GYMIC: An OpenAI Gym Environment for Simulating Sepsis Treatment for ICU Patients

Amirhossein Kiani [1] Tianli Ding [1] Peter Henderson [1]

1. Introduction

Sepsis is a life-threatening illness caused by the body's response to infection and a leading cause of patient mortality. The task of treating patients with sepsis is very challenging. Aside from different dosage of antibiotics and controlling the sources of infection, sepsis treatment involves administering intravenous fluids and vasopressors. These procedures have however shown to have drastically different results on different patients and there is a lack of efficient real-time decision support tools to guide physicians (Raghu et al., 2017).

MIMIC is a large and comprehensive dataset consisting of de-identified health data associated with 40,000 critical care patients (Johnson et al., 2016). It includes features such as patient demographics, vital signs, laboratory tests, medications and medical interventions.

Deep Reinforcement Learning (RL) has been applied towards learning optimal policies for patient treatment using the MIMIC dataset in recent works (Raghu et al., 2017; Prasad et al., 2017). Existing approaches however rely on Off Policy Evaluation (OPE) methods which have limited power in evaluating the value of deterministic policies due to their reliance on importance sampling. Due to this limitation, most existing works resort to providing a rather qualitative explanation for their models performance. To get around this limitation, there are at least three different alternative approaches: (1) learning a stochastic policy using e.g. an Actor-Critic method (Wang et al., 2016), (2) utilizing alternative evaluation methods to overcome limitations of OPE methods for deterministic policies (Liu et al., 2018), or (3) building a simulator for the patient states in the ICU and using the simulator for policy evaluation.

Given the pressing need for more medical simulators in the RL field and their applicability to a wide range of off-the-shelf methods, we decided to take approach (3). Upon verifying the usability of our simulator, we plan to provide this work as an open source environment for simulating sepsis treatment in the ICU. In addition to building the simulator, this project also includes two different reference agents that are built on top of the environment as proofs of concept.

2. Approach

The source code for this project can be accessed at `https://github.com/akiani/rlsepsis234`.

2.1. The Environment

Our approach is centered on building a custom OpenAI Gym environment that simulates sepsis treatment trajectories in the ICU. This environment consists of four key components: (1) State Model (2) Episode Termination Model (3) Episode Outcome Model (4) OpenAI Gym Wrapper. Each component is described in further detail in the following sections.

2.1.1. State Model

We developed a Recurrent Neural Network trained using pre-processed MIMIC features from our training data. The architecture of this model is described in Figure 1.

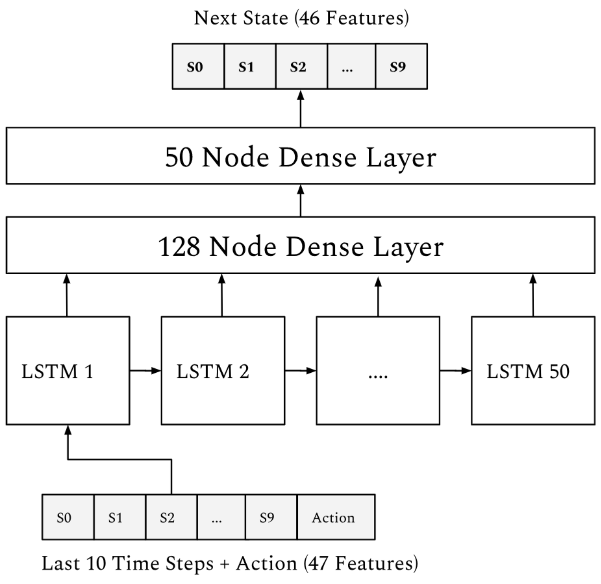

Figure 1. Simulator's State Model Architecture

The model was implemented with Tensorflow using Adam Optimizer (default parameters) for 10 epochs. The network was optimized to minimize the value of Mean Squared Error. The input to this model consisted of the 46 normalized clinical features and the action value (0-24) for the 10 previous time steps (zero padded). The output of the model was the 46 normalized features for the next step.

