Human in the loop Reinforcement Learning

Sandip Patel, Shuran Song, Matei Ciocarlie
Columbia University

ROAM LAB Meeting Memo

1. Introduction

Reinforcement Learning has shown success with complex problems both in research as well as commercial settings. Current Reinforcement Learning Policies are great at learning policies for fairly complex problems in a deterministic environment. However, some sets of problems are so complex that the agent will not be able to always interact with the environment optimally. Agents still learn acceptable policy. Yet, during run time, it can incur heavy penalties. Through this study, we tried to address this problem by developing an algorithm for 'Human-in-the-loop' Reinforcement Learning, where an agent can call an expert, who has complete knowledge of the domain, when in a state associated with high risk. The expert can take correct action with better odds than the agent. Hence we rely on the expert's domain knowledge and experience to take the favorable action when required. We will present two algorithms to implement the concept and analyze them.

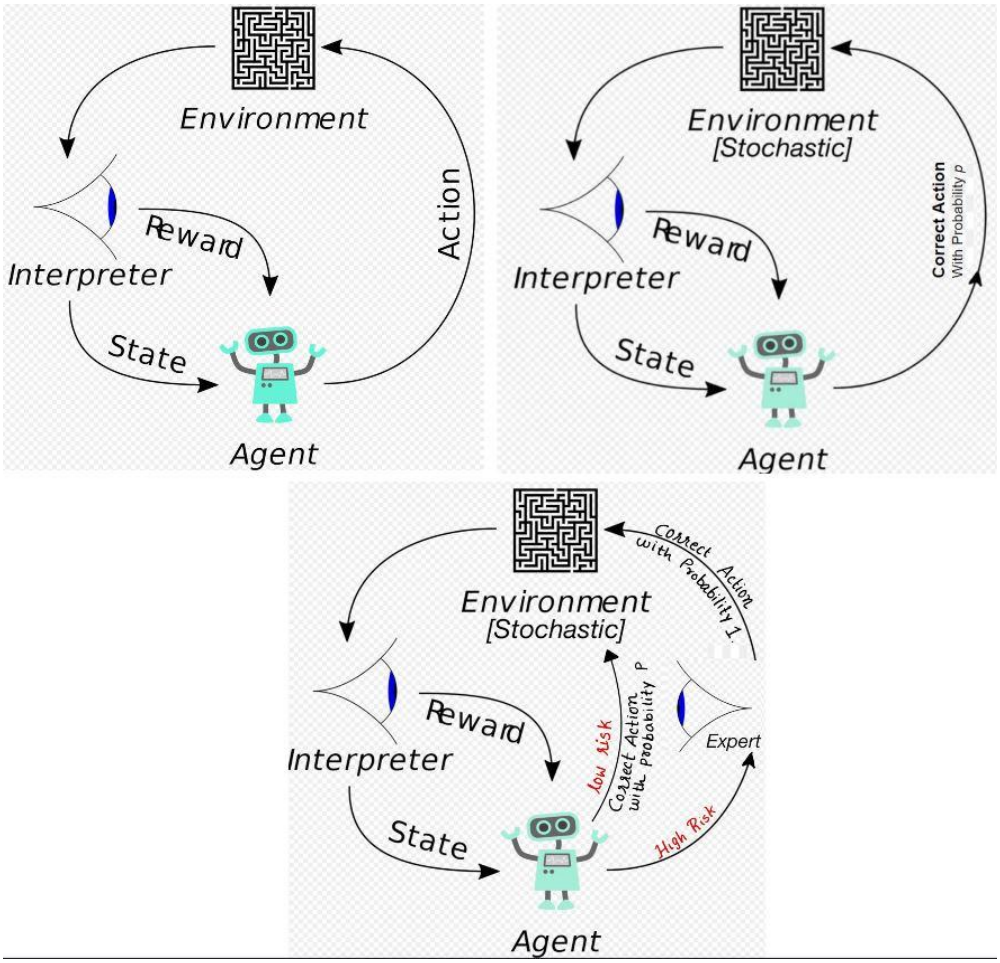


Figure 1. Human-in-the-Loop Concept :- (top-left)- Regular RL, (top right)- Regular RL in non deterministic environment, (bottom) - proposed Human-in-the-Loop RL

2. Detailed Background of the Problem.

Most of the Industry ready Reinforcement Learning solutions rely heavily on accuracy of the training and precise dynamics of the environment. It is assumed that once the optimal policy is learned, the agent is always able to follow the policy and act according to it. However there may be situations where this assumption can pose a big threat. For example driving a car on a Thunderstorm night with heavy downpour (slippery road) or say walking on a slippery frozen lake. In such conditions dynamics is not deterministic and your commanded action will not necessarily be the resultant action. Under such circumstances following learned policy cannot be guaranteed. Taking a step further what if there is some risk associated with the action taken. Extending the example of frozen lake, if there are pot holes in the lake, one wrong slip when around the hole may result in extremely high damage.

