XQMIX: Extended QMix for StarCraft Multi-Agent Challenge

Mohammad Mahdi Rahimi

Electrical Engineering Department
Korea Advanced Institute of Science and Technology
291 Daehak-ro, Eoeun-dong, Yuseong-gu, Daejeon, South Korea
mahi@kaist.ac.kr

Abstract. This paper represents a detailed description of improving state of the art decentralized multi-agents in Starcraft II. Improvements and developments that seemed innovative and effective in the learning process are *Multi-Step Learning* and *Noisy Networks* which will be discussed in detail. Other approaches such as *Changing Optimizer*, *Learning Rate Decay Schedule*, *Change Loss Functions*, and *Add Regularization Term* are also applied, but, none of these attempts seemed to improve upon the baseline significantly. Each change perhaps required significant hyperparameter tuning, which was difficult given the costly amount of time it takes to show meaningful results in some environments.

Keywords: Multi-agent Reinforcement Learning, Game Theory, StarCarft Challenge, Q-Learning

1 Introduction

Multi-agent reinforcement learning is an exciting and challenging sub-field of RL, unlike single agents that optimize their personal rewards, multi-agents in cooperative settings optimize for the team's reward, which may involve self-sacrifices, developing strategies, and so forth. Multi-agents can be useful in any situation involving more than one AI, whether network packet routing, or playing a strategic video game.

This project focuses on improving the state of the art deep reinforcement learning AI for the game of StarCraft II, and concentrate on a mini-challenge of StarCraft proposed in the StarCraft Multi-Agent Challenge (SMAC) Paper.
The goal was to improve upon the baseline QMIX's test time win rate in terms of training speed by the number of timesteps.

2 Baseline Model

The **IQL** and **QMIX** are selected as baselines for this project. The reason is that these two are best-performing algorithms, while **IQL** is a very naïve approach for Multi-Agent, but it out-performs other methods in a few scenarios, and it is also a solid baseline to show how much cooperation between agents can improve the quality compared to independent decision making.

2.1 IQL

The most straightforward approach to learning in multi-agent settings is to use independently learning agents. This was attempted with Q-learning but did not perform well in practice. One issue is that each agent's policy changes during training, resulting in a non-stationary environment and preventing the naïve application of experience replay.

2.2 QMIX

The baseline model is called QMIX (Rashid et al., 2018), the best-performing SMAC model. (Fig. 1) is a visual representation of the model architecture from the original paper.

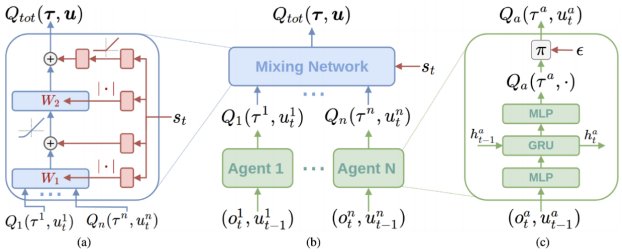

Fig. 1. The previous version of Parsian robots

Each agent (shown in green) that controls a unit is a Deep Recurrent Q-Network (Hausknecht Stone, 2015), which represents the agent's individual value function, Q_a. It takes as input o^{a_t}, the partial observation of the unit at timestep t, and $u^{a_{t-1}}$, the last action taken by the unit. The recurrent unit helps provide information that happened in previous timesteps.
The big picture is that we build a model that estimates the joint-action value function, $Q_{tot(t,u)}$, through a complex non-linear combination of the individual Q_a functions. Our policy is then the $argmax_u Q_{tot(t,u)}$, where u is the joint action

of all agents. We use a feed-forward network to handle the non-linear combination. This network is termed the "mixing network", shown in blue in (Fig. 1).