2.1.2. Episode Termination Model

A separate model was developed to detect episode transitions. The transitions were defined as two mutually exclusive cases of (1) terminating the episode (2) continuing the episode. We accounted for the length of the episodes by adding an episode number feature to the space and action features for this model. The intuition for this feature was to model the distribution of episode lengths seen in our training data and without this feature our generated episodes ended up with mostly unreasonable lengths. The detailed architecture of the episode termination model is demonstrated in Figure 2.

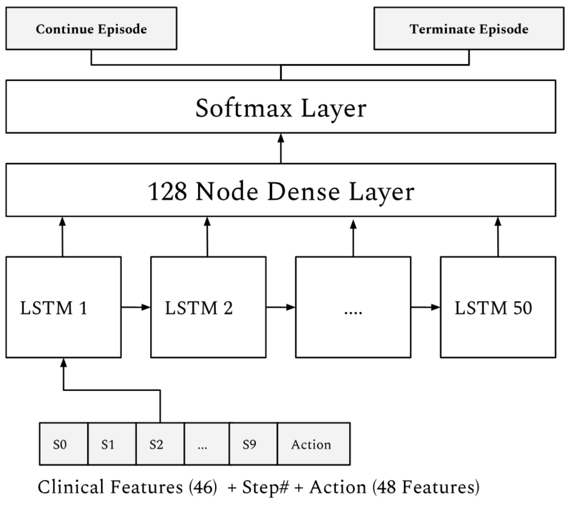

Figure 2. Simulator's Termination Model Architecture

2.1.3. Episode Outcome Model

A third model with the same features and architecture as the Episode Termination Model was developed to predict the two mutually exclusive outcomes of death or release from hospital. This model was used in the environment to decide the reward values at the end of each episode based on the final 10 steps (zero padded) in the episode.

2.1.4. OpenAI Gym Wrapper

We developed a custom Gym environment accessible on `https://github.com/akiani/gym-sepsis`. We followed the directions on OpenAI gym Github [1]to set up the project structure and implemented the required interfaces. The key function implemented in the OpenAI Gym environment is the `step()` function which given an action a, primarily returns an observation o and a reward r. The agents state is initialized with a randomly sampled state from our test datas initial steps. At each step, the agent runs the memory of 10 last step-action values in its history (zero padded) through the aforementioned models and outputs the next state and decides if this state is the final state for the current episode. If the state is deemed to be the final step, then the outcome model is used to generate a sparse reward. The reward is set to +15 for a release event and -15 for a death event.

2.2. Leveraging OpenAI Baseline

A major benefit of implementing our simulator with an OpenAI Gym interface is the ability to run off-the-shelf algorithms implemented by the OpenAI Baselines project [2]. This project consists of a set of high-quality implementations for popular reinforcement learning algorithms.

This process was relatively seamless after building OpenAI Baseline with our custom environment. We tested the DQN (Off-Policy Deep Q-Learning) on our gym environment.

3. Data Processing

We attempted to follow the footsteps of Raghu et al. in defining our state features, action space and reward function. Specifically, we used 46 clinical features to denote each state as defined in (Raghu et al., 2017). These features have been used in major studies for sepsis cohort selection such as those by Johnson et al.. For actions, we took the same approach as Raghu et al. by defining a discrete 5×5 action space for the medical interventions spanning the space of intravenous (IV) fluid and maximum vasopressor (VP) dosage in a given 4 hour window. Action space was discretized into per-drug quartiles based on all non-zero dosages of the two drugs. Each drug at every timestep was converted into an integer representing its quartiles bin number. A special case of no drug given as bin 0 was also included in the action space. Actions were represented as tuples of (total IV in, max VP in) at each time step. Rewards were set to +15 for the last step in episodes where the patient was released from the hospital and to -15 for when the hospital stay had resulted in death of the patient. For all other steps the reward issued was set to 0.

