# Deep Reinforcement Learning for Stabilization of Large-scale Probabilistic Boolean Networks

Sotiris Moschoyiannis, Evangelos Chatzaroulas, Vytenis Sliogeris, Yuhu Wu

***Abstract*—The ability to direct a Probabilistic Boolean Network (PBN) to a desired state is important to applications such as targeted therapeutics in cancer biology. Reinforcement Learning (RL) has been proposed as a framework that solves a discrete-time optimal control problem cast as a Markov Decision Process. We focus on an integrative framework powered by a model-free deep RL method that can address different flavours of the control problem (e.g., with *or* without control inputs; attractor state *or* a subset of the state space as the target domain). The method is agnostic to the distribution of probabilities for the next state, hence it does not use the probability transition matrix. The time complexity is only *linear* on the time steps, or interactions between the agent (deep RL) and the environment (PBN), during training. Indeed, we explore the *scalability* of the deep RL approach to (set) stabilization of large-scale PBNs and demonstrate successful control on large networks, including a metastatic melanoma PBN with *200 nodes*.**

***Index Terms*—Network dynamics, Gene Regulatory Networks, perturbation, Markov Decision Process, control policies**

## I. Introduction

Recent efforts to produce tools to effectively control the dynamics of complex networked systems draw from control theory, numerical methods and, more recently, network science and machine learning. A dynamical system is controllable if by intervening on the state of individual nodes the system as a whole can be driven from any initial state to a desirable state, within finite time [1]. This notion of control finds application in engineered systems but also biological networks, and is often referred to as *stabilization*.

Probabilistic Boolean Networks (PBNs) were introduced in [2] for modelling Gene Regulatory Networks (GRNs) as complex dynamical systems. Nodes represent genes in one of two possible states of activity; 0 (not expressed), 1 (expressed). Edges indicate that genes act on each other, by means of rules represented by Boolean functions. The state of the network at each time step comprises the states of the individual nodes. PBNs extend Kauffman's Boolean Networks (BNs) [3] by associating more than one function with each node, one of which executes at each time step with a certain probability. This stochasticity in the network model accounts for uncertainty in gene interaction, which is inherent not only in data collection but also in cell function. Both PBNs and BNs have been extensively used to model well-known regulatory networks, see seminal work in [4]. The dynamics of PBNs adhere to Markov chain theory [5] and dictate that the network, from any initial state, will eventually settle down to one of a limited set of steady states, the so-called *attractors* (fixed points or cyclic), that it cannot leave without intervention [6].

Genes in biological systems experience sudden emergence of ordered collective behaviour, which corresponds to steady state behaviour in PBNs and can be modelled via attractor theory [7], e.g., the emergence of heterogeneous small cell lung cancer phenotypes [8]. Therapeutic strategies may involve switching between attractors, e.g., proliferation, apoptosis in cancerous cells [9], or directing the network to a (set of) steady states that are more desirable than others, e.g., exhibit lower levels of resistance to antibiotics [10].

While steady states are understood in a small set of experimental GRNs in the literature, they are not generally available and it is computationally intractable to compute them in larger networks. However, attractors with larger basins of attraction tend to be more stable. Therefore, the long-term dynamical behavior can be described by a steady-state distribution (SSD) [11], [12] which effectively captures the time spent at each network state. Hence, the control problem in large-scale PBNs takes the form of set stabilization of the network to a pre-assigned subset of network states. It transpires that the ability to direct the network to a specific attractor, or a subset of network states, by means of intervention on individual nodes (genes), is central to GRNs and targeted therapeutics.

In this context, intervention at a certain time takes the form of effecting a perturbation on the state of a node, which in GRNs translates to knocking-out a gene (switch to 0) or activating it (to 1). A lot of changes in cancer are of a regulatory epigenetic nature [13] and thus cells are expected to be re-configured to achieve the same outcome, e.g., inputs to gene regulatory elements by one signalling pathway can be substituted by another signalling pathway [14]. Hence, cancer biology suggests that perturbations may be transient, e.g., see single-step perturbations in [15]. More concretely, this means that the network dynamics may change the state of a perturbed node in subsequent time steps.

Existing work in control systems engineering typically restricts perturbations to a subset of nodes, typically the *control nodes* of a network [1], [16] or nodes that translate biologically, e.g., pirin and WNT5A driven induction of an invasive phenotype in melanoma cells [11], [17], [18]. The general case where perturbations are considered on the full set of nodes is less studied, with the exception of [19], even though it is relevant in contexts where control nodes are not available, or computationally intractable to obtain. Further,

Sotiris Moschoyiannis, Evangelos Chatzaroulas, and Vytenis Sliogeris are with the School of Computer Science & Electronic Engineering, University of Surrey, GU2 7XH, UK (e-mail: s.moschoyiannis@surrey.ac.uk). Sotiris Moschoyiannis and Vytenis Sliogeris have been partly funded by UKRI Innovate UK, grant 77032. Yuhu Wu is with the School of Control Science and Engineering, Dalian University of Technology, 116024, Dalian, China (e-mail: wuyuhu@dlut.edu.cn).

motivated by the biological properties found in various target states, different approaches perturb individual nodes' states in a PBN in order to either drive it to some attractor within a finite number of steps (*horizon*), or change the network's long-run behaviour by affecting its steady-state distribution (by increasing the mass probability of the target states).

Seminal work by Cheng *et al.* on an algebraic state space representation (ASSR) of Boolean networks, based on semi-tensor product (STP) [20], stimulated an avalanche of research on such networks, including controllability [21]–[24]. However, ASSR linearises logical functions by enumerating their state spaces, hence it is model-based, and such methods require estimating $2^N \times 2^N$ probabilities, which quickly becomes intractable for a large number of nodes $N$.

Attempts to overcome this barrier include work on a polynomial time algorithm to identify pinned nodes [25], [26], which led to subsequent developments in pinning control to rely on local neighbourhoods rather than global state information [27], with perturbations taking the form of deleting edges, to generate an acyclic subgraph of the network. Control of larger BNs (*not* PBNs which are stochastic, rather than deterministic) has been attempted in [28], [29]. However, no guarantees are provided that the original network and its acyclic version, which is eventually controlled, have the same dynamics. In addition, there are concerns over how such changes in the network topology translate in biology since cycles are inherent in most biological pathways, e.g., see [30].

The pinning control strategy has been applied to PBNs by Lin *et al.* [31] but the approach is only demonstrated on a real PBN with $N$=9 nodes, which is hardly large-scale. Moreover, the target domain is an attractor (cyclic attractor comprising 2 states) and not a subset of the state space, validated by a favourable shift in the steady state distribution (SSD) of the PBN, which is amenable to larger networks. Further, the complexity of this most recent approach is $O(N^2 + N2^d)$, hence exponential on the largest in-degree $d$ of the pinned nodes in the acyclic version of the original PBN.

It transpires that the main challenge in dealing with the Boolean paradigm, as the computational counterpart of gene regulatory networks, has to do with the sheer scale of the network state space, i.e., the state transition graph or probability transition matrix, which provides a model of the system's dynamics, grows exponentially on the number of nodes. The primary objective is to develop optimal control methods that can systematically derive the series of perturbations required to direct the PBN from its current state to a target state or subset of states, thus rendering the system stabilizable at state(s) where the cell exhibits desired biological properties. In this paper we demonstrate stabilization of a Melanoma PBN within a space of $2^{200}$ states (Section IV).

Reinforcement Learning (RL) [32], by inception, addresses sequential decision-making by means of maximising a cumulative reward signal (indicator of how effective a decision was at the previous step), where the choice of action at each step influences not only the immediate reward but also subsequent states, and by virtue of that, also future rewards. Most importantly, RL provides a model-free framework and solves a discrete-time optimal control problem cast as an MDP.

In the context of stabilization of PBNs model-free means that the distribution of probabilities of the next state, and that from each state, is not known. The RL agent learns to optimise its reward through continuous interaction with the environment - the PBN, here - from an initial state towards a terminal condition (e.g., reaching the target domain or exceeding the horizon), by following some policy on selecting actions at each state along the way (which of the $m \leq N$ nodes' state to flip). Such an *episodic* framework can feature in Q-Learning, which combines learning from experience (like Monte Carlo methods) *with* bootstrapping (like Dynamic Programming). This means there is no need to wait until the end of the episode to update the estimates of the action-values at each state (Q function). Estimates of the current step are based on estimates of the next time step, until the policy converges to the optimal policy. In this way, the PBN dynamics are learned, by means of learning the reward function.

Previous work on control that utilises Q-Learning includes the work of Karlsen, *et al.* [33] on rule-based RL, in the form of an eXtended Classifier System (XCS) which was also applied to the yeast cell cycle BN ($N$=11) in [34]. The stabilization of PBNs with Q-Learning is studied in [35] but that work also only address a small apoptosis PBN ($N$=9). It transpires that Q-Learning RL struggles to converge to an optimal solution in complex and high-dimensional MDPs.

Naturally, the interest shifts towards combining Q-Learning (and its model-free promise) with deep learning for scalability. Papagiannis & Moschoyiannis in 2019 [36] first proposed a control method based on Deep Q-Learning with experience replay, namely DDQN with Prioritized Experience Replay (PER). This was applied to control of BNs in [36] and then to PBNs (synthetic $N$=20 and a real Melanoma $N$=7) in [19]. Subsequently, this deep RL method was applied to solve the output tracking problem in a reduced version of the T-cell receptor kinetics model (PBCN with $N$=28) in [37]. Batch-mode RL has been used in [38] to control the Melanoma PBN ($N$=28) also studied here. Nevertheless, the advantages of combining Q-Learning with neural network function approximation to provide efficiently scalable RL control methods applicable to *large-scale* PBNs remain largely unexplored.

In this article, we take this stream of research a step further by addressing *large-scale* PBNs through the application of model-free Deep Reinforcement Learning. In comparison to previous work [19], [37], we present an integrated control framework for set stabilization of large PBNs, based on model-free deep RL (DDQN with PER) which (i) can address different flavours of the control problem, with regard to the control inputs as well as the target domain, (ii) can validate successful control in large PBNs, where computing attractors is not feasible, and (iii) has time complexity which is *linearly* dependent on the number of time steps and not the number of nodes in the PBN, hence a clear advancement on polynomial [26] and exponential (on largest in-degree) [28] complexity.

