InferNet for Delayed Reinforcement Tasks: Addressing the Temporal Credit Assignment Problem

1st Markel Sanz Ausin
North Carolina State University
Raleigh, USA
msanzau@ncsu.edu

2nd Hamoon Azizsoltani
North Carolina State University
Raleigh, USA

3rd Song Ju
North Carolina State University
Raleigh, USA

4th Yeo Jin Kim
North Carolina State University
Raleigh, USA

5th Min Chi
North Carolina State University
Raleigh, USA
mchi@ncsu.edu

***Abstract*—Rewards are the critical signals for Reinforcement Learning (RL) algorithms to learn the desired behavior in a sequential multi-step learning task. However, when these rewards are *delayed and noisy* in nature, the learning process becomes more challenging. The temporal Credit Assignment Problem (CAP) is a well-known and challenging task in AI. While RL, especially Deep RL, often works well with immediate rewards but may fail when rewards are delayed or noisy, or both. In this work, we propose delegating the CAP to a Neural Network-based algorithm named *InferNet* that explicitly learns to infer the immediate rewards from the delayed and noisy rewards. The effectiveness of InferNet was evaluated on three online RL tasks: a GridWorld, a CartPole, and 40 Atari games; and two offline RL tasks: GridWorld and a real-life Sepsis treatment task. The effectiveness of InferNet rewards is compared to that of immediate and delayed rewards in two settings: with and without noise. For the offline RL tasks, it is also compared to a strong baseline, InferGP [7]. Overall, our results show that InferNet is robust to delayed or noisy reward functions, and it could be used effectively for solving the temporal CAP in a wide range of RL tasks, when immediate rewards are not available or they are noisy.**

***Index Terms*—Credit Assignment Problem, Deep Reinforcement Learning**

I. Introduction

A large body of real-world tasks can be characterized as sequential multi-step learning problems, where the *outcome* of the selected actions is *delayed*. Discovering which action(s) are responsible for the delayed outcome is known as the ***(temporal) Credit Assignment Problem (CAP)*** [5], [25]. Solving the CAP is especially important for *delayed reinforcement* tasks [40], in which r_t, a reward obtained at time t, can be affected by all previous actions, a_0, a_1, ..., a_{t-1}, a_t and thus we need to assign credit or blame to each of those actions individually.

Prior research has explored solving the CAP by formulating it as a Reinforcement Learning (RL) problem [10], [39], in which an agent learns how to interact with a potentially non-stationary, stochastic, and partially observable environment to maximize the long-term cumulative reward. Such delayed reinforcement tasks become extremely challenging if there are long delays between the actions and their corresponding outcomes. Furthermore, noise is a common problem that machine learning systems need to deal with in many real-world situations. In RL, noise can be present in either the observations or the rewards. When the reward signal is noisy, the agent can have a difficult time learning the optimal behavior, since the rewards can be biased or have large variances. Previously, Temporal Difference (TD) learning methods [41] have been widely used to resolve CAP problems [42]. In particular, the TD(λ) algorithm [41], [43] uses eligibility traces to update the value of a state by using all the future rewards in the episode, which makes it easier to assign credit for long trajectories. For example, Mesnard et al used Counterfactual Credit Assignment to create a future-conditional value function that considers the delayed reinforcement [24].

One way to mitigate the CAP is to use model-based RL or simulations, which allow collecting vast amounts of data. However, in many real-life domains such as healthcare, building accurate simulations is especially challenging because disease progression is a rather complex process; moreover, it is not feasible or ethical to induce medical treatment policies while interacting with patients. In addition, reward functions in such domains are often both delayed and noisy, so solving the CAP is essential. Typically, in healthcare, the most appropriate rewards are patient outcomes, which can only be obtained after the entire trajectory has been completed. The reason for this delay is that the subtle yet complex progression of the disease makes it difficult to assess patient health state moment to moment, and that many clinical or medical interventions that boost short-term performance may not be effective over the long run. Moreover, immediate reward mechanisms in these domains are often imperfect representations of underlying true reward mechanisms. As an example of inaccurate or noisy

patient outcomes, the 30-day readmission rate among sepsis survivors across 633,407 hospitalizations and 3,315 hospitals is 28.7% [27].

A lot of prior research has investigated modifying the reward function through *reward shaping* or *reward engineering* [16], [22], [23], [38], [46]. One main application of reward shaping has been including intrinsic rewards to the agent's reward signal, where these intrinsic rewards encourage the agent to explore unseen states [29]. This approach improves the exploration policy of the agent and mitigates the *sparse reward* problem. Chen and Lin used self-imitation learning to help a robot learn in a sparse reward setting [11]. Other work by Trott et al. [44] used a distance-based metric to shape a reward function that enhances the sparse rewards from the environment and prevents the agent from getting stuck in local optima. Most prior methods develop a separate reward function that *augments* the existing reward function.

To mitigate the temporal CAP, our approach is to *distribute or infer* the final delayed reward into the immediate rewards along the trajectories. Previously, we presented InferGP [7], which uses Gaussian Processes (GP) to infer unobservable immediate rewards from delayed rewards and then the inferred rewards can be used in standard RL algorithms for policy induction. That work, however, had *three crucial limitations*: 1) it did not examine how InferGP handled noisy reward functions, even though GP is widely recognized to be robust to noise; 2) InferGP is not very scalable due to its poor time and space complexity, as shown in section IV-A; and 3) While most RL algorithms are trained online, InferGP can only be used for offline-RL since it incorporates all training data when applying Bayesian inference to infer the rewards. In offline RL, the agent learns the policy from pre-collected data, while in online RL, the agent learns while interacting with the environment. Many RL tasks involve online learning, and many RL algorithms, especially Deep RL (DRL) algorithms often require millions or billions of interactions in order to develop competitive policies. As InferGP can only work offline, it becomes impractical for large datasets or for online RL.

In this work, we propose an intuitive and powerful *Neural Network (NN) based* approach named **InferNet**, which infers "immediate rewards" from the delayed and noisy rewards and can be used to train an RL agent. InferNet is a general, *scalable* mechanism that works alongside any online and offline RL algorithms. It is an easy yet effective add-on mechanism for mitigating the temporal CAP. The effectiveness of InferNet was evaluated on three online RL tasks: a GridWorld, the CartPole task and 40 Atari games; and two offline RL tasks: GridWorld and a real-world Sepsis treatment task. For simplicity, in the following, policies resulting from InferGP and InferNet inferred rewards will be called *InferGP* and *InferNet policies*, respectively. Likewise, policies that are generated by using immediate or delayed rewards are referred to as *immediate* or *delayed policies*. Note that 1) the InferGP policy is only applicable to offline RL tasks, and 2) immediate rewards are not available for the Sepsis treatment task.

