Generalized QMDP-Net

Xiaobai Ma, Zhenkai Wang, and **Sheng Li**
Stanford University
Stanford, California 94305

Abstract

In this paper, we extend the work of QMDP-net (Karkus, Hsu, and Lee 2017) for more general POMDP problems. Inheriting the advantage of embedding the structure prior of a QMDP solver from the original QMDP-net, we further generalize its structure to deal with continuous state space based on the idea of state abstraction. In addition, by allowing the network to learn initial parameters of its embedding QMDP solver, we are able to improve both the speed and performance of the network. Our experiment shows that the modified network is able to outperform the GRU-network on two out of the three test Atari games.

1 Introduction

Sequential decision making problems could be very challenging due to uncertainties and dimension. A common approach is to model such problems as a Markov Decision Process (MDP). If partial observability is involved, in which case the agent receives information that only partially describes the current state of affairs, Partial Observable Markov Decision Process (POMDP) model is usually used. Solving POMDP problems with exact methods is computationally intractable due to the curse of dimension. Approximate methods such as FSVI (Shani, Brafman, and Shimony 2007), SARSOP (Kurniawati, Hsu, and Lee 2008), and PLEASE (Zhang et al. 2015) are developed to improve efficiency. Most of them use sample-based methods to generate a reachable belief space, but the performance decreases as the belief space grows. Deep neural networks have shown strong performance in many domains and provide a distinct new approach to POMDP problems. The deep Q-network (DQN) has successfully solved many Atari games with complex visual input through convolutional neural networks (Mnih et al. 2015). The Deep Recurrent Q-learning Network (DRQN) replaces the fully connected layer of DQN by a recurrent LSTM layer, and is able to deal with partial observaiblity (Hausknecht and Stone 2015). However, DRQN doesn't exploit the underlying structure of a POMDP problem.

QMDP-net is a recently introduced neural network architecture for POMDP problems. It is designed to embed the QMDP (Cassandra and Kaelbling 2016) structure, which is a famous model-based off-line POMDP solver, in its network architecture. The QMDP-net replaces the belief update and planning module in QMDP with neural networks and thus enables model-free end-to-end training. Limited by its structure, the original QMDP-net could only work on 2D grid world problems. We extend its structure so that it could handle continuous state space problems. Another limitation of the original QMDP-net is that it needs environment parameters (e.g. the initial belief, the map and the goal location in a maze navigation problem) as an input, and uses constant initialization on the state value at each step. We change these inputs into trainable parameters and thus making the network require less prior knowledge and can be trained more efficiently.

In this paper, we use Vanilla Policy Gradient (VPG) algorithm to train our modified QMDP-net on three Atari games and compare its performance against the GRU-net (Cho et al. 2014) which is a famous recurrent neural network showing great performance on many sequential tasks.

2 Background

POMDP

A POMDP is defined by a tuple (S, A, O, T, Z, R), where S, A, and O are state, action, and observation spaces. $T(s' \mid s, a)$ is the transition model giving the probability of reaching state s' by executing action a at state s. $Z(o \mid s, a)$ is the observation model giving the probability of receiving observation o given s and a. $R(s, a)$ is the immediate reward after the agent taking action a at s.

QMDP

QMDP is a traditional MDP solver developed for POMDPs with simple discrete state, action, and observation space. It contains a Bayesian filter and a QMDP planner.

* Source code of the project is available at `https://github.com/maxiaoba/QMDPNET`

Bayesian Filter

In QMDP solver, a belief b is maintained and updated using Bayesian filter:

$$b_t(s') = \alpha Z(o \mid s', a) \sum_{s \in S} T(s' \mid s, a) b_{t-1}(s), \quad (1)$$

where α is a normalizing factor.

QMDP Planner

QMDP is a simple, but fast approximate planner for POMDP problems. Instead of adopting belief over state (i.e. uncertainty about the true state), QMDP assumes full observability. The QMDP algorithm iteratively performs Bellman updates on the underlying MDP:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s \mid s', a) V(s'), \quad (2)$$

$$V(s) = \max_{a \in A} Q(s, a). \quad (3)$$

Given the current belief b and the most updated $Q(s, a)$, the value of taking action a at b is calculated by:

$$q(a) = \sum_s Q(s, a) b(s)$$

An action is chosen by the probability distribution got from the softmax on action values.

