DEEPCAS: A Deep Reinforcement Learning Algorithm for Control-Aware Scheduling

Burak Demirel, Arunselvan Ramaswamy, Daniel E. Quevedo and Holger Karl

***Abstract*—We consider networked control systems consisting of multiple independent controlled subsystems, operating over a shared communication network. Such systems are ubiquitous in cyber-physical systems, Internet of Things, and large-scale industrial systems. In many large-scale settings, the size of the communication network is smaller than the size of the system. In consequence, scheduling issues arise. The main contribution of this paper is to develop a deep reinforcement learning-based *control-aware* scheduling (DEEPCAS) algorithm to tackle these issues. We use the following (optimal) design strategy: First, we synthesize an optimal controller for each subsystem; next, we design a learning algorithm that adapts to the chosen subsystems (plants) and controllers. As a consequence of this adaptation, our algorithm finds a schedule that minimizes the *control loss*. We present empirical results to show that DEEPCAS finds schedules with better performance than periodic ones.**

***Index Terms*—Deep learning, reinforcement learning, optimal control, networked control systems, scheduling, communication**

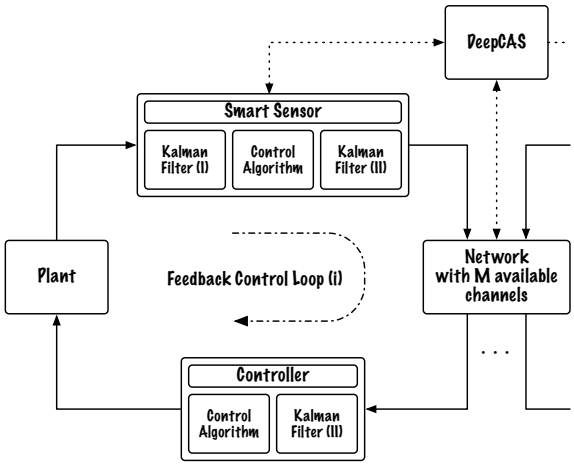

Fig. 1. Networked control system (NCS) that consists of N control subsystems closed over a shared communication network.

I. INTRODUCTION

ARTIFICIAL intelligence (AI) offers an attractive set of tools that are mostly model-free, yet useful in solving stochastic and optimal control problems arising in cyber-physical systems (CPS), Internet of Things (IoT), and large-scale industrial systems. AI-based solutions have seen a major resurgence in recent years, partly owing to recent advances in computational capacities and owing to advances in deep neural networks for function approximation and feature extraction. Oftentimes, the use of reinforcement learning algorithms or AI in conjunction with traditional controllers reduces the complexity of system design while boosting efficiency.

The abovementioned systems are all characterized by large sizes. However, typical resources, such as communication channels, computational resources, network bandwidth etc., do not scale with system size. In other words, resource allocation is an important problem in this setting. In addition, in a distributed control setting that involves feedback, resource allocation is required to be "control-aware", i.e., it is needed to aid in optimizing closed-loop control performance. In such feedback driven systems, controllers often rely on information collected from various sensors to make intelligent decisions. Hence, efficient information dispersion is essential for decision making over communication networks to be effective. As noted earlier, this is a hard problem since the number of communication channels available is much smaller than what is ideally required to transfer data from sensors to controllers.

Fig. 1 illustrates a simplified representation of the class of CPS and IoT systems of interest. The system consists of N *independent* subsystems that communicate over a shared communication network, which contains M channels. We assume that $M \ll N$ (M is much smaller than N), and that transmissions are via error-free channels. Each subsystem consists of one smart sensor, one controller, and one plant. Within each subsystem, there is feedback from the sensor to the controller. These feedback loops are closed over this resource-constrained communication network.

At every stage, DEEPCAS, our deep reinforcement learning-based model-free scheduling algorithm, decides which M of the N subsystems are allocated channels to close the feedback loop. DEEPCAS takes scheduling decisions by adapting to the control actions while trying to minimize the control loss. At every stage, the smart sensors compute estimates of the subsystem states, using *Kalman Filter (I)*, for transmission to the corresponding controller, see Fig. 1. The controller runs *Kalman Filter (II)* to estimate the subsystem state in the absence of transmissions. In addition to *Kalman Filter (I)*, each smart sensor also implements a copy of *Kalman Filter (II)* and the control algorithm. In other words, the smart sensor is cognizant of the state estimate used by the

Manuscript received August 31, 20xx; revised November 12, 20xx; accepted November 28, 20xx. Date of publication December 4, 20xx; date of current version December 21, 20xx. Recommended by Senior Editor G. Cherubini. (Corresponding author: Burak Demirel.)