Algorithm 1 InferNet Online

Initialize InferNet buffer $D \leftarrow ()$
// Pretrain InferNet
for $episode \leftarrow 1$ to K **do**
  Play an episode randomly and collect the data
  Delayed reward $R_{del} = r_0 + r_1 + .. + r_{T-1}$
  $D \leftarrow D \cup (s_0, a_0, ..., s_{T-1}, a_{T-1}, R_{del})$
  Train InferNet on a batch of episodes $B \sim D$: $L(\theta) = (R - \sum_{t=0}^{T-1} f(s_t, a_t)|\theta))^2$
end for
for $episode \leftarrow 1$ to M **do**
  Set episode data sequence $tmp \leftarrow ()$
  while not end of episode **do**
    Get state s from env, select action $a \sim \pi$, and execute on environment $s', r \sim env(s, a)$
    $tmp \leftarrow tmp \cup (s, a, r, s')$
    Train RL agent on historical data with inferred rewards
    Train InferNet on a batch of episodes $B \sim D$: $L(\theta) = (R - \sum_{t=0}^{T-1} f(s_t, a_t)|\theta))^2$
  end while
  Use InferNet to infer rewards for the steps in tmp
  $D \leftarrow D \cup tmp$
  Replace rewards in tmp with InferNet rewards
  Store the tmp data (with the InferNet rewards) to train the RL agent later on
end for

For the three online RL tasks, the InferNet policy is evaluated against the immediate and delayed policies under two reward settings: Noise-free (no noise) and Noisy. On the offline RL tasks, InferNet is evaluated using both noise-free and noisy reward settings on the GridWorld first, and then on the sepsis shock prevention task, where the rewards are delayed and noisy. Overall, our results show that the effectiveness of InferNet is robust against noisy reward functions and is an effective add-on mechanism for solving temporal CAP in a wide range of RL tasks, from classic RL simulation environments to a real-world RL problem and for both online and offline learning.

II. InferNet: Neural Net Inferred Rewards

a) Problem Definition: In conventional RL, an agent's interactions with an environment are often framed as a Markov decision process (MDP), where at each time-step t the agent observes the environment in state s_t, it takes an action a_t and receives a scalar reward r_t and the environment moves to state s_{t+1}. In the discrete case, a_t is selected from a discrete set of actions $a_t \in A = \{1, ..., |A|\}$. The RL agent is tasked with maximizing the expected discounted sum of future rewards, or return, defined as $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_t$, where $\gamma \in [0, 1]$ is the discount factor and T is the last timestep in the episode. A value function is commonly used to estimate the expected return for each state or state-action pair. The optimal action-value function is defined as $Q^*(s, a) = max_\pi Q^\pi(s, a)$, where

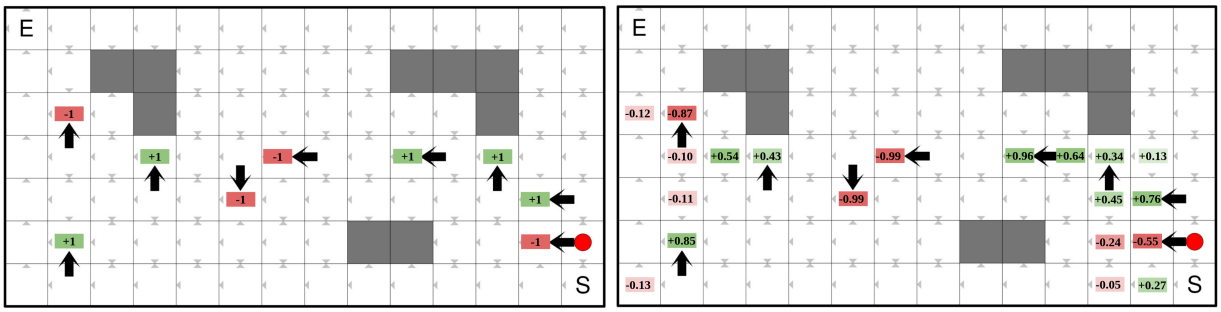


Fig. 1: Left: The original GridWorld with the true immediate rewards. Right: The InferNet predicted rewards.

Q^π estimates the long-term reward the agent would observe after taking action a from state s and following policy π thereafter.

b) InferNet: The intuition behind InferNet is rather straightforward. InferNet uses a deep neural network to infer the immediate rewards from the delayed reward in an episode. At each timestep, the observed state and action are passed as input to the neural network, which will output a single scalar, the inferred immediate reward for that state and action: $r_t = f(s_t, a_t|\theta)$. Here θ indicates the parameters (weights and biases) of the neural network. To address the credit assignment problem, InferNet distributes the final delayed reward among all the states in the episode. More specifically, the network learns to infer the immediate rewards from the delayed reward by applying a constraint on the predicted rewards: the sum of all the predicted rewards in one episode must be equal to the delayed reward, as shown in Equation 1 where R_{del} indicates the delayed reward. This way, the network needs to model the reward function, conditioned on the state-action pair for each timestep, and it will minimize the loss between the sum of predicted rewards and the delayed reward for each episode.

$$R_{del} = f(s_0, a_0|\theta) + f(s_1, a_1|\theta) + ... + f(s_{T-1}, a_{T-1}|\theta) \quad (1)$$

We used the *TimeDistributed* layer on TensorFlow Keras [1], [12] so that we can repeat the same neural network operation multiple times, sharing weights across time, and pass the entire episode at once as input to the neural network. It should be noted that despite sharing weights across time, there is no internal state that is passed to the next timestep (as in a recurrent neural network). Each output is only dependent on the state and action passed as inputs at that timestep. We train InferNet by minimizing the loss function shown in Eq. 2.

$$Loss(\theta) = (R_{del} - \sum_{t=0}^{T-1} f(s_t, a_t|\theta))^2 \quad (2)$$

Algorithm 1 shows the pseudo-code for training InferNet alongside an RL algorithm. In this algorithm, the first loop is used to collect data and pre-train InferNet, without training the RL agent. The second loop, is the standard RL loop. The agent interacts with the environment collecting data, and both the agent and the InferNet neural network get trained. The main difference in our algorithm is that the RL agent is trained on the InferNet rewards instead of the regular rewards.

This process can be seen as making the neural network learn a function that outputs a reward for each state-action pair, subject to the constraint of all rewards in an episode summing up to the delayed reward for that episode. To evaluate the effectiveness of InferNet, we divide the experimental evaluation into online and offline RL tasks. Note that **Table I** summarizes all the hyper-parameters used in different experiments.

III. Online RL Experiments

The effectiveness of InferNet is investigated on three types of tasks: a GridWorld, the CartPole, and 40 games using the Atari 2600 Learning Environment. We compare the following reward settings: 1) *Immediate rewards:* when available, they are the gold standard. 2) *Delayed rewards:* these rewards are used as a baseline. All the intermediate rewards will be zero and the reward that indicates how good or bad the intermediate actions were will only be provided at the end of the episode. When the rewards are not delayed by nature, we simulate the delayed rewards by "hiding" the immediate rewards and providing the sum of all the immediate rewards at the end of the episode, as one big delayed reward, following the idea of reward accumulation proposed by Arjona et al. [4]. 3) *InferNet rewards:* our proposed method which uses a Neural Network to predict the immediate rewards from the delayed reward.