As such, the main contributions of this paper are as follows:

1) We show that a model-free Deep RL control method for directing a PBN to a target state is *scalable*, with time complexity only *linearly* dependent on the number of time steps during training

2) We show the method to be versatile in that it can address:
(a) control input nodes (when known) but also consider the full set of nodes (when not known)
(b) the target domain for control to be a specific attractor (stabilization) but also a pre-assigned subset of the network state space (set stabilization)
3) We demonstrate the approach in successfully determining a control policy for PBNs and PBCNs, including stabilization of a Melanoma PBN with *200 nodes*.

The rest of this paper is structured as follows: Section II sets out key concepts behind PBNs, formulates the control problem, and outlines Deep Reinforcement Learning, focusing on DDQN with PER. The method for deriving series of perturbations (*control policies*) for stabilization of PBNs is developed in Section III. The main results of applying the control method to large PBNs are presented in Section IV, including comparison and discussion. Finally, Section V presents some concluding remarks and possible extensions.

## II. Preliminaries

### A. Probabilistic Boolean Networks (PBNs)

PBNs are a class of discrete dynamical systems characterised by interactions over a set of $N$ nodes, each taking a Boolean value $x_i$ in $\mathcal{D} = \{0, 1\}$. Hence, $x_i(t), i \in [1, N]$, denotes the state of the $i$-th node at time instance $t$, and represents the expression level of the $i$-th gene in the GRN being modelled. The update rule of $x_i(t)$ is determined by the Boolean function $f_i([x_j(t)]_{j \in N_i})$, and the value of $f_i$ is assigned to next state of node $x_i$, the set $N_i \subset [1, N]$ contains the subscript indices of in-neighbours, and $f_i : \mathcal{D}^{N_i} \to \mathcal{D}$ is the logical function chosen for node $i$ at time step $t$. The state of a PBN at time $t$, is denoted by $\mathcal{X}_t = [x_1(t), x_2(t), ..., x_N(t)]^\top$. Then the evolution (dynamics) of the BN is represented by the following vector form:

$$\mathcal{X}_{t+1} = \begin{bmatrix} x_1(t+1) = f_1([x_j(t)]_{j \in N_1}) \\ x_2(t+1) = f_2([x_j(t)]_{j \in N_2}) \\ \vdots \\ x_N(t+1) = f_N([x_j(t)]_{j \in N_N}) \end{bmatrix} \in \mathcal{D}^N \quad (1)$$

Each logical function $f_i$ has $l_i$ possibilities and is chosen from the finite set of Boolean functions $\mathbf{F}_i = \{f_i^1, f_i^2, \ldots, f_i^{l_i}\}$ (hence, $|\mathbf{F}_i| = l_i$) that the node is associated with.

Each function $f_i^k \in \mathbf{F}_i, k \in [1, l_i]$ is chosen with probability $Pr[f_i = f_i^k] = p_i^k$ with $\sum_{k=1}^{l_i} p_i^k = 1$. In this article, we assume that the assignment of logical functions for each node $i$ is independent. Hence, the probability of Boolean function selections, over $N$ nodes, is given by the product $p_1^{\mu_1} \cdot p_2^{\mu_2} \cdots p_N^{\mu_N}$, where $\mu_i \in [1, l_i]$. Different $f_i^k$ selections lead to different PBN realizations, which occur under different probabilities, resulting in stochastic state evolution of the network. Consequently, the possible realizations of a PBN are defined as $\mathcal{R} = \prod_{i=1}^{N} l_i$.

Thus, the probability $P_{\mathcal{X}_\mathbf{t}, \mathcal{X}_{\mathbf{t+1}}}$ of transitioning from state $\mathcal{X}_\mathbf{t} \to \mathcal{X}_{\mathbf{t+1}}$ at the next time step is: $P[x_1(t+1), x_2(t+1), \ldots, x_N(t+1) | x_1(t), x_2(t), \ldots, x_N(t)] = P_{\mathcal{X}_\mathbf{t}, \mathcal{X}_{\mathbf{t+1}}}$ where $P[x_1, x_2, \ldots, x_n | y_1, y_2, \ldots, y_n]$ denotes the joint probability of $x_1, x_2, \ldots, x_n$ conditioned to $y_1, y_2, \ldots, y_n$. We can now construct the transition probability matrix $\mathcal{P}$ comprised of $2^N \times 2^N$ entries where entry $\mathcal{P}_{m,n}$ indicates the probability of transitioning from current state $m$ to possible next state $n$.

### B. Control problem formulation

In the context of PBNs, and consequently GRNs, control takes the form of discovering policies, or series of interventions (perturbations) to the state of a node (gene), aiming to drive the network from its current state to a desirable state, where the network exhibits desirable biological properties.

**Definition 1.** *Consider a PBN at state $\mathcal{X}_\mathbf{t}$. Then, define intervention $\mathrm{I}(\mathcal{X}_\mathbf{t}, i_t)$, $0 \leq i \leq N$, as the process of flipping the binary value $x_i(t)$ associated with node $i$, at time $t$.*
*$i_t = 0$ denotes no intervention at time $t$.*

Since an intervention strategy in gene therapies should be the least intrusive to the GRN, only a single $\mathrm{I}(\mathcal{X}_\mathbf{t}, i_t)$ is allowed at each time step $t$. This means an intervention is followed by a natural network evolution step according to the PBN internal transition dynamics[1]. Hence, we resist operating in a more aggressive intervention mode although it is favourable from a computational viewpoint.

**Definition 2.** *Consider $\mathrm{I}(\mathcal{X}_\mathbf{t}, i_t)$, and intervention horizon $H \in \mathbf{Z}^+$. Define $\mathbf{S} = \{\mathrm{I}(\mathcal{X}_1, i_1), \mathrm{I}(\mathcal{X}_2, i_2), \ldots \mathrm{I}(\mathcal{X}_{h \leq H}, i_h)\}$,* where $0 \leq i_t \leq N$, and $h \leq H \in \mathbf{Z}^+$. Again, $i_t = 0$ denotes no intervention, at time $t$.

The objective is to obtain the sequence of interventions $\mathbf{S}$ that directs the network from the current state - sampled from a uniform distribution of all states $p(\mathcal{X}_0)$ - to a desirable state.

When control input nodes are not known, $\mathbf{S}$ is formed by perturbations effected on any node, hence there are $N + 1$ actions for the RL agent at each state. The objective is to determine the sequence $\mathbf{S}$ within a finite number of interventions, the horizon $H$, assuming that the MDP is ergodic. The experiments reported in the next section show this not to be a restricting assumption.

When control input nodes are known, or can be computed, $\mathbf{S}$ is formed by perturbations effected only on the control nodes. The objective here is to determine the sequence $\mathbf{S}$ required to increase the steady-state probability mass of desirable network states. In the study of control of the melanoma PBN [11], [38] these are the states where the gene *WNT5A*, which is central to the induction of an invasive phenotype in melanoma cells, is OFF. Perturbations are only allowed on the *pirin* gene, as in [11], [38]. Since the target domain does not assume knowledge of attractors, this allows addressing larger networks (cf. we demonstrate control of the Melanoma PBN $N$=200).

### C. Deep Reinforcement Learning

The central task of Reinforcement Learning [32] is to solve sequential decision problems by optimising a cumulative future reward. This can be achieved by learning estimates for the optimal value of each action which is typically defined as the sum of future rewards when taking that action and following the optimal policy afterwards.

[1]We stress that in this article the probability distribution of successor states, from each state, is unknown when learning the control policy.

The strategy that determines which action to take is called a *policy*. Hence, an *optimal policy* results from selecting the actions that maximize the future cumulative reward.

Q-Learning [39] maintains an estimate of the optimal value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ which is updated towards the target:

$$\begin{aligned} Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \\ + \alpha[r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a^{'}) - Q(s_t, a_t)] \end{aligned} \tag{2}$$

where $Q(s_t, a_t)$ is the expected reward of taking action $a_t$ at state $s_t$ at time step $t$, $r_{t+1}$ is the reward received at the next time step after taking action $a_t$, $\gamma$ is the discount factor that trades off the importance of immediate and later rewards, $r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a^{'})$ is the TD[2]-Target, $r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a^{'}) - Q(s_t, a_t)$ is the TD-Error $\delta$, and $0 < \alpha \leq 1$ is a constant that determines how fast the agent forgets past experiences. Contextually, given state $s_t$, Eq. (2) improves the estimate of the value associated with action $a_t$. Note Eq. (2) does not use any probabilities defined by the MDP.

The true value of each state-action pair, $Q^*(s, a)$ can be approximated iteratively by selecting actions at each time step. In this work we select actions following the $\varepsilon$-greedy *policy* algorithm, where $\varepsilon$ is the probability of randomly selecting an action $a$ at each time step, and $(1 - \varepsilon)$ of greedily performing the action with the maximum expected reward determined by $max_a Q(s_t, a)$. The value of $\varepsilon$ starts from 1 and decreases towards a constant value $min_\varepsilon$ at every time step. After an action is selected $Q(s_t, a_t)$ is updated according to Eq. (2). Note that $Q$ has been shown to converge to $Q^*$ [32], [39].

The objective is to determine a policy $\pi : \mathcal{S} \rightarrow \Delta_{\mathcal{A}}$ that maximises the expected return $\mathcal{J}$, for each initial state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, by direct interaction with the environment:

$$\mathcal{J} = \mathbb{E}_\pi[r(s, a)] = \mathbb{E}[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t) | \mathcal{P}, \pi] \tag{3}$$

where $H$ is the intervention horizon. A range of methods have been studied for this problem [40], [32]. Here, we appeal to a model-free RL method which draws upon Double Deep Q-Learning, with experience replay.

