Reinforcement Learning as Probabilistic Inference

Afuad-Abrar Hossain [1] **Arnold Kokoroko** [1] **Tony Brière** [1]

Abstract

In recent years, formulating Reinforcement Learning as PGMs have gained considerable traction due to the availability of powerful inference techniques useful in solving various RL related queries. In this paper, we explore the *RL as Probabilistic Inference* framework and outline both an implementation of the traditional framework for RL and the PGM representation of RL, thereafter comparing their performances on the Frozen Lake game. We show that the the optimal policy retrieved via exact inference from the PGM framework is equivalent to the optimal policy derived from the traditional RL framework in the case of deterministic dynamics. Moreover, in the case of stochastic dynamics, we show that the policy retrieved via exact inference exhibits *optimistic (risk-seeking)* behavior, but is once again equivalent to the traditional RL optimal policy when retrieved via variational inference.

1. Introduction

Reinforcement Learning frameworks over the last few years have seen tremendous success in fields such as robotics and game-playing. The general paradigm of Reinforcement Learning, or hereafter also referred to as "RL", comprises of having some agent learn a policy so as to maximize the utility in a given environment the agent can interact with. RL frameworks have commonly been formulated as Markov Decision Processes (MDP) that describe both the set of states and actions an agent can take, with some rewards as a function of said states and actions. However, in recent years, formulating RL as probabilistic graphical models have gained considerable traction. In Sergey Levine's review on the topic, *"Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review"* (Levine, 2018), Dr. Levine outlines a detailed derivation of this framework and its merit over more traditional approaches to RL. Our paper will explore the *RL as Probabilistic Inference* framework, as described by Dr. Levine, and outline both an implementation of the traditional framework for RL and the PGM representation of RL (Hossain et al.), thereafter comparing their performances on the Frozen Lake game available on OpenAI Gym, a toolkit for testing reinforcement learning algorithms.

[1]Department of Computer Science and Operations Research, Université de Montréal, Montréal, Canada. Correspondence to: Simon Lacoste-Julien <slacoste@iro.umontreal.ca>.

2. Review of Current Literature

In Sergey Levine's review, *"Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review"* (Levine, 2018), Dr. Levine outlines a detailed derivation of the PGM representation of RL and its merit over more traditional approaches to RL. This includes demonstrating that RL is equivalent to exact inference for PGMs in the case of deterministic dynamics, and variational inference in the case of stochastic dynamics. While Dr. Levine presents several newer RL and Control algorithms inspired from the PGM framework, we will focus on section 2 and 3 of his review, where Dr. Levine outlines the derivations needed to find an agent's optimal policy—a set of optimal actions (one that maximizes utility) given an agent's state. Dr. Levine provides derivations for both the case of deterministic dynamics and stochastic dynamics. It will be shown that these derivations are very similar to more traditional derivations seen in standard RL.

2.1. Standard Approaches to RL

As stated previously, RL can be formulated as a Markov Decision Process (MDP) that describes both the set of states and actions an agent can take, with some rewards as a function of said states and actions. The MDP is summarized below:

- Environment has a set of states S
- Agent is given a set of possible actions A
- **Environment dynamics:** transition from state s_t to s_{t+1} with prob. $p(s_{t+1}|s_t, a_t)$ after taking action a_t.
- **Reward function:** $r(s, a) = E[r_{t+1}|s_t = s, a_t = a]$ provides a scalar reward after each step.

- Trajectory of an agent:
  $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \dots\}$

The goal for any agent is to take the most optimal action for each respective state so as to maximize the utility/reward of the environment. We thus search for a policy $\pi : S \to A$ that maximizes the rewards along our trajectory. In order to find the most optimal policy, we must first introduce the Bellman equations:

Bellman eq. for value function of a state s:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p\left(s'|s,a\right)\left[r(s,a) + \gamma V_\pi\left(s'\right)\right]$$

Bellman eq. for value function of a state-action pair (s, a)

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p\left(s'|s,a\right) \sum_{a'} \pi\left(a'|s'\right) Q_\pi\left(s',a'\right)$$