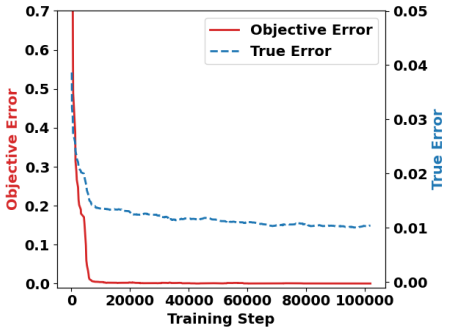
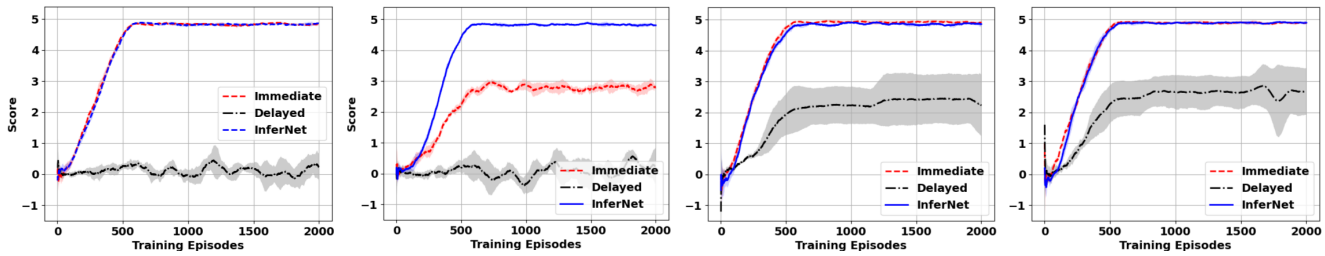


Fig. 2: Training process for InferNet. Minimizing the objective loss (red line) results in also minimizing the true loss (blue line), which is what we want to ultimately achieve.



(a) Q-Learning (no noise) (b) Q-Learning (noisy) (c) TD(λ) (no noise) (d) TD(λ) (noisy)

Fig. 3: Results of training a Q-Learning agent (a) and (b) and a TD(λ) agent (c) and (d) on the GridWorld task, for immediate, delayed, and InferNet rewards. No noise in the rewards (a) and (c). Gaussian noise ($\mathcal{N}(0, 0.08^2)$) added for (b) and (d).

For each type of tasks, we also evaluate the power of InferNet when the reward function is noisy.

A. *Grid World*

Fig. 1 (left) shows the GridWorld environment used in this work as a standard RL testbed. Here, we can compare the true immediate rewards to the inferred rewards produced by InferNet. The three available actions are: move up, left, and down. To prevent the agent from collecting the same reward more than once, the states that have a reward of +1 or -1 only have one action available: move left. The inferred rewards predicted by InferNet are shown in Fig. 1 (Right), and it shows that InferNet can recover the original reward setting effectively. The rewards in the range (-0.03, 0.03) have been intentionally left blank to simplify the visualization. It should be noted that in this environment, the rewards are a function of the state and the action. For this reason, Fig. 1 (Right) only shows the largest (positive or negative) reward in each state.

Fig. 2 shows that by minimizing the training error in Eq. 2 (the difference between the delayed reward and the sum of immediate predicted rewards) (red line), InferNet minimizes the true error (the difference between the predicted and true immediate rewards) (blue line). We then evaluate the effectiveness of the rewards predicted by InferNet when used to train an RL agent, and compare them to the immediate and delayed rewards. We repeated each experiment five times with different random seeds, and show the mean and standard deviation of those runs. We explored Q-Learning and TD(λ) as RL agents; the latter is known for being able to mitigate the temporal CAP.

Q-Learning is a TD-Learning method, but it does not employ any eligibility traces, and it usually only uses a 1-step reward. Fig. 3 (a) shows the result of training different Q-Learning agents on the three reward settings without noise. These results clearly show that the Q-Learning agent that uses the delayed rewards cannot learn to solve the simple GridWorld task; it is no better than a random agent. However, when we first use InferNet to infer the "immediate" rewards from the delayed ones, the agent is able to solve the task as effectively as the agent uses the immediate rewards. When the reward function is noisy, Fig. 3 (b) shows that the immediate reward agent suffers significantly, and cannot solve the environment completely, while InferNet is much more robust to the noise.

TD(λ) is known to be one of the strongest methods to tackle the CAP. This algorithm takes advantage of the benefits of TD methods and includes eligibility traces, which allows the agent to look at all the future rewards to estimate the value of each state. This makes propagating the delayed reward easier than in the case of 1-step rewards. Despite all these advantages, Fig. 3 (c) shows that when the rewards are delayed, the agent is not able to learn as effectively as the agent that has access to the true immediate rewards. However, the agent that uses the InferNet predicted reward achieves the same performance as the agent that uses the immediate rewards; they can both fully solve the environment. When the reward function is noisy, Fig. 3 (d) shows that none of the agents suffer. This result shows that TD(λ) is more robust to noisy rewards than Q-Learning.

B. *CartPole*

Fig. 4 (left) shows the CartPole environment. The goal in CartPole is to move a cart left and right in order to keep a pole balanced in its vertical position. The reward function provides a rewards of +1 for each step where the pole is kept vertical. This means that the reward function InferNet needs to learn is pretty simple, a reward of +1 for each timestep, regardless of the state and action passed as inputs. The difficulty in this environment lies on the continuous state space, that makes tabular RL methods inoperable. For this reason, we compared the same three reward settings using the Deep Q-Network (DQN) algorithm [26]. For the InferNet setting, we trained

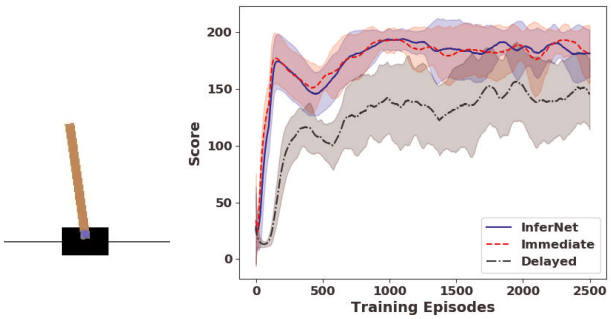


Fig. 4: Left: CartPole environment. Right: Comparison of immediate, delayed and InferNet policies in the CartPole task.

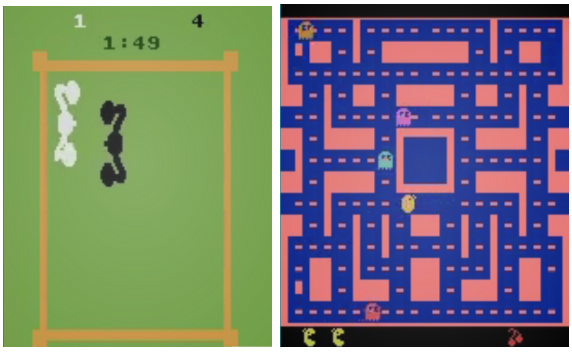


Fig. 5: Examples of two Atari Learning Environment games. Left: Boxing. Right: MsPacman.

the DQN agent [26] and InferNet simultaneously, providing the DQN agent with the rewards produced by InferNet.

Fig. 4 (Right) shows the results of using the immediate, delayed and InferNet rewards. It shows the mean and standard deviation of training each agent with 20 different random seeds as a function of the number of training steps. These results show that InferNet can work effectively in combination with a Deep RL agent, to mitigate the credit assignment problem and boost the performance of the agent when only delayed rewards are available, and can perform as well as the agent that uses the immediate rewards. Meanwhile, the agent that learns from the delayed rewards cannot completely solve this task, although it is able to get a reasonable score. We repeated the experiment with noisy rewards, but none of the agents' results were affected by the noise, so we did not include a separate comparison. It's probably because a reward of +1 is given at each step in the CartPole. Thus the agent can still figure out that its goal is to balance the cart as long as possible, even if there is noise on the rewards.

