QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning

Tabish Rashid [*1] Mikayel Samvelyan [*2] Christian Schroeder de Witt [1]
Gregory Farquhar [1] Jakob Foerster [1] Shimon Whiteson [1]

Abstract

In many real-world settings, a team of agents must coordinate their behaviour while acting in a decentralised way. At the same time, it is often possible to train the agents in a centralised fashion in a simulated or laboratory setting, where global state information is available and communication constraints are lifted. Learning joint action-values conditioned on extra state information is an attractive way to exploit centralised learning, but the best strategy for then extracting decentralised policies is unclear. Our solution is QMIX, a novel value-based method that can train decentralised policies in a centralised end-to-end fashion. QMIX employs a network that estimates joint action-values as a complex non-linear combination of per-agent values that condition only on local observations. We structurally enforce that the joint-action value is monotonic in the per-agent values, which allows tractable maximisation of the joint action-value in off-policy learning, and guarantees consistency between the centralised and decentralised policies. We evaluate QMIX on a challenging set of StarCraft II micromanagement tasks, and show that QMIX significantly outperforms existing value-based multi-agent reinforcement learning methods.

1. Introduction

Reinforcement learning (RL) holds considerable promise to help address a variety of cooperative multi-agent problems, such as coordination of robot swarms (Hüttenrauch et al., 2017) and autonomous cars (Cao et al., 2012).

In many such settings, partial observability and/or communication constraints necessitate the learning of *decentralised policies*, which condition only on the local action-observation history of each agent. Decentralised policies also naturally attenuate the problem that joint action spaces grow exponentially with the number of agents, often rendering the application of traditional single-agent RL methods impractical.

*Equal contribution [1]University of Oxford, Oxford, United Kingdom [2]Russian-Armenian University, Yerevan, Armenia. Correspondence to: Tabish Rashid <tabish.rashid@cs.ox.ac.uk>, Mikayel Samvelyan <mikayel@samvelyan.com>.

(a) 5 Marines map (b) 2 Stalkers & 3 Zealots map

Figure 1. Decentralised unit micromanagement in StarCraft II, where each learning agent controls an individual unit. The goal is to coordinate behaviour across agents to defeat all enemy units.

Fortunately, decentralised policies can often be learnt in a centralised fashion in a simulated or laboratory setting. This often grants access to additional state information, otherwise hidden from agents, and removes inter-agent communication constraints. The paradigm of *centralised training with decentralised execution* (Oliehoek et al., 2008; Kraemer & Banerjee, 2016) has recently attracted attention in the RL community (Jorge et al., 2016; Foerster et al., 2018). However, many challenges surrounding how to best exploit centralised training remain open.

One of these challenges is how to represent and use the action-value function that most RL methods learn. On the one hand, properly capturing the effects of the agents' actions requires a centralised action-value function Q_{tot} that conditions on the global state and the joint action. On the other hand, such a function is difficult to learn when there are many agents and, even if it can be learned, offers no obvious way to extract decentralised policies that allow each agent to select only an individual action based on an individual observation.

The simplest option is to forgo a centralised action-value function and let each agent a learn an individual action-value function Q_a independently, as in *independent Q-learning* (IQL) (Tan, 1993). However, this approach cannot explicitly represent interactions between the agents and may not converge, as each agent's learning is confounded by the learning and exploration of others.

At the other extreme, we can learn a fully centralised state-action value function Q_{tot} and then use it to guide the optimisation of decentralised policies in an actor-critic framework, an approach taken by *counterfactual multi-agent* (COMA) policy gradients (Foerster et al., 2018), as well as work by Gupta et al. (2017). However, this requires on-policy learning, which can be sample-inefficient, and training the fully centralised critic becomes impractical when there are more than a handful of agents.

In between these two extremes, we can learn a centralised but factored Q_{tot}, an approach taken by *value decomposition networks* (VDN) (Sunehag et al., 2017). By representing Q_{tot} as a sum of individual value functions Q_a that condition only on individual observations and actions, a decentralised policy arises simply from each agent performing greedy action selection with respect to its Q_a. However, VDN severely limits the complexity of centralised action-value functions that can be represented and ignores any extra state information available during training.