3. What are we Proposing

We are proposing that if the **expert, who has complete knowledge and experience of working in the environment is available when required**, the agent can dial up and call the expert for its guidance when things are at stake. It is like calling up an expert when an agent feels the need and handing over the control of the dynamics trusting the expert's knowledge and experience. This is analogous to the situation when our computer systems with a lot of valuable data and work go crazy and we prefer not to take chances by trying to fix it on our own (and lose the data). Then we make a call to a **"tech-expert"** who is somewhere remote to ask for help. We trust his/ her knowledge and experience to deliver results with better odds.

So, Basically we will have an Reinforcement learning algorithm where the expert is within the RL loop and can be called when the agent deems necessary. This can be easily understood using the **figure 2**.

Key Points of the Problem statement:

- If Agent and environment interact in a non-deterministic, the resulting action cannot be predicted with certainty.
- This could result in huge loss if there are associated risks with the states in the environment.
- If the expert is available on call to perform actions with certainty, performance of the agent can be increased drastically.

4. Objective

Our objective is to develop a Reinforcement Learning algorithm to learn a policy that achieves higher returns with expert, compared to policy learned without the expert in a non-deterministic environment, and also minimize the expert calls at the same time. We also aim at Reducing expert calls as expert calls can be expensive in nature. Hence, one always needs to evaluate the size of the risk to understand whether we really need an expert or rely on our own skills in order to maximise the rewards. This is intuitive from our real world scenarios as well. On a side note, consider the following analogy. A businessman needs to evaluate the size of the problem when he/she encounters one to see if it is really worth it to call an expert or save profit from draining into the expert’s pocket. So we need our agent to learn the thin threshold where it calls an expert but not too often.

Our Algorithm succeeds if the expected returns with the expert calls are higher than the expected returns without the expert calls. We have developed two algorithms targeted towards the objective mentioned, with the core difference being whether the expert is available during training time or not.

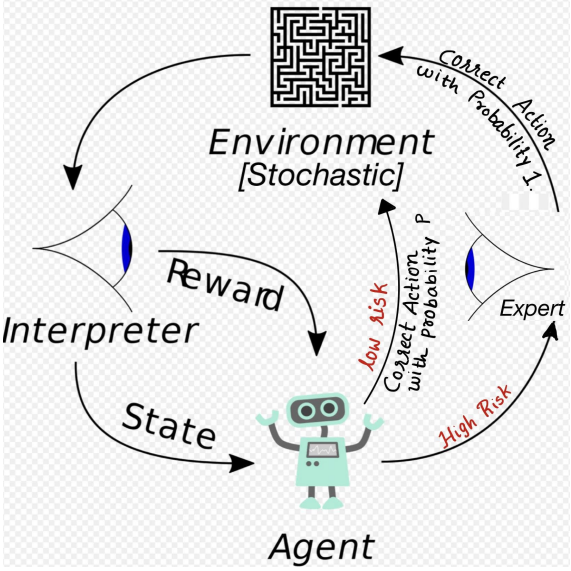


Figure 2. Proposed "Human-in-the-loop" RL

5. Two Paths Ahead

We will be developing two algorithms to achieve the objective namely **Algorithm1 (ALG1)** and **Algorithm2 (ALG2)**. As mentioned earlier, the difference between the two is whether the expert is available during training period or not.

a. Algorithm 1 (ALG1)

Algorithm 1 is based on the assumption that with enough exploration steps during training, the agent can learn to make expert calls judiciously. We are relying on the existing Learning framework (learning appropriate action from available action space with exploration) to learn policy with the expert available during training. Agent is given the privilege to call an expert throughout the training process. An additional action, i.e., "expert call," with some associated extra cost (else expert will be called at every step) is added to action space and the agent just rolls out with the training process as usual.

b. Algorithm 2 (ALG2)

Algorithm 2 agent does not have the privilege to make an expert call during training time. Hence we need to figure out a way to incorporate something that can quantify risk based on experience in the training phase, based on which agent can make an expert call during run time. We were betting high on ALG2 as we were keeping agents blind from actual runtime scenarios and still expect it to achieve high returns then algorithms without the experts.

We came up with an idea to learn Variance of the returns during training and use it to make expert calls during runtime. We designed an algorithm to learn the variance of returns online during training just like value functions. During training, the agent learns two functions namely **action-value function** and **variance**.

How about using entropy instead to get a baseline of how well are we performing?