γ is a discount factor between 0 and 1 (optional to include). The Bellman equations for the value functions of states and state-action pairs simply describe the value an agent gets for being in a state and for taking an action in said state. The optimal value functions (Bellman optimality equations) are expressed by finding policy π that maximizes the Bellman equations:

$$\begin{aligned} V^*(s) &= \max_\pi V_\pi(s) \\ &= \max_a \sum_{s'} p\left(s'|s,a\right)\left[r(s,a) + \gamma V^*\left(s'\right)\right] \end{aligned}$$

$$\begin{aligned} Q^*(s,a) &= \max_\pi Q_\pi(s,a) \\ &= \sum_{s'} p\left(s'|s,a\right)\left[r(s,a) + \gamma \max_{a'} Q^*\left(s',a'\right)\right] \end{aligned}$$

Thus, we see that the most optimal policy $\pi^*(a|s)$ can be found from the optimal $Q^*(s,a)$:

$$\pi^*(a|s) = \delta\left(a = \arg\max_a Q^*(s,a)\right)$$

The optimal trajectory can be found by an agent simply executing the optimal policy.

2.2. RL as a Probabilistic Graphical Model

The previously described MDP can also be modeled as a PGM. The first step is to model the states and actions at some time step t as random variables.

The initial state s_1 can be sampled according to some distribution $s_1 \sim P_1(s)$. As seen in the left PGM in Figure 1, each successive state s_{t+1} is dependant on the previous state s_t and previous action taken a_t. As such, we define the transition dynamics $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. For action a_t,

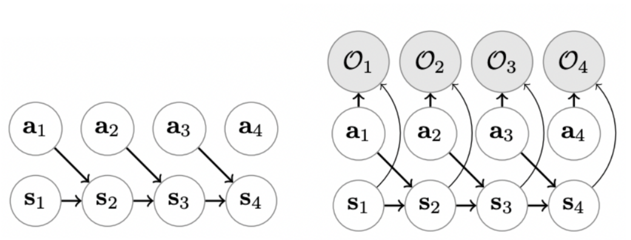

Figure 1. Left: Basic structure of the PGM with states and actions. *Right:* PGM with the the optimality variables introduced (rewards)

we can simply sample from some uniform prior for now. As such, based on the left PGM in Figure 1, we are taking random actions at each time step. This is evidently not useful in our case. In order to sample optimal trajectories and not random trajectories, we need to introduce a reward function to our PGM. In the right PGM in Figure 1, we do exactly that by introducing an optimality binary random variable O_t that depends on the reward. O_t is true when it takes the value 1 (optimal action taken) and false when 0. O_t's relation to the reward function is defined as follows:

$$P(O_t = 1|s_t, a_t) = exp\left(r(s_t, a_t)\right)$$

We now see that we have a PGM configuration resembling that of an HMM, where we can now condition on the optimality variables being true to find the most optimal set of actions—one that maximizes the rewards associated with the environment, as was the case in standard RL.

2.3. Optimal Policy Search as Probabilistic Inference

The PGM framework for RL allows us to explore different types of queries. However, for the purpose of this paper, we will focus on retrieving the optimal policy, as was the case for the standard RL framework. Based on the PGM, we can define the optimal policy as $P(a_t|s_t, O_{t:T})$, that is, the action to take given the state and given that the optimality variable is true at all time steps t. As mentioned, the resemblance of the PGM to an HMM allows us to use inference tools such as sum-product inference algorithms to compute the optimal policy. As such, we define the backward messages to derive:

$$\begin{aligned} \beta_t(s_t, a_t) &= p(O_{t:T}|s_t, a_t) \\ \beta_t(s_t) &= p(O_{t:T}|s_t) \end{aligned}$$