C. Atari Learning Environment (ALE)

The ALE provides visually complex environments in which the state space is very high dimensional, represented by pixels on a screen. Fig. 5 shows two examples of these games. It is important to note that in some games, an episode consists of thousands of steps, so learning from a single delayed reward is no trivial task. We used OpenAI gym [9] to simulate the games, and the stable baselines library [14] to train the Deep RL agent. Here, we evaluate the performance of InferNet in conjunction with a Prioritized Dueling DQN agent [34], [45].

Network Architecture The NN architecture for Deep RL agent was the same as in [45]. InferNet also uses the same architecture as the agent. The only differences were 1) the output layer consists of a single value for InferNet, 2) InferNet uses dropout during training for regularization, and 3) we wrapped all the InferNet layers in the TimeDistributed layer, in order to be able to pass all the steps in an episode as input at once, and infer the immediate reward for each of those steps (see Algorithm 1). It should be noted that the hyper-parameters (including the replay buffer size) are different from those in the Dueling DQN paper. We allowed for these changes because our goal is not to outperform the current state of the art method or to improve upon some prior Deep Reinforcement Learning algorithm. Our task is to create a method that infers better rewards and can be used by other RL algorithms in order to learn more effectively. Modifying some of these hyper-parameters allowed for less expensive training.

Fig. 6 shows the results of a Prioritized Dueling DQN agent on 40 Atari games with the two reward settings: noise free (Left) and noisy (Right). The black vertical line at $x = 0$ in each subgraph represents the baseline performance obtained when training DQN using delayed rewards. This allows us to compare the three types of rewards, after normalizing the performance for each game. It is important to know that as we did not intend to outperform any other algorithm or state-of-the-art method, our evaluation was different from the Dueling DQN paper in several ways: 1) the size of the experience replay buffer was 10,000 instead of 1,000,000; 2) the evaluation was performed by making the loss of a life indicate the end of the episode; 3) we used the default hyper-parameters in the stable baselines library.

Noise-free Rewards: Fig. 6 (Left) shows the results of training the Dueling DQN agent on the three different reward settings. The agent trained on the rewards provided by InferNet performs as well as or better than the agent which uses the delayed rewards in almost all games. In some cases, it can even match the performance of the agent in the immediate setting. These results clearly show that when immediate rewards are not available, using our InferNet is preferable to overly using the delayed rewards.

Noisy Rewards: We repeated the Atari experiments after adding Gaussian noise to the observed rewards. As the noise is unbiased, the expectation of the sum of rewards is the same with and without noise, as shown in Eq. 3.

$$\begin{aligned}\mathbb{E}[R] = \mathbb{E}[r_0 + ... + r_{T-1}] = \\ \mathbb{E}[r_0 + \mathcal{N}(0, \sigma^2) + ... + r_{T-1} + \mathcal{N}(0, \sigma^2)]\end{aligned} \tag{3}$$

Fig. 6 (right) shows the results of training the same Prioritized Dueling DQN agent on noisy rewards with immediate, delayed, and InferNet rewards. When compared to the noise-free immediate rewards (Left), the agent trained on noisy immediate rewards performs significantly worse. When compared to the noise-free setting, InferNet provides better performance in many games. Fig. 7 shows two examples of this (Seaquest and Freeway).

Impact of Different Noise Levels We also examine how different levels of noise affect different reward settings. Three distinctly different Atari games were studied: Centipede, Freeway, and Seaquest. The six levels of standard deviations (0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6) of the Gaussian distribution that generated the white noise were explored (the mean remained zero). Figs. 8, 9, and 10 show the mean performance of the agent over two runs with different random seeds. Overall, among the three types of rewards, using InferNet rewards results in the best overall performance and robustness against different levels of noise.

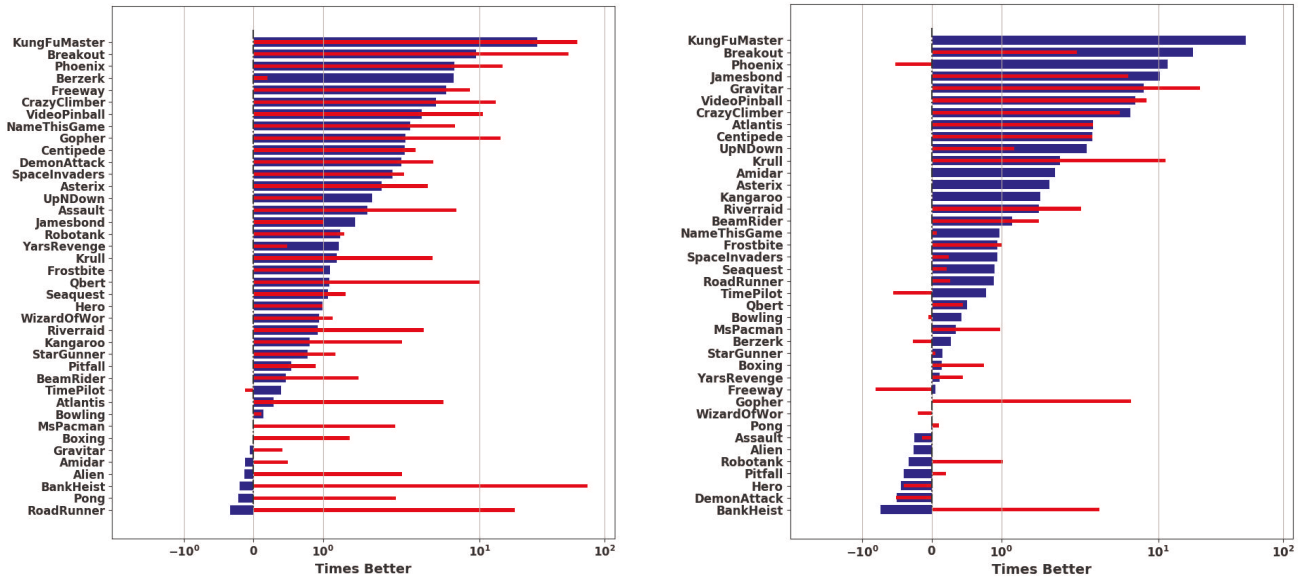
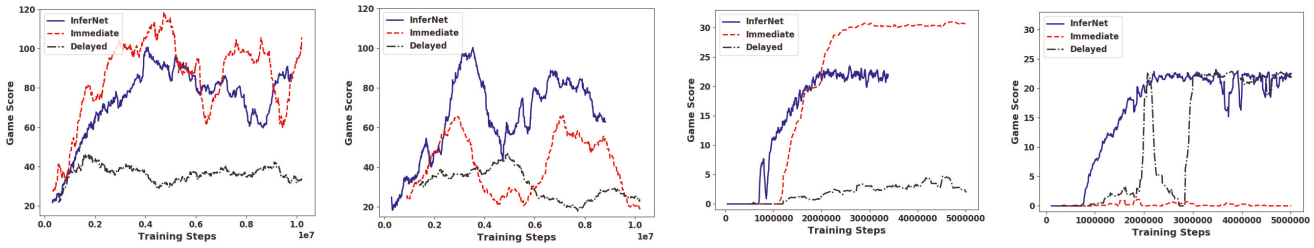


Fig. 6: Performance of the Prioritized Dueling DQN agent on the different Atari games with the three reward settings: Delayed (vertical line at x=0), Immediate (red) and InferNet (Blue). Noise-free rewards (Left) and with noisy rewards (Right). The results have been normalized to show the Delayed rewards at x=0 as a vertical line, and each bar shows how many times better the other agents are.