The space features were generated by setting up a local instance of the MIMIC database (around 60GB in size) on a

[1]`https://github.com/openai/gym/wiki/Environments`

[2]`https://github.com/openai/baselines`

Linux system and involved leveraging reference data extraction queries from Raghu et al., Johnson et al., and reference SQL scripts[3]. Although we had access to reference SQL queries for extracting the sepsis cohort, successfully running these queries was a daunting task involving major reconfigurations and multiple days of work. We did contact Raghu et al. to get access to their raw dataset for their paper but did not hear back from the authors.

We were not able to exactly mirror the cohort built by Raghu et al. due to lack of specifics in the paper or their released code in regards to:

- Initial cohort selection: the authors cite Johnson et al. in regards to their cohort selection strategy. This method however generated 18345 patients during our replication using available Sepsis3 cohort generation script [4] whereas their method yielded 17,898 patients.
- Missing data: the authors did not indicate how they handled missing data for intervals during which a measurement for the features is missing. We performed a Last Value Carried Forward and mean value imputation as it is a common practice in EHR data processing.
- After further examining the results released by the authors on Github, we also observed that the number of actions for each discrete action value is different between their approach and ours. We spent ample time to resolve this discrepancy, however due to the fact that the authors did not release their data extraction code (written in MATLAB), we were not able to understand the root cause.

Aside from the above uncertainties, we do see a relatively similar distribution of discrete action values between those generated by Raghu et al. and ours in the test dataset as demonstrated in Figure 3.

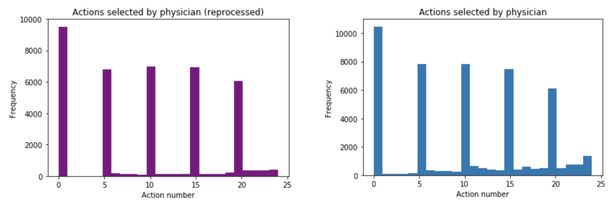

Figure 3. Actions distribution

Both project members went through a mandatory training process in order to receive permissions to access the MIMIC dataset.

4. The Agent

4.1. Q-learning Network

As our first reference methods for an RL agent built on top of our simulation environment, we built a Dueling DQN agent. In order to be able to compare our performance to those of Raghu et al., our model has a similar configuration compare to theirs. The agent has a value network with 2 hidden layers of size 128. Each hidden layer uses a Leaky ReLU as activation function with $\alpha = 0.5$. Following each hidden layer, batch normalization was applied. The output layer of the value network has a scalar output for the estimated value of each state. The advantage network has the same configuration for the hidden layers with an output layer the size of the action space. Finally, our Q output layer adds the estimated value and normalized advantage estimate to produce a Q value for each action.

$$[H]\hat{Q}(s,a) = \hat{V}(s) + \hat{A}(s,a) - \frac{1}{|A|}\sum_{a' \in A} \hat{A}(s,a') \quad (1)$$

At every training step we used a random sampling replay buffer for Experience Replay with a batch size of 32. This implementation is the same as the standard Experience Replay Buffer on an online DQN agent, and noticeably different from the approach in Raghu et al., which samples directly from the training data. As a result, in time steps where the number of samples in replay buffer is less than batch size, it is possible to have zero paddings, which is not the case in the original paper.

4.2. Agent implementation

We used the Q-learning agent in Assignment 2 as our starting point, and progressively added more modifications. First, we implemented the Q network with the same architecture as described above. We tested our agent on the TestEnv environment from Assignment 2, and observed that the agent was able to converge to optimal returns as shown in Figure 4.