Deriving the above backward messages, we get:

$$\begin{aligned} \beta_t(s_t, a_t) &= p(O_{t:T}|s_t, a_t) \\ &= \int_S p(O_{t:T}, s_{t+1}|s_t, a_t) ds_{t+1} \end{aligned}$$

$$
\begin{aligned}
&= \int_S p(O_{t:T}|s_{t+1})p(s_{t+1}|s_t,a_t)p(O_t|s_t,a_t)ds_{t+1} \\
&= \int_S \beta_{t+1}(s_{t+1})p(s_{t+1}|s_t,a_t)p(O_t|s_t,a_t)ds_{t+1} \\
&= p(O_t|s_t,a_t)E_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[\beta_{t+1}(s_{t+1})\right]
\end{aligned}
$$

as well as

$$
\begin{aligned}
\beta_t(s_t) &= p(O_{t:T}|s_t) \\
&= \int_A p(O_{t:T}|s_t,a_t)p(a_t|s_t)da_t \\
&= E_{a_t\sim p(a_t|s_t)}\left[\beta_t(s_t,a_t)\right]
\end{aligned}
$$

We now derive the optimal policy $P(a_t|s_t,O_{t:T})$ using the backward messages:

$$
\begin{aligned}
p\left(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{t:T}\right) &= \frac{p\left(\mathbf{s}_t,\mathbf{a}_t|\mathcal{O}_{t:T}\right)}{p\left(\mathbf{s}_t|\mathcal{O}_{t:T}\right)} \\
&= \frac{p\left(\mathcal{O}_{t:T}|\mathbf{s}_t,\mathbf{a}_t\right)p\left(\mathbf{a}_t|\mathbf{s}_t\right)p\left(\mathbf{s}_t\right)}{p\left(\mathcal{O}_{t:T}|\mathbf{s}_t\right)p\left(\mathbf{s}_t\right)} \\
&\propto \frac{p\left(\mathcal{O}_{t:T}|\mathbf{s}_t,\mathbf{a}_t\right)}{p\left(\mathcal{O}_{t:T}|\mathbf{s}_t\right)} \\
&= \frac{\beta_t\left(\mathbf{s}_t,\mathbf{a}_t\right)}{\beta_t\left(\mathbf{s}_t\right)}
\end{aligned}
$$

We see that the optimal policy can be expressed using the two previous backward messages we had derived. As a result of using Baye's rule to flip the conditions, the term $p(O_{t:T})$ and $p(s_t)$ cancels out since they both appear in the numerator and denominator. $p(a_t|s_t)$ can also be removed, seeing that removing the term leads to another proportional expression, as we had previously mentioned that $p(a_t|s_t)$ follows a uniform distribution.

2.4. Deterministic vs. Stochastic Dynamics

We will see that there are slight differences in the behavior of our policy depending on the environment dynamics. To understand this, we must first observe what happens when deriving the backward messages in log-space. To that effect, let us define:

$$Q(s_t,a_t) = \log\beta_t(s_t,a_t)$$

$$V(s_t) = \log\beta_t(s_t)$$

In this case, if we marginalize the actions out from $\beta_t(s_t,a_t)$ in log-space, we get the following equivalent expression:

$$V(s_t) = \log\int_A \exp\left(Q(s_t,a_t)\right)da_t$$

If we were to re-derive the backward message $\beta(s_t,a_t)$ as we've done previously, but now in log-space, we get the following expression:

$$Q(s_t,a_t) = r(s_t,a_t) + \log E_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[\exp\left(V\left(s_{t+1}\right)\right)\right]$$

Our first observation is that the relationship between $V(s_t)$ and $Q(s_t,a_t)$ is very similar to the one seen in the standard RL framework, but with the log-exponential terms approximating maxima (e.g. soft-max). The most interesting thing to note is that, in the case of deterministic dynamics, $Q(s_t,a_t)$ reduces to $r(s_t,a_t)+V(s_{t+1})$. The equation is now equivalent to the Bellman equation seen in standard RL. In the case of stochastic dynamics, $Q(s_t,a_t)$ does not reduce. The log-expectation-exponential in the expression is not the expected value at the next state, but a "soft-max" of the expected value at the next state. This is problematic as the expression encourages risk-seeking behavior in our agent, that is, the agent will choose actions to reach high-reward states, regardless of the probability of reaching that state. As such, the derivations using backward messages are not ideal for stochastic dynamics.

It will be later shown in the experimentation section that this optimistic policy behavior for stochastic dynamics does in fact occur.

