Query network based limited guidance for RL agents

Akshay Sharma
Graduate Student, MSME-R
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
Email: akshaysh@andrew.cmu.edu

Dr. Katia Sycara
Advanced Agent-Robotics Lab
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Email: katia@andrew.cmu.edu

ABSTRACT

The last few years have seen a lot of progress in deep reinforcement learning (DRL) which have allowed autonomous agents to learn difficult tasks. Though these algorithms boast of a very good performance, these require large amounts of training samples to achieve it, and even then such trained agents when deployed can still at times get confused and take a lot of wrong actions. In this work we have tried to use an expert agent to help agents trained using Reinforcement Learning (RL). We work in a limited communication setup i.e. the expert can not help the agent all the time, and try to learn a "Query network" which can learn when the RL agent gets most confused and then queries the expert only at those steps. Our experiments on a grid world navigation task, shows an improvement in the number of steps the agent takes to reach its goal.

1 INTRODUCTION

Humans have the ability to explore their environment and learn tasks using trial and error. To replicate this behavior in robots people have tried to use Reinforcement Learning (RL). RL algorithms allow agents to explore the environment, and find behaviors which can maximize some underlying reward function. The last few years have seen a lot of new and powerful RL algorithms like DDPG [10], PPO [9], TRPO [8], which have allowed artificial robots to learn and preform a plethora of tasks which have been earlier used to judge human intelligence. Even though these algorithms boast of a very good performance, these require a tremendous amounts of training samples to achieve it. Such trained agents when deployed can still at times get confused and take a lot of wrong actions. If one tries to compare such behaviors with a human's performance who has been trained in doing a certain task, one potential workaround to this problem comes out as the presence of a teacher or an expert which always knows the best possible action in each state. We built up on the same idea, and tried to build a "Query network" which can be trained on top of the agent and learn when is the agent getting confused, and query the expert at those states. For this setup we consider a limited communication scenario, i.e. the expert can not be queried all the times, and work on a grid world based navigation task in the MiniGrid [2] based BabayAI [1] environment. We use PPO to train an agent in the said environment and then train the Query network using the agent's observations, and its policy distribution over possible actions to learn whether it needs expert's help or not.

2 RELATED WORK

Confusion in a RL agent can be thought of its need to explore its environment more. Exploration helps a RL agent in finding behaviors which can help them in getting high task rewards. [5,7] deal with this as curiosity of the RL agent, and have come up with agents which can explore their environment better. Working on the same idea of extracting good behaviors. [3] try to randomize the action selection of agent as much as possible which allows the agent to visit more diverse states and possibly reduce its confusion when it is deployed.
People have also tried to observe and learn agent behavior, for the task of differentiating between different agent types in [6]. We also have tried to observe agent behavior but with the intention of judging the agent's inability to take a decisive action and intervene in those scenarios by taking an expert's help.

3 METHOD

We divide the whole process into 2 parts: (1) Training an agent using PPO, (2) Training a Query network on top of this trained agent.