[3]https://github.com/spfohl/cs238_sepsis_rl

[4]https://github.com/alistairewj/sepsis3-mimic

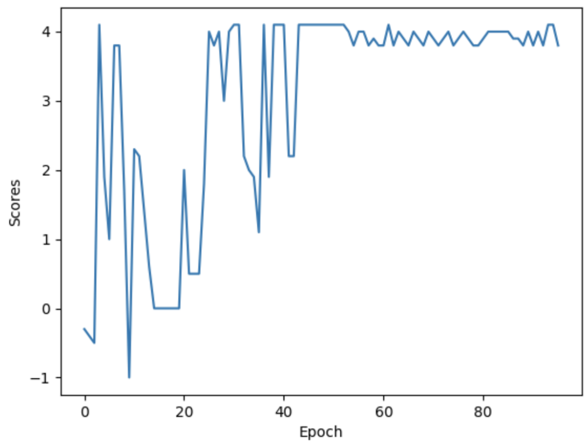

Figure 4. Performance on Assignment 2 TestEnv

Our replay buffer was initially optimized specifically for the Atari games use case. One of the optimizations was to convert the state representation to a uint8 type to save memory. In order to handle the various types of variables in our state representation, we modified the replay buffer to store a complete representation of state without downsizing each number.

We decided to opt for a simple end-of-episode reward of ± 15 instead of using the complex reward function based on intermediate SOFA and lactate levels.

4.3. Offline training

During our initial iterations, we discovered that the Q value estimates were increasing monotonically over training steps and was not able to converge. We think that the combination of function approximation, shifting training target with off-policy learning was the main reason for this to happen. The same issue also occurred in Raghu et al.. As stated in the paper, two compensation mechanisms are used to mitigate this issue. First, we added a regularization term in our loss function to encourage the Q value to stay in the appropriate range.

$$\begin{aligned}\mathcal{L}(\theta) = &\mathbb{E}[(Q_{target} - Q(s,a;\theta)^2)] \\ &+ \lambda \cdot max(|Q(s,a;\theta)| - Q_{thresh}, 0)\end{aligned} \tag{2}$$

Then we also clipped the target Q value at a threshold of 20, to ensure that our Q estimate stays at a reasonable range.

After adding the limits, our agents average estimated Q value stabilized and stayed close to the threshold after a few thousand steps. However, the end result was not satisfactory as it deviated significantly from the results on the paper. We observed that the Q value converged relatively quickly in the first few thousand steps, and then hovered around the maximum cut-off value.

We spent several days trying to understand the cause of discrepancy between our results and the results on the paper, by first debugging our code and then digging into the accompanying code for Raghu et al.. We came to the hypothesis that by clipping off the target Q value, previously accumulated information for distinguishing each of the next states was lost, which impairs the capability of the network to predict a good action. Indeed, when we trained our agent with a few thousand steps, we were able to consistently produce similar results as demonstrated in the paper. As it turned out, digging into the authors code [5] has shown that they have only trained for a few thousand steps before terminating.

5. Results

5.1. Offline Off-Policy Learning Results

After training over 2000 samples with replay $batch\_size = 32$, we were able to achieve similar results as demonstrated in Raghu et al.. As demonstrated in Figure 5, for low and medium SOFA, the physician almost never administers vasopressors, while IV is used relatively frequently. For high SOFA, the agent did not learn a similar behavior as physicians, which could be attributed to the sparsity of these episodes in our training dataset, as mentioned in the original paper.

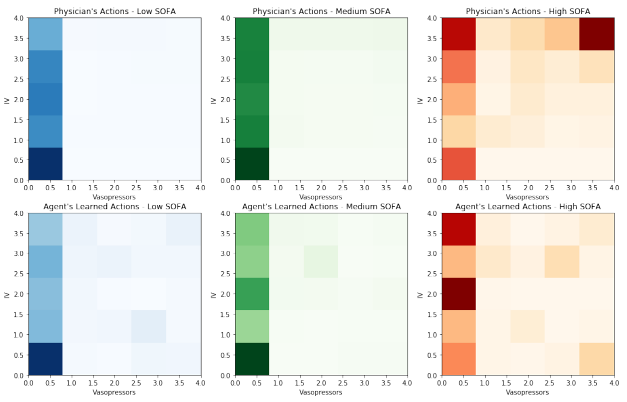

Figure 5. DQN agent actions and physician actions. Dosages are discretized into quantiles, with 0 representing no action.