2.5. Variational Inference for Stochastic Dynamics

At last, it is possible to amend this risk-seeking behavior. As a matter of fact, the author suggests using variational inference methods that leads to maximizing an objective as similar as possible to the Bellman objective of maximizing the *expected* reward. Formally, we are looking to minimize the KL divergence between the actual distribution

$$p(\tau) = \left[p\left(s_1\right)\prod_{t=1}^{T} p\left(s_{t+1}|s_t,a_t\right)\right]\exp\left(\sum_{t=1}^{T} r\left(s_t,a_t\right)\right)$$

and the approximate distribution

$$\hat{p}(\tau) \propto 1[p(\tau)\neq 0]\prod_{t=1}^{T}\pi\left(\mathbf{a}_t|\mathbf{s}_t\right)$$

In this case, we can show that minimizing the divergence is equivalent to maximizing the expected reward as desired and the entropy of the approximate distribution as a bonus. In fact, the latter promotes exploration which is recommended according to frameworks such as maximum entropy reinforcement learning. Constraints for partial observability or particular probability distributions could be added.

$$
\begin{aligned}
-D_{\mathrm{KL}}(\hat{p}(\tau)\|p(\tau)) = \sum_{t=1}^{T} & E_{\left(\mathbf{s}_t,\mathbf{a}_t\right)\sim\hat{p}\left(\mathbf{s}_t,\mathbf{a}_t\right))}\left[r\left(\mathbf{s}_t,\mathbf{a}_t\right)\right] \\
& + E_{\mathbf{s}_t\sim\hat{p}\left(\mathbf{s}_t\right)}\left[\mathcal{H}\left(\pi\left(\mathbf{a}_t|\mathbf{s}_t\right)\right)\right]
\end{aligned}
$$

Multiple iterative methods exist that maximize the same objective or another very similar objective. Those methods include, but are not limited to, MaxEnt Policy Gradients, MaxEnt Actor-Critic and Soft Q-Learning.

We decided to implement the Soft Q-Learning algorithm for our project. This method samples many trajectories, runs the game for each sampled trajectory and learns from the obtained reward. The optimal policy is then a weighted average of the previous optimal policy and the new expected cumulative reward estimate where the weight correspond to a learning rate that is left to our discretion.

2.6. Benefits: a PGM View of RL

The obvious benefit to a PGM view of RL is the availability of powerful inference techniques that can be used to solve various RL-related queries. In our case, the optimal policy was retrieved with the use of backward messaging for deterministic dynamics and variational inference for stochastic dynamics. Other queries include to infer optimal trajectories given the environment's rewards and to infer an environment's rewards given a sample of trajectories. As the PGM model is simply a probability distribution over all trajectories, not only can the optimal trajectory be retrieved but all other trajectories. This allows for the modelling of sub-optimal behavior, useful for certain contexts (Levine, 2018). In short, the PGM view of RL allows for a flexible framework that can be extended to multiple fields in RL.

3. Implementing RL Algorithms

To demonstrate the application of probabilistic graphical models in RL, we implemented and compared 3 different algorithms notably the Standard RL framework solving the Bellman equation, the aforementioned PGM framework using exact inference and the Soft-Q Learning using variational inference. The goal of implementing these three models was to retrieve and compare their optimal policy on the popular Reinforcement Learning game: Frozen Lake. The code-base for the three implementations can be found on GitHub (Hossain et al.).

3.1. Standard RL: Value Iteration Algorithm

The first model implemented, and baseline method used in this review, was the Standard RL framework using the Bellman equation to find the optimal policy. In order to solve this equation and find the optimal policy, we used the *value iteration* algorithm (Ravichandiran, 2018). The alternative method *policy iteration* could have also been used to find the optimal policy as their implementation is similar.

Moreover, to implement the *value iteration* algorithm in an efficient manner we used dynamic programming. Indeed, using such approach optimizes the computation time by dividing the problem into simpler sub-problems, where each sub-problem can be optimized iteratively and its optimal value stored for later use.