3 Generalized QMDP-Net

QMDP solver requires prior knowledge of the model (T and Z). Instead of using model knowledge, QMDP-net encodes the POMDP model structure into a recurrent neural network, as illustrated in Figure 1, and learn the model during training. This gives QMDP-net ability to combine the advantage of both model-based planning (i.e. the POMDP model structure) and model-free learning (i.e. no knowledge on model parameters).

The QMDP-net embeds a POMDP model to approximate the true POMDP model. We denote this encoded POMDP model as (S, A, O, T, Z, R) and the true POMDP model as $(\underline{S}, \underline{A}, \underline{O}, \underline{T}, \underline{Z}, \underline{R})$.

State Abstraction

Although many practical POMDP problems have continuous state space. The state space can usually be clustered into finite abstract states. In each abstract state, the included states have similar values and would prefer similar actions. Traditional solutions along this path need to handcraft state abstraction for neural network. In our network, this abstraction is learned and embedded in the structure. This is the foundation of our idea of using QMDP-net to encode the state space into finite abstract states and solve POMDPs with the QMDP planner.

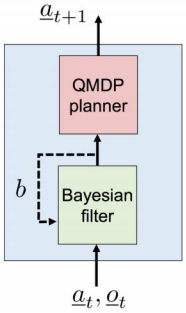


Figure 1: A QMDP-net is an RNN that imposes structure priors for sequential decision making under partial observability. It embeds a Bayesian filter and the QMDP algorithm in the network. The hidden state of the RNN encodes the belief for POMDP planning.

Bayesian Filter Module

The filter module takes in belief $b_t(s)$, action a_t, environment parameter θ, observation o_t and returns the belief of next time step b_{t+1}. In QMDP-net, the initial belief b_0 and the environment parameter θ are inputs to the network. We change them into trainable parameters. We choose the dimension of θ the same as b_0, which is the number of abstract states, $|S|$.

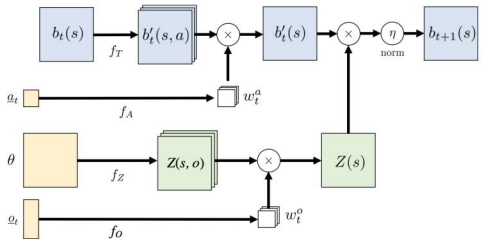


Figure 2: Bayesian filter module.

As illustrated in Figure 3, there are four layers performing the belief update task.

- Filter f_T encodes state transition model appears in the Bayesian filter, as shown in Equation 1. In QMDP-net, f_T is a 2D convolution layer. Although the 2D convolution layer are very intuitive and efficient on representing the transition model of a 2D grid navigation problem, it restricts the QMDP-net to be applied on other POMDP problems. We change f_T to fully connected layers with input dimension $|S|$ and output dimension $|S| \times |A|$, apply softmax on $|S|$ dimension for representing the valid transition probability. Since generally, $T(s' \mid s, a) \neq 0$ for all or random tuples of (s, s', a), a fully-connected layer is more suitable than a partially connected convolutional layer.
- Filter f_A performs "soft indexing" by mapping $\underline{a}_t \in \underline{A}$ to w^a_t, a probability distribution of actions in A. This allows

$|A| \neq |\underline{A}|$. The soft indexing is done by:

$$b'_t(s) = \sum_{a \in A} b'_t(s,a) w_t^a \tag{4}$$

f_A is a fully-connected layer. In the experiment, we let $|A| = |\underline{A}|$.

- Filter f_Z maps the environment variable θ to $Z(o|s)$ representing the probability of receiving observation o at state s. Similar to f_T, f_Z is a 2D convolution layer in the original QMDP-net and we change it to a fully connected layer with input dimension $|S|$ and output dimension $|S| \times |O|$.
- Filter f_O works similarly as f_A. It also performs "soft indexing" that maps $\underline{o}_t \in \underline{O}$ to $o_t \in \underline{O}$ through:

$$Z(s) = \sum_{o \in O} Z(o \mid s) w_t^o \tag{5}$$

Filter f_O allows the network to take continuous observation and map it to a discrete observation space. This part is the same as the original QMDP-net.

The next belief state $b_{t+1}(s)$ is obtained by multiplying $Z(s)$ and $b'_t(s)$ element-wisely,.

