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. Perhaps it required significant hyperparameter tuning.

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 rewards, multi-agents in cooperative settings optimize for the team's reward, involving self-sacrifices and developing strategies.

This project focuses on improving the state-of-the-art deep reinforcement learning AI for StarCraft II and concentrates on a mini-challenge of StarCraft, known as StarCraft Multi-Agent Challenge (SMAC).
The goal was to improve 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 outperforms other methods in a few scenarios. 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. IQL 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 is QMIX (Rashid et al., 2018), the best-performing SMAC model. (Fig. 1).

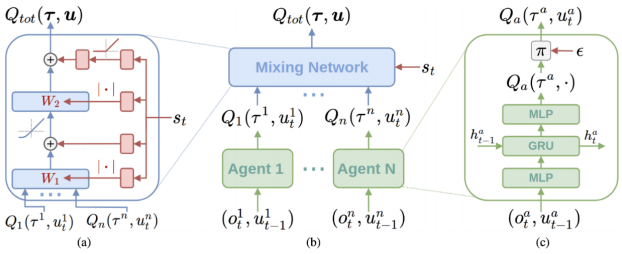

Fig. 1. QMIX Architecture

Each agent (shown in green) controls a unit is a Deep Recurrent Q-Network (Hausknecht & Stone, 2015), representing the agent's value function, Q_a. It takes o^{a_t}, the partial observation of at timestep t, and $u^{a_{t-1}}$, the last action taken. The recurrent unit will provide information about previous timesteps.
The model is built to estimate the joint-action value function, $Q_{tot(t,u)}$, through a mixture of the individual Q_a functions. The policy is the $argmax_u Q_{tot(t,u)}$, where u is the joint action of every agent.
A feed-forward network is used to handle the non-linear combination. which shown in blue (Fig. 1).

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

Fig. 2. QMix Objective Function

However, as agents are decentralized, we have no access to the joint value function while testing. But the paper extract the policy by keeping (Fig. 8):
If (Fig. 8) is correct, then evaluation of Q values of all joint actions u is not needed, which could grow exponentially for all the agents. Since we can take the individual *argmax* action for each agent and get the same result.

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 a different combination of these 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.

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

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, we found that a significant hyperparameter tuning is needed to choose the a optimal 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 using. ϵ-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,

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

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.
We only used this Noisy Net instead of linear layers of RNN agents, which improved agents' exploration and better performance for some challenging 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 hyper network weights with the quadratic function $0.5x^2$. This replacement's performance 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*). We theorize that since weight values tend to be lower than one, the quadratic function makes gradients smoother at these values than the absolute value function.

4 Results

4.1 Data Preparation

Environment was provided from the SMAC library. For every frame, 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

The process and run have been doing on a laptop with 15GB ram and 12 Core CPU, accelerated by NVIDIA 1060 GPU. Each training session takes about five to seven hours, and a single training can be run simultaneously, all of the training results captured for two million episodes.

4.3 Plots

4.3.1 2s3z is the first easy scenario that we experienced. There is no significant improvement to QMix with other methods except using a quadratic mixer and 5-Step Training, which is slightly better than QMix initially.

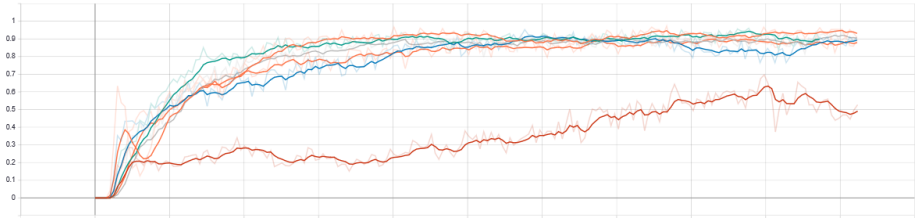

Fig. 3. — QMix, — IQL, — Noise, — Quadratic & 5-Steps, — 5-Steps

4.3.2 10m_vs_11m In this scenario, the 5-Step training outperforms the QMix and any other methods of learning.
Understanding better positioning for agents and dynamically changing it considering the situation is the critical point of winning this scenario.