We chose variance as it can capture fluctuations in returns received taking an action from the particular state and following learned policy from thereon. Fluctuations in the returns is a way to quantify risk and based on

risk, agents can decide whether to make an expert call or not. Conceptually this is very similar to the Action-Value function, that represents the expectation of the returns corresponding to (***state, action***) pair.

Learning Variance

The variance of returns can be known in various ways. One such way is using Monte-Carlo methods[1] to accumulate states and actions the agent takes and resulting rewards in the buffer and at the end of the episode, calculate returns corresponding to the (***state, action***) pair. Using simple statistics, one can calculate variance for each state-action pair throughout the training.

However, we wanted to implement something that is memory efficient and similar to learning an action-value function on the go; online learning. Hence we derived Bellman-like Equation for the variance of the returns[2] to learn and update the variance online as training proceeds.We came up with the following set of equations to update the variance online. Derivation is already uploaded in the meeting notes folder.

shouldn't this be max:
https://medium.com/@curiousily/solving-an-mdp-with-q-learning-from-scratch-deep-rein

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha_q[r(s,a) + \gamma * argmax_a(Q_{t-1}(s',a')) - Q_{t-1}(s,a)] \quad \text{---- 1}$$

$$M_t(s,a) = M_{t-1}(s,a) + \alpha_m[r(s,a)^2 + 2\gamma r(s,a) * argmax_a(Q_{t-1}(s',a')) + \gamma^2 M_{t-1}(s',a') - M_{t-1}(s,a)] \quad \text{---- 2}$$

$$V_t(s,a) = V_{t-1}(s,a) + \alpha_v[M_{t(s,a)} - Q^2{}_t(s,a) - V_{t-1}(s,a)] \quad \text{---- 3}$$

Will these equations allow for negative variance?

Where,

- s - state of agent before the step.
- a - action taken by agent in state s
- s'- state of agent after after taking action a from state
- a' - action from state s' as dictated by policy
- $r(s,a)$ - reward obtained by taking action a from state s
- α_v - learning rate for variance
- α_q - learning rate for action-value
- α_m - learning rate for second moment
- γ- discount factor
- $Q_t(s,a)$- action value of for (s,a) at time t after the update
- $Q_{t-1}(s,a)$ - action value of for (s,a) at time $t-1$ before the update
- $M_t(s,a)$- Second moment of the returns for (s,a)at time t after the update
- $M_{t-1}(s,a)$- Second moment of the returns for (s,a)at time t after the update
- $V_{t-1}(s,a)$- Variance of returns for (s,a)before making and update
- $V_t(s,a)$- Variance of returns for (s,a) after making and update.

6. Implementation

The toy problem of grid navigation is selected to implement and demonstrate the algorithms developed. A 10x10 grid world is created as a custom **OpenAI Gym** Environment. The grid contains sparsely located obstacles and traps, as seen in **Figure 3 below**. The agent starts from a random "safe" position at the beginning of each episode and learns to navigate towards the goal.

We narrowed down to this problem as we want to make the problem as simple as possible for initial implementation but not simpler (as Einstein said). We wanted to formulate a problem where the expert is useful but yet simple. Presence of traps combined with non-deterministic dynamics amplifies the need for the expert guidance when around traps (high risk).

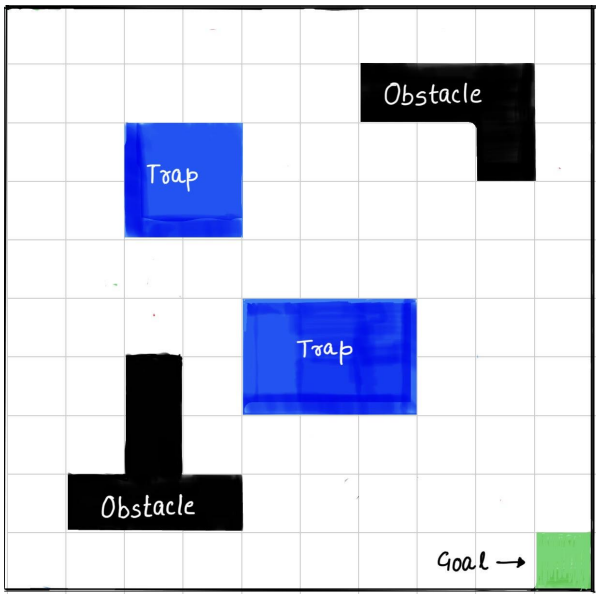


Figure 3. Grid world

a. About Environment

The agent starts from a random "safe" position at the beginning of each episode and learns to navigate towards the goal.When the agent collides with obstacles, it gets a penalty and stays in the same state, but falling into one of the traps receives high negative rewards, and the episode is considered over. An episode can get over if the agent falls into the trap or safely navigates the grid to reach the goal state. However, the environment is stochastic, and dynamics is faulty; thus, the agent is not always able take steps guided by the policy learned (epsilon greedy) but takes random action with probability $p = \varepsilon_2$. This stochasticity is on top of one introduced due to epsilon greedy policy during learning (random exploration with $p = \varepsilon_1$**). Throughout the study** $\varepsilon_1 = 0.1$ **and** $\varepsilon_2 = 0.3$**.**

How exactly would epsilon 2 work in the environment will it randomly throw the agent in any possible state?