QMDP Planner Module

The QMDP planner module illustrated in Figure 3 encodes the Bellman updates and action extraction using neural networks.

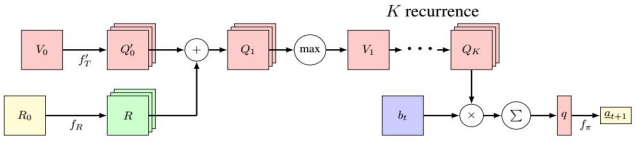


Figure 3: QMDP module.

Starting from the value function V_0 and reward function R_0, the planner module computes the next horizon value Q_1 and V_1 by executing equation 2 and 3. By repeating this process K times, the network calculates the state-action value Q_K of horizon K.

The transition probability is calculated through transition layer f'_T similar to f_T. In the original work, QMDP-net requires the prior knowledge of environment parameter θ as the input for a convolutionaly layer f_R to calculate the reward R. Here we make θ a trainable variable R_0 and change f_R to a fully-connected layer.

The original QMDP-net initializes V_0 to zeros at each step and uses $K = 30$ to get a near-to-converge state-action value. However, the ideal state value should remain "near" constant at each step. In a real POMDP, the state value is constant. We use the word "near" here since the embedded POMDP model is changing gradually as the network is being trained. Even though, using a constant initial value for V at each step is not the most efficient way since it fails to utilize the calculation from previous steps. Thus, we change the V_0 to a trainable variable hoping it to learn the ideal state value through training. By doing so, we are able to reduce horizon K since V_0 is faster to converge, which also speeds up the forward and backward propagation.

After K recurrence, the QMDP module uses filter f_π to map Q_K to the actual action space $\underline{A}$, where f_π is a softmax layer. The action with maximum probability is then chosen as next action $\underline{a}_{t+1}$.

Training Algorithm

In the original paper, the authors train the QMDP-net using imitation learning methods which requires expert trajectories. We want to validate the performance of training QMDP-net with reinforcement learning methods. Thus, we use Vanilla Policy Gradient (VPG) shown in Algorithm 1 (Sutton et al. 2000) to train the network.

Algorithm 1 Vanilla Policy Gradient

1: **Input:** (Environment E, Stochastic policies π_θ, Baseline value function V)
2: **for** $i = 1, \ldots, N_{iter}$ **do**
3: $\quad paths \leftarrow \textsc{Rollout}(E, \pi_\theta)$
4: $\quad ad \leftarrow \textsc{ProcessPaths}(paths, V)$
5: $\quad \pi_\theta \leftarrow \textsc{PolicyOptimization}(\pi_\theta, paths, ad)$
6: **end for**

In $Rollout$, we collect a sequence of (s, a, r, s') tuples by executing the current policy π_θ in environment E and receiving observation o.

In $ProcessPaths$, we calculate the advantage estimator with $ad = r + \gamma V(s') - V(s)$, where V is a function approximator (e.g. a neural network) that estimates the state value function. Calculating the advantage estimator this way is also referred as Advantage Actor-Critic (A2C). This is shown to be very useful in reducing the variance in reinforcement learning(Schulman et al. 2015). The value function V is then improved with the new paths. Now the paths contain a sequence of (s, a, ad, s') tuples.

In $PolicyOptimization$, we want to maximize the object function

$$J = \frac{1}{BatchSize} \sum_{(s,a,ad,s') \in paths} ad \cdot Pr(s, a, s' \mid \pi_\theta), \tag{6}$$

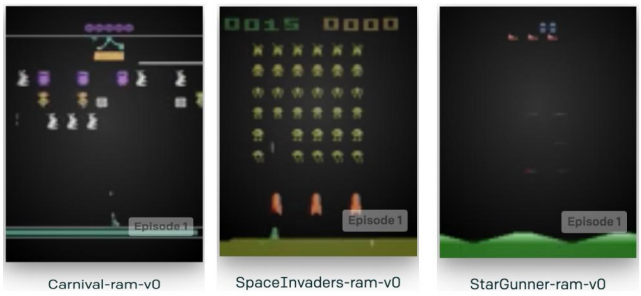
which is a differentiable function with respect to the policy's parameter θ.