**Double Deep Q Network (DDQN).** Deep Reinforcement Learning combines classical reinforcement learning with neural network function approximation. More specifically, Double Q-Learning [41] is combined with Deep Q-Learning [42] to address the issue of overestimating $Q$ values. A parametric form of the state-action value function of Eq. (2), $Q(s, a; \theta)$, with parameters $\theta$ is often represented using neural networks:

$$\begin{aligned} \theta_{t+1} = \theta_t + \alpha[r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a^{'}; \theta_t) - \\ - Q(s_t, a_t; \theta_t)] \nabla_{\theta_t} Q(s_t, a_t; \theta_t) \end{aligned} \tag{4}$$

A Deep Q Network (DQN) is used here for this purpose, which is a multi-layer neural network with inputs being the current state observation of the environment and outputs a vector of the expected action values for that state $Q(s, \cdot; \theta)$. Training involves direct interaction with the (PBN) environment. The goal of the DQN is to iteratively update $\theta$ in order to approximate $Q^*(s, a; \theta)$. The DQN is trained by minimizing a (*different*) sequence of loss functions at each iteration:

$$L(\theta_t) = (r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a^{'}; \theta_t^-) - Q(s_t, a_t; \theta_t))^2 \tag{5}$$

where $\theta_t$ denotes the parameters of $Q$ of Eq. (2) and $\theta_t^-$ is a periodic copy of $\theta_t$. It is worth noting here that following the suggestion in [42] a separate network is used to determine the TD-Target. This *target* DQN is initialised with the same parameters as the main DQN (the so-called *policy* DQN), but has its parameters updated every $k$ iterations. That is, the expected $Q$ values of the *target* DQN are fixed and every $k$ iterations the parameters of the *policy* DQN are copied to the *target* DQN.

By differentiating Eq. (5) we obtain:

$$\begin{aligned} \nabla_{\theta_t} L(\theta_t) = (r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a^{'}; \theta_t^-) \\ - Q(s_t, a_t; \theta_t)) \nabla_{\theta_t} Q(s_t, a_t; \theta_t) \end{aligned} \tag{6}$$

which can be used to update the DQN parameters using stochastic gradient descent. In the context of Deep Reinforcement Learning [41] two networks corresponding to the two Q functions are used (as in [43]) and Eq. (6) becomes:

$$\begin{aligned} L(\theta_t) = (r_{t+1} + \gamma Q(s_{t+1}, argmax_{a'} Q(s_{t+1}, a^{'}; \theta_t); \theta_t^-) \\ - Q(s_t, a_t; \theta_t))^2 \end{aligned} \tag{7}$$

The update process remains the same where the parameters $\theta_t$ are copied to $\theta_t^-$ periodically, every $k$ iterations.

**Prioritized Experience Replay (PER).** During the training (cf Section III) the agent interacts with the environment by observing state $s_t$ and performing action $a_t$, either randomly with probability $\varepsilon$ or greedily with probability $(1-\varepsilon)$ by selecting the action with the highest $Q$ value, then the environment transitions to state $s_{t+1}$ and the agent receives reward $r_{t+1}$. Thus, the agent experiences a sequence of transition tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ which are stored in a replay buffer $\mathcal{B}$. The loss function of Eq. (5) is obtained as the expectation over a batch of such tuples capturing the experiences of the agent. Every time step $t$ a batch of experiences is sampled from $\mathcal{B}$ and used to update the Double DQN parameters using Prioritized Experience Replay (PER) [44].

Remember our approach does not use the Probability Transition Matrix, i.e., it is *model-free*, hence the agent learns the control policy by sampling the environment. However, training a network from consecutive samples directly obtained from the environment is inefficient because of the strong correlations in this data, which can cause a high variance in the network parameter updates [42].

Also, rare experiences can be forgotten rapidly because of the way $Q$ values are updated. Such experiences are useful because if the sampled experiences are dominated by those frequently occurring, the Double DQN can become biased towards selecting actions useful only for those experiences. PER breaks this correlation and avoids large oscillations of the network parameters leading to more stable learning, while allowing experiences to be seen more often, hence, also avoiding rare experiences from being forgotten fast [44].

[2]Temporal-Difference (TD) learning refers to algorithms where estimates at the current time step are based on those of the next time step [32].

In our implementation we use *proportional* prioritization compared to rank-based prioritization, as experimentally it yielded slightly better results. The probability $P(i)$ of a tuple $i$ being sampled is given by:

$$P(i) = \frac{p_i^\omega}{\sum_{z \in \mathcal{B}} p_z^\omega} \tag{8}$$

where $p_i$ is the priority value of the $i$-th tuple, given by $p_i = |\delta_i| + c$, where $|\delta_i|$ is the absolute TD-error associated with the $i$-th tuple, $c$ is a small constant to prevent experiences with zero TD-Error from never being replayed, and exponent $\omega$ determines the magnitude of prioritization, with $\omega = 0$ corresponding to the uniform case.

However, prioritizing replay creates a bias in learning towards samples with high TD-Error as they are sampled more often. To avoid this, importance weights [44] are attached to each tuple, given by:

$$w_i = (\frac{1}{|\mathcal{B}| \cdot P(i)})^\beta \tag{9}$$

where $|\mathcal{B}|$ is the size of the buffer, and $\beta$ is a hyperparameter used to anneal the amount of importance sampling over training episodes.

## III. Deep RL to control large PBNs

To address the control problem, a Deep Neural Network was constructed with an input layer of size $N$, for a PBN with $N$ nodes, two hidden layers with $M$[3] rectifier units each, and a linear output unit of size $N+1$ where $N$ corresponds to the expected Q values of the $N$ possible perturbations at each time step, while the extra unit corresponds to taking no action, i.e., no perturbation.

The behavior policy applied during training was $\varepsilon$-greedy [42] and our parameters are largely in line with the literature. $\varepsilon$ was set to start at 1 and decay to 0.05 over a given fraction of the training steps, where it stays fixed. In practice this fraction was set to 0.75 when the target domain was an attractor, and 0.1 and 0.5 when the target was a pre-assigned subset of the state space (depending on the PBN). The selected discount factor $\gamma$ was set to 0.99 to weigh future rewards strongly, as the agent may direct the network to the desired state after a large number of steps.

For Prioritized Experience Replay we set $\omega = 0.6$ and linearly anneal $\beta$ from 0.4 to 1 after 75% of training as done in [44]. Experimentally these hyperparameter choices yielded the best results. Intuitively, we want to prioritize aggressively, but not dominate the sampled experiences with large TD-Error experiences. The size of the replay buffer selected varies depending on the experiment, see Section IV.

During training the algorithm used for optimizing the network parameters is Adam, with $\alpha$ = 0.0001. Huber loss is used to minimize the network's TD-Error. The reason behind this choice is to avoid exploding gradients by error clipping [45]. We also sample a batch of 256 or 128 experiences from the replay memory every time we update the network parameters. Please see the Supplementary Material for further details on hyperparams.

**Reward Scheme & Episode Design** Measured design of the reward assignment is required to approximate $Q(s, a; \theta)$ during training and guide the agent towards the goal, i.e., derive the shortest sequence of perturbations that direct to the target domain. Consider $\mathcal{Y}$ to be the set of desirable states, $s_t = \mathcal{X}_\mathbf{t}$ and $a_t = i$. We define the following reward function:

$$r(s_t, a_t) := \begin{cases} r > 2 & \text{if } s_{t+1} \in \mathcal{Y} \\ -2 & \text{if } s_{t+1} \text{ is in a undesirable attractor} \\ -1 & \text{if } s_{t+1} \text{ is any other state} \end{cases} \tag{10}$$

Moreover, there is an additional cost to the agent taking an action by returning $r(s_t, a_t) - 1$ if $a_t \neq 0$. The action cost encourages minimum interventions. This scheme leads to successful control, as shown in the experiments (Section IV).

When the target domain is an attractor, an episode ends either when the desired attractor $\mathcal{Y}$ is reached or the horizon $H$ has been reached. The PBN dynamics entail that the network gravitates towards some attractor state, which may not be the desirable one. In such cases, the policy gets a reward of -2 so as to learn to avoid transitioning to non-desirable attractor states. Finally, to compensate for the fact the desirable state will rarely be achieved initially, especially in larger state spaces, the policy gets a reward $r(s_t, a_t) > 2$ whenever it does achieve it - in practice, this was set to 5. This encourages experience tuples with high priority to be sampled more often due to the PER mechanism.

Note that in cases where only the desirable attractor is known the scheme can be readily adapted so that the policy gets a reward of -1 for all states except for those in the desirable attractor. It may be instructive to read Eq. (10) in terms of the policy $\pi$ maximising the expected return $\mathcal{J}$ of Eq. (3) discussed earlier.

Following Eq. (10), maximising the performance objective $\mathcal{J}$ of Eq. (3) corresponds to the process of finding a sequence of perturbations that drive the PBN to the desirable states $\mathcal{Y}$ and achieve that with the least possible perturbations.

When the target domain is a pre-assigned subset of the state space, the reward scheme and episode design are slightly adjusted. The positive reward for being in a favourable state is retained, but in practice set to a larger value, 10. Additionally, there is no notion of an "undesirable attractor" now. Instead, there is a larger cost for being in an undesirable state, namely the inverse of the one for being in a desirable state, i.e., $-10$. Finally, the action cost is retained. Similar to typical RL continuous control environments, an episode is only terminated in this case when the horizon $H$ has been reached.

**Training.** Following standard practice, we train the network for a number of episodes–or time steps–and update the second estimator (*target* DQN) periodically.

When the target domain is an attractor, we train for up to 670,000 episodes in total depending on the network, updating the *target* DQN every 5,000 episodes, which we call an "epoch". During training we evaluate the performance of the learning agent by plotting the average number of perturbations per epoch, and the average reward per epoch. Fig. 1 shows the

[3]The number of rectifier units $M$ depends on $N$; through experimentation we found that $M = 64$ worked well in the PBNs under study (Section IV).

average number of perturbations decrease as training evolves, hence the agent learns the policy for the target network.

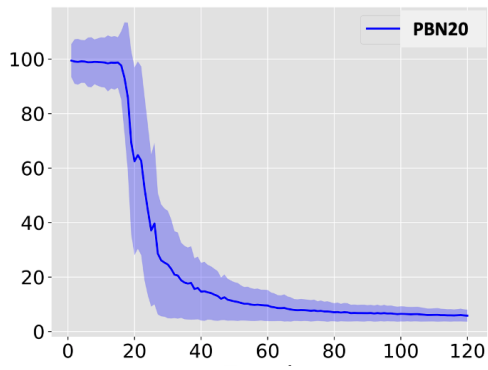


Fig. 1. Avg number of perturbations (y-axis) per epoch (x-axis) decreases as agent learns control policy during training (shown here for PBN $N$=20)

It may be worth noting the sharp drop in the average number of perturbations around the 18th epoch, i.e., after the first 90,000 episodes. We return to this point when we discuss stabilization of the $N$=20 PBN (cf. Section IV).