Algorithm 1 Computing Backward Messages

```
for $t = T - 1$ to $1$ do
    $\beta_t(s_t, a_t) = p(O_t|s_t, a_t) E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)} [\beta_{t+1}(s_{t+1})]$
    $\beta_t(s_t) = E_{a_t \sim p(a_t|s_t)} [\beta_t(s_t, a_t)]$
end for
```

The value iteration algorithm can be divided into 4 main steps. The initialization of the value function V(s) to random values, followed by storing all the state-action pairs Q(s,a) in order to improve our value function. Then, iteratively improve the value function with the maximum value from Q(s,a). Finally, we retrieve the optimal value function once its change in values become very small. Using the resulting optimal value function we can easily derive the optimal policy by updating the state-action pairs Q(s,a) with the optimal V(s) and choosing the best actions for each state.

3.2. RL as PGM: Exact Inference

In order to represent a graphical model, we utilized object-oriented programming principles: the nodes of a PGM are class structures and the PGM itself is also a class structure which houses the data structures of the nodes. The edges of the graph (connectivity) are stored within the same structure as the nodes. Furthermore, each PGM structure has at its disposal several methods, the most notable of which is the inference method. This method allows for the computation of backward messages, needed in the derivation of the optimal policy. The algorithm for computing backward messages is given by the figure (**Algorithm 1**). This simple approach (using backward messages) is elegant, easy to implement, compact, easy to explain and allows for the implementation of any graph structure.

The OpenAI module provides a set of methods which allows for the easy setup and modification of the game-playing environment. Utilizing the PGM class structure and the OpenAI methods, it became very easy to implement multiple agents (trained via inference) and multiple environments necessary in conducting our experiments.

3.3. Soft Q-Learning - Variational inference

The implementation of Soft Q-Learning was surprisingly easier to implement than the Bellman equations. Our team wrote a script that would play the game a 1000 times by sampling random actions at every step (there would be a maximum of 10 steps per game so that the running times are reasonable). At every step, a function would update the policy estimate by adding the reward corresponding to the action taken with the current policy estimate for that action-state pair. In particular, the update is a weighted-average of both using a learning rate.

As for the implementation of the Bellman equations, it is possible to add a parameter γ that acts as a discount factor to minimize the number of steps.

4. Experiments with the Frozen Lake Game

Using the three implementations of RL (one using the standard RL framework, the other two using varieties of the PGM framework), we retrieve the optimal policies for two versions of the Frozen Lake Game on OpenAI Gym, the first game with deterministic dynamics and the second game with stochastic dynamics. We thereafter compare the policy behaviors, primarily to show that the policy retrieved via exact inference (Implementation 2) exhibits optimistic (risk-seeking) behavior in stochastic dynamics, whereas the optimal policy retrieved via approximate/variational inference (Implementation 3) does not.

4.1. A Word On OpenAI Gym

OpenAI Gym is a toolkit for developing and testing Reinforcement Learning algorithms (Brockman et al., 2016). It is maintained by OpenAI, a research laboratory based out of San-Francisco, California. It provides a collection of test environments, some of which are in the form of popular games. For this project, we use the Frozen Lake Game environment, mainly due to its ease-of-use and effectiveness in conveying RL principles.

4.2. Frozen Lake Game Set-up

The Frozen Lake game is a grid game where an agent has to traverse the grid in search of a *goal* state while avoiding *hole* states. The game ends when the agent reaches a goal state (agents receives a reward) or a hole state (agent receives no reward). We have setup two versions of the grid, one to test our implementations in deterministic settings (hereafter called *deterministic grid*), the other to test in stochastic settings (hereafter called *stochastic grid*). The deterministic grid follows the classic 4x4 setup of the Frozen Lake Game. In the deterministic grid, the only goal has a reward of 1.

The stochastic grid was modified to include 2 goal states, along with one stochastic state (ICE) next to one of the goal states. Goal #1 has a reward of 8 and Goal #2 has a reward of 1. With the induced stochastic setting (ICE), an agent has only a 10% of reaching Goal #1 and a 90% chance of falling into a hole should it try to go towards Goal #1. However, should it try to go towards Goal #2, the agent is assured to reach it, as the nearby state dynamics are entirely deterministic (WOOD). The setup for the stochastic grid allows to test risk-seeking behavior as $Goal\#1 > Goal\#2$ but the expected reward $(0.10 * Goal\#1)$ of Goal #1 is lower than expected reward of Goal #2 $(Goal\#2)$.

