

Emulation Learning for Neuromimetic Systems

Zexin Sun & John Baillieul

Abstract—Building on our recent research on neural heuristic quantization systems, results on learning quantized motions and resilience to channel dropouts are reported. We propose a general emulation problem consistent with the neuromimetic paradigm. This optimal quantization problem can be solved by model predictive control (MPC), but because the optimization step involves integer programming, the approach suffers from combinatorial complexity when the number of input channels becomes large. Even if we collect data to train a neural network, collection of training data and the training itself are still time-consuming. Therefore, we propose a general Deep Q Network (DQN) algorithm that not only learns trajectories but also exhibits the advantages of resilience to channel dropout. Furthermore, to transfer the model to other emulation problems, a mapping-based transfer learning approach can be used directly on the current model to obtain the optimal trajectory steps for the new emulation problems.

I. INTRODUCTION

The work being reported continues our effort to develop the theoretical foundations of control system designs that exhibit key features of the neural mechanisms that govern movement and other behaviors in animals. One such feature is control modulation involving actions of very large numbers of simple inputs and outputs that are effective in influencing the system dynamics only in their aggregate operation. The goal of research on such control systems is to understand the engineering analogs of neuroplasticity, learning and re-learning, memory and adaptation. Previously reported work has introduced what we call *neuromimetic* linear models. The focus has been on finite dimensional linear systems that are *overcomplete* in that they have many more input and output channels than the dimension of the state [1],[6]. Such systems have been shown to have reduced cost of operation in terms of standard L_2 metrics, reduced uncertainty in the face of input channel noise, and resilience with respect to drop-outs of input or output channels, [9]. With such qualities in mind, research is now aimed at overcomplete models with simple neuron-like discrete inputs taking values $\{-1, 0, 1\}$ and exploring how well these are able to emulate the behaviors of systems with standard continuous feedback designs. In [10], algorithms based on ideas from machine learning were used to solve a restricted emulation problem. The solutions were useful in comparing optimal and nearby suboptimal designs as well as in suggesting approaches

to understanding the complexity of neuromimetic feedback designs.

One of the approaches to feedback control using the quantized control inputs of [10] involved reinforcement learning (RL)—specifically, a DQN-like algorithm [13],[14]. RL can solve continuous decision-making tasks remarkably effectively. Agents can improve their performance by interacting with the environment through trial-and-error to minimize an emulation error metric [2]. However, when the state and action sets become large or possibly infinite, tabular solution methods must be abandoned in favor of approaches that lead only to approximate solutions. The approach here, following basic ideas from [13], is to replace the Q-table of basic RL with a deep Q neural network (DQN). Revisiting [5] in what follows, we examine the inherent complexity arising from the set of state-action pairs being infinite and the decision space being discrete. Even in this somewhat simple case, obtaining a sufficient sample of interactions as the emulation problem under study changes remains challenging since it is a time-sensitive problem, [7]. For this reason, we consider transfer learning (TL) [3] as a way to utilize the experience from other emulation problems to accelerate the learning process for the new problem, [8].

The rest of the paper is organized as follows. Section II describes the emulation problems, our system models, and the design goals of the learning algorithm. Section III introduces two approaches: an MPC-data based learning method and a generalized deep Q-network (DQN). Section IV presents the transfer learning algorithm to adapt the learned model a new emulation problem. Section V shows simulation results that illustrate the proposed approaches, and we make a summary in Section VI.

II. PROBLEM DESCRIPTION

Consider linear time-invariant (LTI) systems of the form

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), & x \in \mathbb{R}^n, & u \in \mathbb{R}^m, \text{ and} \\ y(t) &= Cx(t), & y \in \mathbb{R}^q. \end{aligned} \quad (1)$$

By applying the simple feedback control law $u = Kx$, where K is a stabilizing gain chosen as in [10], we obtain the closed-loop LTI system

$$\dot{x}(t) = Hx(t), \quad x(0) = x_0, \quad (2)$$

where $H = A + BK \in \mathbb{R}^{n \times n}$ is Hurwitz. Following [9], [10], we consider the problem of emulating (2) by means of a discrete-time system with quantized inputs

$$x_{qs}(k+1) = e^{Ah}x_{qs}(k) + \int_0^h e^{A(h-s)}Bu(k)ds, \quad (3)$$

Zexin Sun is with the Division of Systems Engineering at Boston University. John Baillieul is with the Departments of Mechanical Engineering, Electrical and Computer Engineering, and the Division of Systems Engineering at Boston University, Boston, MA 02215. The authors may be reached at {zxsun, johnb}@bu.edu.

Support from various sources including the Office of Naval Research grant number N00014-19-1-2571 is gratefully acknowledged.

where $x_{qs} \in \mathbb{R}^n$ is the system state, $u \in \mathbb{U} = \{-1, 0, 1\}^m$ is the set of possible quantized inputs. Following [10], we refer to these inputs $u(k)$ as activation patterns. h is the time step and $m \gg n$ denotes the large number of input channels. Here we provide an example.

Example 1: Suppose the $n = 2, m = 4$ and that

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Then the update law (3) is written as

$$x_{qs}(k+1) = x_{qs}(k) + hBu(k). \quad (4)$$

Taking the 81 activation patterns $\{-1, 0, 1\}^4$ as inputs, we see that at each time step, (3) can move in any of the 25 directions (including a zero vector) depicted in Fig. 1.