4 Experiments and Discussion

Training Environment

OpenAI Gym (Brockman et al. 2016) provides a rich pool of environments for reinforcement learning (RL). One of the most famous types is the Atari games. Atari games are a series of simple 2D video games developed by Atari, Inc. When an Atari game is used as a RL environment, its state is fully

described by a ram of 128 bits. Although it is a discrete state space, the total state number is so large (2^{128}) that we could almost treat it as a continuous state space environment. To introduce the partial observability in the environment, we add a random mask to the ram such that certain number of random bits are set to 0 and use the masked ram as the observation to the agent. For the same reason, we could treat the observation space as continuous. The action space for Atari games is discrete. We test our network structure on 3 interesting Atari games (the Carnival, the Space Invaders, and the Star Gunner) since their rewards are relatively dense compared to other games.



(a) The Carnival. (b) The Space Invaders. (c) The Star Gunner.

Figure 4: The Atari Games.

Baseline

The baseline model we picked is the famous GRU-net (Cho et al. 2014). It has a one-directional single-layer recurrent unit structure. The number of hidden units in the GRU-net is set to 32.

Training Hyperparameters

We use VPG to train both our network and the GRU-net. The hyperparameters we use are listed in table 1

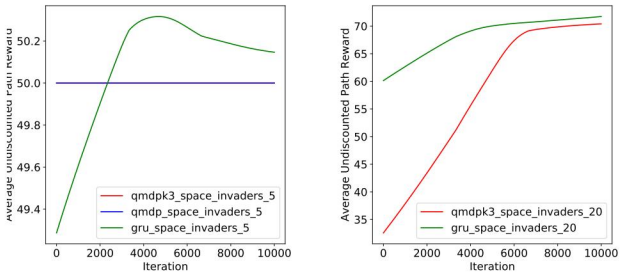
Number of Iterations	10000
Batch Size	2048
Max Path Length	400
Step Size	0.01
Discount	0.95

Table 1: Training Hyperparameters

Results

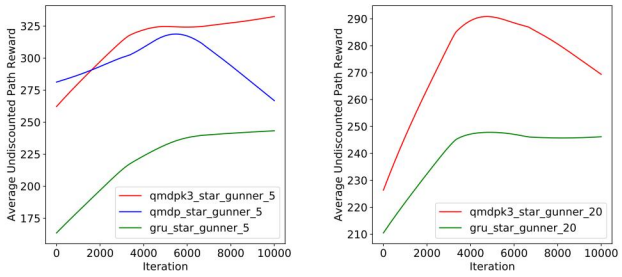
We test the performance of our QMDP-net in the three Atari games. In the experiment, we randomly mask 5 or 20 bits of the ram to introduce partial observability. Three different network structures are tested. The first one is the baseline model, the GRU-net with 32 hidden units. The second network is **the fix initial value QMDP-net**, in which we modified filter structures while keeping constant values for θ, R_0, V_0, b_0 and setting the recurrence in the QMDP planner $K = 30$. In the third network, **the trainable initial value QMDP-net**, we further make these values (R_0, V_0, b_0) as trainable parameters and set the recurrence in the QMDP planner $K = 3$. Both QMDP-nets use an embed POMDP of $|S| = 32$, $|O| = 17$, and $|A| = |\underline{A}|$.

The results for the Space Invader game are in Figure 5, the results for the Star Gunner game are in Figure 6, and the results for the Carnival game are in Figure 7.



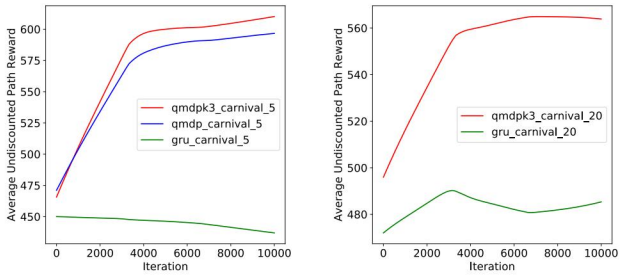
(a) Masking 5 bits. (b) Masking 20 bits.

Figure 5: Results of the Space Invader.



(a) Masking 5 bits. (b) Masking 20 bits.

Figure 6: Results of the Star Gunner.



(a) Masking 5 bits. (b) Masking 20 bits.

Figure 7: Results of the Carnival.