Stochasticity due to ε_2 stays during run time unlike ε_1, as it is property of the environment or the agent dynamics or both tending to lower the expected returns as an agent might incur a huge penalty by falling into a trap or colliding with obstacles.

b. Reward Structure

Reward engineering is at the center of every reinforcement learning problem as it decides what the agent is trying to accomplish. Hence we took at most care to craft a meaningful reward structure. Prime considerations were to set reward structure such that the expected reward maximizes when the agent reaches goal in a minimum number of steps with minimum possible expert calls. Utmost care is taken to keep logical relations between events intact with rewards. For example, a reward of -2 is associated with every step taken by the agent to make sure the agent reaches the goal in minimum steps and the cost of calling an expert for the step is set to -5, in addition to the regular reward of -2. This incorporates the consideration that calling an expert for the step incurs extra cost and is expensive. **Table 1** contains the reward associated with the events.

Events	**Associated Rewards**
Each step in environment	-2
Collision with obstacle	-3
Falling into the trap	-100
Reaching the Goal state	100
Calling Expert for next step	-5

Table 1. Reward structure

c. Design of Expert

Although our aim is to include human as an expert within the RL loop, hard coding the expert for the purpose of toy problems can make the architecture relatively simple. Although we are aware that hard coding is not scalable and reusable. However it will serve the purpose for the problem at hand.

Next major challenge is to design an expert with a certain attitude. We can either model experts as aggressive or cautious. An aggressive expert would make sure the agent takes the shortest path towards the goal even if that means getting or staying close to the traps. A cautious agent will try to always take steps to maximise the distance from traps when called, even if that is at the cost of taking a step not towards the goal. We modeled a cautious expert. Our agent is defined as:

> **“When called, agent dictates step that aims to take the agent away from the trap even at the cost of moving away from the goal state”**

Figure 8 in appendix shows how an expert is modeled with numbers assigned to each action as shown in the table that follows.

d. Training

For a start, we developed algorithms based on tabular Q-learning. Updates are made to the learned value function and are made online using temporal difference after every step.

Once the Custom Environment is setup in OpenAI gym and Algorithms are ready! We ran 10 Million episodes for training to teach agents how to navigate to the Goal for both ALG1 and ALG2,running with the same hyper parameters. In order to compare parameters and hyper-parameters for both the algorithms refer to the table below. Only stark difference between both the training is an extra action available to the agent in ALG1 compared to ALG2.

Experience Replay not mentioned here so we are going to assume that we are not to use that.

Is it possible for ALG1 to call expert_action while choosing a random action under epsilon2?

Hyperparameter	**ALG1**	**ALG2**
Observation Space	(agent's state, reward)	(agent's state, reward)
Action Space	Up, Down, Left, Right, Expert Call	Up, Down, Left, Right
ε1	0.1	0.1
ε2	0.3	0.3
Number of training Episodes	10 Million	10 Million
Discount Factor	0.9	0.9
α_q, α_m, α_v	0.1	0.1

Table 2. Tables with hyper parameters value

We terminated training after making sure the update value is less than 1\% of the current value function or variance function.

7. Results

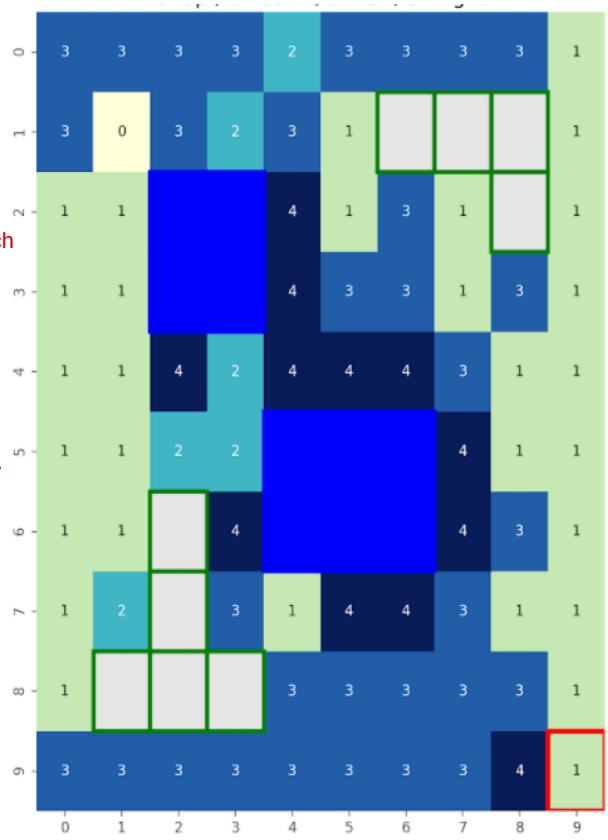
Results for both algorithms are plotted as heat maps of 10 X 10 each square represent the corresponding state in the grid world. Values inside are of the value functions or variance estimated by algorithms. After taking a glance at the results, we will discuss elaborately on interpretation of the results.