A. Ramaswamy was supported by the German Research Foundation (DFG) (project number 315248657).

B. Demirel is with Autonomous Motion, ATS Pre-Development and Research, Scania CV AB (e-mail: burak.demirel@scania.com).

A. Ramaswamy, D. E. Quevedo, and H. Karl are with Paderborn University, Warburger Straße 100, 33098, Paderborn, Germany (e-mail: arunr@mail.uni-paderborn.de, dquevedo@ieee.org, holger.karl@upb.de).

controller at every time instant. DEEPCAS obtains feedbacks (i.e., rewards) from sensors for taking scheduling decisions.

Previously, several scheduling strategies have been proposed to determine the access order of different sensors and/or actuators; see [1] and references therein. A popular approach is to use periodic schedules [2]–[5] since they are easy to implement and they facilitate stability analysis of networked control systems. Unfortunately, finding optimal periodic schedules for control applications may not be easy since both period and sequence need to be found. Further, restricting to periodic schedules may lead to performance loss [6]. With a handful of exceptions, the determination of optimal schedules indeed requires solving a mixed-integer quadratic program, which is computationally infeasible for all but very small systems; see [6], [7].

Event- and self-triggering algorithms present popular alternatives to periodic scheduling; see [8] and references therein. Linear, quadratic optimal control problems subject to such scheduling schemes have been investigated in [9]–[12]. Many of the aforementioned results only consider single-loop control systems. There exists limited literature that study multi-loop control systems [9]–[11]. One limitation is that many of these results only investigate linear scalar systems.

Our contribution in the present work is in the development of a deep reinforcement learning-based control-aware scheduling algorithm, DEEPCAS. At its heart lies the Deep Q-Network (DQN), a modern variant of Q learning, introduced in [13]. In addition to being readily scalable, DEEPCAS is completely model-free. To optimize the overall control performance, we propose the following sequential design of control and scheduling: In the first step, we design an optimal controller for each independent subsystem. As discussed in [12], under limited communication, the *control loss* has two components: **(a)** best possible control loss **(b)** error due to intermittent transmissions. If $M = N$, then **(b)** vanishes. Since we are in the setting of $M \ll N$, the goal of the scheduler is to minimize **(b)**. To this end, we first construct an associated Markov decision process (MDP). The state space of this MDP is the difference in state estimates of all controllers and sensors (obtainable from the smart sensors). The single-stage reward is the negative of the loss component **(b)**. Since we are using DQN to solve this MDP, we do not need the knowledge of transition probabilities. The goal of DEEPCAS is to find a scheduling strategy that maximizes the reward, i.e., minimizes **(b)**.

II. NETWORKED CONTROL SYSTEM: MODEL, ASSUMPTIONS, AND GOALS

A. Model for each subsystem

As illustrated in Fig. 1, our networked control system consists of N independent closed-loop subsystems. The feedback loop within each subsystem (plant) is closed over a shared communication network. For $1 \leq i \leq N$, subsystem i is described by

$$\boldsymbol{x}_{t+1}^{(i)} = A^{(i)}\boldsymbol{x}_t^{(i)} + B^{(i)}\boldsymbol{u}_t^{(i)} + \boldsymbol{w}_t^{(i)} \,, \tag{1}$$

where $A^{(i)}$ and $B^{(i)}$ are matrices of appropriate dimensions, $\boldsymbol{x}_t^{(i)} \in \mathbb{R}^{n_i}$ is the state of subsystem i, $\boldsymbol{u}_t^{(i)} \in \mathbb{R}^{m_i}$ is the control input, and $\boldsymbol{w}_t^{(i)} \in \mathbb{R}^{n_i}$ is zero-mean i.i.d. Gaussian noise with covariance matrix $W^{(i)}$. The initial state of subsystem i, $\boldsymbol{x}_0^{(i)}$, is assumed to be a Gaussian random vector with mean $\bar{\boldsymbol{x}}_0^{(i)}$ and covariance matrix $X_0^{(i)}$ and of each other.

At a given time t, we assume that only noisy output measurements are available. We, thus, have:

$$\boldsymbol{y}_t^{(i)} = C^{(i)}\boldsymbol{x}_t^{(i)} + \boldsymbol{v}_t^{(i)} \,, \tag{2}$$

where $\boldsymbol{v}_t^{(i)} \in \mathbb{R}^{p_i}$ is zero-mean i.i.d. Gaussian noise with covariance matrix $V^{(i)}$. The noise sequences, $\boldsymbol{w}_t^{(i)}$ and $v_t^{(i)}$, are independent of the initial conditions $\boldsymbol{x}_0^{(i)}$.

B. Control architecture and loss function

The dynamics of each subsystem is a stochastic linear time-invariant (LTI) system given by (1). Further, each subsystem is independently controlled. Dependencies do arise from sharing a communication network. Subsystem i has a smart sensor which samples the subsystem's output $\boldsymbol{y}_t^{(i)}$ and computes an estimate of the subsystem's state. This value is then sent to the associated controller, provided a channel is allocated to it by DEEPCAS. If the controller obtains a new state estimate from the sensor, then it calculates a control command based on this state estimate. Otherwise, it calculates a control command based on its own estimate of the subsystem's state.