Based on the results for masking 5 bits, the trainable initial value QMDP-net performs consistently better than the fix initial value QMDP-net. In addition, with the reduced

recurrence number K, the trainable initial value QMDP-net is trained faster in one iteration. Therefore, for the masking 20 bits experiments, we only compare the trainable initial value QMDP-net against the baseline model and omit the fix initial value QMDP-net. For language simplicity, in the following page, we would refer the trainable initial value QMDP-net as QMDP-net unless specially mentioned.

The results show that the QMDP-net performs generally better than the GRU-net. It outperforms the GRU-net on the Star Gunner and the Carnival for both masking settings. Although on the Space Invader game, its reward doesn't exceed that of the GRU-net, the difference is small.

There are several unexpected results. Firstly, the average accumulated rewards for the two QMDP-nets on the Space Invader with 5 bits mask remains constant throughout the training. We found that all trajectory rewards are actually constant for the two QMDP-nets in the Space Invader with 5 bits mask experiments. This might be caused by the fact that most trajectories in Space Invader game converge to a same final reward and the one needs to be lucky to find outliers. Secondly, the performance of the GRU-net on 20 bits mask seems to be better than that on 5 bits mask. More experiments are required to determine whether this is a common case.

5 Conclusion and Future Work

In this paper, we explore the generalization of QMDP-net on Atari game environments with partial observability using reinforcement learning method. We also study the influence on the ability of learning the initial values. We compare the QMDP-net against a GRU-net baseline. The results show that the QMDP-net is generally more efficient than the baseline model.

There are many interesting expansions we could make based on the results of this paper.

More state-of-the-art policy optimization methods like Proximal Policy Optimization (PPO) (Schulman et al. 2017), and Advantage Asynchronous Actor-Critic (A3C) (Mnih et al. 2016) could be applied to further improve the performance.

It seems unnecessary to keep the f_Z and f_R in the network when we treat θ and R_0 as trainable variables. In future experiments, we would like to see how the performance is if we drop f_Z and f_R (and consequently dropping θ and R_0) and directly use R and $Z(s|o)$ as trainable variables.

Another limitation on the QMDP-net is the QMDP solver itself. The QMDP planner assumes no partial observability when computing the state values. This is proven to give a very loose upper bound on the actual value at each state. We are considering using the Fast Informed Bound (FIB) instead of QMDP to estimate the state value which gives a more tight upper bound. The equation for FIB is given as:

$$Q(s,a) = R(s,a) + \gamma \sum_{o \in O} \max_{a'} \sum_{s' \in S} Z(o|s',a) T(s \mid s',a) Q(s',a')$$

For the state abstraction, in the experiment, we kind of randomly choose the $|S|$ and $|O|$ for the embeded POMDP. If there is a systematic way of determining the size of the state and observation abstraction according to the real POMDP to solve, we could efficiently adjust the network so that it has high performance while using as least parameter numbers as possible.

Finally, although with our extensions, the QMDP-net could work on continous state and observation spaces, it is still limited to discrete action space. We could apply similar ideas of soft indexing on the action space and make it work on continuous space. For example, we could change f_π to a mean and a variance layer and thus output a Gaussian probability distribution on the continuous action space $|\underline{A}|$.

References

[Brockman et al. 2016] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym.

[Cassandra and Kaelbling 2016] Cassandra, A. R., and Kaelbling, L. P. 2016. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995: Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, July 9-12 1995*, 362. Morgan Kaufmann.

[Cho et al. 2014] Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Hausknecht and Stone 2015] Hausknecht, M., and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*.

[Karkus, Hsu, and Lee 2017] Karkus, P.; Hsu, D.; and Lee, W. S. 2017. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*.

[Kurniawati, Hsu, and Lee 2008] Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

[Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

[Schulman et al. 2015] Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[Schulman et al. 2017] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Shani, Brafman, and Shimony 2007] Shani, G.; Brafman, R. I.; and Shimony, S. E. 2007. Forward search value iteration for pomdps. In *IJCAI*, 2619–2624.

[Sutton et al. 2000] Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

[Zhang et al. 2015] Zhang, Z.; Hsu, D.; Lee, W. S.; Lim, Z. W.; and Bai, A. 2015. Please: Palm leaf search for pomdps with large observation spaces. In *Eighth Annual Symposium on Combinatorial Search*.