Consider solutions to the linear ordinary differential equation (ODE) (2). The goal of the general emulation problem is to find piecewise constant quantized inputs with sampling interval $h > 0$ such that the resulting trajectories of (3) with initial state $x_{qs}(0) = x_0$ approximate the continuous system (2). Here, we solve the emulation problem that finds a partition of the state space $\{U_i : \cup U_i = \mathbb{R}^n; U_i^o \cap U_j^o = \emptyset; U_i^o = \text{interior } U_i\}$ and a selection rule depending on the current state $x_{qs}(k)$ for assigning values of the input at the k -th time step to be $u(k) \in \mathcal{U} = \{-1, 0, 1\}^m$, so that for each $x_{qs} \in U_i$, $e^{Ah}x_{qs}(k) + \int_0^h e^{A(h-s)}dsBu(k)$ is as close as possible to the emulated LTI system (2). We have defined various metrics in terms of which we determine the fitness of an approximation such as the direction and magnitude between two systems as was studied in [9]. When m is large (B is $n \times m$ with $m \gg n$), it will frequently be the case that several activation patterns have comparable fitness. Because two or more activation patterns may give approximately equal quality of emulation, it may be the case that over the course of a trajectory, we need to consider multi-step fitness. This is illustrated in the Fig.1, where we see that the red approximants have the best first vector step matching the vectorfield (black vector), but in considering pairs of steps, the blue pair of vectors end up closer to the black pair than the red. Hence, when applying algorithms to learn to optimally emulate a given system, we need to focus on methods that consider multiple steps, and we need to be aware that there may be multiple solutions of approximately equal value.

Our goal is to design algorithms that can meet the following objectives:

- The quantized system (3) can emulate the continuous-time LTI system (2), i.e., at any given point, the model generated by the algorithm can compute an optimal quantized direction taking into account multiple steps.
- The emulation task can still be achieved when there are channel dropouts in the quantized system during the emulating process.
- When the emulated LTI system changes, the model still works well under new circumstances directly without the need of retraining.

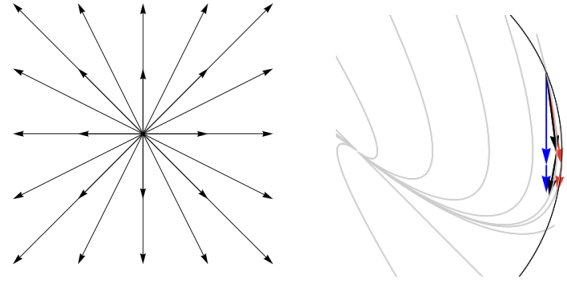


Fig. 1. The figure on the left is an example of a set of quantized directions (as considered in [10] and related to the 81 activation patterns of Example 1), while the figure on the right shows choices of vector pairs (red vs blue) chosen for emulating a LTI vectorfield (gray curves and black tangents) when considering multiple steps.

III. LEARNING METHODS FOR EMULATION PROBLEMS

For emulation learning of trajectories of a given LTI system, we propose two approaches: an MPC data-based supervised learning approach that trains a neural network using training data collected from solving the MPC optimization problem and a generalized DQN algorithm by exploring the environment. We shall discuss the advantages of these approaches in terms of the optimality of trajectory, computational efficiency and resilience to channel dropouts.

A. MPC Data-based Supervised Learning

Many works utilize MPC with neural networks in control problems, including [11], [12]. However, the notable difference from what we are proposing is that the basic model formulated for our neuromimetic quantized control systems is far more complex, and the solutions cannot be expressed as affine functions as was done in [12]. Despite this, the potential savings in real-time implementations of neural network solutions are of considerable significance in our quantized inputs. Results obtained from neural networks to realize the trajectories are more substantial than when problems involve continuous control variables.

As described in [19], the Model Predictive Control (MPC) approach can be formulated as solving the following optimization problem:

$$\begin{aligned} \min_{u_{0|k}, \dots, u_{N-1|k}} J &= |u_{n|k}|_R^2 + |x_{N|k} - x_{ref}(N|k)|_P^2 \\ &+ \sum_{n=0}^{N-1} |x_{n|k} - x_{ref}(n|k)|_\Gamma^2 \\ \text{s.t.} \quad u_{n|k} &\in \{-1, 0, 1\}^m, \\ x_{n+1|k} &= Ax_{n|k} + Bu_{n|k}, \\ x_{ref}(n+1|k) &= e^{Hh}x_{ref}(n|k), \\ &\forall n = 0, 1, \dots, N-1, \quad (5) \end{aligned}$$

where P, Γ are positive semi-definite matrices, R is positive definite and the function $|x|_P^2 = x^T Px$, with similar definitions for Γ and R . The reference values x_{ref} are sampled values of trajectories of the linear time-invariant system (2).

By solving the optimization problem (5), we obtain a sequence of activation patterns generating a sequence of p.w.-constant vector directions such that (3) tracks the reference generated by (2). Unfortunately, this approach involves integer programming, which is a computational challenge, especially since the dimension m is large in our case.