The control actions and scheduling decisions (of DEEPCAS) are taken to minimize the total control loss given by

$$J = \sum_{i=1}^{N} J^{(i)}, \tag{3}$$

where $J^{(i)}$ is the expected control loss of subsystem i and is given by

$$J^{(i)} = \mathbb{E}\Bigg[\boldsymbol{x}_{\mathrm{T}}^{(i)\top} Q_f^{(i)} \boldsymbol{x}_{\mathrm{T}}^{(i)} + \sum_{t=0}^{\mathrm{T}-1} \left(\boldsymbol{x}_t^{(i)\top} Q^{(i)} \boldsymbol{x}_t^{(i)} + \boldsymbol{u}_t^{(i)\top} R^{(i)} \boldsymbol{u}_t^{(i)}\right)\Bigg] \,,$$

where $Q^{(i)}$ and $Q_f^{(i)}$ are positive semi-definite matrices and $R^{(i)}$ is positive definite.

C. Smart sensors and pre-processing units

Within our setting, the primary role of a smart sensor is to take measurements of a subsystem's output. Also, it plays a vital role in helping DEEPCAS with scheduling decisions. It is from the smart sensors that DEEPCAS gets all the necessary feedback information for scheduling. For these tasks, each smart sensor employs two Kalman filters: (1) *Kalman Filter (I)* is used to estimate the subsystem's state, (2) a copy of *Kalman Filter (II)* is used to estimate the subsystem's state as perceived by the controller. Note that the controller employs *Kalman Filter (II)*. Below, we discuss the set-up in more detail.

Kalman filter (I): Since we assume that the sensors have knowledge of previous plant inputs, the sensors employ standard Kalman filters to compute the state estimate $\hat{\boldsymbol{x}}_{t|t}^{(i)s}$ and covariance $P_{t|t}^{(i)s}$ recursively as:

$$\begin{aligned}
\hat{\boldsymbol{x}}_{t|t-1}^{(i)s} &= A^{(i)}\hat{\boldsymbol{x}}_{t-1|t-1}^{(i)s} + B^{(i)}\boldsymbol{u}_{t-1}^{(i)} \\
P_{t|t-1}^{(i)s} &= A^{(i)}P_{t-1|t-1}^{(i)s}A^{(i)\top} + W^{(i)} \\
K_t &= P_{t|t-1}^{(i)s}C^{(i)\top}\big(C^{(i)}P_{t|t-1}^{(i)s}C^{(i)\top} + V^{(i)}\big)^{-1} \\
\hat{\boldsymbol{x}}_{t|t}^{(i)s} &= \hat{\boldsymbol{x}}_{t|t-1}^{(i)s} + K_t^{(i)}\big(\boldsymbol{y}_t^{(i)} - C\hat{\boldsymbol{x}}_{t|t-1}^{(i)s}\big) \\
P_{t|t}^{(i)s} &= \big(I - K_t^{(i)}C^{(i)}\big)P_{t|t-1}^{(i)s}\,,
\end{aligned}$$

starting from $\hat{\boldsymbol{x}}_{0|-1}^{(i)s} = \bar{\boldsymbol{x}}_0^{(i)}$ and $P_{0|-1}^{(i)s} = X_0^{(i)}$.

Kalman filter (II): The controller runs a minimum mean square error (MMSE) estimator to compute estimates of the subsystem's state as follows:

$$\hat{\boldsymbol{x}}_{t|t-1}^{(i)c} = A^{(i)}\hat{\boldsymbol{x}}_{t-1|t-1}^{(i)c} + B^{(i)}\boldsymbol{u}_{t-1}^{(i)}\,, \tag{4}$$

$$\hat{\boldsymbol{x}}_{t|t}^{(i)c} = \begin{cases} \hat{\boldsymbol{x}}_{t|t}^{(i)s} & \text{if the MMSE estimate received}\,, \\ \hat{\boldsymbol{x}}_{t|t-1}^{(i)c} & \text{otherwise}\,, \end{cases} \tag{5}$$

with $\hat{\boldsymbol{x}}_{0|-1}^{(i)c} = \bar{\boldsymbol{x}}_0^{(i)}$.

D. Goal: minimizing the control loss

For the control problem studied, the certainty equivalent (CE) controller is still optimal; see [12] for details. Using the control commands, generated by the CE controllers, the minimum value of the total control loss, (3), has two components: **(a)** best possible control loss **(b)** error due to intermittent communications. Hence, the problem of minimizing control loss has two separate components: (i) designing the best (optimal) controller for each subsystem and (ii) scheduling in a control-aware manner.