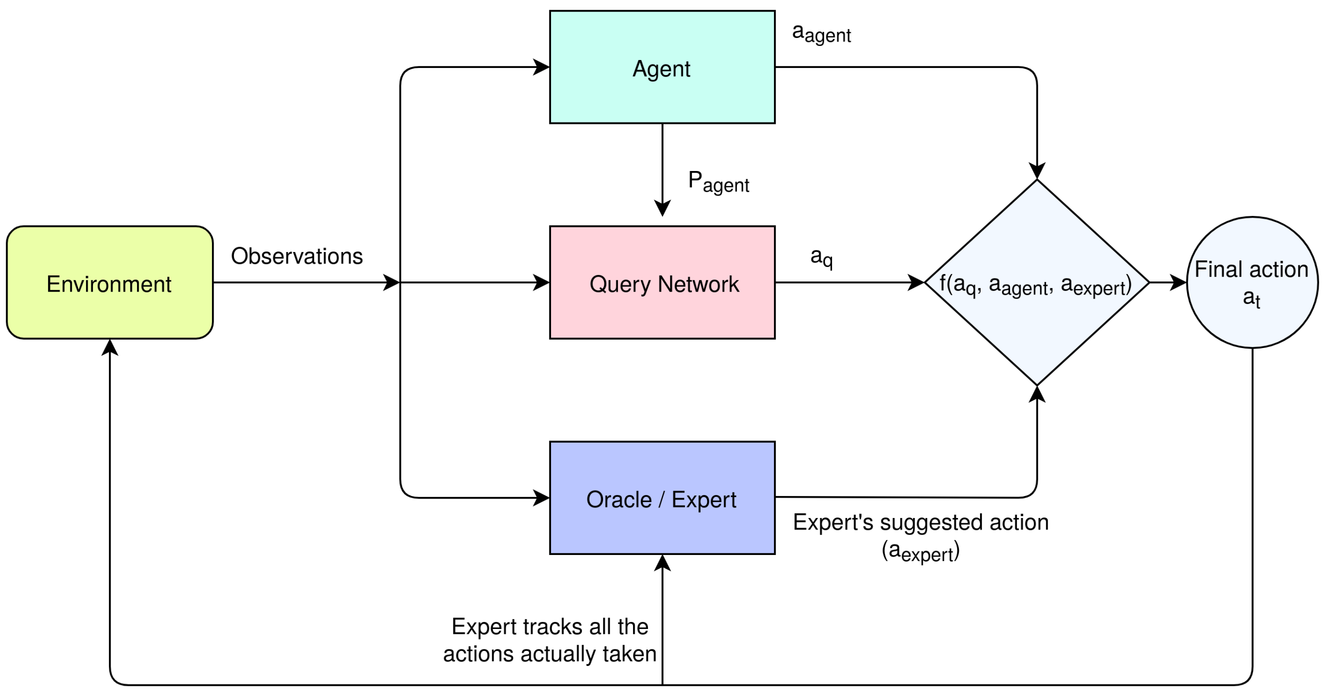
Agent
a_{agent}
P_{agent}
Environment
Observations
Query Network
a_q
$f(a_q, a_{agent}, a_{expert})$
Final action a_t
Oracle / Expert
Expert's suggested action (a_{expert})
Expert tracks all the actions actually taken

Fig. 1. Overall architecture of the training procedure for Query network

3.1 Agent

We used a RL agent which is trained using Proximal Policy Optimization (PPO) [9] algorithm. For this we used the existing PPO training framework already implemented in BabyAI. This algorithm takes the partially observable view, and the task instruction text as the input, and gives a probability distribution over the actions.

3.2 Expert

The BabyAI [1] environment provides an expert agent for each task. This expert agent is a heuristic based agent and has access to the whole environment, and can run a shortest path algorithm to find the best sequence of actions to the goal. This expert can also work along with the RL agent by taking in as input the history of actions taken so far, and providing corrective actions if required.