We approach the problem by means of a neural network with training data comprised of many solutions that have been obtained off-line. The idea is that at run time the trained model will directly select appropriate activation patterns at each time step, without the need to resolve (5). Once the model is trained, it can steer (3) to emulate (2) in a way that is optimal or nearly optimal in terms of (5). Unfortunately, collecting sufficient training samples remains prohibitive. Nevertheless, the kinds of resilience encountered in previously studied overcomplete systems, [9], [10] are seen to be present. From the simulation experiments in Section V below, we find that even when there were channel dropouts (simulating neuronal damage), the systems were able to choose new quantized trajectories emulating the desired asymptotically stable motion.

In an attempt to find an approach that is both more computationally tractable and more similar to learning mechanisms of neurobiology, we next propose a deep Q-learning method that works well in the presence of channel dropouts at both the training and final execution stages.

B. Generalized DQN-like Algorithm

Reinforcement learning using deep neural networks has been proven successful from experiments in various areas recently [22]. The basic idea of RL is that an agent and an environment interact continuously. The agent receives a state (s_t) at step t from the environment and chooses an action (a_t), then the environment reacts to this action leading to a new state (s_{t+1}) for the agent along with a reward (r_t) that reflects the value of the action taken. Q-learning is one of the most efficient strategies for carrying out this type of learning [23]. It operates using a Q function that stores state-action pairs and maps them to Q values defined by

$$Q_\pi(s_t, a_t) = \mathbb{E}[r_t + \sum_{i=1}^{T-t} \gamma^i r_{t+i}], \quad (6)$$

where $\gamma < 1$ is a discount factor and π is the policy for choosing actions in terms of states. Since in what follows we only consider the deterministic case, the notation $\mathbb{E}[\cdot]$ can be ignored. The goal is to find an optimal policy π^* that maximizes Q_π . The optimal Q function satisfies the Bellman's equation:

$$Q^*(s_t, a_t) = r_t + \gamma \max_{a'} Q^*(s_{t+1}, a'). \quad (7)$$

An iterative update of the Q function is given by

$$loss = (r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))^2. \quad (8)$$

Iterating updates using this state-action pair expression is only practical when the number of these pairs is finite and not too large. When the state and action space have large

numbers of elements or perhaps take on continuous values, it becomes infeasible to construct a Q table. Therefore, a neural network with learnable parameters ϕ is used to approximate the complex nonlinear Q function. The learning rule for ϕ with learning rate α is constructed as

$$\phi_{i+1} = \phi_i - \alpha \nabla_{\phi_i} (r_t + \gamma \max_{a'} Q_{\phi_i}(s_{t+1}, a') - Q_{\phi_i}(s_t, a_t))^2. \quad (9)$$

It is well-known that direct iteration of (9) may not converge since the term $\max_{a'} Q_{\phi_i}(s_{t+1}, a')$ also depends on ϕ_i , [21]. The deep Q network (DQN) algorithm solves this problem by introducing another neural network called the *target network* with parameters $\hat{\phi}$ to predict the target Q value. It can stabilize the learning by replacing the term $\max_{a'} Q_{\phi}(s_{t+1}, a')$ with $\max_{a'} Q_{\hat{\phi}}(s_{t+1}, a')$. Therefore, when it learns parameters ϕ , only the part $Q_{\phi}(s_t, a_t)$ changes, keeping the target function ($r_t + \gamma \max_{a'} Q_{\hat{\phi}}(s_{t+1}, a')$) fixed to avoid oscillations. Therefore, the loss function of the DQN algorithm is

$$l_{DQN}(\phi; \hat{\phi}) = (r_t + \gamma \max_{a'} Q_{\hat{\phi}}(s_{t+1}, a') - Q_{\phi}(s_t, a_t))^2. \quad (10)$$

For every C steps, the parameters ϕ from the prediction network Q_{ϕ} are copied to the target network $Q_{\hat{\phi}}$.

The efficiency of the DQN algorithm has been illustrated in several experiments, such as playing Atari games [13] and planning vehicle routing [18]. Inspired by these successes, we proposed a DQN-like algorithm to solve the restricted emulation problem in [10], which focused on one-step optimization of quantized system motion starting from points on the unit sphere. In what follows, we provide an extension by proposing a generalized DQN-like Algorithm 1 for the quantized system to learn the whole emulation trajectory for any linear dynamic system. M, T, C in this algorithm are the hyperparameters, which denote the total number of episodes, iterations and steps. To guarantee this emulation problem is a Markov Decision Process (MDP) [4], the state s_t needs to be designed carefully so as to incorporate information regarding states of both the emulating and emulated systems as well as the time. Therefore, $s_t \in \mathbb{R}^{2n}$ in this paper contains two parts: the error vector between two systems, i.e., $x(t) - x_{qs}(t)$ and the state of the emulated LTI system $x(t)$. The action space is a set of quantized directions (Dir) so that we use d_t (as illustrated in Fig. 1) instead of a_t . Therefore, in each transition, when s_t and the action d_t are given, we can determine the next state s_{t+1} . The reward r_t is a function involving the L_2 norm of the error, which is also determined. The quantized direction is obtained by following an ϵ -greedy policy, which is

$$d_t = \begin{cases} \text{choose } d \in Dir \text{ randomly,} & p = \epsilon \\ \arg \max_d Q_{\phi}^*(s_t, d), & p = 1 - \epsilon \end{cases} \quad (11)$$