a. Results - Algorithm 1

Figure 9 in the appendix shows heat maps of converged action value (Q-value or simply Qvalue) function for ALG1. While **Figure 4 below** plots the learned policy based on action value. We can see that the agent learns to figure out that expertise is required when around traps to prevent high penalties of falling into the trap. Yet in some states (like (2,1)), expert is not called and an agent assumes risk of falling into the trap by being confident in dynamics and its knowledge.

Without experience replay I think there are a lot coorelated samples being trained which can result in different and incorrect final values every time
For ex: cell (9,8) should not be calling the expert at all.

Also I do not know how consistent these values are for every run.
Our learning rates are also very high which may lead to even more inconsistent results.



Numbers	Corresponding Action
0	Up
1	Down
2	Left
3	Right
4	Expert call

Figure 4. ALG1 policy leaned

b. Results - Algorithm 2

Since algorithm 2 is not as straightforward as Algorithm 1 in terms of figuring **s**out when to call an expert. During training for algorithm 2 agent is trained for 10 million episodes with 4 actions at hand and agent learns two functions - namely Variance of the returns from each state and action value function. **Figure 5(a)** and **figure 5(b)** hold Variance calculated through Sampling and variance learned using Bellman-like equations respectively. One can see that although values differ but the trend of large variance around the traps in both methods agree with each other.

For the purpose of sanity check we calculated variance using both methods and compared them in terms of pattern as well as values. As expected both come out to be close and online learning of variance is verified.

Figure 10 in appendix contains the MaxQ plot learned in ALG2 just like we did in ALG1. We can notice that the value function is very similar to the one learned in ALG1 except around the states where the expert is called in ALG1.

We are calculating the variance for every state action pair, so how are the below diagrams being made, are they the average variance for every action on that cell?