3.3 Query network

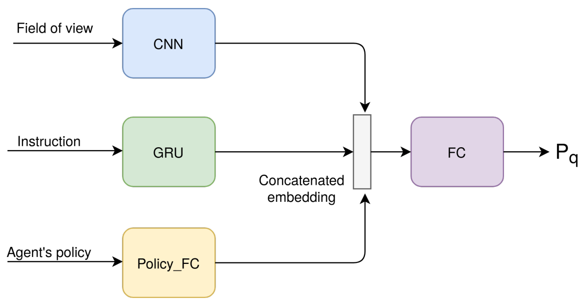

Fig. 2. Query network architecture

After we have trained the agent, we freeze its weights and deploy it in the environment, and train the Query network on this setup. The Query network is a deep neural network which takes in as input the same information as the agent along with the agent's output probability distribution over its actions as shown in Fig 1 at each step. At each step, the environment gives the current observations of the agent which include agent's current field of view, and the instructions defining the goal. These are used by the agent to give a probability distribution P_{agent} over its actions, from which an action a_{agent} can be selected by taking the argmax of the probability distribution.
As shown in Fig 2 the query network it self consists of a convolution neural network (CNN) block which processes the field of view of the agent, a gated recurrent unit (GRU) to process the task instruction, one fully connected (Policy_FC) block to process the agent's policy distribution, and another fully connected (FC) block which processes the concatenated field of view, instruction, and agent policy embeddings to output a probability value regarding whether to query the expert or not.
So it takes in the observations and P_{agent} and gives an output probability, $P_q \in [0,1]$, where 0 means that the query network thinks that the RL agent is not confused at all, and 1 means that the RL agent is completely confused and the expert should be queried. The final decision of the query network is found by thresholding the probability at 0.5, i.e.

$$a_q = \begin{cases} 0 & if\, P_q < 0.5 \\ 1 & if\, P_q \geq 0.5 \end{cases} \tag{1}$$

The final action a_t used to actually take a step in the environment is selected by taking into account decision of the query policy on whether to query the expert or not,

$$a_t = f(a_q, a_{agent}, a_{expert}) = (1 - a_q) * a_{agent} + a_q * a_{expert} \quad (2)$$

The target output or the ground truth for the query network is decided by comparing the agent's action against the expert's action.

$$a_{q,target} = \begin{cases} 0 & \text{if } a_{agent} = a_{expert} \\ 1 & \text{if } a_{agent} \neq a_{expert} \end{cases} \quad (3)$$

3.4 Loss function

The query network is trained using a custom loss function. For this we first take the RL agent's logit output of the last layer i.e. the raw scores before probability computation over the action space, and denote it by λ. Then we calculate a probability distribution only on two actions, one selected by the expert a_{expert}, and the action with the highest logit value which is not equal to the a_{expert}, and denote it by a'

$$\mathcal{P}_{a'} = \frac{\lambda_{a'}}{\lambda_{a'} + \lambda_{a_{expert}}}, \mathcal{P}_{a_{expert}} = \frac{\lambda_{a_{expert}}}{\lambda_{a'} + \lambda_{a_{expert}}} \quad (4)$$

Then we calculate the loss such that the network learns that if the logit score of a' is not very far from that of the one selected by the expert, the agent might be confident enough and does not need help.

$$L = -(1 - P_q) * (\mathcal{P}_{a_{expert}}) - (P_q) * (\mathcal{P}_{a'}) \quad (5)$$

If we are using a trained RL agent, it might happen that the number of times it is confused is low, in those cases we have to deal with class imbalance, i.e. the number of positive samples (where $a_{q,target} = 1$) are less that negative samples (where $a_{q,target} = 0$). In these cases the network can become biased towards always giving $P_q = 0$, as most samples do not require assistance. To counter this we also keep a running average of the number of positive, and negative samples.

$$w_{pos} = \frac{number\ of\ positive\ samples}{total\ samples} \quad (6)$$

$$w_{neg} = \frac{number\ of\ negative\ samples}{total\ samples} \quad (7)$$

Using these weights we do an inverse weighing, i.e. multiply the loss term for positive samples with w_{neg}, and that of negative samples with w_{pos}. So our final loss function looks like:

$$L = -w_{pos} * (1 - P_q) * (\mathcal{P}_{a_{expert}}) - w_{neg} * (P_q) * (\mathcal{P}_{a'}) \quad (8)$$

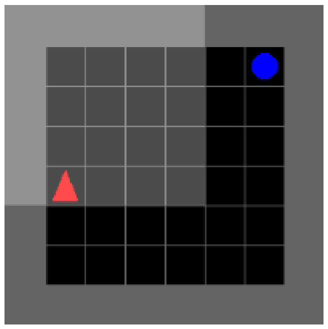

Fig. 3. Example of a generic navigation task in BabyAI. Here the agent is given the instruction, "Go To the Blue Ball".