Since the convergence of this DQN algorithm has only been demonstrated by experiments and is known sometimes diverge [16], [17], a new loss function from [21] is utilized:

$$Loss(\phi; \hat{\phi}) = \max\{l_{DQN}(\phi; \hat{\phi}), l_{MSBE}(\phi)\}, \quad (12)$$

where

$$l_{MSBE}(\phi) = (r_t + \gamma \max_{d'} Q_\phi(s_{t+1}, d') - Q_\phi(s_t, d_t))^2 \quad (13)$$

denotes the mean squared Bellman error. By using this loss function (12) and since $\hat{\phi}_{i+1} = \arg \min_{\phi} \text{Loss}(\phi; \hat{\phi}_i)$, the non-increasing relation

$$\begin{aligned} \min_{\phi} \text{Loss}(\phi; \hat{\phi}_{i+1}) &\leq \text{Loss}(\hat{\phi}_{i+1}; \hat{\phi}_{i+1}) = l_{MSBE}(\hat{\phi}_{i+1}) \\ &\leq \text{Loss}(\hat{\phi}_{i+1}; \hat{\phi}_i) = \min_{\phi} \text{Loss}(\phi; \hat{\phi}_i) \end{aligned} \quad (14)$$

is obtained, which means the iteration $\hat{\phi} \leftarrow \arg \min_{\phi} \text{Loss}(\phi; \hat{\phi})$ is bounded and non-increasing. Numerical experiments in Section V show that the proposed algorithm is convergent in the sense that the loss is monotonically decreasing up to a limiting value, after which it is no longer guaranteed to decrease.

Algorithm 1 Learning optimal path to emulate dynamic systems

- 1: Input: Activation patterns $U = \{u_1, u_2, \dots, u_K\}$ and its direction vectors of quantization output alphabet to form an action space $Dir = \{d_1, d_2, \dots, d_{K'}\}$, a learning metric G ;
 - 2: Initialize replay memory D with capacity N ;
 - 3: Initialize action-value Q function with parameter ϕ and target- Q function with parameter $\hat{\phi} = \phi$;
 - 4: **for** episode= 1, M **do**
 - 5: Start from the initial state $s_0 = \{e(0), x_{ref}(0)\}$, which contains the initial error vector between emulated and quantized systems and the location of the reference system;
 - 6: **for** $t= 1, T$ **do**
 - 7: Observe two systems and record the error vector and location of emulated system as $s_t = \{e(t), x_{ref}(t)\}$. Choose direction $d_t = \pi^e(s_t)$, get reward $r_t = -G(s_t)$ and new state $s_{t+1} = \{e(t+1), x_{ref}(t+1)\}$;
 - 8: Store (s_t, d_t, r_t, s_{t+1}) as a memory cue to D ;
 - 9: Sample random minibatch of cues $(s_j, d_j, r_j, s_{j+1}), j \in [0, t]$ from D ;
 - 10: Apply the loss function $\text{Loss}(\phi; \hat{\phi}) = \max\{l_{DQN}(\phi; \hat{\phi}), l_{MSBE}(\phi)\}$ to train Q network and every C steps, $\hat{\phi} \leftarrow \phi$;
 - 11: **end for**
 - 12: **end for**
-

Remark 1: It is noted that Algorithm 1 can not only be used for the quantized system to learn the trajectory of LTI systems but also any given system. An example of using this algorithm to track a nonlinear system around equilibria will be considered elsewhere.

Remark 2: (On resilient learning) When using our deep Q network to learn the trajectories, channels can drop out at any time without causing problems. Since at each step, it chooses the optimal available action (direction) by (11).

When there channels have dropped out and the optimal quantized direction is unavailable, the policy can generate the direction with the optimal Q value from all available remaining candidates.

IV. MAPPING-BASED TRANSFER LEARNING

In addition to the ability to learn, neurobiological systems have the ability to generalize and adapt what they have learned to new but similar problem domains. To explore such *transfer learning* in the context of our emulation problems, suppose that a DQN has been trained for emulating a particular LTI system. Suppose another LTI system has the form

$$\dot{z} = H_o z, \quad (15)$$

and $H_o = OHO^{-1}$ where O is known and invertible. To emulate this LTI system, we utilize a mapping-based transfer learning method [15]. The coordinate is transformed by setting $z = Ox$, then the dynamic equations of these two LTI systems have the following relation:

$$\begin{aligned} \dot{z} &= H_o z = OHO^{-1}z = OHO^{-1}Ox = OHx, \\ \dot{z} &= O\dot{x} = OHx \end{aligned} \quad (16)$$

Assume we have already obtained a trained model from the MPC data-based or the DQN-like algorithm for the emulation problem of (2) using (3), say $F : \mathbb{R}^{2n} \rightarrow \mathbb{R}^n$, which is expressed by a neural network. $2n$ is the dimension of state s and n is the dimension of the quantized direction d . The model can predict the optimal quantized direction at any given state s . For example, $d = \arg \max_d Q_\phi^*(s, d)$ if the DQN model is used.