However, because our agents are decentralized, we have no access to the joint value function during testing. The authors of the paper extract the policy by ensuring (Fig. 8):

$$\underset{\mathbf{u}}{\operatorname{argmax}}\, Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \tag{1}$$

Fig. 2. The previous version of Parsian robots

If (Fig. 8) is true, then we do not need to evaluate Q values of all joint actions u for all the agents (an exponential search space), since we can instead just take the individual $argmax$ action for each agent and get the same result.

The baseline model also uses RMSProp, a fixed learning rate of .0005, double Q-learning, and L2 loss. The target Q-network is updated every 200 timesteps. Epsilon is decayed from 1 to .05 linearly over 50k timesteps. Gamma is chosen as 0.99. A replay buffer contains 5,000 of the most recent episodes, and 32 episodes are sampled uniformly at random for each update step.

3 XQMIX: Extended Model

XQMIX borrowed ideas and improvements from the "Rainbow" paper. The initial goal was to implement two main improvements: Multi Step Learning and Noisy Networks. And then try hyperparameter tuning or different combination of this ideas.

3.1 Multi-Step Learning

Q-learning accumulates a single reward and then uses the greedy action at the next step to bootstrap. Alternatively, forward-view *multi-step* targets can be used (Sutton 1988). We define the truncated n-step return from a given state S_t as

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}. \tag{1}$$

A multi-step variant of QMIX is then defined by minimizing the alternative loss.

$$\left(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t)\right)^2. \tag{2}$$

As (Sutton and Barto 1998) said in the original paper: Multi-step targets with suitably tuned n often lead to faster learning.
After adding multi-step learning to the model, I found The significant tuning was needed to choose the appropriate value of n for each scenario. (e.g. The $n = 5$ worked best for the *10m_vs_11m* scenario and $n = 3$ worked best for the *3s_vs_5z* scenario).

3.2 Noisy Net

The limitations of exploring usingϵ-greedy policies are clear in harder games, where many actions must be executed to collect the first reward. Noisy Nets(Fortunato et al. 2017) propose a noisy linear layer that combines a deterministic and noisy stream,

$$y = (b + W_x) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^w)x), \tag{3}$$

where ϵ^b and ϵ^w are random variables, and $\odot$ denotes the element-wise product. This transformation can then be used in place of the standard linear $y = b+Wx$. Over time, the network can learn to ignore the noisy stream, but will do so at different rates in different parts of the state space, allowing state-conditional exploration with a form of self-annealing.
I only used this Noisy Net instead of linear layers of RNN agents, which improved the exploration of agents and leads to better performance for some hard tasks, but a delayed convergence on easy ones.

3.3 Quadratic Mixer

A minor change is introduced to the QMIX mixer itself. The change involved replacing the absolute value function applied to the hypernetwork weights with the quadratic function $0.5x^2$. The performance of this replacement was equal to using the absolute value on some scenarios (e.g. *10m_vs_11m*) and extraordinarily better on the others. (e.g. *3s_vs_5z*). I theorize this is due to the fact that weight values tend to be low (¡ 1), and the quadratic function provides smoother gradients at these values than the absolute value function.

3.4 Other Experiments

Some other attempts to improve the model include creating a learning rate decay schedule (the baseline's learning rate is fixed at .0005), using the Adam optimizer instead of RMSProp, using Huber loss instead of L2 loss, and introducing a regularization term. However, none of these attempts seemed to improve upon the baseline. Each change perhaps required significant hyperparameter tuning, which was difficult given the costly amount of time it takes to even begin seeing results on some environments.

4 Results

4.1 Data Prepration

Data was provided from the SMAC environment. For every frame of the episode, the environment provides each agent with a feature vector corresponding to their local observations. During training time, global state information is also available. Due to limited time and resources, we mainly developed on 2 scenarios from each level:

- Easy
  - *2s3z* (symmetric)
  - *10m_vs_11m* (asymmetric)
- Hard
  - *3s_vs_5z* (asymmetric)
  - *2c_vs_64zg* (asymmetric)
- Super Hard
  - *MMM2* (asymmetric)
  - *corridor* (asymmetric)

4.2 Processing Unit

All the process and run has been doing on a laptop with 15GB ram and 12 Core CPU, accelerated by NVIDIA 1060 GPU. Each training session take about five to seven hours and a single training can be run at the time, all of training result captured for two million episode.