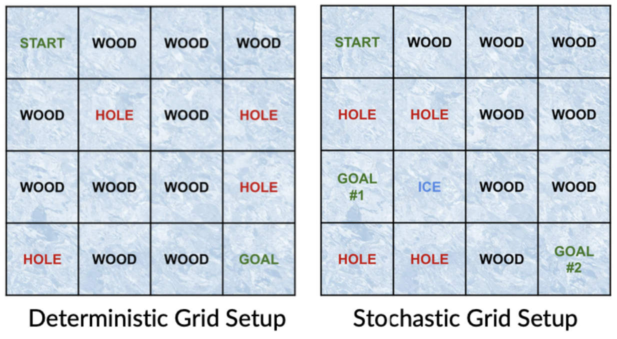

Figure 2. Left: Grid used for deterministic games. Grid has one goal state (Goal #1=1) and all states have transition probabilities of 1. *Right:* Grid used for stochastic games. Grid has two goal states (Goal #1 = 8, Goal #2 = 1) and one stochastic state (ICE) with transition dynamics: $p(s_{t+1} = GOAL\#1 | s_t = ICE, a_t = LEFT) = 0.10$.

4.3. Results

Based on *Figure 3*, it is seen that for the deterministic grid, both the PGM-Exact-Inference implementation and the Soft Q-Learning implementation are equivalent to that of the Bellman implementation: all three policies lead to the goal state in the same number of steps. As the environment is deterministic, the result can be duplicated 100% of the time.

As for the stochastic grid, we finally observe differences in policy behavior. For the Bellman implementation, the policy leads towards Goal #2, whereas for the PGM-Exact-Inference implementation, the policy leads towards Goal #1, even with the transition probability of attaining said goal being small (10%). Finally, while Bellman and Soft Q-Learning policy may differ for some states, both policies ultimately lead to Goal #2 within the same number of steps, entirely avoiding Goal #1, hence they are considered to be equivalent once again.

In observing *Figure 4*, we see that in 1000 episodes of the stochastic game, both Soft Q-Learning and Bellman will accumulate a reward at every episode, hence receive 1000 points, whereas PGM-Exact-inference will miss out on rewards on the majority of episodes as its policy leads to the more risky goal state #1 (10% of getting the higher reward). This explains the discrepancy in accumulated rewards between PGM-Exact-inference and Bellman/Soft Q-Learning.

4.4. Discussion

Based on the results obtained, we can analyze the behavior of our agent in the deterministic case, and in the stochastic

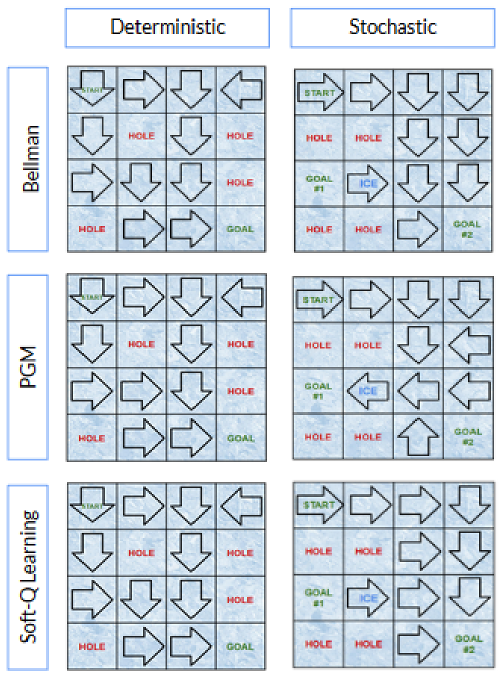

Figure 3. Optimal policies of the 3 RL implementations (Bellman, PGM-Exact Inference, SoftQ-Learning) on the Frozen Lake Game in both deterministic and stochastic environments.

case. In the former, all three agents manage to get to the goal successfully without falling into a hole, using an optimal amount of steps. Consequently, there is no special pattern in the agents' actions for the deterministic case of this game, and the optimal policy derived by the Standard RL algorithm using Bellman equation is equal to the optimal policy of the PGM-Exact-Inference. An interesting case for further experiments could be to add smaller rewards around the path to the goal to see the changes in the policy of the three algorithms.