Instead of learning F_o by constructing another dataset by solving a new MPC problem or learning a new deep Q network—which are time-consuming, we try to obtain a selection policy F_o from F . The strategy will be to use mapping-based transfer learning, [14]. The main idea is to express the relationship of features. Therefore, F can be used directly by the following steps to obtain the learned policy F_o :

- Record the learning metric like the error vector between the new LTI system (15) and the quantized system (3) as feature $f_1 \in \mathbb{R}^n$, the direction (i.e., $[x(k+1) - x(k)]/h$) or the location of the LTI system (i.e., $x(k)$) as feature $f_2 \in \mathbb{R}^n$. Combine these features to be $f_o = [f_1; f_2] \in \mathbb{R}^{2n}$;
- Transform the coordinate of features f_o to the coordinate system of (2): $f = \begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} f_o$;
- Predict the optimal direction by $\vec{d} = F(f)$;
- Change back the predicted direction to the coordinate system of (15): $\vec{d}_o = O\vec{d}$. It is noted that \vec{d}_o may not be in the quantized direction set formed in the current coordinate;
- Find the nearest neighbor of \vec{d}_o in the direction set by kd-tree or Hebbian-Oja algorithm [10], and denote this by \vec{d}^* . The mapping rule that computes the nearest

neighbor is denoted by $M : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and Fig. 2 is an example.

- It is the (sub)optimal output direction for emulating the system (15).

Lemma 1: Following the above steps, the learned policy that gives \vec{d}_o is $\tilde{F}_o = OF(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} f_o)$ and the learned policy for the new emulation problem is $F_o = M(OF(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} f_o))$.

Proof: By applying the transformation steps on the given model, the formula for F_o is easily derived, and the details are omitted. ■

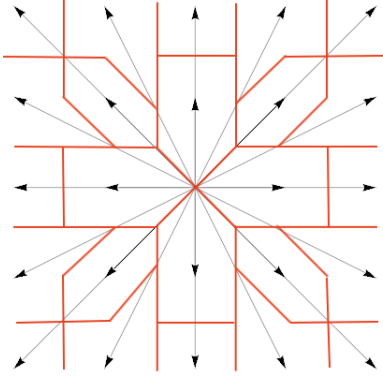


Fig. 2. An example of a mapping function that partitions the space according to the closest quantized direction (not including zero direction). The black vectors are the quantized directions formed by Bu corresponding to Fig. 1(a), and the red lines are the division boundaries between the cells U_i .

It can be observed that there is a special case when the transformation matrix O makes no change of the action space, which is the set of all combinations of Bu . In this case, Dir is the same as the set Dir_o formed by OBu . Therefore, $\vec{d}_o = \vec{d}^*$ and the last step to find the nearest neighbor is no longer needed. Section V provides a simulation of such a case.

Let π^* , Q_ϕ^* denote the optimal policy and optimal Q function for the original emulation problem, and π_o^* , $Q_{\phi_o}^*$ for the new one. According to the definition,

$$\begin{aligned} \pi^*(s) &= \arg \max_d Q_\phi^*(s, d), \\ \pi_o^*(s_o) &= \arg \max_{d_o} Q_{\phi_o}^*(s_o, d_o). \end{aligned} \quad (17)$$

Theorem 1: The optimal learning policy for this new emulation problem can be expressed as $\pi_o^*(s_o) = O\pi^*(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o)$, when the following conditions are satisfied:

- The direction space is invariant before and after the transformation.
- The first layer of the Q_ϕ network is linear and Q_{ϕ_o} has the same structure as Q_ϕ .
- The reward function for the new problem is designed to

be $r(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o)$, where $r(\cdot)$ is the reward function of original problem.

Proof: From the basic setup of Q-learning, we have

$$Q_\phi^*(s, d) = r(s) + \gamma \max_{d'} Q_\phi^*(s', d'),$$

where s' is the state after the system at state s taking direction d . Here we only consider deterministic policy so that s' is also deterministic. For the new problem, we first transfer the coordinate to the original one. Since state $s \in \mathbb{R}^{2n}$ contains the error vector as well as the current location of the LTI system, $s_o \rightarrow \begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o$ and $d_o \rightarrow O^{-1}d_o$. Then,

$$\begin{aligned} Q_\phi^*\left(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o, O^{-1}d_o\right) &= r\left(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o\right) \\ &+ \gamma \max_{O^{-1}d'} Q_\phi^*\left(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s'_o, O^{-1}d'\right). \end{aligned} \quad (18)$$

If the first layer of Q_ϕ is linear, say $\{\omega_{1i}s + b_{1i}\}$, $\forall i = 1, 2, \dots, k$, then ω_{1i} can absorb the transformation matrix and becomes $\omega_{1i} \begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix}$, where k is the number of units in the first layer. Assume all other parameters are the same as in Q_ϕ except ω_{1i} . We denote this new family of parameters to be ϕ_n . Fig. 3 shows the detail of such parameters transformation. Then, (18) becomes

$$\begin{aligned} Q_{\phi_n}^*(s_o, O^{-1}d_o) &= r\left(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o\right) \\ &+ \gamma \max_{O^{-1}d'} Q_{\phi_n}^*(s'_o, O^{-1}d'). \end{aligned} \quad (19)$$

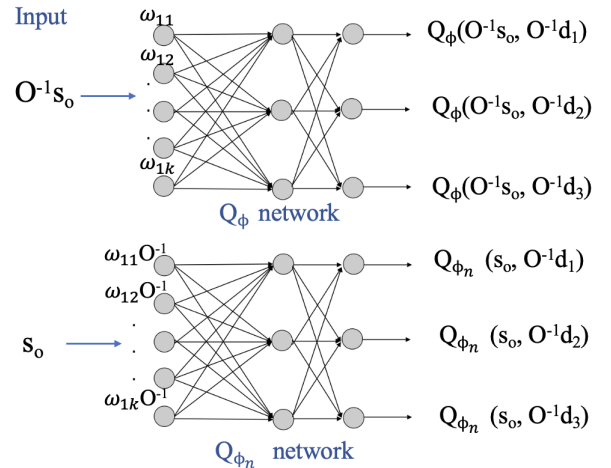


Fig. 3. The parameter transformations from Q_ϕ to Q_{ϕ_n} .

