Supplementary Material

1. DESCRIPTION OF THE INNATE IMMUNE RESPONSE AGENT-BASED MODEL

The innate immune response agent-based model (IIRABM) simulates the hospitalization of a patient diagnosed with sepsis. The virtual environment consists of a 101×101 grid representing a two-dimensional abstraction of the human endothelial–blood interface. Each grid point contains an endothelial cell agent and may contain additional immune cell agents. Each grid point also contains 14 scalar state variables: 12 cytokine concentrations, a measure of infection, and a measure of tissue damage. Agents (cells) contain additional state information, including age, activation status, and cytokine receptor concentrations.

The mechanisms of the IIRABM capture many cell–cell and cell–environment interactions related to the innate immune response and sepsis. The IIRABM includes mechanisms for infection spreading, tissue damage, chemotactic cell movement, leukocyte activation, leukocyte extravasation, respiratory burst, apoptosis, antibiotic administration, cytokine receptor trafficking, and T cell differentiation. Details of these mechanisms are provided in An (2004).

Most relevant to the control problem is the mechanism by which cytokines interact with each other. Cytokine signaling ultimately governs the immune response; indeed, cytokine dysregulation is a root issue underlying sepsis. Furthermore, the simulated cytokine interventions we explore directly modulate these interactions. Thus, we describe this mechanism in detail. Under certain conditions, a cell agent updates the values of one or more cytokines at its grid point. In the absence of cytokine intervention, the general formula for updating the ith cytokine is linear: $c_i \leftarrow \boldsymbol{\lambda} \cdot \mathbf{c}$, where c_i is the cytokine being updated, $\boldsymbol{\lambda}$ is a vector of regulatory constants, and $\mathbf{c}$ is the vector of cytokine values at that grid point. The elements in $\boldsymbol{\lambda}$ represent the degree to which each cytokine up- or downregulates cytokine i through the cell's mediation. In addition, direct cytokine secretion/elimination (the constant portion of the linear updates) is afforded by appending $\mathbf{c}$ with a dummy value of unity and appending $\boldsymbol{\lambda}$ with a value that represents the degree to which the cell directly secretes/eliminates cytokine i.

Cytokines are also altered through passive diffusion, spontaneous degradation, and external interventions. External interventions—or actions—map to putative cytokine-specific mediation therapies that either inhibit or augment cytokine signaling. Programmatically, an action $a \in \mathbb{R}$, specific to the ith cytokine, alters that cytokine's update rule by wrapping it in a function parameterized by a. Specifically, the update becomes $c_i \leftarrow f(\boldsymbol{\lambda} \cdot \mathbf{c}; a)$, where f is defined as

$$f(x;a) = \begin{cases} 10^a x & a \le 0 \quad \text{(inhibition)} \\ x + (10^a - 1) & a \ge 0 \quad \text{(augmentation)} \end{cases} .$$

Thus, an action taken by the reinforcement learning (RL) agent for a given cytokine i corresponds to choosing parameter a_i that determines the amount of inhibition or augmentation by the update function $f(\lambda \cdot \mathbf{c}; a_i)$. Note that the wrapper function is multiplicative for negative (inhibitory) actions and additive for positive (augmentative) actions. Exponentiation is introduced so that the action space is symmetric and centered around $a = 0$. Note that when $a = 0$, the wrapper function becomes the identity function, that is, it recovers the former update in which no cytokine mediation is applied. The control problem deals with dynamically choosing values of a for each cytokine in a way that drives the simulated patient toward a healthy state.

A simulation begins with an applied bacterial infection. As it begins to spread, it quickly triggers cytokine signaling and an immune response. After 12 hours of simulated time, actions may be applied. This time delay reflects an approximation of the minimal time from onset of infection to presentation to a health care facility, identification of infection, and initiation of treatment. The simulation ultimately ends with one of two outcomes: complete healing or death. The health condition occurs when the system's total damage plus the total infection level (each expressed as a percentage of its maximum possible value) drops below a threshold of 0.02%. Once cytokine mediation stops, to ensure that it provided a nontransient lasting effect, a patient must proceed without further cytokine mediation for at least 12 hours before a less stringent health threshold of 0.8% is checked. The less stringent threshold is due to small transient infection and damage caused by recurrent injuries to the host due to environmental conditions. The death condition occurs when total damage exceeds 80% (regardless of infection level) (An et al., 2017; Cockrell and An, 2018).