Component I: Controller design. The controller in feedback loop i takes the following control action, $\boldsymbol{u}_t^{(i)}$, at time t:

$$\boldsymbol{u}_t^{(i)} = -L_t^{(i)}\hat{\boldsymbol{x}}_{t|t}^{(i)c}, \tag{6}$$

where $\hat{\boldsymbol{x}}_{t|t}^{(i)c}$ is the state estimate used by the controller,

$$L_t^{(i)} = (B^{(i)\top}S_{t+1}^{(i)}B^{(i)} + R^{(i)})^{-1}B^{(i)\top}S_{t+1}^{(i)}A^{(i)} \tag{7}$$

and $S_t^{(i)}$ is recursively computed as

$$\begin{aligned}
S_t^{(i)} = A^{(i)\top}S_{t+1}^{(i)}A^{(i)} + Q^{(i)} - A^{(i)\top}S_{t+1}^{(i)}B^{(i)} \\
\times (B^{(i)\top}S_{t+1}^{(i)}B^{(i)} + R^{(i)})^{-1}B^{(i)\top}S_{t+1}^{(i)}A^{(i)},
\end{aligned} \tag{8}$$

with initial values $S_N^{(i)} = Q_f^{(i)}$. Let $\hat{\boldsymbol{x}}_{t|t}^{(i)s}$ be the state estimate of *Kalman Filter (I)*, as employed by the sensor. We have $\hat{\boldsymbol{x}}_{t|t}^{(i)c} = \hat{\boldsymbol{x}}_{t|t}^{(i)s}$ when the sensor and controller of the feedback loop i have communicated. Otherwise, $\hat{\boldsymbol{x}}_{t|t}^{(i)c}$ is the state estimate obtained from *Kalman Filter (II)*. The minimum value of the control loss of subsystem i is given by

$$\begin{aligned}
J^{(i)} = \bar{\boldsymbol{x}}_0^{(i)\top}S_0^{(i)}\bar{\boldsymbol{x}}_0^{(i)} + \mathrm{Tr}\big(S_0^{(i)}X_0^{(i)}\big) + \sum_{t=0}^{\mathrm{T}-1}\mathrm{Tr}\big(S_{t+1}^{(i)}W^{(i)}\big) \\
+ \sum_{t=0}^{\mathrm{T}-1}\mathrm{Tr}\big(P_{t|t}^{(i)s}\Gamma_t^{(i)}\big) + \sum_{t=0}^{\mathrm{T}-1}\mathbb{E}\Big[\boldsymbol{e}_{t|t}^{(i)\top}\Gamma_t^{(i)}\boldsymbol{e}_{t|t}^{(i)}\Big]\,,
\end{aligned} \tag{9}$$

where $\Gamma_t^{(i)} \triangleq L_t^{(i)\top}(B^{(i)\top}S_{t+1}^{(i)}B^{(i)} + R^{(i)})L_t^{(i)}$ and $\boldsymbol{e}_{t|t}^{(i)} \triangleq \hat{\boldsymbol{x}}_{t|t}^{(i)s} - \hat{\boldsymbol{x}}_{t|t}^{(i)c}$ stems from communication errors in subsystem i. Recall that there are N subsystems and $M << N$ communication channels.

Component II: Control-aware scheduling. The main aim of the scheduling algorithm is to help minimize J of (3). To this end, one must minimize

$$\sum_{t=0}^{\mathrm{T}-1}\mathbb{E}\Big[\boldsymbol{e}_{t|t}^{(i)\top}\Gamma_t^{(i)}\boldsymbol{e}_{t|t}^{(i)}\Big] \tag{10}$$

of (9) for every $1 \le i \le N$. Note that T in (10) is the control horizon. At any time t, the scheduler decides which M among the N subsystems may communicate. Note that $\boldsymbol{e}_{t|t}^{(i)} = 0$ when a communication channel is assigned to subsystem i at time t.

In the following section, we present a deep reinforcement learning algorithm for control-aware scheduling called DEEPCAS. DEEPCAS communicates only with the smart sensors. At every time instant, sensors are told if they can transmit to their associated controllers. Then, the sensors provide feedback on the scheduling decision for that stage. Note that we do not consider the overhead involved in providing feedback.

III. DEEP REINFORCEMENT LEARNING FOR CONTROL-AWARE SCHEDULING

As stated earlier, at the heart of DEEPCAS lies the DQN. The DQN is a modern variant of Q-learning that effectively counters Bellman's curse of dimensionality. Essentially, DQN or Q-learning finds a solution to an associated Markov decision process (MDP) in an iterative model-free manner. Before proceeding, let us recall the definition of an MDP. For a more detailed exposition, the reader is referred to [14]. An MDP, $\mathcal{M}$, is given by the following tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where

- $\mathcal{S}$ is the state-space of $\mathcal{M}$;
- $\mathcal{A}$ is the set of actions that can be taken;
- P is the transition probability, i.e., $P(s, s'; a)$ is the probability of transitioning to state s' when action a is taken at state s;
- r is the one stage reward function, i.e., $r(s, a)$ is the reward when action a is taken at state s;
- γ is the discount factor with $\gamma \in [0, 1]$.

Below, we state the MDP $\mathcal{M}_d$ associated with our problem.