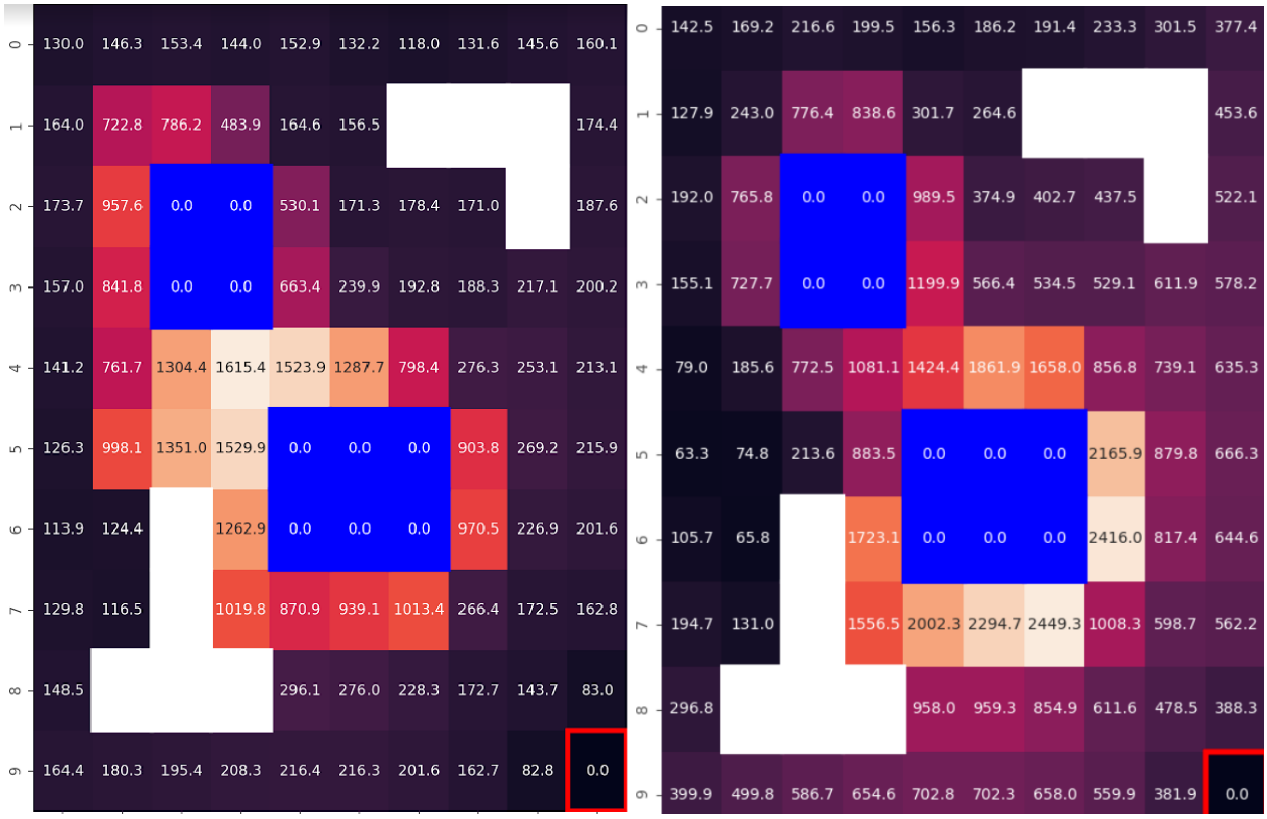


Figure 5. left-a) - Variance using sampling and (right -b) - learning method

Finding new variance online: https://www.quora.com/Is-it-possible-to-calculate-variance-using-old-variance-and-a-new-value

c. Threshold on variance for expert calculation

So far we have learned Q-values function and variance of returns for ALG2 but still we have not formulated criteria of when the expert should be called. Having come so far and based on the figures it is safe to expect that we need to set and fine tune the threshold on variance above which the expert will be called during run time (based on the argument that the expert will be called when risk is higher).

Hence, to find the threshold on variance to make expert calls such that we achieve maximum return, we ran the test on ALG2 where the agent follows a learned policy for 200 episodes and calls experts only if Variance in the current state is equal or higher than the set threshold. Returns for the threshold is reported as the average from 200 episodes. The result is mentioned in the plot below showing plot for Returns vs.Threshold.

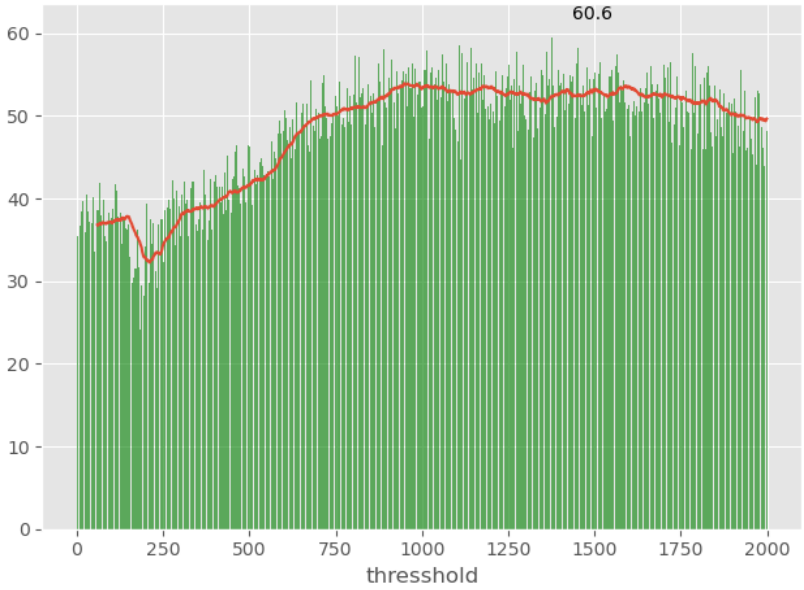


Figure 6. Returns for various thresholds on variance.

We can see that the returns slowly decrease after threshold limit of about 900 in variance and at that limit algorithm achieves the returns of around 52-54. This is expected as one can argue that a lower than optimal threshold agent is less confident in the policy and relies heavily on agent hence taking costly steps. With higher than optimal thresholds the agent is over confident on its policy and dynamics and tries to be autonomous even though its in a critical state, again achieving lower returns. The policy learned for ALG2 for optimal threshold is as shown in **figure 7** below.