Our result for comparing the mortality rate based on differences of dosage between physician and agent shows a similar trend as what was shown in Raghu et al., as seen in Figure 6.

[5] `https://github.com/aniruddhraghu/sepsisrl/blob/master/continuous/q_network.ipynb`

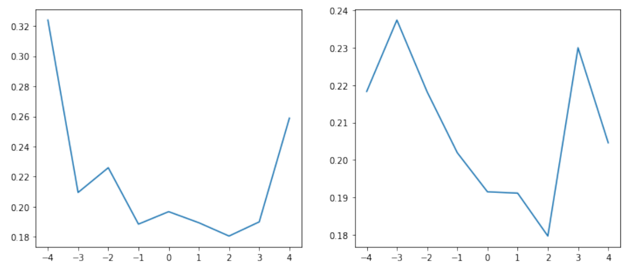
Figure 6. Comparison of mortality based on differences in actions.

We have omitted the comparison since as noted in the original paper, there is no interesting results seen due to sparse death for low SOFA and rare data points for high SOFA. Noticeably, the difference in mortality rate started to raise with larger delta between the action value for the doctors and the action value for agent. We attribute the lack of exact reproduction of these figures to the differences in data processing.

5.2. Simulator Results

In order to measure the correctness of our simulator, we attempted three different tasks which are described in the following sections.

Table 1 describes the accuracy and loss values for the best chosen models for the state, transition and outcome models. We ran all models using Tensorflow, and used early stopping with a patience value of 3. This means that if the models validation metric did not improve for 3 epochs, we used the last best parameters based on this metric and automatically terminated the training. All models reached this state within the first 10 epochs.

5.2.1. Trajectory Loss

In order to measure the compounding error for the simulator, we defined the Normalized Trajectory Loss metric. This metric is defined as the mean squared error of the episodic values generated by the model when the physicians historical actions are played on a simulator initialized with identical initial state. We measured this value for different features as indicated in Figure 7. When episodes ended earlier in the simulator compared to the real world or vice versa, we assumed a value of zero for the missing values. We also normalized each features mean squared error by the sum of squares of the feature values in the real dataset to generate a comparable weight for each feature. The results such as TempC, SpO2 and Glucose are very hard to model for. This could act as a sanity check for our models performance and provide a direction for prioritizing future improvements.

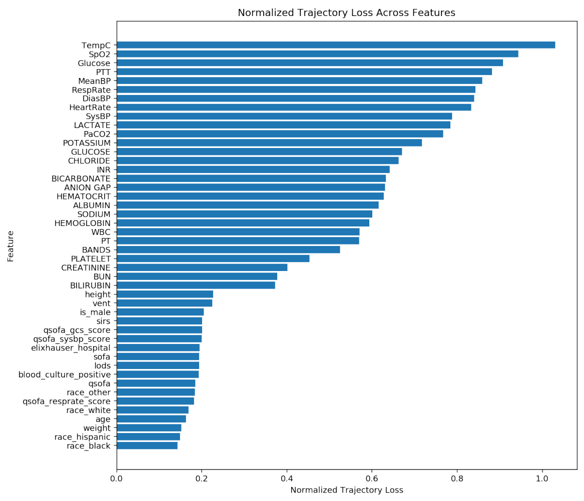

Figure 7. Normalized Trajectory Loss

After playing the physicians actions on our simulation environment, we also compared the distribution of episode lengths as well as results between the real and simulated worlds. The results demonstrated in Figure 8 seem to suggest a visually similar trend between the two worlds which provides a promising signal for the performance of our simulator.