- $\mathcal{S}$: The state space $\mathcal{S}$ consists of all possible *augmented error vectors*. Hence, the state vector s_t at time t is given by $(\boldsymbol{e}_{t|t}^{(1)}, \ldots, \boldsymbol{e}_{t|t}^{(N)})$.
- $\mathcal{A}$: Action space is given by the M-size subsets of the channels: $\{\mathcal{S} \mid \mathcal{S} \subset \{1, 2, \ldots, N\},\ |\mathcal{S}| = M\}$. Hence, the cardinality of the action space is given by $|\mathcal{A}| = \binom{N}{M}$.

r: At time t, the reward $r(t)$ is given by $-\sum_{i=1}^{N} \boldsymbol{e}_{t|t}^{(i)} \Gamma_t^{(i)} \boldsymbol{e}_{t|t}^{(i)}$.

γ: Although it would seem natural to use $\gamma = 1$, we use $0 < \gamma < 1$ since it hastens the rate of convergence.

Note that the scheduler (DEEPCAS) takes action just before time t and receives rewards just after time t, based on transmissions at time t. Also, note that DEEPCAS only gets non-zero rewards from non-transmitting sensors. DEEPCAS is model-free. Hence, it does not need to know transition probabilities.

Let us suppose we use a reinforcement learning algorithm, such as Q-learning, to solve $\mathcal{M}_d$. Since the learning algorithm will find policies that minimize the future expected cumulative rewards, we expect to find policies that minimize scheduling effects on the entire system. This is a consequence of our above definition of reward r. Below, we provide a brief overview of Q-learning and DQN, the reinforcement learning algorithm at the heart of DEEPCAS. *Simply put,* DEEPCAS *is a DQN solving the above defined MDP* $\mathcal{M}_d$.

DEEPCAS. At any time t_0, the scheduler is interested in maximizing the following expected discounted future reward:

$$R(t_0) := \mathbb{E}\left[\sum_{t=t_0}^{T-1} \gamma^{t-t_0} r(t)\right].$$

Recall that $r(t)$ is the single stage cost given by $-\sum_{i=1}^{N} \boldsymbol{e}_{t|t}^{(i)} \Gamma_t^{(i)} \boldsymbol{e}_{t|t}^{(i)}$. Q-learning is a useful methodology to solve such problems. It is based on finding the following Q-factor for every state-action pair:

$$Q^*(s,a) := \max_{\pi} \mathbb{E}\left[R_t \mid s_t = s, a_t = a, \pi\right],$$

where π is a policy that maps states to actions. The algorithm itself is based on the Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \mid s,a].$$

Note that DEEPCAS has no knowledge of networked control system dynamics. This unknown dynamics is represented by $\mathcal{E}$, in the above equation. Since our state space is continuous, we use a deep neural network (DNN) for function approximation. Specifically, we try to find good approximations of the Q-factors iteratively. In other words, the neural network takes as input state s and outputs $Q(s,a,\theta)$ for every possible action a, such that $Q(s,a,\theta) \approx Q^*(s,a)$. This deep function approximator, with weights θ, is referred to as a Deep Q-Network. The Deep Q-Network is trained by minimizing a time-varying sequence of loss functions $L_t(\theta_t)$ given by

$$L_t(\theta_t) = \left(1/2\right) \mathbb{E}_{s,a \sim \rho(s,a)} \left[(Q(s,a,\theta_t) - y_t)^2\right],$$

where $y_t := \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a',\theta_{t-1}) \mid s,a\right]$ is the expected cost-to-go based on the latest update of the weights; ρ is the behavior distribution [13]. Training the neural network involves finding θ^*, which minimizes the loss functions. Since the algorithm is run online, training is done in conjunction with scheduling. At time t, after feedback (reward) is received, one gradient descent step can be performed using the following gradient term:

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}_{s,a \sim \rho(\cdot); s \sim \mathcal{E}}\Big[\Big(Q(s,a;\theta_t) - r - \gamma \max_{a'} Q(s',a',\theta_{t-1})\Big) \nabla_{\theta_t} Q(s,a;\theta_t)\Big]. \quad (11)$$

To make the algorithm implementable, we update the weights θ_t using samples than finding the above expectation exactly. At each time, we pick actions using the ϵ-greedy approach [13]. Specifically, we pick a random action with probability ϵ, and we pick a greedy action with probability $1-\epsilon$. This ϵ-greedy approach for picking actions induces the behavior distribution ρ. In other words, the actions at every stage are picked using distribution ρ. Note that a greedy action a_t at time t is one that maximizes $Q(s_t,a;\theta)$. Initially it is desirable to *explore*, hence ϵ is set to 1. Once the algorithm has gained some experience, it is better to *exploit* this experience. To accomplish this, we use an attenuating ϵ to 0.