Since the direction space is invariant under matrix O , $O^{-1}d_o$ and d lie in the same space and we can write (19) as

$$Q_{\phi_n}^*(s_o, d) = r\left(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o\right) + \gamma \max_{d'} Q_{\phi_n}^*(\tilde{s}_o, d'), \quad (20)$$

where \tilde{s}_o is the state after the new system at state s_o taking direction d . From the uniqueness of the optimal Q-network, it is proved that $Q_{\phi_o}^*(s_o, d_o) = Q_{\phi_n}^*(s_o, d_o)$ as long as the reward function in the new problem is $r(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o)$.

From (17) and the invariant direction space, we have

$$\begin{aligned} \pi_o^*(s_o) &= \arg \max_{d_o} Q_{\phi_o}^*(s_o, d_o) \\ &= O \arg \max_{O^{-1}d_o} Q_{\phi_n}^*(\begin{bmatrix} O & 0 \\ 0 & O \end{bmatrix} s, O^{-1}d_o) \\ &= O \arg \max_d Q_{\phi}^*(s, d) = O\pi^*(s) \\ &= O\pi^*(\begin{bmatrix} O^{-1} & 0 \\ 0 & O^{-1} \end{bmatrix} s_o). \end{aligned} \quad (21)$$

Therefore, the optimal policy $\pi_o^*(s_o)$ of Theorem 1 coincides with the policy prescribed by applying Algorithm 1 directly. ■

Corollary 1: Q_{ϕ_o} is convergent that is $\lim_{k \rightarrow \infty} (Q_{\phi_o})^k = Q_{\phi_o}^*$.

Proof: From the proof of Theorem 1, we obtain the relationship between ϕ_o and ϕ when both of them construct the optimal Q function. Therefore, from the convergence of Q_{ϕ}^* , $Q_{\phi_o}^*$ is also convergent. ■

For general cases, in the last step above, finding the nearest neighbor of \vec{d}_o is a challenging task when there are numerous candidate quantized directions if we use exhaustive search. We may use the Hebb-Oja algorithm which has been introduced in [10], [20] to compute the nearest direction through iterations. Another approach is constructing a kd-tree, which is a binary space partitioning data structure. Each non-leaf node can be thought of as a dividing hyperplane. The points on one side of this hyperplane are represented by the left subtree of the node, while the right subtree represents the points on the other side of the hyperplane. Each node in the tree is associated with one of the k dimensions, and the hyperplane is perpendicular to the axis of that dimension. Here, the quantized directions can be viewed as nodes and the direction \vec{d}_o is the search key. This tree structure is well suited for channel dropout situations because when nodes in the tree have been removed, instead of destroying the whole structure, we only need to form the set of all nodes and leaves from the children of the removed nodes and recreate that part of the tree.

By applying either of two approaches mentioned above, the mapping function M is obtained. We can directly use this model in Lemma 1 to compute a (sub)optimal direction. In addition, it can also be used to learn the Q-network by the same structure of loss function (12) and the initial $\phi_o = \phi$. From experiments in Section V, it can be observed that with these initial parameters, the training efficiency improves compared with a randomly generated initialization.

V. SIMULATION AND ANALYSIS

In this section, we provide simple simulations of the learned trajectory per the MPC-based method and per the DQN algorithm. The LTI system to be emulated is $\dot{x} =$

$Hx = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} x$ with all its eigenvalues located in the left-half plane. To simplify the example, we let A and B be as in Example 1 and choose time step $h = 0.05$. In the cost function, we design $P = \Gamma = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$ and R to be $0.05 * \mathbb{I}_m$. The result is shown in Fig. 4(a), which is generated by adopting the solver Cplex [24] to solve the integer programming optimization problem (5). It can be observed that by solving the MPC optimization problem, we generate a sequence of vector steps that emulate the LTI system moving towards the origin.

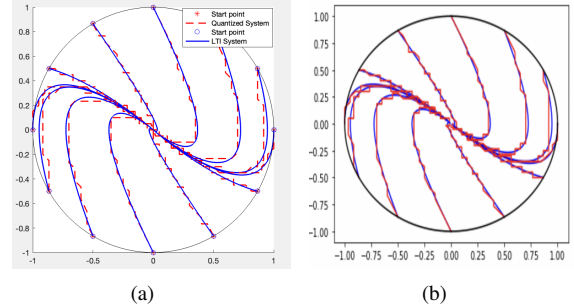


Fig. 4. Figure (a) is the emulation result when solving the integer optimization problem (5) directly. Blue trajectories are the LTI system and red ones are the quantized system. Figure (b) is the emulation result using the MPC data-based learned model.