When the target domain is a subset of the state space, we opt to train for a number of time steps rather than a number of episodes, as the episode itself has lost the semantics attached to it. We train for 150,000 time steps and update the *target* DQN every 10,000 time steps with the exception of the $N$=70 PBN where we do so every 1,000 time steps.

**Successful control.** After training is complete, when the target is an attractor, we evaluate our control method by initializing the PBN in question from every possible state and attempting to direct it to the desirable state multiple times. Each attempt terminates when (i) the desirable attractor (fixed point or cyclic) is reached, or (ii) the agent exhausts the number of perturbations available to it, i.e., exceeds the horizon $H$. Whenever the attractor is reached, this counts as a successful control attempt.

When the target is a subsrt of the state space, we evaluate our method by computing the steady state distribution histogram of the network averaging across 300 distinct runs of 4,000 time steps each.

The set up and training of the DDQN with PER is given in Algorithm 1, which is what has been implemented to control the PBNs discussed in the following section.

Time complexity has been challenging in existing works. The time complexity of Algorithm 1 is best approximated by:

$$\mathcal{O}(t \cdot \mathcal{B} \cdot |\theta|) \tag{11}$$

where $t$ is the number of time steps, $\mathcal{B}$ is the batch size (samples fed into the neural network (of the DQN) for gradient descent), and $|\theta|$ is the number of parameters in the neural network.

It can be seen that our approach is *linearly* dependent on the time steps needed in training, for learning the optimal policy in reaching the target domain (selecting actions with the maximum reward). While this grows with the size of the network, the time complexity of the approach is a significant advance to existing approaches which are exponential or polynomial. This is in fact illustrated in the following section.

## IV. Experiments & Results

We report on experiments of applying the model-free deep RL (DDQN with PER) control method outlined in this paper on a number of networks of varying sizes (e.g., 7, 20, 28, 70, 200 nodes) and cover both the case where the goal is to reach a specific attractor and where the goal is to shift the probability mass favourably. More specifically, we report on applying it to two synthetic PBNs of $N$=10 and $N$=20, as well as four PBNs generated directly from gene data studied in the literature, with $N$=7, $N$=28, $N$=70 and $N$=200.

These experiments were run on standard issue hardware and took anywhere from 20 minutes to 6 hours depending on the training length.

### A. Attractor states as the target

We start with results on PBNs where the agent was allowed to perform perturbations on any node of the network.

**PBN $N$=10.** The connectivity of the PBN with 10 nodes, is given as follows: $inp^1 = [1, 10], inp^2 = [3, 8], inp^3 = [8, 10], inp^4 = [7, 8], inp^5 = [9, 6], inp^6 = [8, 2], inp^7 = [10, 4], inp^8 = [5, 9], inp^9 = [10, 9], inp^{10} = [4, 7]$ where $inp^i : N \rightarrow [1, N]$ denotes the set of nodes that provide input to node $i$.

We compute the attractors in a standard way, e.g., using the NetworkX [46] library, and find this PBN has 3 attractors, namely the network states $A_1 = \{0000000000\}$, $A_2 = \{10000000000\}$ and a cyclic attractor $A_3$ comprising 143 states $A_3 = \{1111001111, 1111001110, \ldots, 1010010110, 1000101110\}$; this can be easily reproduced by rolling out the PBN with the connectivity given earlier and the dynamics shown in Table I.

| $\mathbf{F}_i$ | OR | AND | XOR | $F^i$ | OR | AND | XOR |
|---|---|---|---|---|---|---|---|
| $\mathbf{F}_1$ | 1 | - | - | $\mathbf{F}_6$ | 0.82 | 0.15 | 0.03 |
| $\mathbf{F}_2$ | 0.5 | 0.25 | 0.25 | $\mathbf{F}_7$ | 0.48 | 0.52 | - |
| $\mathbf{F}_3$ | 0.71 | 0.29 | - | $\mathbf{F}_8$ | 0.28 | 0.45 | 0.27 |
| $\mathbf{F}_4$ | 0.52 | 0.48 | - | $\mathbf{F}_9$ | 1 | - | - |
| $\mathbf{F}_5$ | 0.36 | 0.05 | 0.59 | $\mathbf{F}_{10}$ | 0.99 | 0.01 | - |
| | $p_i^1$ | $p_i^2$ | $p_i^3$ | | $p_i^1$ | $p_i^2$ | $p_i^3$ |

TABLE I

Dynamics of the PBN $N$=10; each function $\mathbf{F}_i$, $i = 1..10$, with the probability $p_i^{1..3}$ of it being assigned to node $i$ is shown.

In order to select a target attractor, we allowed the PBN to *naturally* evolve assuming a uniform starting distribution for all states. We noted that $A_1$ only occurs with probability $P = 0.0097$, $A_2$ with probability $P = 0.0107$ and $A_3$ with probability $P = 0.9796$. This was to be expected as attractor $A_3$ comprises 144 network states. In our experiments with this PBN we set our desired state to be attractor $A_1$ as its probability of *naturally* occurring is the least.

In order to set the horizon $H$, we attempt to control the PBN starting from every possible state multiple times by applying random perturbations and note that the average interventions needed was 1,387. We decided to set the horizon at 11

**Algorithm 1** DDQN with PER Training Algorithm

**Input**: $\gamma$, $\min_\epsilon$, $|\mathcal{B}|$, $\beta$, $\omega$, $\alpha$, $N$, $N_{episodes}$, $N_{epochs}$, `horizon`, `batchSize`, c, `updateInterval`
**Output**: $\theta^*$

```
θ ← rand([0,1]), θ⁻ ← θ                                   ▷ Initialize network weights
B ← ∅, inc_β ← β / (0.75 × N_episodes × N_epochs), max_p ← 1   ▷ Initialize PER
ε ← 1, dec_ε ← (1 − min_ε) / (N_episodes × N_epochs)      ▷ Initialize ε-greedy
trainCount ← 0
for epoch ∈ [0, N_epochs] do
    for episode ∈ [0, N_episodes] do
        t ← 0, s_t ← ∅
        X_t ← rand(D^N)                                   ▷ Initialize PBN to a random state
        while s_t ∉ Y ∧ t ≠ horizon do
            s_t ← read(X_t), a_t ← ε−greedy(ε, s_t)
            s_{t+1}, r(s_t, a_t) ← apply(action)          ▷ Apply the chosen action to the environment
            saveToReplayBuffer(B, (s_t, a_t, r(s_t, a_t), s_{t+1}, max_p))
            if |B| ≥ batchSize then
                Sample (T = {S, A, R, S'}, W) where ∀i ∈ T, P(i) = p_i^ω / Σ_{z∈B} p_z^ω, ∀w_i ∈ W, w_i = (|B| · P(i))^{−β}
                L(θ) ← (R + γ max_{a'} Q(S', a'; θ⁻) − Q(S, A; θ)) · W
                θ ← θ − α∇_θ L(θ)                         ▷ Gradient Descent
                ∀i ∈ T update priorities with p'_i ← L(θ) × c, max_p ← max(p_i ∀i ∈ B)   ▷ Update PER priorities
                trainCount ← trainCount + 1
                if trainCount mod updateInterval = 0 then ▷ Update second DQN
                    θ⁻ ← θ
                end if
            end if
            t ← t + 1
        end while
        β ← min(β + inc_β, 1), ε ← max(ε − dec_ε, min_ε)  ▷ Update annealed parameters
    end for
end for
```

perturbations - that is, approximately $0.8\%$ of the average random perturbations needed to land on $A_1$. This PBN makes an interesting case for controllability as its possible network realizations, at each time step, are $N$=1,296, which makes state transitions highly non-deterministic.

For this experiment, with reference to Section III, we run 300,000 episodes for training, we set the reward $r$=5, the discount factor $\gamma$ to 0.99, parameter $c$ in PER to 500. The size of the input and output units have been set to match the size of the PBN and number of possible perturbations, i.e., 11. The size of the memory buffer was set to $10,000$ and we sampled experiences in batches of 128. Finally, we let the network train for $300,000$ episodes and update the second estimator network every $400$ episodes.

The agent achieves on average 99.8% successful control on the horizon of 11 perturbations. This result indicates that the network fails to always control 0.02% of the possible PBN states. As already mentioned in Section III there is a tradeoff between the number of allowed perturbations and maximizing the probability of successful control. It is interesting to note that if we allow the same DDQN with PER (trained on a maximum of 11 interventions) to perform 14 interventions during testing, it achieves 100% successful control.

The reason behind this result lies in the mechanics of Q-Learning. As explained in Section IIC, Q-Learning aims to maximize the cumulative reward. Hence, even if the agent was not successful in finding a policy that always achieves 100% control by at most 11 perturbations, it manages after the 11th perturbation to drive the PBN to a state that has previously been shown to evolve to states that are controllable. Thus, if the state naturally occurring after the 11th intervention has previously been controlled and the agent is allowed perturbation on that state, it will successfully drive it to the desired attractor.

The fact that 14 perturbations was the maximum number required to achieve control, with 100% success rate, means that the DDQN drives the states that failed to be controlled within 11 perturbations to states that at the next time step will always naturally transition to states that can be controlled within a maximum of 3 perturbations.

Further, it can be seen in Fig.2 that the average number of perturbations needed to control the PBN begin to decrease after training has begun - sharp decrease after the first 10 epochs in Fig.2. This would not be justifiable if the RL agent did not succeed in finding a control policy that drives the PBN to the desired attractor within the horizon of 11 perturbations.

**PBN $N$=20.** The connectivity of the PBN with 20 nodes is as follows: $inp^1 = [3, 6], inp^2 = [7, 14], inp^3 = [3, 5], inp^4 = [7, 4], inp^5 = [9, 6], inp^6 = [3, 11], inp^7 = [11, 3], inp^8 = [10, 9], inp^9 = [14, 7], inp^{10} = [8, 19], inp^{11} = [8, 6], inp^{12} =$

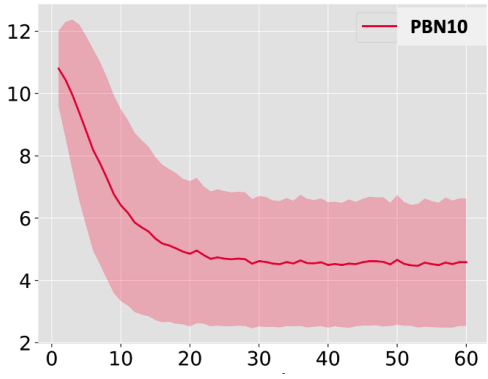


Fig. 2. The avg number of perturbations (y-axis) per epoch (x-axis) decrease as the agent determines the control policy (PBN $N$=10).