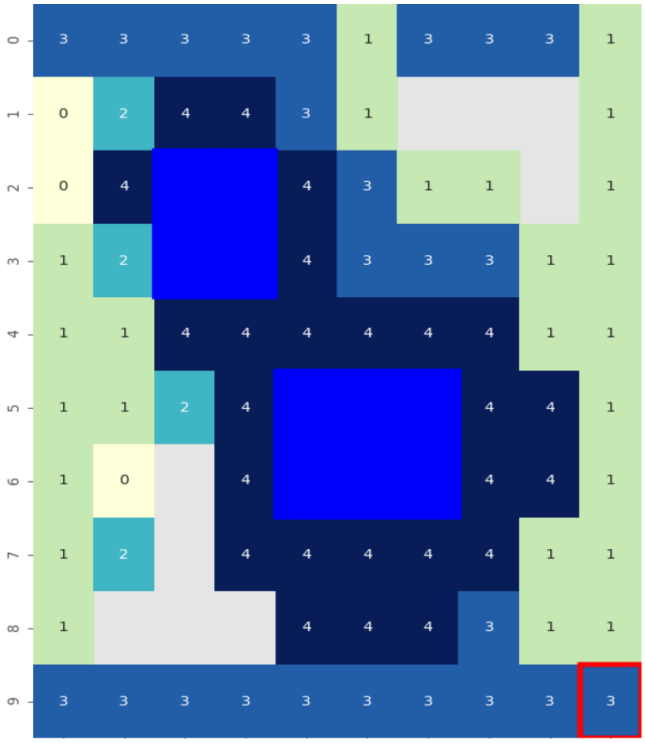


Figure 7. ALG2 learned policy with optimal threshold

8. Comparing with Baseline, ALG1 and ALG2

Final step would be to compare that our algorithm (ALG2) outperforms the baseline i.e. without the expert as well as ALG1 in a non-deterministic environment. Following **Table 3** shows the returns from trained agent on respective algorithms.

Algorithm	Average Return during run time (200 episodes)
Baseline (without expert)	36.8
ALG1 (with expert)	41.8
ALG2 (with expert)	53.2

Table 3. Algorithms Comparison

We can see that ALG2 outperforms the baseline by about 50% in terms of returns and ALG1 by 25%. ALG2 can be considered cost effective on another note as well, based on the fact that it does not call expert even once throughout the training. While agent calls the expert around 1.4 million times during its training period of 10 million episodes resulting in expert fatigue if the expert is a human. Hence ALG2 can be considered superior to its competitor and helps in achieving returns for task with faulty dynamics or where high risk are involved.

9. Current Work and Future Plan

Now we are working on to push the algorithm over Deep Reinforcement Learning using Duelling Deep Q networks also known as duelling DQN. First, We have applied DQN on this toy problem. We are currently in the process of verifying the results obtained through Deep Q networks. Once this is done we are planning to push this algorithm on a more complex problem over some Physics simulator, say Mujoco or Bullet.

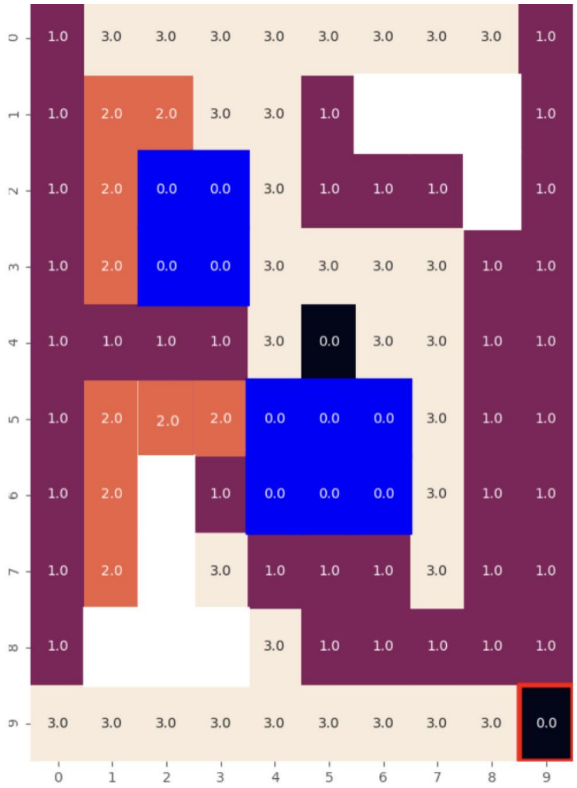
Reference

1. *David B. Thomas and Wayne Luk, "Estimation of Sample Mean and Variance for Monte-Carlo Simulations", Imperial College London, {dt10,wl}@doc.ic.ac.uk*

2. *Aviv Tamar, Dotan Di Castro, Shie Mannor, "Learning the Variance of the Reward-To-Go", JMLR:v17:14-335, 2016, http://jmlr.org/papers/v17/14-335.html*

3. *Shie Mannor, John N. Tsitsiklis, "Algorithmic aspects of mean–variance optimization in Markov", 2013, European Journal of Operational Research, Algorithmic aspects of mean–variance optimization in Markov decision processes - ScienceDirect*

Appendix



Although the agent is not exactly trying to imitate the expert, regardless the actions taken by the expert, will affect the subsequent q value updates in the agent's model. So clearly the experts hardcoded actions affect the policy. How are we sure that our hardcoded actions are optimal given our reward structure and that these actions are not confusing the agent we are trying to train.