Clinically, this threshold represents the ability of current medical technologies to keep patients alive (i.e., through organ support machines) in conditions that previously would have been lethal.

2. REINFORCEMENT LEARNING WITH AGENT-BASED MODELS

An RL problem is formulated as a Markov decision process, in which an *environment* (defined by a state space $\mathcal{S}$, action space $\mathcal{A}$, state transition function P, and reward function r) interacts with an RL agent. (Note that the environment is distinct from the virtual environment in which IIRABM agents interact, and the RL agent is distinct from agents of the IIRABM.)

- A *state* $s \in \mathcal{S}$ is a complete specification of all quantities of the agent-based model (ABM) at each time step. Although the ABM maintains its own internal state, the RL agent's observation $o = O(s)$ may be limited by a function $O(\cdot)$ that reduces the amount of accessible information. This is analogous to a clinician's limited observation of a patient's complete physiological state. To maintain a degree of clinical relevance, our RL agent will treat observations o as the complete state of the system, using $s \in \mathcal{S}$ to denote the RL agent's observation hereafter.
- An *action* $a \in \mathcal{A}$ is an external operation applied to the ABM at each time step, chosen by the RL agent rather than generated by the internal mechanisms of the ABM.
- The *state transition function* $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ represents the simulation mechanisms, which transition the ABM from one state to the next, given the chosen action.
- The *reward function* $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is a mapping from current state, the action taken, and the resulting state to a scalar signal at each time step. The *return* is the discounted cumulative future reward from time t until the end of the episode, $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the terminal time step and $\gamma \in (0, 1]$ is a *discount factor* that determines the degree to which immediate rewards are favored over delayed rewards.
- The *RL agent* interacts with the environment by following a *policy* $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$, which, in the general case, maps a state to a distribution over actions. In this work, we consider deterministic policies $\mu : \mathcal{S} \mapsto \mathcal{A}$, which can be learned with many fewer samples than stochastic policies (Silver et al., 2014).

The RL agent interacts with its environment in the following way: given a current observation s_t, the RL agent selects an action $a_t \sim \pi(s_t)$ and applies it to the environment, which produces a new observation s_{t+1} and an associated reward r_t. The objective of an RL algorithm is to find the optimal policy $\pi^\star$ that maximizes the expected return from the distribution over starting states. It does so by interacting with the environment across many episodes, using the reward as a learning signal.

3. DEEP DETERMINISTIC POLICY GRADIENT

RL algorithms can be broadly classified as value-function methods or policy search methods (Sutton and Barto, 1998). Among value-function methods, Q-learning is a widely used algorithm for estimating an optimal action–value function $Q^\star : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, defined as $Q^\star(s, a) := \max_\mu \mathbb{E}[R_t | s_t = s, a_t = a]$. It gives the maximum expected return for executing action a at state s and following an optimal policy thereafter, and it induces a deterministic optimal policy $\mu^\star : \mathcal{S} \mapsto \mathcal{A}$ defined by $\mu^\star(s) := \arg\max_{a \in \mathcal{A}} Q^\star(s, a)$.

In many systems, including the IIRABM, the state space $\mathcal{S}$ and/or action space $\mathcal{A}$ are both continuous and high dimensional, rendering traditional tabular Q-learning approaches infeasible. Mnih et al. (2013) developed the *deep Q-networks* (DQN) algorithm that adapted Q-learning to continuous and high-dimensional state spaces by utilizing an artificial neural network called the *Q-network*, $Q(s, a; \theta)$, to parameterize the action–value function with neural network weights θ. Training is executed using transitions (s_t, a_t, r_t, s_{t+1}) and stochastic gradient descent on a loss function $L(\theta) := (y_t - Q(s_t, a_t; \theta))^2$, where $y_t := r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta)$. RL algorithms utilizing such networks are known as *deep RL* (DRL) algorithms. Two main contributions led to the success of DQN: (1) the use of a *target network* $Q'(s, a; \theta')$, which is a copy of the Q-network used to evaluate y_t, but whose distinct weights θ' slowly update toward the learned weights θ; and (2) the use of a *replay buffer* containing a history of transitions, from which minibatches are sampled for network training. DQN had great success in learning to play Atari video games, often achieving superhuman performance.