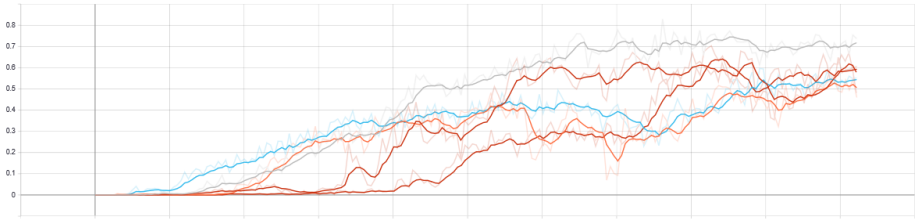

Fig. 4. — QMix, — Noise, — Quadratic, — 5-Steps, — 3-Steps

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

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

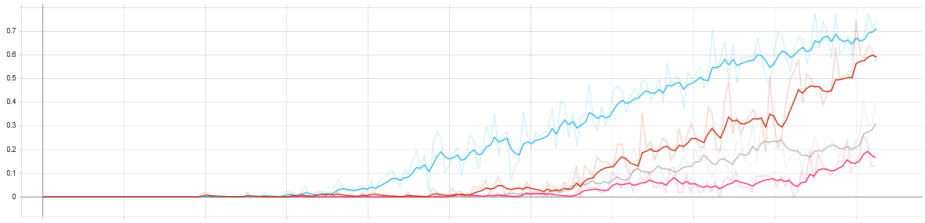

Fig. 5. — QMix, — Noise, — 3-Steps, — 3-Steps & Noise

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

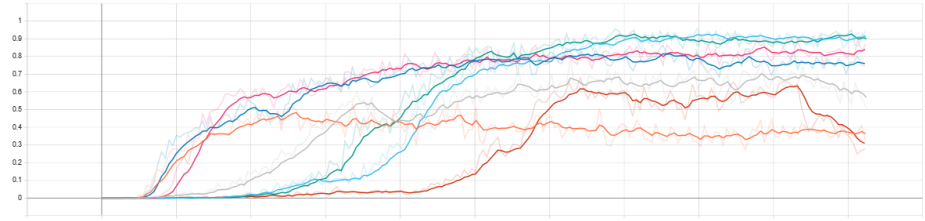

Fig. 6. — IQL, — QMix, — Noise, — 3-Steps, — 5-Steps, — 10-Steps, — Quadratic,

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

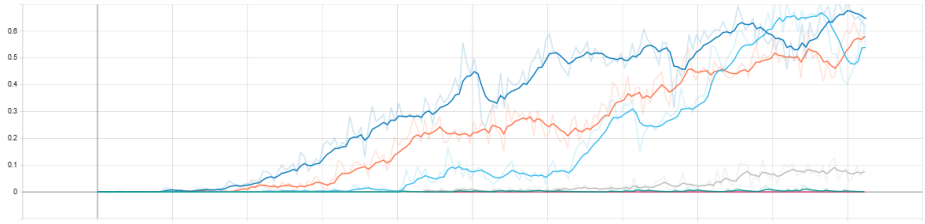

Fig. 7. — IQL, — QMix, — Noise, — 3-Steps, — 5-Steps, — Quadratic

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

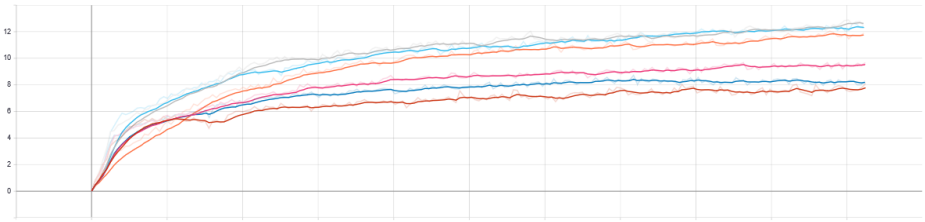

Fig. 8. Corridor Number of enemies Killed (— IQL, — QMix, — Noise, — 3-Steps, — 5-Steps, — Quadratic)

5 Conclusion

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

Our 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 on this buggy implementation was still able to learn, albeit worse than the baseline.

Another point was that excellent performance in one scenario does not necessarily translate to another. To gauge early on whether a model would perform, we tested the model on many known scenarios, and methods performed poorly on some shows excellent performance on others and vice-versa. However, the

combination of methods that performed better than baseline mostly leads to a significantly 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 factorization 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/