(a) Seaquest (no noise) (b) Seaquest (noisy) (c) Freeway (no noise) (d) Freeway (noisy)

Fig. 7: Training process for Seaquest (a) and (b), and Freeway (c) and (d). No noise added for (a) and (c). Gaussian noise ($\mathcal{N}(0, 0.6^2)$) added for (b) and (d).

IV. Offline RL Experiments

When applying RL to solve many real-life tasks such as healthcare, we have to perform offline learning. That is, the training of the RL agent needs to be done from a fixed training dataset, and no further exploration of the environment is possible. In these situations, having a method that effectively solves the CAP is crucial. For the offline experiments, InferNet is also compared against InferGP. Our prior work has shown that InferGP works reasonably well in a wide range of offline RL tasks [7]. Our goal is to determine the effectiveness of InferNet for offline RL tasks when compared to immediate, delayed, and InferGP rewards. We use the same GridWorld environment as in Section III-A. Additionally, we want to evaluate our method in a real-world problem: a healthcare task where the goal of the agent is to induce a policy for sepsis treatment and septic shock prevention.

A. GridWorld

In this offline RL experiment, we use the same GridWorld as in section III-A. However, to make the training happen offline, we first use a random policy to interact with the GridWorld randomly and collect gameplay data. Then, we use the delayed rewards in the data to infer the immediate rewards, and finally we train an RL agent with the newly inferred rewards.

Root Mean Squared Error (RMSE): We evaluated the amount of data needed for InferNet and InferGP to approximate the true immediate rewards. We calculated the RMSE between the inferred rewards and the true immediate rewards in the training dataset by varying the amount of training data. Fig. 11 (Left) compares this RMSE for InferNet and InferGP. Overall, with 100 or more trajectories, InferNet consistently has a lower RMSE than InferGP. Fig. 11 (Right) compares the RMSE of the two approaches with noisy rewards. Adding

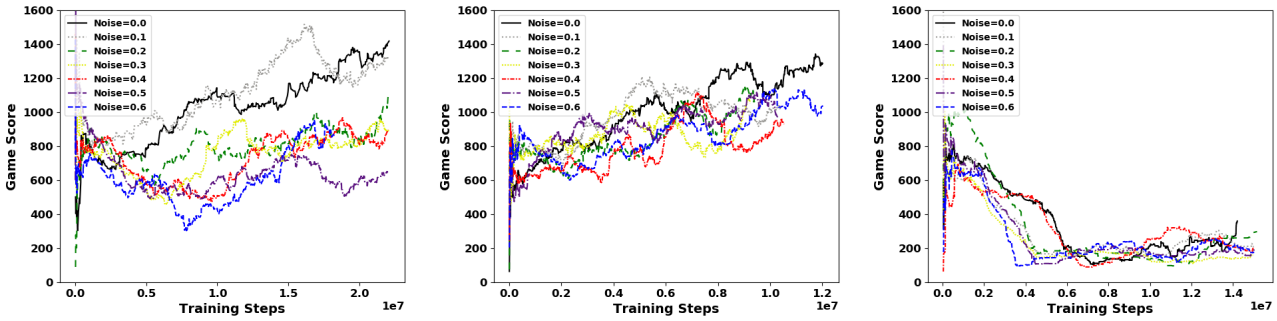


Fig. 8: Training on Centipede with different noise levels. Left: Immediate. Center: InferNet. Right: Delayed.

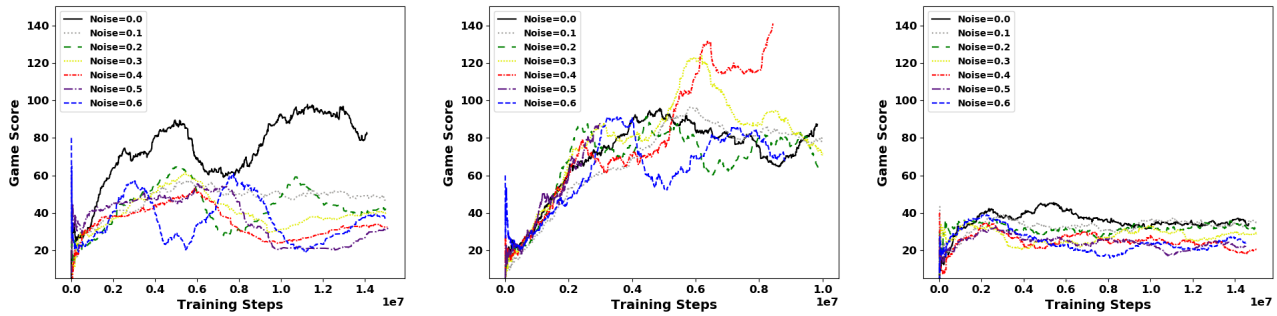


Fig. 9: Training on Seaquest with different noise levels. Left: Immediate. Center: InferNet. Right: Delayed.

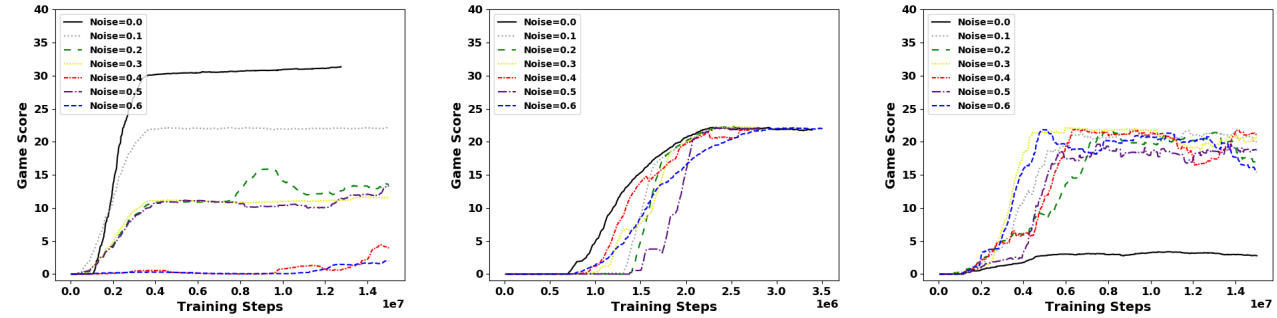


Fig. 10: Training on Freeway with different noise levels. Left: Immediate. Center: InferNet. Right: Delayed.

noise makes the CAP more challenging, and InferGP cannot adapt as well as InferNet (InferGP increases from 0.15 to 0.5 and InferNet increases from 0.13 to 0.2), despite the fact that GP is known to be able to handle noisy data.