In this paper, we propose a new approach called QMIX which, like VDN, lies between the extremes of IQL and COMA, but can represent a much richer class of action-value functions. Key to our method is the insight that the full factorisation of VDN is not necessary to extract decentralised policies. Instead, we only need to ensure that a global $\arg\max$ performed on Q_{tot} yields the same result as a set of individual $\arg\max$ operations performed on each Q_a. To this end, it suffices to enforce a monotonicity constraint on the relationship between Q_{tot} and each Q_a:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0. \tag{1}$$

QMIX consists of *agent networks* representing each Q_a, and a *mixing network* that combines them into Q_{tot}, not as a simple sum as in VDN, but in a complex non-linear way that ensures consistency between the centralised and the decentralised policies. At the same time, it enforces the constraint of (1) by restricting the mixing network to have positive weights. As a result, QMIX can represent complex centralised action-value functions with a factored representation that scales well in the number of agents and allows decentralised policies to be easily extracted via linear-time individual argmax operations.

We evaluate QMIX on a range of unit micromanagement tasks built in StarCraft II[1]. (Vinyals et al., 2017). Our experiments show that QMIX outperforms IQL and VDN, both in terms of absolute performance and learning speed. In particular, our method shows considerable performance gains on a task with heterogeneous agents. Moreover, our ablations show both the necessity of conditioning on the state information and the non-linear mixing of agent Q-values in order to achieve consistent performance across tasks.

2. Related Work

Recent work in multi-agent RL has started moving from tabular methods (Yang & Gu, 2004; Busoniu et al., 2008) to deep learning methods that can tackle high-dimensional state and action spaces. In this paper, we focus on cooperative settings.

On the one hand, a natural approach to finding policies for a multi-agent system is to directly learn decentralised value functions or policies. *Independent Q-learning* (Tan, 1993) trains independent action-value functions for each agent using Q-learning (Watkins, 1989). *Distributed DQN* (Nair et al., 2015) extends this approach to deep neural networks using the DQN algorithm (Mnih et al., 2015). While trivially achieving decentralisation, these approaches are prone to instability arising from the non-stationarity of the environment induced by simultaneously learning and exploring agents. Omidshafiei et al. (2017) and Foerster et al. (2017) address learning stabilisation to some extent, but still learn decentralised value functions and do not allow for the inclusion of extra state information during training.

On the other hand, centralised learning of joint actions can naturally handle coordination problems and avoids non-stationarity, but is hard to scale to many agents, as the joint action space grows exponentially in the number of agents. Classical approaches to scalable centralised learning include *coordination graphs* (Guestrin et al., 2002), which exploit conditional independencies between agents by decomposing a global reward function into a sum of agent-local terms. *Sparse cooperative Q-learning* (Kok & Vlassis, 2006) is a tabular Q-learning algorithm that learns to coordinate the actions of a group of cooperative agents only in the states in which such coordination is necessary, encoding those dependencies in a coordination graph. All of these methods require message passing during execution and so cannot learn fully decentralised policies.

More recent approaches for centralised learning require even more communication during execution: CommNet (Sukhbaatar et al., 2016) uses a centralised network architecture to exchange information between agents. BicNet (Peng et al., 2017) uses bidirectional RNNs to exchange information between agents in an actor-critic setting. This approach additionally requires estimating individual agent

[1]StarCraft and StarCraft II are trademarks of Blizzard Entertainment™.

rewards.

Some work has developed hybrid approaches that exploit the setting of centralised learning with fully decentralised execution. COMA (Foerster et al., 2018) uses a centralised critic to train decentralised actors, estimating a counterfactual advantage function for each agent in order to address multi-agent credit assignment. Similarly, Leibo et al. (2017) present a centralised actor-critic algorithm with per-agent critics, which scales better with the number of agents but mitigates the advantages of centralisation. Lowe et al. (2017) learn a centralised critic for each agent and apply this to competitive games with continuous action spaces. These approaches use on-policy policy gradient learning, which can have poor sample efficiency and is prone to getting stuck in sub-optimal local minima.

