control strategies for each state, however as mentioned in Sect. 5.4 with no guarantee that it does so for *every* state due the fact that we are using a non-linear function approximator during training (Double DQN). This can also be deduced by observing Fig. 3 that shows the average interventions per epoch during training. Interventions drop to an average of slightly above 4. For a more concrete elaboration on *optimality* the reader is referred to [22, 24].

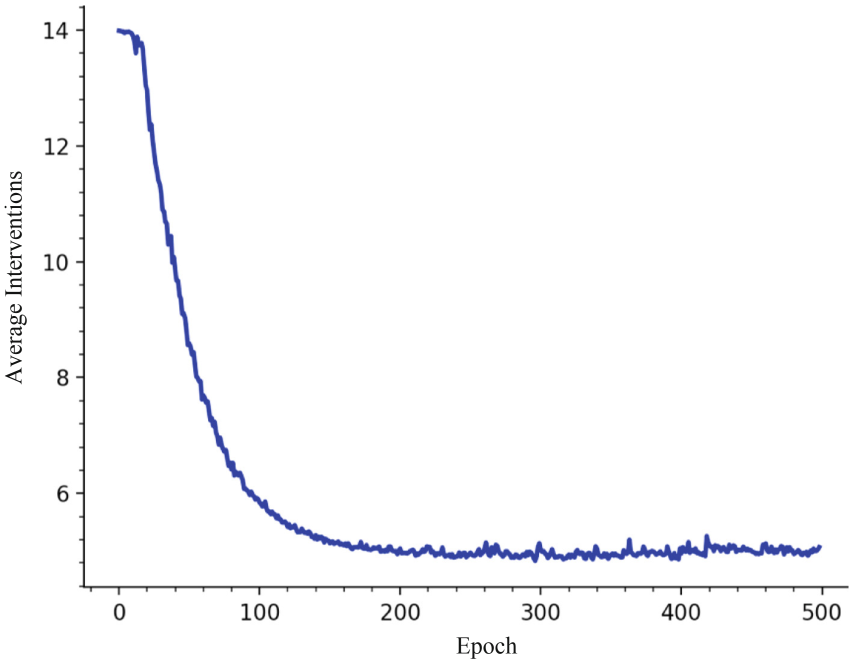

Fig. 3. The average episodic interventions per epoch (epoch = 5000 episodes) sharply decreases after a few thousand episodes and reaches an average value slightly above 4 further indicating that the agent finds a control policy with significantly less interventions than the specified.

Comparing this approach with XCS [12], XCS was able to control RBNs of size maximum of 5 nodes due to scalability issues faced by Classifier Systems as the state space of the problem of application increases [22]. While a direct comparison is not feasible, it interesting to note the efficiency of Double DQN by comparing the result of an average of approximately 4 interventions for the RBN with 25 nodes to the one presented in [12]. Table 3 [12] shows average interventions on 5 node RBNs ranging from 1.2 to 1.7. Even though we increased the state space by a factor 1048576 (2^{20}) the proposed approached determined a control strategy that requires only about 2.5 more interventions than XCS needed for RBNs with 5 nodes.

6.4 Conclusion and Future Work

We introduced in this paper a novel *learning* approach to the problem of RBN control that uses Double DQN and PER to directly interact with an RBN and achieve control with no knowledge of the underlying connectivity, structure or Boolean functions of the RBN. This can be proved to be very useful when a control strategy needs to be developed for the states of an RBN, but no prior knowledge exists about the structure or dynamics of the network. In addition, we showed that this approach achieves control in significantly larger RBNs compared to previous learning approaches and does so in significantly fewer time steps than random interventions would achieve control.

Finally, we argued that the reason behind achieving control in such few steps is because the developed policy tends to find *near* optimal control strategies.

For future work we aim to investigate further the application of Deep Reinforcement Learning algorithms on the problem of control of RBNs, where we will focus on whether we can use this technique to determine a network's driver nodes [5, 18] and attempt to control RBNs which are updated asynchronously. In addition, we aim to improve our current control implementation to scale to significantly larger RBNs as well as study other approaches to address the current limitations of the approach as discussed in Sect. 5.4. Finally, we will be generalising our proposed solution by applying our work to the problem of controlling Probabilistic Boolean Networks.