Although we train our DNN in an online manner, we do not perform a gradient descent step using (11), since it can lead to poor learning. Instead, we store the previous K experiences (s_t, a_t, r_t, s_{t+1}), $t_0 - K + 1 \le t \le t_0$, in an *experience replay memory* $\mathcal{D}$. When it comes to training the neural network at time t, it performs a single mini-batch gradient descent step. The mini-batch (of gradients) is randomly sampled from the aforementioned experience replay $\mathcal{D}$. The idea of using experience replay memory, to overcome biases and to have a stabilizing effect on algorithms, was introduced in [13].

DQN for control-aware scheduling

1: Initialize the replay memory $\mathcal{D}$ to capacity K.
2: Initialize the weights, θ, of the Q-Network.
3: **for** the entire duration **do**
4: With probability ϵ select a random action a_t.
5: With probability $1-\epsilon$ pick a_t that maximizes $Q(s_t,a,\theta)$.
6: Execute action a_t to obtain reward r_t and observe s_{t+1}.
7: Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$.
8: Sample random mini-batch transitions $((s_j, a_j, r_j, s_{j+1}))$ from $\mathcal{D}$.
9: Corresponding to (s_j, a_j, r_j, s_{j+1}), set

$$y_j := r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta).$$

10: Perform a gradient descent step with loss given by $(y_j - Q(s_j, a_j; \theta))^2$.

IV. EXPERIMENTAL RESULTS

Recall that DQN is at the heart of our DEEPCAS, which uses a deep neural network to approximate Q-factors. The input to this neural network is the appended error vector. The hidden layer consists of 1024 rectifier units. The output layer is a fully connected linear layer with a single output for each of the $\binom{N}{M}$ actions. The discount factor γ in our Q-learning algorithm is fixed at 0.95. The size of the experience

replay buffer is fixed at $20,000$. The exploration parameter ϵ is initialized to 1, then attenuated to 0.001 at the rate of 0.9. For training the neural network, we use the optimizer ADAM [15] with a learning rate of e^{-4} and a decay of 0.001. The control horizon is set to $T = 500$. Note that we used the same set of parameters for all of the experiments presented below.

We conducted three sets of experiments. For the first two sets, we used the reward described in Section III. For the last experiment, we used the total control cost as the reward. The reader is referred to (9) *in Section II-D for the control cost associated with subsystem i. Using the full control cost as the reward allows us to discuss the stability of the networked control system, see Section V for details.*

A. Experiment 1 (*N*=3, *M*=1, and *T*=500)

For our first experiment, we used DEEPCAS to schedule one channel for three subsystems. We considered three second-order single-input-single-output (SISO) subsystems consisting of one stable (subsystem 2) and two unstable subsystems (subsystems 1 and 3). If there were three channels, then there would be no scheduling problem and the total optimal control loss J would be 13.8487. Since there is only a single channel available, one expects a solution to the scheduling problem to allocate it to subsystems 1 and 3 for a more substantial fraction of the time, as compared to subsystem 2. This expectation is fair since subsystems 1 and 3 are unstable while subsystem 2 is stable. *Once trained, on an average* DEEPCAS *indeed allocates the channel to subsystem 1 for 52% of the time, to subsystem 2 for 12% of the time, and to subsystem 3 for 36% of the time.*

We train DEEPCAS continuously over many epochs. Each epoch corresponds to a single run of the control problem with horizon 500. At the start of each epoch, the initial conditions for the control problem are chosen as explained in § II. The *black-curve* in Fig. 2 illustrates the learning progress in *Experiment 1*. The abscissa axis of the graph represents the epoch number while the ordinate axis represents the average control loss. The plot is obtained by taking the mean of 30 Monte Carlo runs. Since DQN is randomly initialized, scheduling decisions are poor at the beginning, and the average control loss is high. As learning proceeds, the decisions taken improve. After only 10 epochs, DEEPCAS converges to a scheduling strategy with an associated control loss of around 21.

Traditionally, the problem of scheduling for control systems is solved by using control theoretic heuristics to find periodic schedules. For *Experiment 1*, we exhaustively searched the space of all periodic schedules, with periods ranging from 2 to 11. Using this strategy, we were able to acheive a minimum possible control loss of $J = 22.8112$. In comparison, DEEPCAS finds a scheduling strategy with an associated control loss of 21.15. *In addition to being faster,* DEEPCAS *does not need any system specification and can schedule efficiently for very long control horizons.*

B. Experiment 2 (*N*=6, *M*=3, and *T*=500)

For our second experiment, we train DEEPCAS to schedule three channels for a system with six second-order SISO subsystems. If $N = M$, then the total control loss would be 18.234. As before, learning is done continuously over many epochs. The *red-curve* in Fig. 2 illustrates the learning progress of DEEPCAS in scheduling three channels among six subsystems. The abscissa and ordinate axes are as before. As evidenced in the figure, DEEPCAS quickly finds schedules with an associated control loss of around 20.