4 EXPERIMENTS

The RL agent's neural network architecture, its training algorithm provided in BabyAI [1], our implementation of the Query network and its training are all written using PyTorch [4]

4.1 Environment

For this work we used BabyAI [1] as we wanted a grid-world environment, which can allow us to have a not too complex environment along with an expert which can provide us with the correct action at every state. BabyAI. As shows in Fig 3 the environment consists of a $7X7$ grid including a boundary, which leaves a $6X6$ grid for the agent to explore. The agent is represented by a red triangle and has a limited field of view of size $5X5$ in front of it marked as the grey shaded area in the Fig 3. We used only the navigation task in our work, which just asks the agent to reach to a certain object in the environment. Here the agent is supposed to reach the blue ball shown in the top right corner of the Fig 3. The task is conveyed as a text according to a predefined grammar in BabyAI [1].

4.1.1 Observations

At every step the environment provides a $5X5$ view of the area ahead of the agent in the agent facing direction. This view clearly marks the agent, the objects, the walls, and the empty space as integers along with the color information of each object.

4.1.2 Action space

The agent has seven actions in its action space: **Forward, Turn left, Turn right, Pick, Drop, Open, Done**. In our context of the navigation task in a $7X7$ grid, the **Open** action is not usable. The agent has to specifically call the **Done** action to indicate that the task is complete.

4.1.3 Reward structure

This setup has a sparse reward structure, i.e. the agent gets a reward of +1 only when it reaches the target object, and 0 otherwise.