At the same time, we collect a total of 10000 data points with error vector, i.e., $x(k) - x_{qs}(k)$ and directions, i.e., $[x(k+1) - x(k)]/h$ and $[x_{qs}(k+1) - x_{qs}(k)]/h$ of two systems as features, and quantized directions as their labels. Then, we construct a regular densely-connected four-layer neural network with ReLU, Sigmoid, or Linear as their activation functions. The number of nodes in each layer is 1200, 1200, 1200, and 25, respectively. After 20 training epochs, we obtain a model with a training accuracy of 94.1%. The test dataset comes from another emulation with all initial points in the unit circle, which contains 840 data points, and the accuracy is 90.7%. The trajectory generated by this model is shown in Fig. 4(b).

Using the same A, B matrices and the same emulated system, we also construct a two linear-layer Q network with a hidden layer with 200 activation units between them to learn the trajectory. The activation function is ReLU. Fig. 5(a) is an emulating system produced by the generalized DQN algorithm. The learning metric in the algorithm only considers the L_2 norm of the error vector between two systems. At time T , the location of the LTI system is $x(T) = e^{THh}x_0$, while the quantized system is in the location $x_{qs}(T) = x_0 + hBu(0) + hBu(1) + \dots + hBu(T-1)$. Therefore, $G(u, T) = \|e^{THh}x_0 - (x_0 + hB \sum_{t=0}^{T-1} u(t))\|^2$, where x_0 is the initial point. In this simple example, there are 25 distinct directions (including zero vector) in the quantization output alphabet formed by Bu , where $u_i \in \{-1, 0, 1\}^4$. Fig. 5(b) provides an example of having one channel randomly dropout at each time instant. Though it

has degraded performance, it still has the ability to emulate.

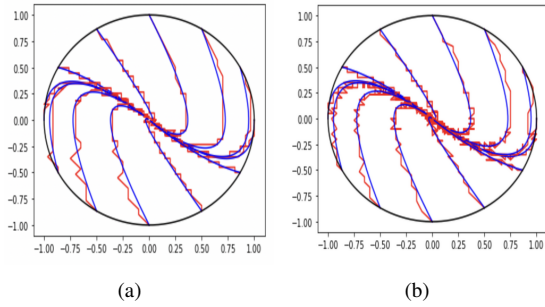


Fig. 5. Figure (a) is generated by the DQN algorithm with time step $h = 0.05$, while figure (b) is using the same model but having channel dropouts.

Next, we use the mapping-based transfer learning method to emulate another LTI system $\dot{z} = H_o z = \begin{bmatrix} -2 & 1 \\ -1 & 0 \end{bmatrix} z$. The transformation matrix is $O = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ in (16), which results in the action space being invariant. The models we use are the MPC data-based learning and the DQN-like algorithm, respectively. The tracking performance is shown in Fig. 6. It illustrates that both of these two models can be transferred to solve another emulation problem.

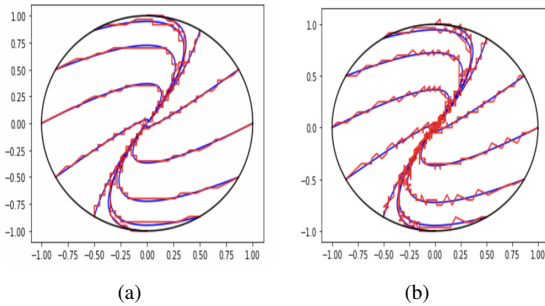


Fig. 6. Figures show the emulation trajectory of another LTI system $\dot{z} = H_o z = \begin{bmatrix} -2 & 1 \\ -1 & 0 \end{bmatrix} z$ by transferring the MPC-based and DQN model.

When the transform matrix is $O = \begin{bmatrix} 1 & 0.5 \\ -0.5 & 1 \end{bmatrix}$ in (16), the action space is no longer invariant as is illustrated in Fig. 7(a). The red vector directions form the new action space, used in emulating a new LTI system. For this example, the new emulated LTI system is $\dot{z} = H_o z = \begin{bmatrix} -0.5 & 0 \\ -1 & -2.5 \end{bmatrix} z$. The model we used is the DQN-like Algorithm 1 and the tracking performance is shown in Fig. 7(b).

To compare the training time of the model with initial weights and the existing model obtained from the previous emulation problem, we still choose the new emulated LTI system to be $\dot{z} = H_o z = \begin{bmatrix} -0.5 & 0 \\ -1 & -2.5 \end{bmatrix} z$. The existing DQN model is obtained from emulating the LTI system $\dot{x} = Hx = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} x$, which generates the trajectory

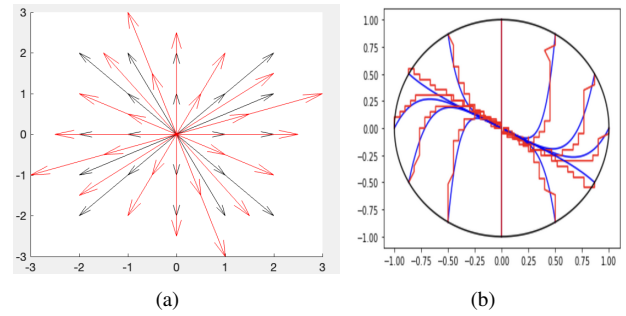


Fig. 7. Figure (a) shows the new sequence of quantized directions in red to emulate the LTI system $\dot{z} = H_o z = \begin{bmatrix} -0.5 & 0 \\ -1 & -2.5 \end{bmatrix} z$, while black ones are the previous directions to be used to train the DQN model. Figure (b) is the emulation trajectory using new vector directions.