4.3 Plots

4.3.1 2s3z is the first easy scenario that I experienced on, there are no significant improvement to QMix with other methods except using quadratic mixer and 5-Step training together which slightly better than QMix in the beginning.

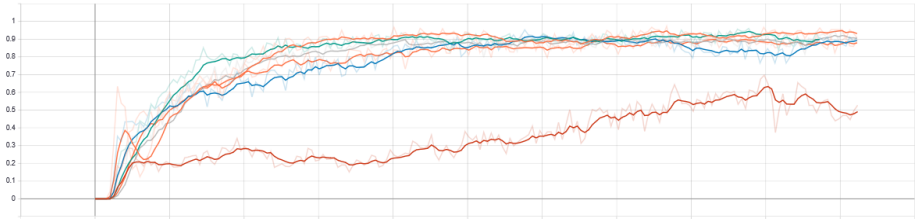

Fig. 3. — QMix, — IQL, — Noise, — Quadratic & 5-Steps, — 5-Steps

4.3.2 10m_vs_11m In this scenario the 5-Step Training outperform the QMix and any other methods of learning.

By my understanding better positioning for agents and dynamically changing it considering the situation is the key point of wining this scenario.

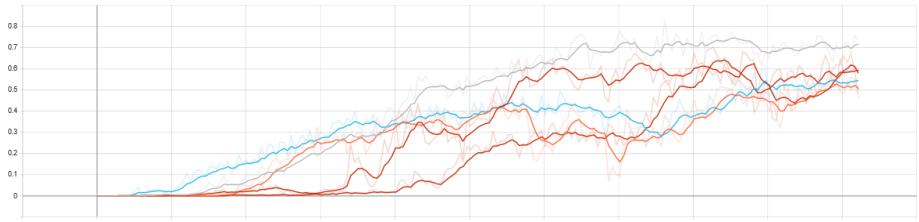

Fig. 4. — QMix, — Noise, — Quadratic, — 5-Steps, — 3-Steps

4.3.3 3s_vs_5z is one of hard scenarios, the final learned strategy was run and hit technique.

Both 3-Step and Noisy Net outperformed the QMix, so I combined two technique and the performance keeps improving even more.

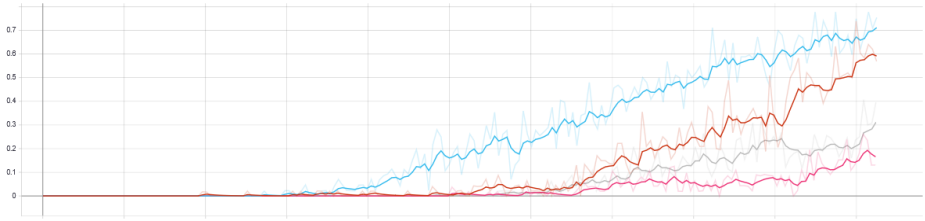

Fig. 5. — QMix, — Noise, — 3-Steps, — 3-Steps & Noise

4.3.4 2c_vs_64zg is another hard scenario that can won by good positioning but it needs a good exploration for fining good positioning strategy. Multi-Step methods performed so well in the beginning of training but in the end they converge to a lower score. On the other hand we have Noisy Net which performed worse than Multi-Step in the beginning of train but converge to better value at the end. Overall Noisy Net outperformed the QMix for this scenario.