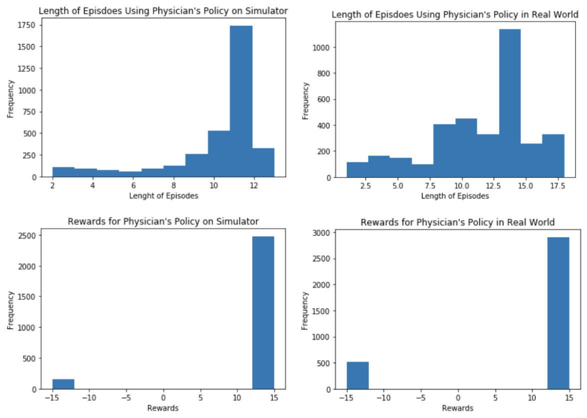

Figure 8. Reward and length of episode distribution

5.2.2. Predictive Power

We studied the predictive power of our state simulator by feeding the simulator with values from the real data trajectories from previous 10 time steps and predicting the next steps. We then plotted these projections against the real values for different clinical signals. Examples from these signals can be seen in Figure 9.

	Performance on Test Dataset	Metric
State Model	0.1161	Mean Squared Error
Episode Termination Model	97.20%	Accuracy
Outcome Prediction Model	86.31%	Accuracy

Table 1. Accuracy and loss values for the best chosen models for the state.

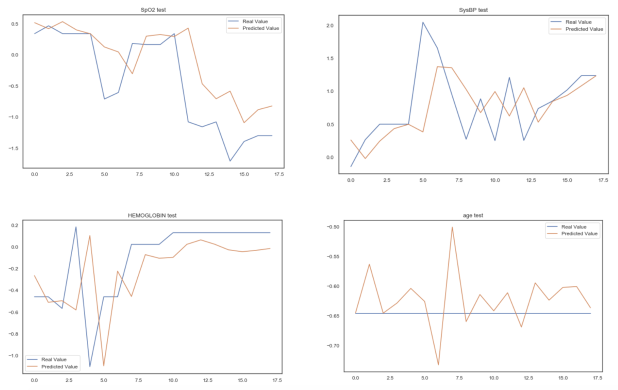

Figure 9. Predictive Power

It is notable that the model does relatively well because it has access to all features in the past 10 steps in the real world data before the prediction. This trend seems to however worsen for features that remain constant during the episode (such as age). To resolve this issue, we set a fixed value for the constant features of 'age', 'race_white', 'race_black', 'race_hispanic', 'race_other', 'height' and 'weight' based on the value in the starting state of the environment.

5.3. Rollout on Physicians Policy

We attempted to visually inspect our simulator on the physicians policy. Specifically, we initialize the model with the starting state of one of the patients. We then performed the actual series of actions that the physician performed on the patient and visualized the state features across the length of the episode. The values generated in the simulator do seem to match the overall trajectory of these values in the real world as demonstrated in Figure 11.

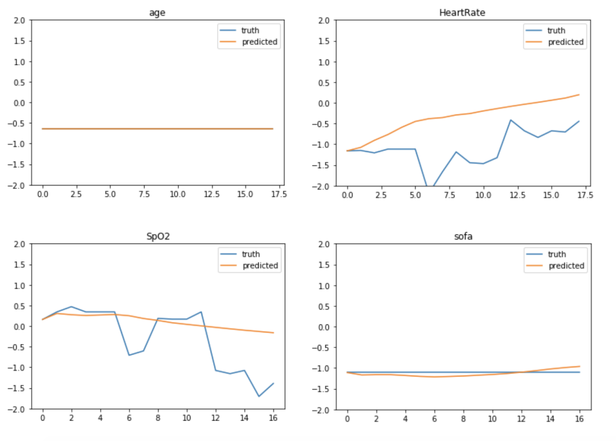

Figure 10. Rollout on Physician's Policy

5.4. Evaluation on OpenAI Baseline Learned Policies

To complete the evaluation of our simulator, we leveraged OpenAI Baselines off the shelf algorithms to train two agents on top of our OpenAI gym environment. We chose the DQN method to learn a comparable policy to our off-policy learned method. As demonstrated in Figure 11 we do observe that the agent is able to learn a very successful policy (almost always patient gets released after a single step) but we suspect that this might be due to the fact that the agent has found an unrealistic vulnerability in the simulation environment that it has learned to exploit.