of Fig.5(a). Then, we train 20 episodes of these two models, respectively. Simulation results are shown in Fig. 8. It can be observed that the existing model has a better emulation performance after the same training episodes. The reason is that borrowing the existing model's structure and parameters for the new problem can reduce the value of the loss function to a certain extent during the initial training to shorten the training time.

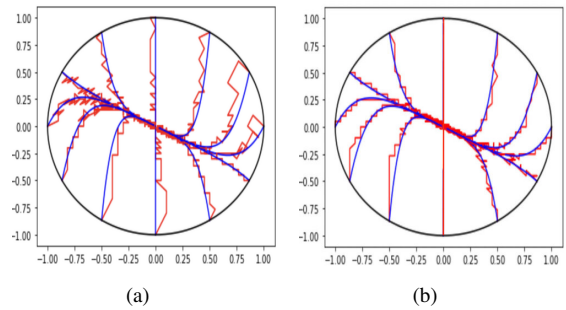


Fig. 8. Figure (a) shows the learned trajectories after 20 training episodes of a DQN model with randomly initialized parameters, while figure (b) uses the previously trained LTI system DQN model as initializations for the new system's model.

VI. CONCLUSIONS AND FUTURE WORK

The work we have reported examines techniques in machine learning by which the neuromimetic linear models introduced in [9] and [10] can learn to emulate stable closed-loop systems. We have proposed a learning model incorporating model predictive control (MPC) as well as a custom deep Q network (DQN) algorithm. For both approaches, we have studied specific cases in which transfer learning is possible for systems related by certain classes of coordinate transformations. For all cases, it is noted that the kinds of resilience to channel dropouts reported previously persist in the learned models. Future work will be aimed at showing how a continuous observer can be used to synthesize quantized trajectories for the class of overcomplete systems under consideration.

REFERENCES

- [1] J. Baillieul and Z. Kong, "Saliency based control in random feature networks," in 53rd IEEE Conference on Decision and Control, Los Angeles, CA, 2014, pp. 4210-4215. doi: 10.1109/CDC.2014.7040045
- [2] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.
- [3] A. Lazaric, "Transfer in reinforcement learning: a framework and a survey," in *Reinforcement Learning: State-of-the-Art*, Springer, 2012.
- [4] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, 1957.
- [5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," arXiv preprint arXiv:1708.05866, 2017.
- [6] J. Baillieul, "Perceptual Control with Large Feature and Actuator Networks," 2019 IEEE 58th Conference on Decision and Control (CDC), Nice, France, 2019, pp. 3819-3826, doi: 10.1109/CDC40024.2019.9029615.
- [7] Zhu, Z., Lin, K. and Zhou, J., "Transfer learning in deep reinforcement learning: A survey", arXiv preprint arXiv:2009.07888, 2020.
- [8] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, 2009.
- [9] J. Baillieul and Z. Sun, "Neuromimetic Control—A Linear Model Paradigm," 2021 60th IEEE Conference on Decision and Control (CDC), 2021, pp. 2709-2716, doi: 10.1109/CDC45484.2021.9683392.
- [10] Z. Sun, and J. Baillieul. "Neuromimetic Linear Systems—Resilience and Learning." 2022 IEEE 61st Conference on Decision and Control (CDC), pp. 7388-7394. IEEE, 2022.
- [11] Parisini, T. and Zoppoli, R.. A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10), pp.1443-1451, 1995.
- [12] Karg, B. and Lucia, S.. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics*, 50(9), pp.3866-3878, 2020.
- [13] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. *Nature* 518, 529–533 (2015). <https://doi.org/10.1038/nature14236>
- [14] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [15] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. "A survey on deep transfer learning." In *Artificial Neural Networks and Machine Learning—ICANN 2018: 27th International Conference on Artificial Neural Networks*, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27, pp. 270-279. Springer International Publishing, 2018.
- [16] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.
- [17] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. arXiv preprint arXiv:1812.02648, 2018.
- [18] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atssc): methodology and large-scale application on downtown toronto," *IEEE Transactions on Intelligent Transportation Systems*, 2013.
- [19] Z. Sun, and J. Baillieul. "Model Predictive Control for Neuromimetic Quantized Systems." arXiv preprint arXiv:2212.09887, 2022.
- [20] E. Oja, "Simplified neuron model as a principal component analyzer," 1982. *Journal of Mathematical Biology*, 15(3), pp.267-273.
- [21] Wang, Z.T. and Ueda, M., 2021. Convergent and efficient deep Q network algorithm. arXiv preprint arXiv:2106.15419.
- [22] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand- eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.
- [23] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Dept. Psychol., Cambridge Univ., Cambridge, U.K., 1989. [Online]. Available: <https://ci.nii.ac.jp/naid/10008997819/>
- [24] Cplex, I.I. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53), 157, 2013.