$[9,4], inp^{13} = [14,16], inp^{14} = [14,18], inp^{15} = [19,15], inp^{16} = [19,2], inp^{17} = [18,4], inp^{18} = [1,20], inp^{19} = [2,5], inp^{20} = [18,20]$, where, again, $inp^i$ denotes the set of nodes that provide input to node $i$.

Note the model-free deep RL method we propose does not require knowledge of the dynamics of the environment, in terms of the distribution of probabilities of successor states, from each state. However, we include here the Boolean function probabilities (see Table II) and the PBN structure to aid with our experiments being reproduced.

| $\mathbf{F}_i$ | OR | AND | XOR | $\mathbf{F}_i$ | OR | AND | XOR |
|---|---|---|---|---|---|---|---|
| $\mathbf{F}_1$ | 0.39 | 0.05 | 0.57 | $\mathbf{F}_{11}$ | - | 1 | - |
| $\mathbf{F}_2$ | 0.70 | - | 0.30 | $\mathbf{F}_{12}$ | - | 1 | - |
| $\mathbf{F}_3$ | 1 | - | - | $\mathbf{F}_{13}$ | 1 | - | - |
| $\mathbf{F}_4$ | 0.18 | 0.82 | - | $\mathbf{F}_{14}$ | 0.01 | 0.98 | 0.01 |
| $\mathbf{F}_5$ | - | 0.11 | 0.89 | $\mathbf{F}_{15}$ | - | - | 1 |
| $\mathbf{F}_6$ | 1 | - | - | $\mathbf{F}_{16}$ | - | 1 | - |
| $\mathbf{F}_7$ | 1 | - | - | $\mathbf{F}_{17}$ | 1 | - | - |
| $\mathbf{F}_8$ | - | 0.44 | 0.56 | $\mathbf{F}_{18}$ | - | 1 | - |
| $\mathbf{F}_9$ | - | - | 1 | $\mathbf{F}_{19}$ | - | - | 1 |
| $\mathbf{F}_{10}$ | 0.82 | 0.09 | 0.09 | $\mathbf{F}_{20}$ | 1 | - | - |
| | $p_i^1$ | $p_i^2$ | $p_i^3$ | | $p_i^1$ | $p_i^2$ | $p_i^3$ |

TABLE II
DYNAMICS OF THE PBN WITH $N$=20

Again, in order to select the specific desirable attractor, we allowed the PBN to *naturally* evolve assuming a uniform starting distribution of all states. The attractor $A = \{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\}$ was shown to occur the least. This was set as the desirable attractor so as to address the most difficult case for control.

We attempted to control the PBN starting from every possible state multiple times by random perturbations and noted that the average perturbations needed was 6,511. The maximum number of allowed perturbations was set at approximately $1.5\%$ of the average random perturbations, i.e., a horizon $H = 100$ perturbations.

Regarding the parameters of the DDQN with PER discussed in the previous section (Section III), for this experiment, the discount factor $\gamma$ was set to $0.99$, $c$ to 5,000 and the reward of Eq. (10) was set to $r$=20. The size of the memory buffer was 500,000. Finally, we let the network train for 670,000 episodes and update the second estimator every 5,000 episodes.

The agent achieves $100\%$ control on the horizon of 100 interventions. Further, it is worth noting that the success rate only drops by a small fraction to just over $99\%$ if we limit the agent to 15 perturbations only. This explains the sharp drop on the (avg) number of perturbations seen earlier in Fig. 1.

**Melanoma PBN $N$=7.** The proposed method was also applied to PBNs of real GRNs. We report here results on stabilization of a 7 node PBN, obtained from the gene expression data on metastatic melanoma found in [17], to a specific attractor. To allow for direct comparative study, the PBN includes the same 7 genes - *pirin, WNT5A, S100P, RET1, MART1, HADHB, and STC2* (in that order), which are also studied in existing literature [11], [38].

The attractors in this melanoma PBN $N$=7 are $A_1 = 1001001$, $A_2 = 0110110$ and $A_3 = 0101111$. Drawing on domain knowledge that a deactivated WNT5A gene can reduce metastasis on the Melanoma GRN [5], the desired state in this experiment was set to be the attractor where WNT5A is $OFF$, i.e., is set to 0, namely: $A_1$ =1**0**01111.

For this experiment, we run 150,000 episodes for training, we set the reward $r$=5, $\gamma$ to $0.9$, $c$ in PER to 500, and allowed up to 7 perturbations. The agent develops a control policy, see Fig. 3, and achieves $99.72\%$ control on the horizon of $H$=7 perturbations. It achieves $100\%$ control if allowed up to 10 perturbations. More details on this experiment found in [19].

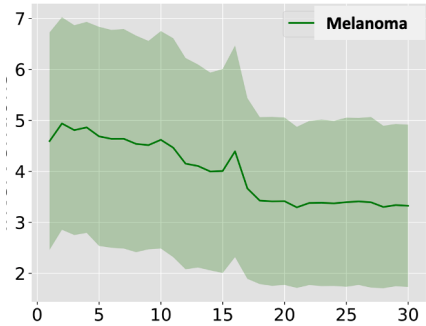


Fig. 3. The avg number of perturbations (y-axis) per epoch (x-axis) required to control the Melanoma PBN $N$=7

In terms of comparative study, the proposed method produces a success rate of $99.72\%$ compared to $96\%$ in [11] and $98\%$ in [38] who incidentally employ a batch-type of Reinforcement Learning. We note the favourable comparison despite the fact the action space is reduced to perturbations on one node only (namely *pirin*) in both works [11], [38] while we considered perturbations on any node here.

### B. Pre-assigned subset of the state space as the target

Next, we demonstrate the DDQN with PER control method in the case where the target is a pre-assigned subset of the state space. In addition, in this section perturbations can only be performed by the agent on a subset of the nodes, the so-called *control* or *input* nodes. The term Probabilistic Boolean Control Networks (PBCNs) is often used in the literature.

We have tested the method on PBNs of varying dimension, i.e., $N$=7, $N$=28, $N$=70 and $N$=200 which have been inferred from real gene expression data, namely the microarray data extracted from metastatic melanoma cells studied in Bittner,

*et al* [17]. Microarrays provide the relative activity of a large number of genes within a sample.

Before presenting the results of the control method on these real PBNs, we briefly outline the network inference method [47]. This builds on standard inference based on Coefficients of Determination (CODs) [48], which describe how good a function $f(X) = Y$ performs when predicting $Y$ using $X$. Here, $Y$ is the target gene (with state 1 or 0), and $X$ is the set of genes that have a directed edge into $Y$ ($inp^Y$). Typically, a linear predictor is used:

$$f(X) = a_1 X_1 \cdot a_2 X_2 \ldots a_n X_n + b \tag{12}$$

CODs are used to measure the effect on the prediction of adding a specific gene $x \in (N \setminus Y)$ to the set $X$, and they take the form

$$\theta = \frac{e_X - e_{X \bigcup x}}{e_X} \tag{13}$$

where $e_X$ is the error of the model when predicting $Y$ given $X$, and $e_{X \bigcup x}$ is the error of the model when predicting $Y$ given $X$ *and* some additional gene $x$.

However, there exist non-linear relationships in the data, hence linear predictors are not sufficient in this context, i.e., if the inputs for $Y$ are $x_1$ and $x_2$, and $Y$ is activated (state=1) when $x_1 \neq x_2$, then the linear predictor would entirely fail inferring this causal relationship. For this reason, we infer the functions straight from the single cell RNAseq data, based on [18] - once the inputs are determined, for each possible input combination we note the rate of the output being True.

This results in a Lookup table (LUT), which holds the rates of the output being True for each input combination. It can be used as a function. To infer such functions from data, given a particular set of input data $X$ and output data $Y$, for each observed input combination, $x$, the probability $p$ is given by:

$$p_x = P(Y = 1|X = x) = \frac{\#(Y = 1|X = x)}{\#(X = x)} \tag{14}$$

where $\#(Z)$ denotes the number of instances where $Z$ is True.

In terms of pre-processing in the data set comprising 8,150 genes and 31 samples, for gene selection discriminative weights are computed, which determine how a gene changes during the experiment compared to the control cells. Since gene expression data shows the relative activation levels of genes within cells, the values of which are between 0 and $\infty$, the gene data has to be quantized, and there are various methods for this [49]. Here, the thresholds were computed by applying k-means clustering, drawing upon Shmulevich & Dougherty in [50]. This results in quantized tuples $(F, m)$ where $F$ is the set of functions with corresponding probabilities (e.g., Table I), and $m$ the input mask.

Next, we present results of the proposed control method on large PBNs from real gene data used in cancer biology.

**Melanoma PBN $N$=28.** We consider a PBN of size 28 from the same metastatic melanoma gene data [17]. We use the exact same genes also studied in [38], for the purpose of direct comparison. Intervention in this case concerns perturbations on the *pirin* gene, and the objective is to arrive at a subset of network states where another gene, namely WNT5A, is OFF. The state space (just under 270 million states) makes calculating attractors prohibitive. Like [11], [38], we compute the steady state distribution (SSD) instead. To control such a network we aim to increase the mass probability of the network ending up in a desirable state.

We have also computed the steady state distribution histogram for this melanoma network but the $N$=28 has $2^{28}$ states on the x-axis making straightforward visualisation rather challenging. Instead, we show the SSD histogram for the Melanoma $N$=7 in Fig. 4 which comprises $2^7 = 128$ states on the x-axis. We note it agrees with the SSD in [38]; there are three prominent states, two of which are desirable. However, where the control of [38] increase most of the prominent states uniformly, our method only amplifies the desired attractor state out of the three prominent states.