Offline Q-Learning: We trained a tabular Q-learning agent for 5000 iterations on the same dataset used to infer the rewards. We compared the four reward settings: immediate, delayed, InferGP, and InferNet. Once our RL policies are trained, their effectiveness is evaluated online by interacting with the GridWorld environment directly. Fig. 12 (left) shows the mean and standard deviation of the performance of the agent when interacting with the environment for 50 episodes, as a function of the number of episodes available in the training dataset. Fig. 12 (left) shows that, as expected, the delayed policy performs poorly, while the Immediate policy can converge to the optimal policy after only 10 episodes of data; additionally, both InferNet and InferGP can converge to the optimal policy but they need more training data (around 150 episodes) than the Immediate policy. Fig. 12 (Right) shows the performance of the policies when the rewards are noisy. It clearly shows that adding noise to the rewards function deteriorates the Immediate policy, while InferNet and InferGP policies also suffer but it seems that InferNet is the best option among the four.

Time Complexity: Fig. 13 empirically compares the time complexity of InferNet and InferGP: the training time of InferNet is less sensitive to the size of the training dataset, while the training time of InferGP increases cubically as the training data increases. Fundamentally, InferGP has an asymptotic time complexity of $O(n^3)$ and asymptotic space complexity of $O(n^2)$, where n refers to the size of the dataset. InferNet has a time complexity of $O(n)$ since we sample a constant amount of mini-batches from the dataset for each gradient descent step, and we only need to train the network

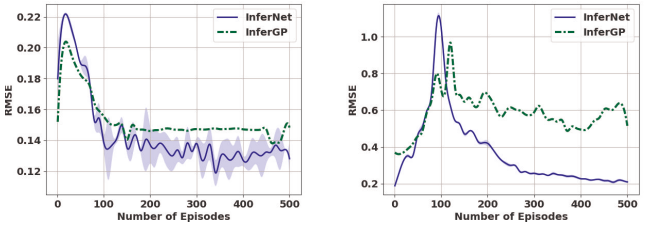


Fig. 11: RMSE between the inferred and the true immediate rewards as a function of the number of episodes collected from the GridWorld environment. Left: No noise. Right: Gaussian noise ($\mathcal{N}(0, 0.3^2)$).

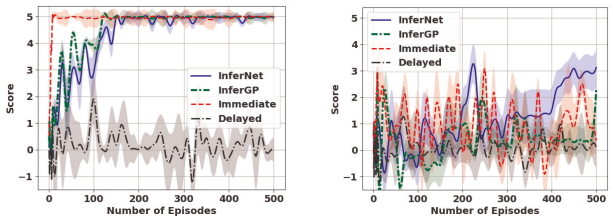


Fig. 12: Performance of Q-Learning agents on the GridWorld environment as a function of the number of training episodes. Left: No noise. Right: Gaussian Noise ($\mathcal{N}(0, 0.3^2)$)

for a constant number of epochs. The space complexity for InferNet is $O(f * l)$, where f is the number of features in the state and action that are passed as inputs, and l is the length of the episode that is passed as input.

B. Healthcare

We evaluate InferNet and InferGP, on a real-world sepsis treatment task. Sepsis is a life threatening infection, and the septic shock, the most severe complication of sepsis, leads to a mortality rate as high as 50% [32]. Our goal is to learn an optimal treatment policy to prevent patients from going into septic shock. Since as many as 80% of sepsis deaths could be prevented with timely diagnosis and treatment [18], it is crucial to monitor sepsis progression and recommend the optimal treatment as early as possible.

In this task, we used Electronic Health Records (EHRs) where the actions (treatment) are sparse and the rewards (patients' health status) are both delayed and noisy. By the nature of human bodies, the reactions to certain treatment such as medication or oxygen assistance would not be immediately observed, and thus it is challenging to make a long-term treatment plan for an individual patient. If the agent can estimate immediate rewards of treatment from delayed rewards, that could help to improve the decision making process of medical treatment. Recently, several DRL approaches have been investigated for septic treatment, utilizing EHRs. However, [17], [30] only considered delayed rewards, while [7] leveraged the Gaussian process based immediate reward inference method, which is one of our baselines.

Data: Our EHRs were collected from Christiana Health Care System (July 2013 to December 2015). We identified $2,964$ septic shock positive visits and sampled $2,964$ negative visits based on the expert clinical rules, keeping the same ratio of age, gender, race, and the length of hospital stay as in the original EHRs. We selected 22 sepsis-related state features such as vital signs, lab results, and medical interventions, and defined four types of treatments as actions: no treatment, oxygen control, anti-infection drug, and vasopressor.

Reward: The rewards were assigned by the expert-guided reward function based on the multiple septic stages. These rewards are delayed in time and noisy. The delayed rewards are given when the patient goes into septic shock or recovers at the end of their stay, and the noise in the rewards is a result of imperfect sensors or incomplete measurements.

Experiment setting: We compared three reward settings: the original delayed, InferGP, and InferNet, since the immediate rewards are not available. InferNet predicts one reward for each timestep. After inferring the immediate rewards, the septic treatment policies were induced using a Dueling DQN agent. We split data into 80% for training, 20% for test using the stratified random sampling keeping the same ratio of septic shock. The hyper-parameters are shown in Table I.

Evaluation metric: In similar fashion to prior studies ([7], [17], [30]), the induced policies were evaluated using the *septic shock rate*, which is the portion of shock-positive visits in each group belonging to the corresponding agreement rate, as the agreement rate increases from 0 to 1 with a 0.1-rate interval. It is desirable that the higher the agreement rate, the lower the septic shock rate. The policy agreement rate in the non-shock patients should be higher, which means that our policy agrees more with the physicians' actions for the non-shock patients.

Results: The underlying assumption is that the agreed treatments are adequate and acceptable, as they were taken by real doctors in real clinical cases, but may not necessarily optimal. Fig. 14 shows the *septic shock rate* in the visiting group for the training (Left) and test (Right) sets, as a function of the corresponding agreement rates.

For InferNet, the septic shock rate almost monotonically decreases in the test set evaluation, as the agreement rate increases and reaches the lowest shock rate of all policies.

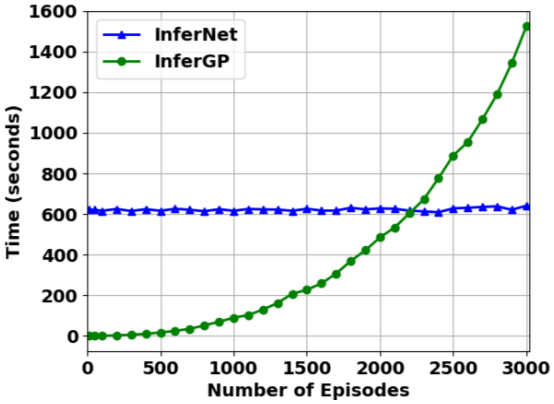


Fig. 13: Time analysis of training InferNet and InferGP.