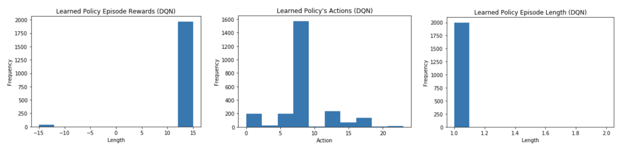

Figure 11. Learned Policy using OpenAI Baseline

6. Conclusion

We reproduced the work done by Raghu et. al and showed a similar pattern in our agents learned behavior. Due to differences in our cohort, we notice some small differences in the final result. The differences could have also been caused by different implementation of the replay buffer, as discussed above. We additionally built a tri-model custom OpenAI Gym environment to simulate the MIMIC Sepsis cohort. While we were able to fit our to the space features for single step predictions based on real patient history, the compounding error effect from a model-based rollout hindered our environments applicability for learning interpretable policies. Further improvements to this model could be built by introducing Variational Autoencoders for building a more generalizable and less noisy environment for the agents to learn from. Additionally filtering the training cohort/dataset to a more relevant set for optimizing the learning process could be beneficial in improving the simulators performance.

7. Future Direction

Works such as those of Ha & Schmidhuber have leveraged Variational Autoencoders to generate dream like episodes for the agents to be used for learning offline policies. Implementing this approach could potentially increase the generalization of our model specially due to the fact that our training dataset seems relatively noisy. Another potential improvement is to optimize the architecture for the episode termination and outcome models. Another key potential improvement is building an stochastic model for the space that accounts of the uncertainties in the feature space such as a Deep Bayesian Neural Network and at every step in the simulation sampling from a distribution of possible next state features. Another useful addition to the simulator could be a visual rendering mode for the environment so that the patient and the actions performed can be graphically visualized similar to those built for Atari games.

Acknowledgements

We would like to thank Professor **Emma Brunskill** and **Peter Henderson** from Stanford Computer Science Department for mentoring us on this project. We also would like to thank **Stephen Pfohl** from Stanford Biomedical Informatics Department for his guidance and sharing his previous work on feature Engineering.

References

Ha, D. and Schmidhuber, J. World models. *CoRR*, abs/1803.10122, 2018. URL http://arxiv.org/abs/1803.10122.

Johnson, A. E., Pollard, T. J., Shen, L., Lehman, L. W., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. MIMIC-III, a freely accessible critical care database. *Sci Data*, 3:160035, May 2016.

Johnson, A. E., Aboab, J., Raffa, J. D., Pollard, T. J., Deliberato, R. O., Celi, L. A., and Stone, D. J. A comparative analysis of sepsis identification methods in an electronic database. *Critical care medicine*, 46(4):494–499, 2018a.

Johnson, A. E., Stone, D. J., Celi, L. A., and Pollard, T. J. The MIMIC Code Repository: enabling reproducibility in critical care research. *J Am Med Inform Assoc*, 25(1): 32–39, 01 2018b.

Liu, Y., Gottesman, O., Raghu, A., Komorowski, M., Faisal, A., Doshi-Velez, F., and Brunskill, E. Representation balancing mdps for off-policy policy evaluation. *CoRR*, abs/1805.09044, 2018. URL http://arxiv.org/abs/1805.09044.

Prasad, N., Cheng, L., Chivers, C., Draugelis, M., and Engelhardt, B. E. A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. *CoRR*, abs/1704.06300, 2017. URL http://arxiv.org/abs/1704.06300.

Raghu, A., Komorowski, M., Ahmed, I., Celi, L. A., Szolovits, P., and Ghassemi, M. Deep reinforcement learning for sepsis treatment. *CoRR*, abs/1711.09602, 2017. URL http://arxiv.org/abs/1711.09602.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL http://arxiv.org/abs/1611.01224.