However, in the case of the stochastic environment, we observe different tendencies when using the PGM-Exact-Inference compared to Bellman. While Bellman's optimal policy heads toward Goal #2 which has the best maximum expected reward, the PGM-Exact-Inference optimal policy heads towards Goal #1 even though its maximum expected reward is smaller. Indeed, the PGM-Exact-Inference agent prioritize the largest rewards even if the probability of reaching them is smaller. This greedy tendency can be characterized as a risk-seeking behavior as it was introduced in the earlier sections of this review.

Nonetheless, this problem arising only in the stochastic version of the game is solved by using the Soft Q-Learning method. The Soft Q-Learning method using variational/approximate inference to maximize the expected

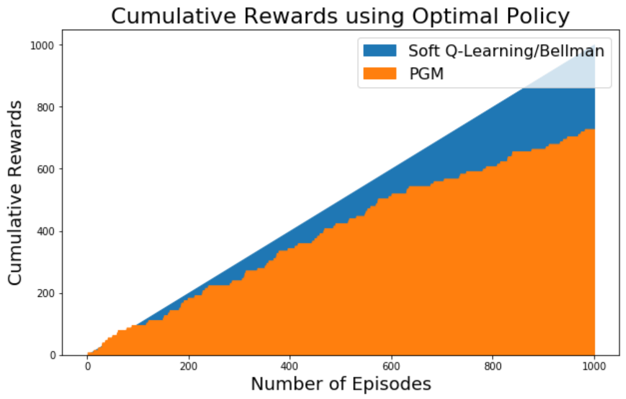

Figure 4. Cumulative rewards of the 3 different implementations using their optimal policy on a 1000 episodes. *Orange:* Cumulative reward using the PGM Exact-inference. *Blue:* Cumulative reward of both the Soft Q-Learning and Bellman algorithm.

reward chooses the path leading to the goal with the best expected reward just like the Bellman method. The sub-optimality of searching for the highest reward regardless of the probability of getting to it, is further demonstrated in Figure 4, where the risk-seeking PGM-Exact-Inference is outclassed in the long run. Thus, the Soft Q-Learning method can be said to be equivalent to the Standard RL algorithm in the stochastic case.

5. Conclusion

In this paper, we explored the *RL as Probabilistic Inference* framework by outlining both an implementation of the traditional framework for RL (as defined by the Bellman equations) and two varieties of the PGM representation of RL. In comparing the implementation results on the Frozen Lake Game, it can be concluded that the optimal policy retrieved via exact inference is equivalent to the optimal policy in standard RL in the case of deterministic dynamics. However, in the case of stochastic dynamics, the policy retrieved via exact inference is shown to exhibit optimistic (risk-seeking) behavior. This risk-seeking behavior was rectified by using approximate/variational inference methods as seen with the Soft Q-learning implementation once again having equivalent results to that of the Bellman implementation in the stochastic case.

This paper mainly focused on the derivation and implementation of optimal policies using the the PGM framework. However, the flexibility of this framework gives rise to a multitude of future research directions in the domain of Reinforcement Learning.

Acknowledgements

We would like to thank Sergey Levine, whose review and tutorial, *"Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review"* serves as the foundation of our explorations and implementations. We would also like to thank Professor Simon Lacoste-Julien and his TA Jose Gallego, for their teachings and guidance throughout the semester.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Hossain, A.-A., Brière, T., and Kokoroko, A. Ift6269-project. https://github.com/afuadhossain/IFT6269-Project. Accessed: 2019-12-19.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *CoRR*, abs/1805.00909, 2018. URL http://arxiv.org/abs/1805.00909.

Ravichandiran, S. *Hands-On Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow*. Packt Publishing, 2018. ISBN 1788836529, 9781788836524.