However, DQN cannot handle large continuous action spaces. Lillicrap et al. (2015) extended DRL to continuous action spaces by utilizing an actor-critic framework that optimizes the performance $J(\theta^{\mu})$ of a policy $\mu(s;\theta^{\mu})$. Their *deep deterministic policy gradient* (DDPG) algorithm uses two main networks. An *actor network* $\mu(s;\theta^{\mu})$ outputs an action $a \in \mathcal{A}$ based on the current state s and parameters θ^{μ}. A *critic network* $Q(s,a;\theta^{Q})$ evaluates the Q value given the state–action pair and parameters θ^{Q}. Actor network weights are updated using the deterministic policy gradient $\nabla_{\theta^{\mu}} J(\theta^{\mu}) = \mathbb{E}_{s\sim\mu}\left[\nabla_{\theta^{\mu}} Q(s,\mu(s;\theta^{\mu});\theta^{Q})\right]$ (Silver et al., 2014). Critic network weights are updating using gradient descent on the loss $L(\theta^{Q}) = (y_t - Q(s_t, a_t;\theta^{Q}))^2$, where $y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1};\theta^{\mu});\theta^{Q})$ is known as the *temporal difference* target. As with DQN, DDPG utilizes a replay buffer and target networks for the actor and critic. We employ DDPG to handle the continuous state and action spaces of the sepsis environment.

4. NETWORK ARCHITECTURE AND HYPERPARAMETER SELECTION

We used the same network architecture and hyperparameters as Lillicrap et al. (2015) with several exceptions. In brief, both the actor and critic networks included two hidden layers (of 400 and 300 nodes, respectively) and rectified linear activation functions. For the critic network, a hyperbolic tangent activation was added to the output layer to bound actions to the environment's action space, $[-1,1]^{12}$. For the critic network, actions were added at the second hidden layer. Differences from Lillicrap et al. (2015) include that the inputs to each network were not batch normalized, which we found to result in instability as the data distribution of the experience replay buffer shifted during training. Furthermore, batch normalization was not applied to hidden layers. For exploration, we added uncorrelated Gaussian noise with a standard deviation of 0.1 independently to each dimension of the selected action vector. This standard deviation was decreased 10-fold every 1000 episodes.

Given a fixed reward value for the health/death outcome of ± 250, the reward function has two hyperparameters: β and λ. To ensure that the overall goal of reaching health/death remains the dominant driver of the reward function, we sought the contributions of potential-based reward shaping and penalizing actions to each be roughly two orders of magnitude less than the terminal rewards. We found that $\beta = 100$ and $\lambda = 1$ typically resulted in $\mathcal{O}(1)$ values for the potential-based term and action penalty term, respectively.

5. SOFTWARE AND COMPUTATION

The IIRABM was implemented in C++ as in Cockrell and An (2018) and An et al. (2017) to maximize performance. We exposed it to Python using the Boost C++ libraries (Abrahams and Grosse-Kunstleve, 2003) and recast it as an OpenAI Gym environment (Brockman et al., 2016). We implemented DDPG in Python, leveraging TensorFlow (Abadi et al., 2016) and Gym (Brockman et al., 2016) packages.

6. RELATED WORK

We review studies that are related in some combination of methodology (i.e., DRL), type of environment (i.e., simulation, ABM), application domain (i.e., biology, sepsis), and use case (i.e., controlling a system to a desirable state).

Perhaps the most similar study is an application of traditional RL (i.e., not DRL) to control an ABM of tumor growth using radiotherapy. The authors employ tabular Q-learning (Jalalimanesh et al., 2017) to learn the optimal policy of radiotherapy, using an environment comprising an underlying ABM of vascular tumor growth. Both the action space (radiation intensity: choice of weak, normal, or intense) and observation space (tumor size: one of 200 bins) are one-dimensional and discretized for simplicity. Thus, DRL was not employed in this study since traditional tabular methods sufficed.