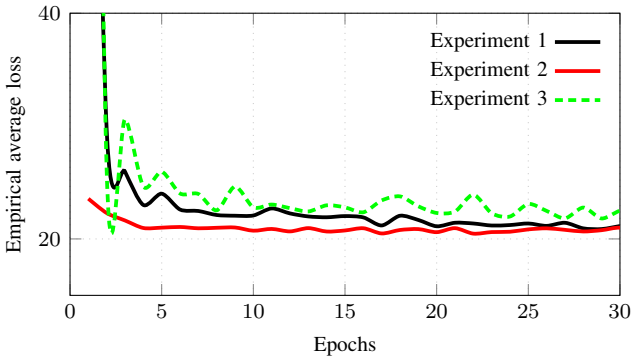

Fig. 2. Convergence of the empirical average control loss.

We are unable to compare the results of *Experiment 2* with any optimal periodic schedules. This is because optimal periodic scheduling strategies do not extend to the system size and control horizon considered here. Further, performing an exhaustive search for finding periodic schedules is not possible since the number of possibilities are in the order of $\binom{6}{3}^n = 20^n$, where n is the period-length.

C. Experiment 3 (same set-up as Experiment 1 but with $-J$ as reward)

The systems considered hitherto have independent subsystems. This facilitates the splitting of the total control cost into two components; see (9). The one-stage reward in our algorithm is the negative of the error due to lack of communication defined in (10). However, in general multi-agent settings, the previously mentioned splitting may not be possible. To show that our results are readily extensible to more general settings, we repeated *Experiments* 1 and 2 with *negative of the one-stage control cost* as the reward. The results of the modified experiments are very similar to the original ones. The learning progress of the modified *Experiment 1*, with full cost, is given by the *green-curve* in Fig. 2.

V. STABILITY ISSUES

In our framework, the controller and scheduler run in tandem. The control policy, π_c, is fixed before the scheduler is trained. As a consequence of training, the scheduler finds a scheduling policy π_s. Thus, the controller-scheduler pair finds a policy tuple (π_c, π_s). To investigate the stabilizing properties of DEEPCAS, we make the following mild assumptions on this policy tuple.

A1 $\liminf_{n\to\infty} \frac{1}{n}\sum_{t=0}^{n} J(t) = \limsup_{n\to\infty} \frac{1}{n}\sum_{t=0}^{n} J(t)$, where $J(t) := \sum_{i=1}^{N} J^{(i)}(t)$ is the single-stage control loss and

$$J^{(i)}(t) = \bar{\boldsymbol{x}}_0^{(i)\top} S_0^{(i)} \bar{\boldsymbol{x}}_0^{(i)} + \mathrm{Tr}\big(S_0^{(i)} X_0^{(i)}\big) + \mathrm{Tr}\big(S_{t+1}^{(i)} W^{(i)}\big) + \mathrm{Tr}\big(P_{t|t}^{(i)s} \Gamma_t^{(i)}\big) + \mathbf{E}\Big[\boldsymbol{e}_{t|t}^{(i)\top} \Gamma_t^{(i)} \boldsymbol{e}_{t|t}^{(i)}\Big]$$

is the single stage loss of subsystem i at time t. In other words, we assume that the limit of the average cost sequence exists. This limit may be infinite, i.e., $\lim_{n\to\infty} \frac{1}{n}\sum_{t=0}^{n} J(t) < \infty$ or $= \infty$.

A2 The discount factor γ used for training is such that $\liminf_{\alpha\uparrow 1}(1-\alpha)\sum_{t=0}^{\infty} \alpha^t J(t) \le \sum_{t=0}^{\infty} \gamma^t J(t) + M_0$, for some $0 < M_0 < \infty$. Again, it could be that $\sum_{t=0}^{\infty} \gamma^t J(t) = \infty$. In which case, **(A2)** is trivially satisfied.

In our framework, the controller uses a control policy, π_c, that solves the average cost control problem. The scheduler learns a scheduling policy, π_s, to solve the discounted cost problem. Since they run in tandem, the control loss value $J(t)$, at any time t, depends on both the control and scheduling actions taken at time t. *Further, we have empirically observed that our scheduler can be successfully trained for all discount factors γ close to* 1. Before proceeding, consider the following theorem due to Abel:

Theorem 1 (Abel, [16]) *Let $\{c_t\}_{t\ge 0}$ be a sequence of positive real numbers, then*

$$\liminf_{n\to\infty} \frac{1}{n}\sum c_t \le \liminf_{\alpha\uparrow 1}(1-\alpha)\sum_{t=0}^{\infty} \alpha^t c_t.$$

It follows from **(A1)** and Abel's theorem that

$$\lim_{n\to\infty} \frac{1}{n}\sum J(t) \le \liminf_{\alpha\uparrow 1}(1-\alpha)\sum_{t=0}^{\infty} \alpha^t J(t)\ .$$