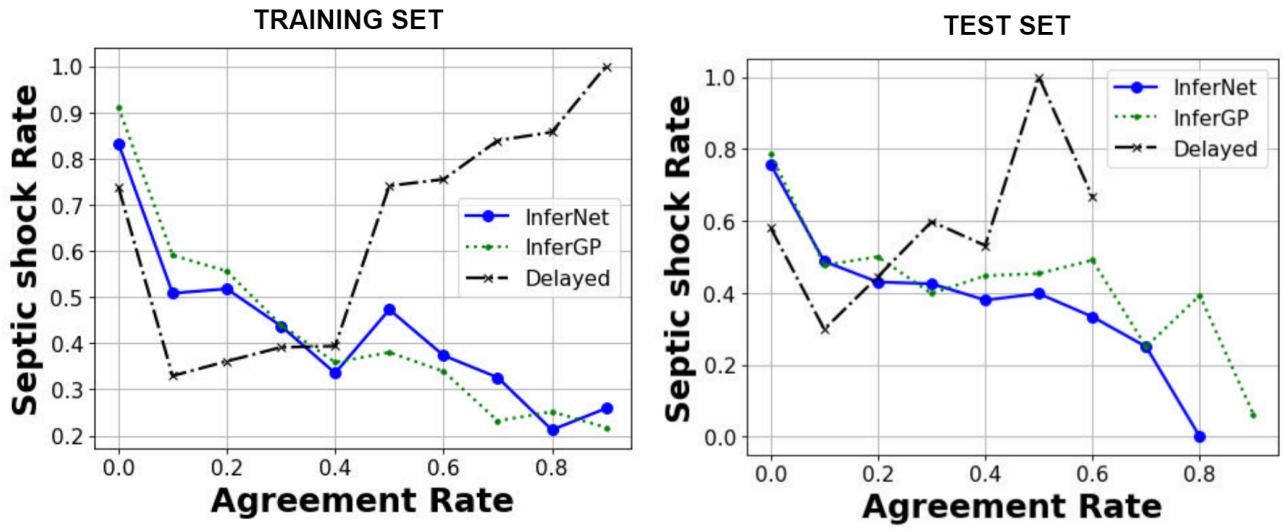


Fig. 14: Healthcare: Septic shock rate as a function of the agreement rate between the policy and the physician actions for the training (Left) and test (Right) sets.

InferGP shows a general trend of decreasing shock rate with a larger variance than InferNet as the agreement rate increases, while delayed fails to learn an effective shock prevention policy. This supports that InferNet significantly improves the policy training process at preventing septic shock, compared with InferGP and delayed. Furthermore, InferNet and InferGP induced policies that agree with the physicians more for the non-shock patients, while when the patients are more likely to go into septic shock, the agents try to search for a different treatment strategy from the given treatments that resulted in septic shock.

V. Hyper-parameters

Table I shows the hyper-parameters for all the experiments in this work. The dashes indicate that the hyper-parameter was not used, either because the training was performed offline or because of a design decision. The hyper-parameters were chosen to balance the computational cost and the performance of the agents. In the case of TD(λ) for the online GridWorld, we performed a grid search with the values for λ and α, as well as the type of traces to be used (we considered accumulating, replacing and dutch traces), and chose the hyperparameters that worked best across the different reward settings.

VI. Related Work

By utilizing deep learning and novel RL algorithms, Deep RL (DRL) has shown great success in various complex tasks [8], [36]. Much of prior work on DRL has focused on online learning where the agent learns while interacting with the environment. Immediate rewards are generally much more effective than delayed rewards for RL because of the CAP: the more we delay rewards or punishments, the harder it becomes to assign credit or blame properly. Different approaches have been proposed and applied for solving the CAP. For example, when applying DRL for games such as Chess, Shogi and Go, the final rewards are determined by the outcomes of the game: $-1/0/+1$ for loss/draw/win respectively; and for each state, Monte Carlo Tree Search (MCTS) was used to learn the likelihood of each outcome [36], [37]. Because of the CAP, RL algorithms often need more training data to learn an effective policy using delayed rewards than using immediate rewards. More importantly, for some extremely complicated games, DRL may fail to learn an effective policy altogether. As a result, prior research used expert-designed immediate rewards, or learned a reward function from expert experience trajectories, using reward engineering methods such as Inverse RL [2], [20], [31], [48]. For example, Berner et al. used human-crafted intermediate rewards to simplify the CAP. They designed a reward function based on what expert players agree to be good in that game [8]. While effective, such expert-designed rewards are often labor-intensive, expensive, and domain specific. Additionally, these expert rewards might introduce expert bias into the process, leading to sub-optimal agent performance, as shown by AlphaGo Zero [37] outperforming the original AlphaGo [35].

The human brain can be very efficient at solving the CAP when learning to perform new tasks [6], [33]. Thus a wealth of neuroscience research focuses on understanding the learning and decision-making process in animals and humans. For example, [3] studied the structural and temporal CAP and suggested unification of the problem for multi-agent, time-extended problems. In RL, the temporal CAP has been widely studied [40], and solutions to it have been proposed in order to more successfully train neural network systems [19], [28].

In machine learning, prior research investigated solving the CAP by formulating it as an RL task [39]. To the best of our knowledge, the best-known family of algorithms to tackle the CAP are the Temporal Difference (TD) Learning methods and TD(λ) in particular [41]. It employs eligibility traces to

TABLE I: Hyper-parameters used for the different experiments.

Parameter Name	GW Online	GW Offline	Healthcare	CartPole	Atari
InferNet Hidden Layers	3 Dense	3 Dense	3 Dense	3 Dense	3 Conv + 1 Dense
InferNet Num. Units	3x256	3x256	3x256	3x64	Conv: 32, 64, 64. Dense: 512
InferNet Activation	Leaky ReLU	Leaky ReLU	Leaky ReLu	ReLU	ReLU
InferNet Dropout Rate	—	—	0.2	0.2	0.2
InferNet Optimizer	Adam	Adam	Adam	Adam	Adam
InferNet Learning Rate	1e-4	1e-3	1e-4	1e-4	3e-3
InferNet Batch Size	32 ep.	32 ep.	20 ep.	10 ep.	1 ep.
InferNet Training Steps	500,000	50,000	1,000,000	60,000	Varying per game
InferNet Buffer Size	500	—	—	500 ep.	500 ep.
Agent Training Steps	2,000 ep.	5,000	1,000,000	150,000	Varying per game
Agent Discount γ	0.9	0.90	0.99	0.99	0.99
Agent Batch Size	—	32	32	32	32
Agent Buffer Size	—	—	—	500,000	10,000
Agent Hidden Layers	—	—	2 Dense	2 Dense	3 Conv + 1 Dense
Agent Num. Units	—	—	2x256	2x32	Conv: 32, 64, 64. Dense: 512
Agent Activation	—	—	ReLU	ReLU	ReLU
Agent Learning Rate	—	—	1e-4	2.5e-4	1e-4
TD(λ): λ	0.91	—	—	—	—
TD(λ): α	0.1	—	—	—	—
TD(λ): trace type	Dutch	—	—	—	—

use all the future rewards when updating the value of each state, resulting in the better assignment of credit/blame for each action. More recently, the Expected TD(λ) or ET(λ) has been proposed [21], where the concept of Expected Eligibility Traces is introduced, to introduce an algorithm that has lower variance than the TD(λ) alternative.