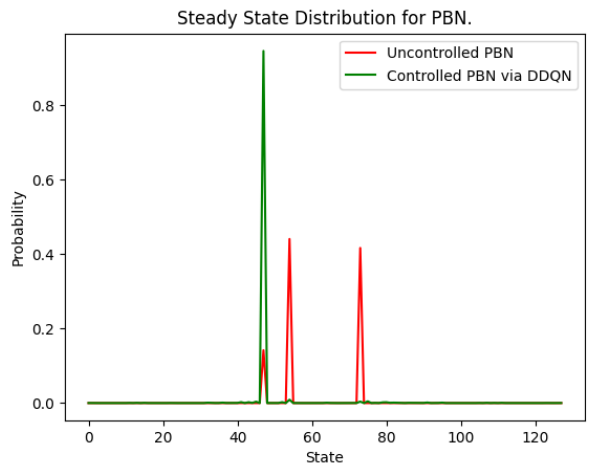


Fig. 4. States left of 64 are desirable; the rest are undesirable.

For the stabilization of the melanoma PBN $N$=28, the output layer of the DDQN is of size 2; one representing the Q value of taking or not taking an action, with an action defined as flipping the state of the *pirin* gene.

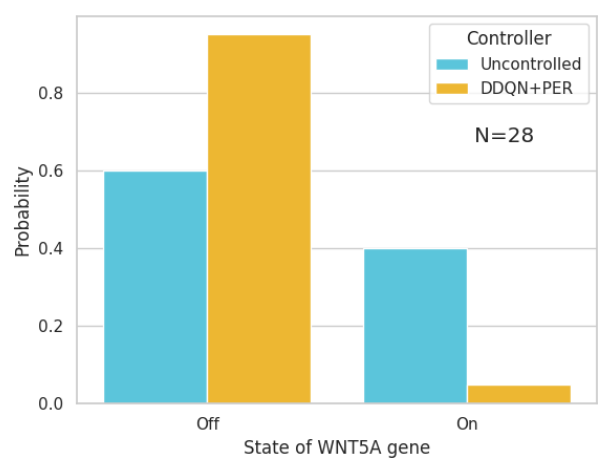


Fig. 5. Increase in probability mass of arriving at desirable state (PBN $N$=28).

The RL environment starts approaching the state space scale of environments tested in the original DQN paper [42] and

thus we adopt its default parameters for the experiments. The exploration fraction was set to 0.1, the *target* Q network is updated every 10,000 training steps and we train for 150,000. The primary adjustment is the use of 256 as the batch size.

The agent achieves 95.1% control, see Fig. 5, which is a significant improvement to the 80% reported previously in the literature, see [38].

**Melanoma PBN $N$=70.** The proposed DDQN with PER control method was tested against a larger PBN consisting of 70 nodes. This network was generated from the same data set on metastatic melanoma provided by Bittner, *et al* [17].

The selection of genes for this PBN comprises the 7 found in the studies of [11], [38] discussed earlier in the context of the smaller Melanoma PBNs, and the rest were appended from the gene list in Bittner, *et al* [17] based on their discriminative weights, which were computed as described in [17].

The SSD for this network agrees with $N$=28, see Fig. 6, which in turn agrees with the literature, as discussed before.

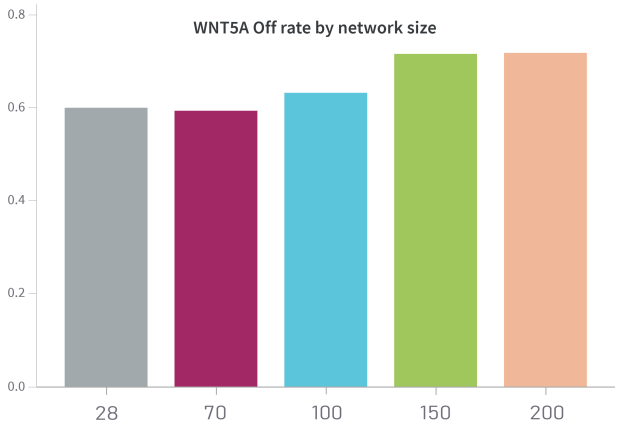


Fig. 6. Steady state distribution for different size PBNs from Bittner's Melanoma gene data [17].

The settings for training the DDQN with PER agent were adjusted so that the MLP architecture of the DQN has a larger amount of neurons in the first hidden layer, resulting in a (128 → 64) hidden layer structure. Additionally, we adjust the batch size back down to 128 and decrease the buffer size to 5,120, as well as update the target network every 1,000 training steps. The exploration fraction was also adjusted to 0.5.

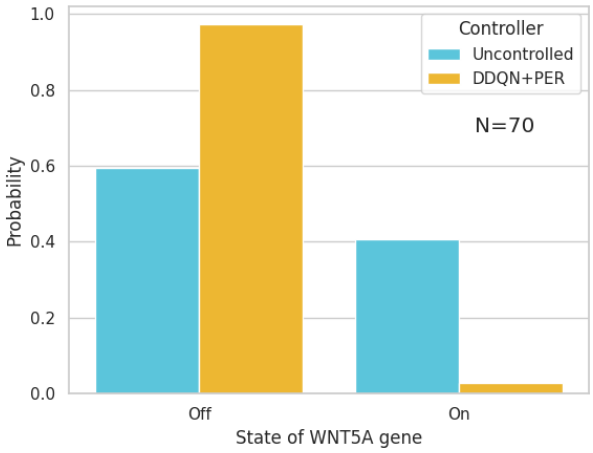


Fig. 7. Increase in probability mass of arriving at desired state (PBN $N$=70).

The agent achieves 97.3% successful control in this large PBN despite operating in a most challenging environment comprising $2^{70}$ states, see Fig. 7.

**Melanoma PBN $N$=200.** In order to truly test the scalability of the approach, we then generated a network of over double the size weighing in at $N$=200 nodes. The SSD for this much larger network does in fact agree with that of the $N$=28 and $N$=70, although it has drifted up slightly, which is in line with the trend observed in SSDs of networks of intermediate size generated from the same data as well (see Fig. 6).

The settings for training the RL agent to operate in the $2^{200}$ state space of this PBN were based on those of the melanoma PBN $N$=28 with some changes: the exploration fraction was adjusted to 0.5 to afford more exploration steps and the MLP architecture extended to 3 layers (256 → 128 → 64 architecture) to accommodate the 200 input size.

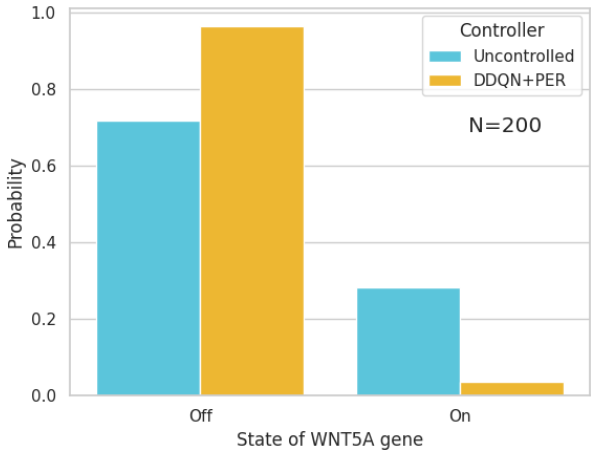


Fig. 8. Increase in probability mass of arriving at desired state (PBN $N$=200).

When this large PBN is controlled by the RL agent, it spends 96.4% of the time in the desirable states, see Fig. 8. To the best of our knowledge, this is the largest PBN to be controlled in the literature to date.

### C. Comparison & Discussion

The model-free, deep reinforcement learning (DDQN with PER) approach to stabilization of PBNs has been tried on a number of synthetic and real PBNs generated directly from gene expression data. On the synthetic PBN $N$=20, which comprises 1,048,576 states, the method achieves stabilization to an attractor that occurs 1 out of 10,000 times, within a horizon of 11. The method achieves stabilization 99% of the time when allowed up to 15 interventions.

On the well known Melanoma PBN $N$=7 [11] and [38], the method produces a policy for stabilization at a rate of 99.72%. This compares favourably to 96% in [11] and 98% in [38].

On significantly larger networks, we have demonstrated over 96.4% successful control on the Melanoma PBN $N$=70

and PBN $N$=200 generated in the same way from the same gene data provided by Bittner, *et al* [17]. By means of direct comparison, on the Melanoma PBN $N$=28 which is also studied in [38], we get 95.1% successful control compared to 80% (see Sec. 4.2 in [38]) hence, a significant improvement.

The approach in [37] applies Deep RL (in fact, the DDQN with PER method originally proposed by Papagiannis & Moschoyiannis in [19], [36] (which is also adapted here), to the output tracking problem in Boolean networks. It is applied only to situations where control inputs are available (hence, PB*C*Ns) and the target domain is an attractor. In contrast, we presented an integrative DRL framework that can address the full set of nodes as potential control nodes. In addition, the target domain in our study can be an attractor (fixed point or cyclic) *but also* a pre-assigned subset of the state space; in the latter case we have shown how we validate the control method using the corresponding steady state distribution before and after control is applied (recall Fig. 5, 7, 8). Finally, for the $N$=28 the approach in [37] requires $2 \times 10^6 = 2$M (million) episodes in training while, indicatively, our approach on a (different) $N$=28 network requires only 150,000 time steps. Finally, we demonstrated the scalability of the approach here by showing successful control of Melanoma PBN with $N = 200$.

| Network | DDQN+PER | DDQN | PPO | TRPO |
|---|---|---|---|---|
| N=28 | **0.951** (std: 0.028) | 0.910 (std: 0.045) | **0.951** (std: 0.008) | 0.947 (std: 0.013) |
| N=70 | **0.973** (std: 0.125) | 0.812 (std: 0.071) | 0.971 (std: 0.008) | 0.970 (std: 0.002) |
| N=200 | **0.964** (std: 0.020) | 0.912 (std: 0.016) | 0.946 (std: 0.006) | 0.956 (std: 0.001) |

TABLE III

COMPARISON OF FINAL PROBABILITY MASS OF ARRIVING AT DESIRED STATE BY APPLYING DIFFERENT RL ALGORITHMS FOR CONTROL, USING THE BEST RUN FOR EACH & STANDARD DEVIATION FROM 5 RUNS

Q-learning is a fundamental approach to RL but we also experimented with other state-of-the-art model-free RL algorithms, such as PPO [51] and its predecessor, TRPO [52]. Table III summarises our experiments that show there is little improvement to be had. It is worth noting that PPO and TRPO demonstrate much lower standard deviation in final performance across multiple training runs compared to DDQN+PER. This is to be expected as those policy gradient methods are revered for their stability, but the best seed for DDQN+PER consistently outperformed them, albeit marginally, for the networks we tested.