Sunehag et al. (2017) propose *value decomposition networks* (VDN), which allow for centralised value-function learning with decentralised execution. Their algorithm decomposes a central state-action value function into a sum of individual agent terms. This corresponds to the use of a degenerate fully-disconnected coordination graph. VDN does not make use of additional state information during training and can represent only a limited class of centralised action-value functions.

A number of papers have established unit micromanagement in StarCraft as a benchmark for deep multi-agent RL. Usunier et al. (2016) present an algorithm using a centralised *greedy MDP* and first-order optimisation. Peng et al. (2017) also evaluate their methods on StarCraft. However, neither requires decentralised execution. Similar to our setup is the work of Foerster et al. (2017), who evaluate replay stabilisation methods for IQL on combat scenarios with up to five agents. Foerster et al. (2018) also uses this setting. In this paper, we construct unit micromanagement tasks in the *StarCraft II Learning Environment* (SC2LE) (Vinyals et al., 2017) as opposed to StarCraft. This is because it is actively supported by the developers of the game, and we found that SC2LE offers a more stable testing environment.

QMIX relies on a neural network to transform the centralised state into the weights of another neural network, in a manner reminiscent of *hypernetworks* (Ha et al., 2016). This second neural network is constrained to be monotonic with respect to its inputs by keeping its weights positive. Dugas et al. (2009) investigate such functional restrictions for neural networks.

3. Background

A *fully cooperative multi-agent task* can be described as a Dec-POMDP (Oliehoek & Amato, 2016) consisting of a tuple $G = \langle S, U, P, r, Z, O, n, \gamma\rangle$. $s \in S$ describes the true state of the environment. At each time step, each agent $a \in A \equiv \{1, ..., n\}$ chooses an action $u^a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. This causes a transition on the environment according to the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$. All agents share the same reward function $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$ and $\gamma \in [0, 1)$, where the γ is a discount factor.

We consider a *partially observable* scenario in which each agent draws individual observations $z \in Z$ according to observation function $O(s, a) : S \times A \to Z$. Each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a(u^a|\tau^a) : T \times U \to [0, 1]$.

The optimisation objective is to maximise the *discounted return*: $R_t = \sum_{l=0}^{\infty} \gamma^i r_{t+l}$. The joint policy π produces a joint *action-value function*:

$$Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}} [R_t|s_t, \mathbf{u}_t]$$

Although training is centralised, execution is decentralised, i.e., the learning algorithm has access to all local action-observation histories $\boldsymbol{\tau}$ and global state s, but each agent's learnt policy can condition only on its own action-observation history τ^a.

3.1. Deep Q-Learning

Deep Q-learning represents the action-value function with a deep neural network parameterised by θ. *Deep Q-networks* (DQNs) (Mnih et al., 2015) use a *replay memory* to store the transition tuple $\langle s, u, r, s'\rangle$, where the state s' is observed after taking the action u in state s and receiving reward r. θ is learnt by sampling batches of b transitions from the replay memory and minimising the squared *TD error*:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{\text{DQN}} - Q(s, u; \theta) \right)^2 \right], \tag{2}$$

where $y^{\text{DQN}} = r + \gamma \max_{u'} Q(s', u'; \theta^-)$. θ^- are the parameters of a *target network* that are periodically copied from θ and kept constant for a number of iterations.

3.2. Deep Recurrent Q-Learning

In partially observable settings, agents can benefit from conditioning on their entire action-observation history. Hausknecht & Stone (2015) propose *Deep Recurrent Q-networks* (DRQN) that make use of recurrent neural networks. Typically, gated architectures such as LSTM (Hochreiter & Schmidhuber, 1997) or GRU (Chung et al., 2014) are used to facilitate learning over longer timescales.