Recall that our scheduler can be successfully trained to solve the discounted cost problem for all discount factors close to (but not equal to) 1. In other words, given a discount factor $\gamma \approx 1$, the scheduler finds a policy $\pi_s(\alpha)$ such that

$$\sum_{t=0}^{\infty} \gamma^t J(t) < \infty.$$

If we couple this observation with **(A2)**, we get:

$$\liminf_{\alpha\uparrow 1}(1-\alpha)\sum_{t=0}^{\infty} \alpha^t J(t) \le \sum_{t=0}^{\infty} \gamma^t J(t) + M(\gamma) < \infty,$$

for some $\gamma \approx 1$ and $M(\gamma) > 0$. If we choose γ as the discount factor for our training algorithm, it follows that:

$$\lim_{n\to\infty} \frac{1}{n}\sum J(t) \le \liminf_{\alpha\uparrow 1}(1-\alpha)\sum_{t=0}^{\infty} \alpha^t J(t) \le \sum_{t=0}^{\infty} \gamma^t J(t) + M(\gamma) < \infty.$$

We claim that system stability follows from this set of inequalities. To see this, observe that $\sum_{i=1}^{N} \|\boldsymbol{x}_t^{(i)}\|_Q \le J(t)$. Hence, $\limsup_{n\to\infty} \frac{1}{n}\sum_{t=0}^{n}\sum_{i=1}^{N} \|\boldsymbol{x}_t^{(i)}\|_Q \le \lim_{n\to\infty} \frac{1}{n}\sum_{t=0}^{n} J(t) < \infty$. In other words, the following claim is immediate.

Claim 1 *Under **(A1)** and **(A2)**, the scheduling algorithm can be successfully trained for discount factors close to* 1*, consequently $\sum_{t=0}^{\infty} \gamma^t J(t) < \infty$. Further, the policy π_s thus found, stabilizes the system, i.e., $\sup_{t\ge 0} \sum_{i=1}^{N} \|\boldsymbol{x}_t^{(i)}\|_Q < \infty$.*

VI. Conclusions

This paper considered the problem of scheduling the sensor-to-controller communication in a networked control system, consisting of multiple independent subsystems. To this end, we presented DeepCAS, a reinforcement learning-based control-aware scheduling algorithm. This algorithm is model-free and scalable, and it outperforms scheduling heuristics, such as periodic schedules, tailored for feedback control applications.

References

[1] P. Park, S. C. Ergen, C. Fischione, C. Lu, and K. H. Johansson, "Wireless network design for control systems: a survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 978 – 1013, Secondquarter 2018.

[2] H. Rehbinder and M. Sanfridson, "Scheduling of a limited communication channel for optimal control," *Automatica*, vol. 40, no. 3, pp. 491–500, March 2004.

[3] D. Hristu-Varsakelis and L. Zhang, "LQG control of networked control systems," *International Journal of Control*, vol. 81, no. 8, pp. 1266–1280, 2008.

[4] L. Shi, P. Cheng, and J. Chen, "Optimal periodic sensor scheduling with limited resources," *IEEE Transactions on Automatic Control*, vol. 56, no. 9, pp. 2190–2195, 2011.

[5] L. Orihuela, A. Barreiro, F. Gómez-Estern, and F. R. Rubio, "Periodicity of Kalman-based scheduled filters," *IEEE Transactions on Automatic Control*, vol. 50, no. 10, pp. 2672–2676, 2014.

[6] M. Zanon, T. Charalambous, H. Wymeersch, and P. Falcone, "Optimal scheduling of downlink communication for a multi-agent system with a central observation post," *IEEE Control Systems Letters*, vol. 2, no. 1, pp. 37–42, Jan. 2018.

[7] T. Charalambous, A. Ozcelikkale, M. Zanon, P. Falcone, and H. Wymeersch, "On the resource allocation problem in wireless networked control systems," in *Proceedings of the 57^{th} IEEE Conference on on Decision and Control*, 2017.

[8] W. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *Proceedings of the 51^{st} IEEE Conference on Decision and Control*, Dec. 2012.

[9] C. Ramesh, H. Sandberg, and K. H. Johansson, "Design of state-based schedulers for a network of control loops," *IEEE Transactions on Automatic Control*, vol. 58, no. 8, pp. 1962–1975, Aug. 2013.

[10] A. Molin and S. Hirche, "Price-based adaptive scheduling in multi-loop control systems with resource constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3282–3295, Dec. 2014.

[11] E. Henriksson, D. E. Quevedo, H. Sandberg, and K. H. Johansson, "Multiple loop self-triggered model predictive control for network scheduling and control," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 6, pp. 2167–2181, 2015.

[12] B. Demirel, A. S. Leong, V. Gupta, and D. E. Quevedo, "Trade-offs in stochastic event-triggered control," *arXiv:1708.02756*, 2017.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[14] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceeding of the 3^{rd} International Conference for Learning Representations*, 2015.

[16] O. Hernandez-Lerma and J. B. Lasserre., *Discrete-time Markov control processes: basic optimality criteria*. Springer Science & Business Media, 2012, vol. 30.