## V. CONCLUSION

We have demonstrated that the model-free, deep RL method (DDQN with PER) is successful in (set) stabilization of large-scale PBNs, of up to 200 nodes, with significant improvements in performance on Melanoma PBNs studied in the existing literature. The method does not assume knowledge of the Probability Transition Matrix and does not require control input nodes to be known, although it can readily work with them if they are available.

The time complexity, given by $\mathcal{O}(t \cdot B \cdot |\theta|)$ (Section III) is *linearly* dependent on the time steps $t$ needed in training, for learning the optimal policy in reaching the target domain.

In our experiments where the goal is to reach a specific attractor, this is chosen to be the one least likely to naturally occur. The intervention horizon was set to 0.8-1.5% of the random perturbations (on average) required to reach the target domain. The PBNs (and the PB*C*Ns) we addressed are highly stochastic; indicatively, the PBN $N$=10, has 1,296 possible network realisations at each step, compared to a handful of possible PBN realisations considered in existing works. Set stabilizaton of a real Melanoma PBN $N$=200 was achieved and was demonstrated via the favourable shift in the corresponding SSD towards the target domain.

Possible extensions include targeting known pitfalls in RL such as the need for frequent observations of the network state, as these can be computationally costly. Work is under way in this direction, on exploring sampled-data control, effectively varying the window of deciding on the next perturbation by considering *options* in RL within a semi-MDP framework [53].

One other interesting extension would be to determine ways to utilize the PBN's transition patterns observed during training to improve the learning efficiency. To this end, *control* nodes, in the sense of [1], [16], [54], [55], may be useful. The identification of pinning control nodes in [27] based on local neighbours rather than global state information could possibly be leveraged in this respect.

Alongside a similar vein of research would be to explore the application of *model-based* RL approaches [56], [57] on real cells directly and infer the model "on-the-go" to apply RL on it. This could lead to increased real-word gene data sample efficiency for deriving a control policy.

Finally, an exciting frontier for RL methods at the moment is using sequential models to learn a policy. Exploring the application of Decision Transformers [58] to the analysis of large-scale PBNs seems worthwhile.

Code and example networks can be found at:
https://github.com/UoS-PLCCN/pbn-rl/ and
https://github.com/UoS-PLCCN/gym-PBN

## REFERENCES


[1] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási, "Controllability of complex networks," *Nature*, vol. 473, no. 7346, p. 167, 2011.

[2] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks ," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, 2002.

[3] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, no. 3, pp. 437–467, 1969.

[4] R. Albert and H. G. Othmer, "The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster," *Journal of Theoretical Biology*, vol. 223, no. 1, pp. 1–18, 2003.

[5] A. Datta, A. Choudhary, M. L. Bittner, and E. R. Dougherty, "External Control in Markovian Genetic Regulatory Networks," *Machine Learning*, vol. 52, no. 1-2, pp. 169–191, 2003.

[6] I. Shmulevich, E. Dougherty, and W. Zhang, "Gene perturbation and intervention in probabilistic boolean networks," *Bioinformatics*, vol. 18, no. 10, pp. 1319–1331, 2002.

[7] S. Huang, I. Ernberg, and S. Kauffman, "Cancer attractors: A systems view of tumors from a gene network dynamics and developmental perspective," *Seminars in Cell & Developmental Biology*, vol. 20, no. 7, pp. 869–876, 2009.

[8] A. R. Udyavar *et al.*, "Novel hybrid phenotype revealed in small cell lung cancer by a transcription factor network model that can explain tumor heterogeneity," *Cancer Research*, vol. 77, no. 5, pp. 1063–1074, 2017.

[9] S. Huang and D. Ingber, "Shape-dependent control of cell growth, differentiation, and apoptosis: Switching between attractors in cell regulatory networks," *Experimental Cell Research*, vol. 261, no. 1, pp. 91–103, 2000.

[10] S. Reardon, "Modified viruses deliver death to antibiotic-resistant bacteria," *Nature*, vol. 546, no. 7660, pp. 58 6–587, 2017.

[11] R. Pal, A. Datta, and E. Dougherty, "Optimal infinite horizon control for probabilistic boolean networks," *IEEE Transactions on Signal Processing*, pp. 2375–2387, 2006.

[12] K. Kobayashi and K. Hiraishi, "Design of probabilistic boolean networks based on network structure and steady-state probabilities," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 8, pp. 1966–1971, 2017.

[13] P. Jones and S. Baylin, "The fundamental role of epigenetic events in cancer," *Nat Rev Genet*, vol. 3, pp. 415–428, 2002.

[14] E. Chatzaroulas, V. Sliogeris, P. Victori, F. M. Buffa, S. Moschoyiannis, and R. Bauer, "A structural characterisation of the mitogen-activated protein kinase network in cancer," *Symmetry*, vol. 14, no. 5, 2022.

[15] A. Baudin, S. Paul, C. Su, and J. Pang, "Controlling large Boolean networks with single-step perturbations," *Bioinformatics*, vol. 35, no. 14, pp. i558–i567, 07 2019.

[16] S. Moschoyiannis, N. Elia, A. Penn, D. J. B. Lloyd, and C. Knight, "A web-based tool for identifying strategic intervention points in complex systems," in *Proc. Games for the Synthesis of Complex Systems (CASSTING @ ETAPS)*, ser. EPTCS, vol. 220, 2016, pp. 39–52.

[17] M. Bittner, P. Meltzer, Y. Chen, Y. Jiang, E. Seftor, M. Hendrix, M. Radmacher, R. Simon, Z. Yakhini, A. Ben-Dor, N. Sampas, E. Dougherty, E. Wang, F. Marincola, C. Gooden, J. Lueders, A. Glatfelter, P. Pollock, C. J., E. Gillanders, D. Leja, K. Dietrich, C. Beaudry, M. Berens, D. Alberts, and V. Sondak, "Molecular classification of cutaneous malignant melanoma by gene expression profiling," *Nature*, vol. 406, no. 6795, pp. 536–540, 2000.

[18] S. Kim, H. Li, E. Dougherty, N. Cao, Y. Chen, M. Bittner, and E. Suh, "Can markov chain models mimic biological regulation?" *Journal of Biological Systems*, vol. 10, no. 4, pp. 337–357, 2002.

[19] G. Papagiannis and S. Moschoyiannis, "Deep reinforcement learning for control of probabilistic boolean networks," in *Complex Networks 2020*, vol. 944. Springer, 2020, pp. 361–371.

[20] D. Cheng, H. Qi, and Z. Li, *Analysis and Control of Boolean Networks: A Semi-tensor Product Approach*. Springer-Verlag, London, 2011.

[21] Y. Liu, H. Chen, J. Lu, and B. Wu, "Controllability of probabilistic boolean control networks based on transition probability matrices," *Automatica*, vol. 52, pp. 340–345, 2015.

[22] J. Lu, J. Zhong, C. Huang, and J. Cao, "On pinning controllability of boolean control networks," *IEEE Transactions on Automatic Control*, vol. 61, no. 6, pp. 1658–1663, 2016.

[23] Y. Wu, Y. Guo, and M. Toyoda, "Policy iteration approach to the infinite horizon average optimal control of probabilistic boolean networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[24] B. Li, J. Lu, Y. Liu, and Z.-G. Wu, "The outputs robustness of boolean control networks via pinning control," *IEEE Transactions on Control of Network Systems*, vol. 7, no. 1, pp. 201–209, 2020.

[25] S.-I. Azuma, T. Yoshida, and T. Sugie, "Structural oscillatority analysis of boolean networks," *IEEE Transactions on Control of Network Systems*, vol. 6, no. 2, pp. 464–473, 2019.

[26] E. Weiss and M. Margaliot, "A polynomial-time algorithm for solving the minimal observability problem in conjunctive boolean networks," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2727–2736, 2019.

[27] J. Zhong, D. W. C. Ho, and J. Lu, "A new approach to pinning control of boolean networks," *IEEE Transactions on Control of Network Systems*, pp. 1–1, 2021.

[28] S. Zhu, J. Lu, J. Zhong, Y. Liu, and J. Cao, "On the sensors construction of large boolean networks via pinning observability," *IEEE Transactions on Automatic Control*, pp. 1–1, 2021.

[29] S. Zhu, J. Lu, L. Sun, and J. Cao, "Distributed pinning set stabilization of large-scale boolean networks," *IEEE Transactions on Automatic Control*, pp. 1–1, 2022.

[30] H. Vundavilli, A. Datta, C. Sima, J. Hua, R. Lopes, and M. Bittner, "Bayesian inference identifies combination therapeutic targets in breast cancer," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 9, pp. 2684–2692, 2019.

[31] L. Lin, J. Cao, J. Lu, J. Zhong, and S. Zhu, "Stabilizing large-scale probabilistic boolean networks by pinning control," *IEEE Transactions on Cybernetics*, pp. 1–13, 2021.

[32] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[33] M. R. Karlsen and S. Moschoyiannis, "Evolution of control with learning classifier systems," *Applied Network Science*, vol. 3, no. 1, p. 30, 2018.

[34] M. Karlsen and S. Moschoyiannis, "Learning versus optimal intervention in random boolean networks," *Applied Network Science*, vol. 4, no. 1, pp. 1–29, 2019.

[35] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio, "Reinforcement learning approach to feedback stabilization problem of probabilistic boolean control networks," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 337–342, 2021.

[36] G. Papagiannis and S. Moschoyiannis, "Learning to control random boolean networks: A deep reinforcement learning approach," in *Complex Networks 2019*, vol. 881. Springer, 2019, pp. 721–734.

[37] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio, "Double deep-q learning-based output tracking of probabilistic boolean control networks," *IEEE Access*, vol. 8, pp. 199 254–199 265, 2020.

[38] U. Sirin, F. Polat, and R. Alhajj, "Employing batch reinforcement learning to control gene regulation without explicitly constructing gene regulatory networks," in *23rd Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 2013, p. 2042–2048.