For ex: the hardcoded action at (2,1) [x,y] is to go left, but what if the optimal action with our particular reward structure was to actually to go up and around the obstacle on the right. How can we be sure?

Numbers	Corresponding Action
0	Up
1	Down
2	Left
3	Right
4	Expert call

Figure 8. Cautious agent coding (ignore the 0's inside the traps) and action encoding

In the cells close to the trap, don't we expect the q values to be higher and sure of to go away from the trap, or call the expert?

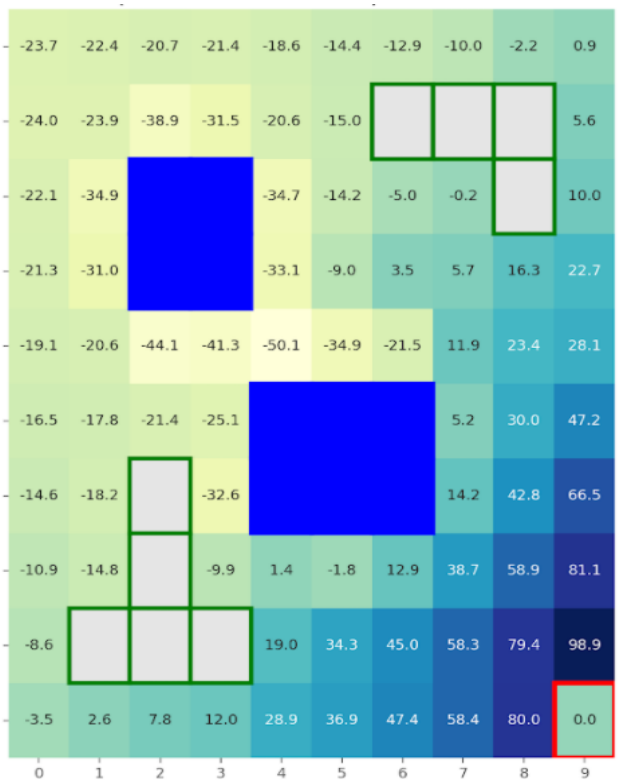


Figure 9. Value inside each square is the action value for the corresponding (state, action) pair where action is the one with highest action value out of the 5 actions available to the agent in the state. "MaxQvalue" for each state.

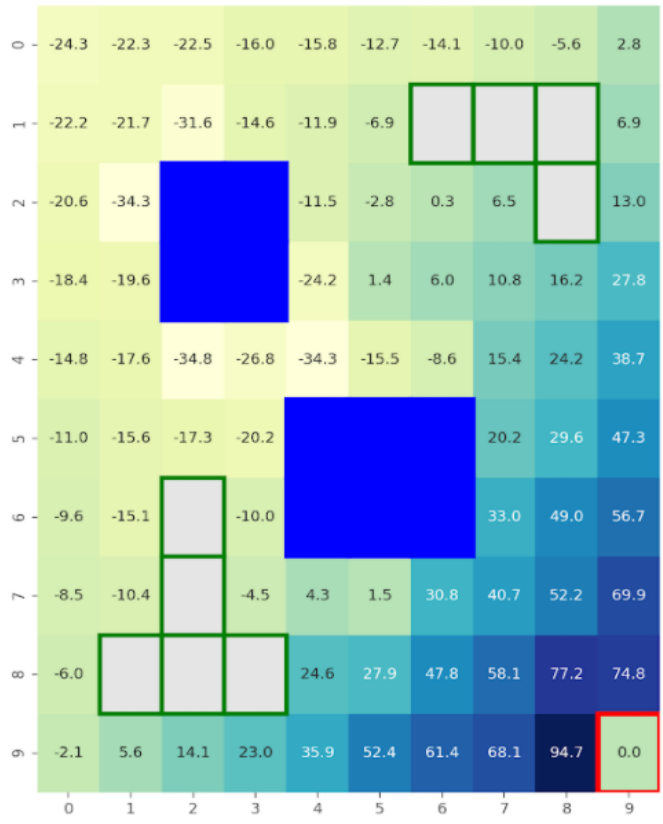


Figure 10. ALG2 MaxQ-value plot