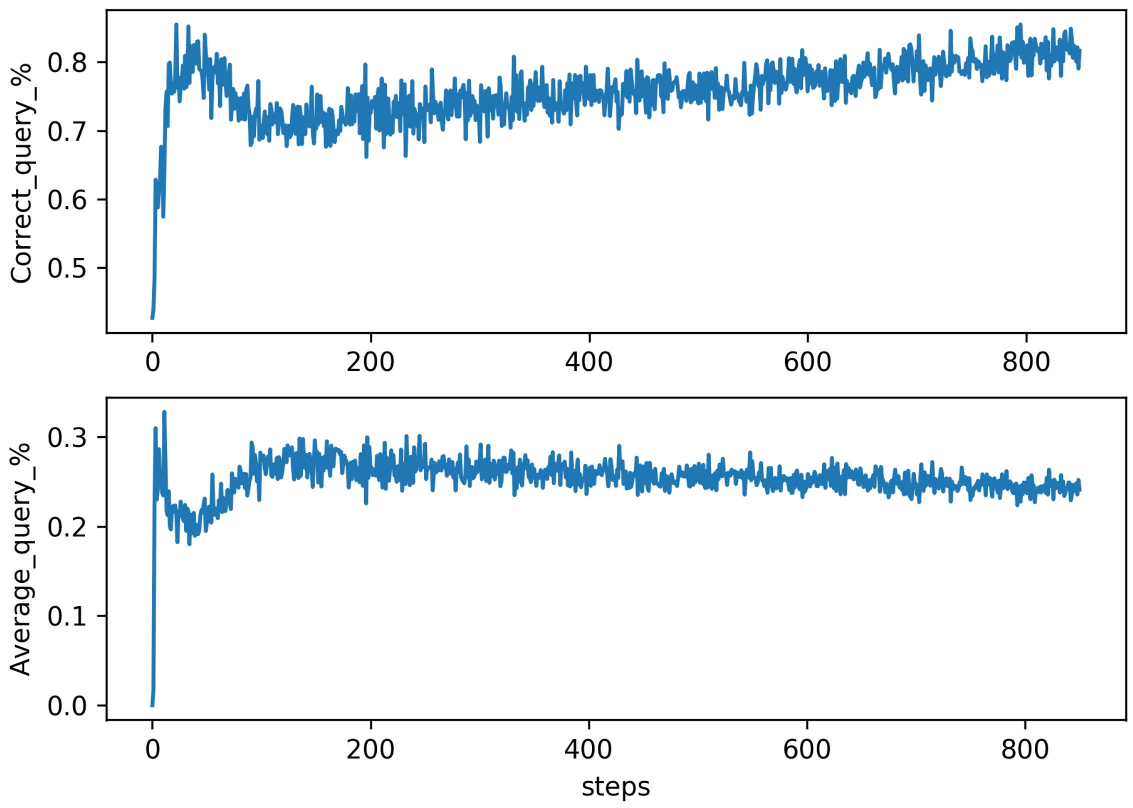

Fig. 4. The plots show the evolution of "Correct query %", and "Average query %" as the training progress.

5 RESULTS AND DISCUSSIONS

We ran our experiments on the navigation task in the BabyAI's [1] $7X7$ grid environment, training the RL agent and the query network as described in the section 3.

5.1 Evaluation Metrics

We evaluated the results using the following metrics:

1. Percentage of queries per episode: As stated earlier that we want to emulate a limited communication setting where the RL agent can't ask the expert at all time steps, so we want our query network to query the expert as less number of times as it can. We measure this criteria by calculating the percentage of steps on which the query network queried the expert in every episode and average this over all the episodes.
2. Percentage of correct queries per episode: We also want the query network to make legitimate queries, i.e. all the queries it makes should be actually required. We check the legitimacy of each query by checking whether at those queried steps, $a_{q,target} = 1$ or not. We calculate this over all the queried steps in an episode and then average it over all the episodes.
3. Average return: The mean return shows the actual performance of the RL agent. If the query network worked it should increase. We measure it by taking the average of the cumulative reward of each episode over all the episodes.
4. Average episode length: A high average episode length indicates that the agent might be taking a lot of wrong actions in the environment and is just roaming around. Query network should ideally lead to a decease in this metric.

Table 1. Comparing RL agent's performance with and without the query network. Here the query network was used after its training has converged. For Average return, higher the value the better it is, and for Average episode length a lower value is better.

Metric	Without Query	With Query
Average query %	-	25%
Correct query %	-	84%
Average return	0.918	**0.925**
Average episode length	5.79	**5.39**

5.2 Results

We trained the query network on top of the trained RL agent, and evaluated the trained query network along the RL agent during training on a test environment. As can be seen from Fig 4, the Average query % initially rises as the query network starts to learn, and then at the end settles down to making queries around 25% of the time. At the same time, the percentage of queries which are correct also increase and settles to around 84%. These results do show promising performance of the query network as we expected.

We also tested whether the query network is actually making a difference by running the trained RL agent with and without the query policy. As can be seen in table 1, the Average return increased from 0.918 to 0.925 when we used the query network along with the agent, and the Average episode length decreased from 5.79 to 5.39.

6 FUTURE WORK

In this work, we have tested the query network setup only on a simple navigation task, but the BabyAI [1] provides a lot of other complex takes like picking and placing objects, multiple room environments, etc which should be further tested out. Another good direction will be to apply this method in environments other than gridworld which will have their own unique challenges.

Acknowledgements

This work has been partially supported by ARL award W911NF-1920146. I would specially like to thank my advisor Dr. Katia Sycara, and my lab-mates Swaminathan Gurumurthy, Siddharth Ghiya, Abhijeet Ghawade, Keitaro Nishimura, Rohit Jena, and Vidhi Jain for helping and collaborating with me on this and different other projects.

References

[1] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. 2018.

[2] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[5] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[6] Neil C. Rabinowitz, Frank Perbet, H. Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. Machine theory of mind. *CoRR*, abs/1802.07740, 2018.

[7] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

[8] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. *32nd International Conference on Machine Learning, ICML 2015*, 3:1889–1897, 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. pages 1–12, 2017.

[10] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1:605–619, 2014.