3.3. Independent Q-Learning

Perhaps the most commonly applied method in multi-agent learning is *independent Q-learning* (IQL) (Tan, 1993), which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. This approach does not address the non-stationarity introduced due to the changing policies of the learning agents, and thus, unlike Q-learning, has no convergence guarantees even in the limit of infinite exploration. In practice, nevertheless, IQL commonly serves as a surprisingly strong benchmark even in mixed and competitive games (Tampuu et al., 2017; Leibo et al., 2017).

3.4. Value Decomposition Networks

By contrast, *value decomposition networks* (VDNs) (Sunehag et al., 2017) aim to learn a joint action-value function $Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$, where $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^n$ is a joint action-observation history and $\mathbf{u}$ is a joint action. It represents Q_{tot} as a sum of individual value functions $Q_a(\tau^a, u^a; \theta^a)$, one for each agent a, that condition only on individual action-observation histories:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i(\tau^i, u^i; \theta^i). \tag{3}$$

Strictly speaking, each Q_a is a *utility function* (Guestrin et al., 2002) and not a value function since by itself it does not estimate an expected return but admits arbitrary scaling. However, for terminological simplicity we refer to both Q_{tot} and Q_a as value functions.

The DQN update rule for VDN is equivalent to (2), where Q is replaced by Q_{tot}. An advantage of this representation is that a decentralised policy arises simply from each agent performing greedy action selection with respect to its Q_a. However, VDN severely limits the complexity of centralised action-value functions Q_{tot} that can be represented and ignores any extra state information available during training.

4. QMIX

In this section, we propose a new approach called QMIX which, like VDN, lies between the extremes of IQL and centralised Q-learning, but can represent a much richer class of action-value functions.

Key to our method is the insight that the full factorisation of VDN is not necessary in order to be able to extract decentralised policies that are fully consistent with their centralised counterpart. Instead, for consistency we only need to ensure that a global $\arg\max$ performed on Q_{tot} yields the same result as a set of individual $\arg\max$ operations performed on each Q_a:

每一个maximize自己的不一定能maximize全局的

$$\arg\max_{\mathbf{u}} Q_{tot} = \begin{pmatrix} \arg\max_{u^1} Q_1 \\ \vdots \\ \arg\max_{u^n} Q_n \end{pmatrix}. \tag{4}$$

This allows each agent a to participate in a decentralised execution solely by choosing greedy actions with respect to its Q_a. As a side effect, if (4) is satisfied, then taking the $\arg\max$ of Q_{tot}, required by off-policy learning updates, is trivially tractable.

VDN's representation is sufficient to satisfy (4). However, QMIX is based on the observation that this representation can be generalised to the larger family of monotonic functions, which are also sufficient but not necessary to satisfy (4). Monotonicity can be enforced through a constraint on the relationship between Q_{tot} and each Q_a:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \; \forall a \in A. \tag{5}$$

To enforce (5), QMIX represents Q_{tot} using an architecture consisting of *agent networks*, a *mixing network*, and a set of *hypernetworks* (Ha et al., 2016). Figure 2 illustrates the overall setup. For each agent a, there is one agent network that represents an individual value function $Q_a(\tau^a, u^a)$. We represent agent networks as DRQNs that receive the current individual observation o_t^a and the last action u_{t-1}^a as input at each time step, as shown in Figure 2b. Additionally, we employ weight sharing across agent networks. In order to allow different agents to execute different behaviours, the agent IDs are included as part of the observations.

The mixing network is a feedforward neural network that takes the agent network outputs as input and mixes them monotonically, producing the values of Q_{tot}. To enforce the monotonicity constraint of (5), its weights are restricted to be non-negative. This allows the mixing network to approximate any monotonic function arbitrarily closely (Dugas et al., 2009).

Each hypernetwork takes extra state information in the form of the global state s as input and generates the weights of one layer of the mixing network. Each hypernetwork is a single linear layer, followed by an activation function that takes the absolute value of the hypernetwork outputs, to ensure that the mixing network weights are non-negative. Figure 2c illustrates the mixing network and the hypernetworks.

Note that the global state is used by the hypernetworks rather than being passed directly into the mixing network. This is because Q_{tot} is allowed to depend on the extra state information in non-monotonic ways, so it would be overly constraining to pass some function of s through the monotonic network alongside the per-agent values. Instead, the use of hypernetworks makes it possible to condition the weights of the monotonic network on s in an arbitrary way, thus integrating the full state s into the joint action-value estimates as flexible as possible.