References

1. Akutsu, T., Hayashida, M., Ching, W., Ng, M.: Control of Boolean networks: hardness results and algorithms for tree structured networks. J. Theoret. Biol. **244**(4), 670–679 (2007). https://doi.org/10.1016/j.jtbi.2006.09.023
2. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (2010)
3. Chen, S., Hong, Y.: Control of random Boolean networks via average sensitivity of Boolean functions. Chin. Phys. B **20**(3), 036401 (2011). https://doi.org/10.1088/1674-1056/20/3/036401
4. Fornasini, E., Valcher, M.: Optimal control of Boolean control networks. IEEE Trans. Autom. Control **59**(5), 1258–1270 (2014). https://doi.org/10.1109/TAC.2013.2294821
5. Gao, J., Liu, Y., D'Souza, R., Barabási, A.: Target control of complex networks. Nat. Commun. **5**(5415) (2014). https://doi.org/10.1038/ncomms6415
6. Gates, A., Rocha, L.: Control of complex networks requires both structure and dynamics. Sci. Rep. **6**, 24,456 (2016). https://doi.org/10.1038/srep24456
7. Hasselt, H.: Double Q-learning. In: Advances in Neural Information Processing Systems 23, pp. 2613–2621. Curran Associates, Inc. (2010)
8. Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094–2100. AAAI Press (2016)
9. Huang, S.: Gene expression profiling, genetic networks, and cellular states: an integrating concept for tumorigenesis and drug discovery. J. Mol. Med. **77**(6), 469–480 (1999). https://doi.org/10.1007/s001099900
10. Huang, S., Ingber, D.: Shape-dependent control of cell growth, differentiation, and apoptosis: switching between attractors in cell regulatory networks. Exp. Cell Res. **261**(1), 91–103 (2000). https://doi.org/10.1006/excr.2000.5044
11. Kauffman, S.: Metabolic stability and epigenesis in randomly constructed genetic nets. J. Theoret. Biol. **22**(3), 437–467 (1969). https://doi.org/10.1016/0022-5193(69)90015-0
12. Karlsen, M.R., Moschoyiannis, S.: Evolution of control with learning classifier systems. Appl. Netw. Sci. **3**(1), 30 (2018). https://doi.org/10.1007/s41109-018-0088-x
13. Karlsen, M.R., Moschoyiannis, S.K.: Optimal control rules for random boolean networks. In: International Workshop on Complex Networks and their Applications, pp. 828–840. Springer (2018)
14. Lin, L.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. **8**, 293–321 (1992). https://doi.org/10.1007/bf00992699

15. Liu, Y., Slotine, J., Barabási, A.: Controllability of complex networks. Nature **473**, 167–173 (2011). https://doi.org/10.1038/nature10011
16. Luque, B., Solé, R.: Controlling chaos in random Boolean networks. Europhys/ Lett. (EPL) **37**(9), 597–602 (1997). https://doi.org/10.1209/epl/i1997-00196-9
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015). https://doi.org/10.1038/nature14236
18. Moschoyiannis, S., Elia, N., Penn, A., Lloyd, D.J.B., Knight, C.: A web-based tool for identifying strategic intervention points in complex systems. In: Proceedings Games for the Synthesis of Complex Systems (CASSTING'16 @ ETAPS 2016), EPTCS, vol. 220, pp. 39–52 (2016)
19. Paul, S., Su, C., Pang, J., Mizera, A.: A decomposition-based approach towards the control of Boolean networks. In: Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 11–20. ACM (2018)
20. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay (2015). https://arxiv.org/abs/1511.05952. Accessed 14 Sep 2019
21. Somogyi, R., Sniegoski, C.: Modeling the complexity of genetic networks: understanding multigenic and pleiotropic regulation. Complexity **1**(6), 45–63 (1996). https://doi.org/10.1002/cplx.6130010612
22. Sutton, R., Barto, A.: Reinforcement Learning. The MIT Press, Cambridge (2018)
23. Szallasi, Z., Liang, S.: Modeling the normal and neoplastic cell cycle with realistic Boolean genetic networks: their application for understanding carcinogenesis and assessing therapeutic strategies. In: Pacific Symposium on Biocomputing, vol. 3, pp. 66–76 (1998)
24. Watkins, C., Dayan, P.: Mach. Learn. **8**, 279–292 (1992). https://doi.org/10.1023/a:1022676722315
25. Watkins, C., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992). https://doi.org/10.1007/bf00992698