With respect to controlling sepsis, a recent study used DRL to discover retrospective policies for treating septic patients given a fixed clinical data set comprising observations (e.g., patient vitals) and actions (i.e., administered medications) (Raghu et al., 2017). Despite a similar goal to this study, such retrospective studies using fixed data sets (a setting known as batch mode RL; Ernst et al., 2005) are quite different in methodology, as there is no underlying simulation and thus no ability to query or explore the environment. Consequently, in contrast to our approach, only previously attempted therapeutics can be considered. Also similar in goal but different in methodology is a recent approach using genetic algorithms to search for a nonadaptive control strategy using the same IIRABM as this study (Cockrell and An, 2018).

There have been many recent advances in state-of-the-art DRL algorithms (Mnih et al., 2013; Lillicrap et al., 2015) and variations of these algorithms (Schaul et al., 2015; Wang et al., 2015; Mnih et al., 2016; Van Hasselt et al., 2016). These approaches are typically benchmarked against standard RL environments (such as those curated by OpenAI Gym; Brockman et al., 2016), including classic control problems, Atari 2600 games using the Arcade Learning Environment (Bellemare et al., 2013), board games (e.g., Go), and the MuJoCo physics engine (Todorov et al., 2012). These benchmark environments are open sourced and well curated, facilitating performance comparisons across different algorithms. There are limited examples of DRL being applied to control a simulation (including an ABM). Notably, Li et al. (2016) and Casas (2017) use DRL for controlling the timing of traffic lights.

SUPPLEMENTARY REFERENCES

Abadi, M., Agarwal, A., Barham, P., et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv* arXiv:1603.04467.

Abrahams, D., and Grosse-Kunstleve, R.W. 2003. Building hybrid systems with Boost.Python. *C/C++ Users J*. 21, 29–36.

An, G. 2004. In silico experiments of existing and hypothetical cytokine-directed clinical trials using agent-based modeling. *Crit. Care Med*. 32, 2050–2060.

An, G., Fitzpatrick, B.G., Christley, S., et al. 2017. Optimization and control of agent-based models in biology: A perspective. *Bull. Math. Biol*. 79, 63–87.

Bellemare, M.G., Naddaf, Y., Veness, J., et al. 2013. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res*. 47, 253–279.

Brockman, G., Cheung, V., Pettersson, L., et al. 2016. Openai gym. *arXiv* arXiv:1606.01540.

Casas, N. 2017. Deep deterministic policy gradient for urban traffic light control. *arXiv* arXiv:1703.09035.

Cockrell, C., and An, G. 2018. Examining the controllability of sepsis using genetic algorithms on an agent-based model of systemic inflammation. *PLoS Comput. Biol*. 14, e1005876.

Ernst, D., Geurts, P., and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res*. 6, 503–556.

Jalalimanesh, A., Haghighi, H.S., Ahmadi, A., et al. 2017. Simulation-based optimization of radiotherapy: Agent-based modeling and reinforcement learning. *Math. Comput. Simul*. 133, 235–248.

Li, L., Lv, Y., and Wang, F.-Y. 2016. Traffic signal timing via deep reinforcement learning. *IEEE/CAA J. Autom. Sin*. 3, 247–254.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., et al. 2015. Continuous control with deep reinforcement learning. *arXiv* arXiv:1509.02971.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. 2013. Playing Atari with deep reinforcement learning. *arXiv* arXiv:1312.5602.

Mnih, V., Badia, A.P., Mirza, M., et al. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937. New York, NY, USA.

Raghu, A., Komorowski, M., Celi, L.A., et al. 2017. Continuous state-space models for optimal sepsis treatment—A deep reinforcement learning approach. *arXiv* arXiv:1705.08422.

Schaul, T., Quan, J., Antonoglou, I., et al. 2015. Prioritized experience replay. *arXiv* arXiv:1511.05952.

Silver, D., Lever, G., Heess, N., et al. 2014. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. pp. 387–395. Beijing, China.

Sutton, R., S., and Barto, A., G. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, MA, USA. 1998.

Todorov, E., Erez, T., and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. *In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Vilamoura, Portugal, pp. 5026–5033.

Van Hasselt, H., Guez, A., and Silver, D. 2016. Deep reinforcement learning with double Q-learning. *In AAAI*. Phoenix, AZ, USA. pp. 2094–2100.

Wang, Z., Schaul, T., Hessel, M., et al. 2015. Dueling network architectures for deep reinforcement learning. *arXiv* arXiv:1511.06581.