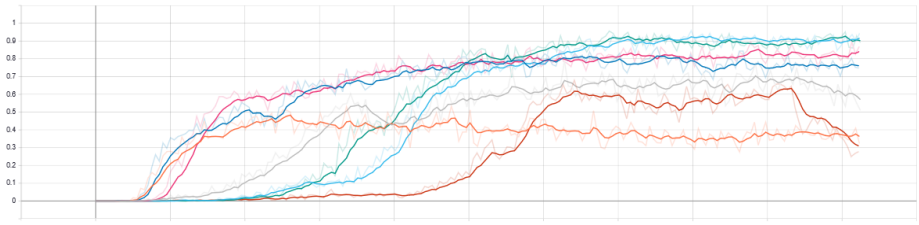

Fig. 6. — IQL, — QMix, — Noise, — 3-Steps, — 5-Steps, — 10-Steps, — Quadratic,

4.3.5 MMM2 is one of super hard scenario in which QMix do not perform very well, but we have Quadratic and 3-Steps training which outperformed the QMix.

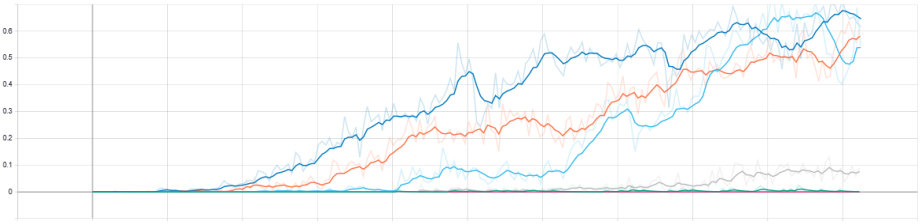

Fig. 7. — IQL, — QMix, — Noise, — 3-Steps, — 5-Steps, — Quadratic

4.3.6 corridor is one of scenarios which QMix can never win in 2Million Training trial and also no other customization of QMix win this scenario so the plot for win rate is always zero. So for a comparison the number of enemies killed are showed.

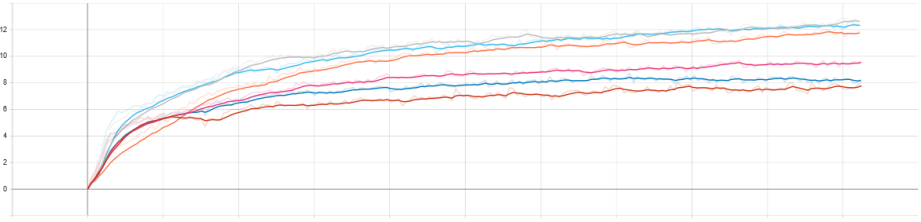

Fig. 8. Corridor Number of enemies Killed (— IQL, — QMix, — Noise, — 3-Steps, — 5-Steps, — Quadratic)

5 Conclusion

Model improvements were made are noisy net, multi-step and hypernetwork replacements, specifically the quadratic function replacement. My results show significant improvement on some tested scenarios after hyperparameter tuning and experimentation.

My first conclusion was seeking out implementation issues is difficult. For example, the first multi-step learning implementation had an issue with handling masking, but the trained model based off this buggy implementation was still able to learn, albeit worse than the baseline.

Another point was that good performance on one scenario does not necessarily translate to another. In order to gauge early on whether a model would perform, I tested the model on a many known scenarioes and methods performed poor on some shows excellent performance on others and vice-versa. However, combination of methods which performed better than baseline mostly leads to a extremely better method.

References

[1] Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. (2018) QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning.

[2] Hessel M, Modayil J, Van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M, Silver D (2017) Rainbow: Combining improvements in deep reinforcement learning

[3] Samvelyan, M., Rashid, T., de Witt, C., Farquhar, G., Nardelli, N., Rudner, T., Hung, C., Torr, P., Foerster, J., Whiteson, S. (2019) The Starcraft Multi-agent Challenge

[4] Hausknecht, M. and Stone, P. Deep Recurrent Q-Learning for Partially Observable MDPs. (2015)

[5] Fortunato, M.; Azar, M. G.; Piot, B.; Menick, J.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; Pietquin, O.; Blundell, C.; and Legg, S. 2017. Noisy networks for exploration. CoRR abs/1706.10295.

[6] https://openai.com/five/