[39] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[40] D. P. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1st Edition, 1996.

[41] H. van Hasselt, "Double Q-learning," *Advances in Neural Information Processing Systems*, vol. 23, pp. 2613–2621, 2010.

[42] V. Mnih and et al, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[43] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning, 30th AAAI," 2016.

[44] T. Schaul, J. Quan, A. I., and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2016.

[45] B. Faryabi, A. Datta, and E. R. Dougherty, "On reinforcement learning in genetic regulatory networks," in *IEEE/SP 14th Workshop on Statistical Signal Processing*, 2007, pp. 11–15.

[46] A. Hagberg and D. Conway, "Networkx: Network analysis with python," *URL: https://networkx. github. io*, 2020.

[47] V. Sliogeris, L. Maglaras, and S. Moschoyiannis, "Inferring probabilistic boolean networks from steady-state gene data samples," in *Complex Networks 2022*. Springer, 2022, to appear.

[48] S. Kim, E. Dougherty, Y. Chen, K. Sivakumar, P. Meltzer, J. Trent, and M. Bittner, "Multivariate measurement of gene expression relationships," *Genomics*, vol. 15, no. 67, pp. 201–209, 2000.

[49] C. Mussel, F. Schmid, T. J. Blutte, M. Hopfensitz, L. Lausser, and H. A. Kestler, "BiTrinA: multiscale binarization and trinarization with quality analysis," *Bioinformatics*, vol. 32, no. 3, pp. 465–468, 10 2015.

[50] I. Shmulevich and E. Dougherty, *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*. SIAM, 2010.

[51] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[52] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[53] A. Yerudkar, E. Chatzaroulas, C. Del Vecchio, and S. Moschoyiannis, "Sampled-data control of probabilistic boolean control networks: A deep reinforcement learning approach," *Information Sciences*, 2022, in press.

[54] S. Savvopoulos *et al.*, "Impact of removing nodes on the controllability of complex networks," in *Complex Networks*, 2017, pp. 361–363.

[55] D. L. Gibbs and I. Shmulevich, "Solving the influence maximization problem reveals regulatory organization of the yeast cell cycle," *PLoS Comput Biol*, vol. 13, no. 6, p. e1005591, 2017.

[56] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv preprint arXiv:1802.10592*, 2018.

[57] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[58] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 15 084–15 097, 2021.

## Appendix

### Hyperparameter Settings

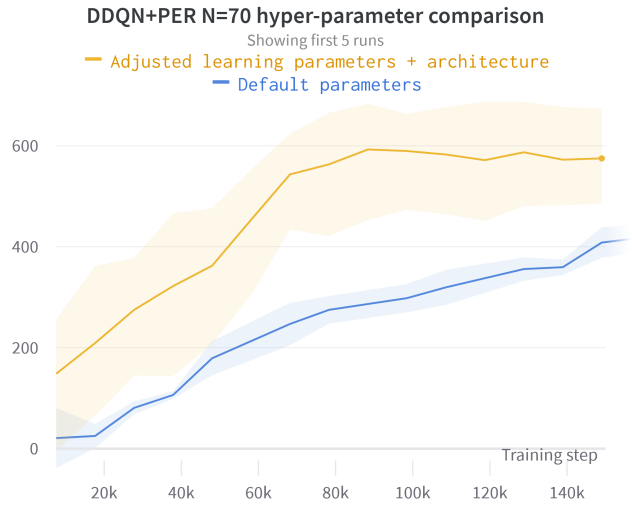


Fig. 9. Comparison of different DDQN+PER parameter settings for the $N$=70 network.

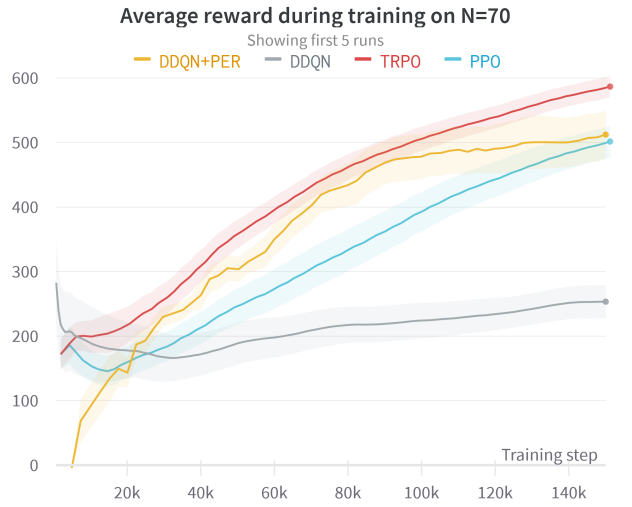


Fig. 10. Comparison of learning performance between the model-free methods on the $N$=70 PBN.

Our hyperparameter optimization process was to start with the parameters recommended in one of the breakout papers that brought the potential of DQN to light[4]. It is worth noting that this set of parameters is also the set of default parameters for the popular Reinforcement Learning library Stable baselines3[5]. This set of hyperparameters worked out of the box for the $N$=28 Melanoma PBN, but that was not the case for $N$=70 and $N$=200, showing limited to no learning signal.

[4]V. Mnih and et al, "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[5]https://github.com/DLR-RM/stable-baselines3

For the $N$=70, we first tried adjusting the "learning" parameters–as in parameters that directly affect how and when the agent learns. These include the batch size, the learning rate, the interval at which the *target* Q network gets updated with the latest parameters and the buffer size. DDQN is an off-policy model-free RL algorithm, but the on-policy baselines we tried in the case of PPO [6] and TRPO[7] did not struggle in this network whatsoever. Thus, we adjusted the DDQN parameters to steer its behaviour closer to an on-policy algorithm. By dramatically reducing the buffer size from 1,000,000 to 5,120, we quickly discard old samples, although not as quickly as on-policy methods. To avoid overfitting on the much smaller dataset, we reduce the batch size from 256–which we used for the $N$=28 PBN–to 128. Finally, observing that the dimension of the input layer (70 neurons) was smaller than that of the first hidden layer (64 neurons), we adjusted the first hidden layer to 128 neurons. Finally, we increased the exploration fraction to 0.5 in order to afford more exploration, and copied over the parameters every 1,000 time steps instead of the default of 10,000.

This set of parameters performed significantly better than the default set of parameters, as shown in Fig. 9, and much closer to the well-performing baselines, as shown in Fig. 10.

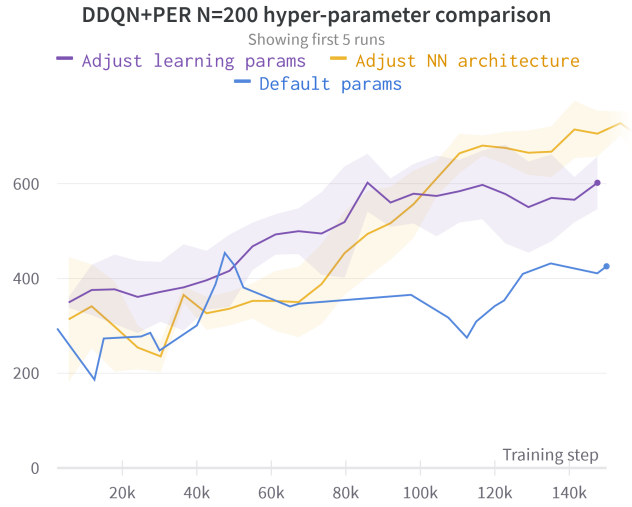


Fig. 11. Comparison of different DDQN+PER parameter settings for the $N$=200 network.

For the $N$=200, we initially began with applying all the learning parameter changes from $N$=70. While this did yield an increase in performance, it was still not up to par with the rest of the baselines. We then tried adjusting the MLP architecture itself in a similar way we did to the $N$=70, except this time add a third hidden layer at the front of 256 neurons to accommodate the 200 inputs. Additionally, we adjusted the buffer size back up to its default value of 1,000,000 and batch size back to 256 as well as the target Q

[6]J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347,2017.

[7]J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in International conference on machine learning. PMLR, 2015, pp. 1889–1897

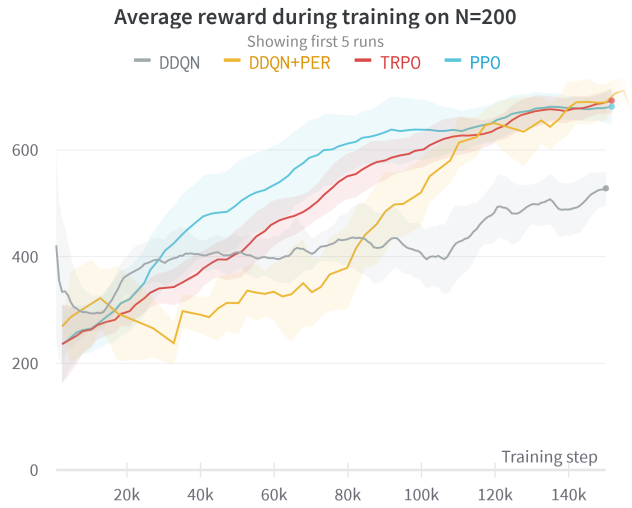


Fig. 12. Comparison of learning performance between the model-free methods on the $N$=200 PBN.

network parameter copy interval to 10,000, motivated by the much larger state space and the fact that the default–more "off-policy"–parameters still showed *some* learning signal, unlike the $N$=70 where they showed almost none at all. This final set of parameters outperformed the defaults and the lightly tuned $N$=70 parameters as shown in Fig. 11, and once again performed similar to PPO and TRPO as shown in 12.

Overall, we suggest starting with the default parameters from [8] and adjusting in the following order:

1) **MLP Architecture** - In both cases, having an architecture that better fit the input and output sizes helped dramatically.
2) **Buffer size** - 1,000,000 is extremely large especially when PER is used. While it might not lead to bad performance in a lot of cases, sometimes reducing this can speed up learning as older experiences are discarded faster since they get wiped out of the buffer due to its size.
3) **Target network update interval** - Lowering this is worth trying alongside making the buffer size smaller as in some cases the target network does not update quickly enough at 10,000 to facilitate Q learning.
4) **Batch size / learning rate** - These do not seem to affect performance when kept within reasonable settings, but it is worth noting that oftentimes 256 batch size might be too large.

[8] V. Mnih and et al, "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.