The sparse reward issue [11] is closely related to the temporal CAP, in that both need to learn from very few and delayed rewards. To mitigate the problem of sparse rewards, *reward shaping* and *reward engineering* have been studied in the past [16], [22], [23], [38], [46]. Prior work has also studied using intrinsic motivation as an added reward, to encourage the agent to explore the environment more effectively [29], [44]. Intrinsic rewards have also been used to improve the performance of RL agents in combination with a policy gradient algorithm [47], and showed that these new rewards results in better performance on the Atari games.

Closely related to this work, RUDDER uses a neural network-based approach for addressing the temporal CAP [4]. However, RUDDER does not predict the immediate rewards directly with the neural network; rather, it predicts the final return, and then uses the differences in predictions to redistribute the immediate rewards. Their results show that RUDDER can greatly improve the training of the RL agent in several simulated tasks as well as four Atari games. Prior work has also helped memory-based RL agents solve the CAP, as shown by the work by Hung et al. [15], where they develop an algorithm named Temporal Value Transport (TVT), which predicts future rewards with a Neural Network, and then adds the prediction term to the bootstrapped term of the Bellman Equation. A different family of algorithms to tackle the CAP was suggested by Harutyunyan et al. [13]. They denote the family of methods they developed as Hindsight Credit Assignment (HCA). The core idea in their work is to sample a trajectory, and use the importance sampling ratio to assign credit to the actions in that trajectory. They show that their method outperforms the default policy gradient method in three simple RL tasks.

VII. Conclusion

We propose and evaluate an intuitive and useful deep learning mechanism named InferNet that explicitly tackles the CAP by generating immediate rewards from the delayed rewards. Our results show that our algorithm makes it easier for the RL algorithm of choice to solve the task at hand, both for online and offline RL while mitigating the problem generated by noisy rewards. We showed that InferNet can accurately predict the true immediate rewards on a simple GridWorld and help Q-Learning and TD(λ) agents solve the environment. We showed that it can be used for slightly more complex tasks such as the CartPole, where the InferNet policy is as good as the immediate policy. Furthermore, we showed that our algorithm scales to large datasets and to online RL, which allows it to help solve more complex pixel-based games such as the Atari games, and it can be especially useful when the reward is noisy as shown by the performance of the agent on the noisy version of the Atari games. Finally, an RL agent that learns a treatment to avoid septic shock from a real-life healthcare dataset can also benefit from the rewards provided by InferNet to make more effective decisions. In the future, we will compare our method to other recently published methods that provide different approaches to tackle the CAP, such as RUDDER or ET(λ).

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, and Eugene Brevdo. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[3] Adrian K Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 980–987. IEEE Computer Society, 2004.

[4] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *arXiv preprint arXiv:1806.07857*, 2018.

[5] Dilip Arumugam, Peter Henderson, and Pierre-Luc Bacon. An information-theoretic perspective on credit assignment in reinforcement learning. *CoRR*, abs/2103.06224, 2021.

[6] Wael F Asaad, Peter M Lauro, János A Perge, and Emad N Eskandar. Prefrontal neurons encode a solution to the credit-assignment problem. *Journal of Neuroscience*, 37(29):6995–7007, 2017.

[7] H. Azizsoltani et al. Unobserved is not equal to non-existent: Using gaussian processes to infer immediate rewards across contexts. In *In Proceedings of the 28th IJCAI*, 2019.

[8] C. Berner, G. Brockman, et al. Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680*, 2019.

[9] G. Brockman et al. Openai gym, 2016.

[10] Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement learning for continuous control. *CoRR*, abs/2005.05440, 2020.

[11] Zhixin Chen and Mengxiang Lin. Self-imitation learning in sparse reward settings. *CoRR*, abs/2010.06962, 2020.

[12] F. Chollet. Keras. https://keras.io, 2015.

[13] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado P van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, et al. Hindsight credit assignment. *Advances in neural information processing systems*, 32:12488–12497, 2019.

[14] Ashley et al. Hill. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[15] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):1–12, 2019.

[16] Yonatan Hutabarat, Kittipong Ekkachai, Mitsuhiro Hayashibe, and Waree Kongprawechnon. Reinforcement q-learning control with reward shaping function for swing phase control in a semi-active prosthetic knee. *Frontiers Neurorobotics*, 14:565702, 2020.

[17] M Komorowski, LA Celi, O Badawi, et al. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nat Med*, 24, 2018.

[18] A. Kumar, D. Roberts, et al. Duration of hypotension before initiation of effective antimicrobial therapy is the critical determinant of survival in human septic shock. *Critical care medicine*, 2006.

[19] Benjamin James Lansdell, Prashanth Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. *arXiv:1906.00889*, 2019.

[20] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[21] S Madjiheurem, H van Hasselt, Matteo Hessel, A Barreto, D Silver, and Diana Borsa. Expected eligibility traces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35. Association for the Advancement of Artificial Intelligence (AAAI), 2021.

[22] Ofir Marom and Benjamin Rosman. Belief reward shaping in reinforcement learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3762–3769. AAAI Press, 2018.

[23] Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Ufuk Topcu, and Scott Niekum. Self-supervised online reward shaping in sparse-reward environments. *CoRR*, abs/2103.04529, 2021.

[24] Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual credit assignment in model-free reinforcement learning. *CoRR*, abs/2011.09464, 2020.

[25] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49:8–30, 1961.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[27] Brett Norman, Colin Cooke, Eugene Ely, and John Graves. Sepsis-associated 30-day risk-standardized readmissions: Analysis of a nationwide medicare sample. *Critical care medicine*, 45, 04 2017.

[28] Alexander G Ororbia, Ankur Mali, Daniel Kifer, and C Lee Giles. Conducting credit assignment by aligning local representations. *arXiv:1803.01834*, 2018.

[29] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR, 2017.

[30] A Raghu, M Komorowski, I Ahmed, et al. Deep reinforcement learning for sepsis treatment. In *Workshop on ML for Health, NeurIPS*, 2017.

[31] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[32] C Rhee, TM Jones, Y Hamad, et al. Prevalence, underlying causes, and preventability of sepsis-associated mortality in us acute care hospitals. *JAMA Netw Open.*, 2(2), 2019.

[33] Blake A Richards and Timothy P Lillicrap. Dendritic solutions to the credit assignment problem. *Current opinion in neurobiology*, 54:28–36, 2019.

[34] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[35] David Silver, Aja Huang, Chris J Maddison, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[36] David Silver, Thomas Hubert, Julian Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[37] David Silver, Julian Schrittwieser, Karen Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[38] Halit Bener Suay, Tim Brys, Matthew E. Taylor, and Sonia Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 429–437. ACM, 2016.

[39] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, USA, 8 2018.

[40] Richard S Sutton. Temporal credit assignment in reinforcement learning. 1985.

[41] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[42] Richard S Sutton and Andrew G Barto. Time-derivative models of pavlovian reinforcement. 1990.

[43] Gerald Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3):257–277, 1992.

[44] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 10376–10386, 2019.

[45] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv:1511.06581*, 2015.

[46] Yuchen Wu, Melissa Mozifian, and Florian Shkurti. Shaping rewards for reinforcement learning with imperfect demonstrations using generative models. *CoRR*, abs/2011.01298, 2020.

[47] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *arXiv preprint arXiv:1804.06459*, 2018.

[48] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.