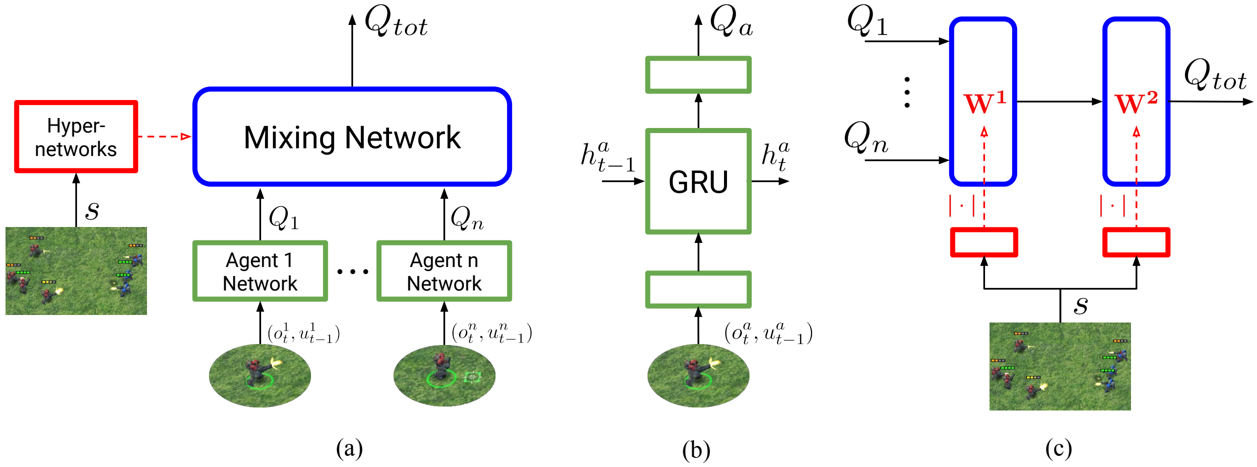


Figure 2. The QMIX architecture (a), the agent networks (b), and the mixing and hypernetworks (c).

QMIX is trained end-to-end to minimise the following loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{\text{tot}} - Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \theta) \right)^2 \right], \tag{6}$$

where b is the batch size of transitions sampled from the replay buffer, $y^{\text{tot}} = r + \gamma \max_{\mathbf{u}'} Q_{tot}(\boldsymbol{\tau}', \mathbf{u}', s'; \theta^-)$ and θ^- are the parameters of a target network as in DQN. (6) is analogous to the standard DQN loss of (2). Since (4) holds, we can perform the maximisation of Q_{tot} in time linear in the number of agents (as opposed to scaling exponentially in the worst case).

5. Experimental Setup

In this section, we describe the decentralised StarCraft II micromanagement problem to which we apply QMIX. We also provide the details of the state features, model architectures, training procedures, and ablations.

5.1. Decentralised StarCraft II Micromanagement

Real-time strategy (RTS) games have recently emerged as challenging benchmarks for the RL community. StarCraft, in particular, offers a great opportunity to tackle competitive and cooperative multi-agent problems. Units in StarCraft have a rich set of complex micro-actions which allow the learning of complex interactions between collaborating agents. Previous work (Usunier et al., 2016; Foerster et al., 2018; Peng et al., 2017) applied RL to the original version of StarCraft: BW, which made use of the standard API or related wrappers (Synnaeve et al., 2016). We perform our experiments on the StarCraft II Learning Environment (SC2LE) (Vinyals et al., 2017), which is based on the second version of the game. Because it is supported by the developers of the game, SC2LE mitigates many of the practical difficulties in using StarCraft as an RL platform, such as the dependence on complicated APIs and external emulation software.

In this work, we focus on the *decentralised micromanagement* problem in StarCraft II, in which each of the learning agents controls an individual army unit. We consider combat scenarios where two groups of identical units are placed symmetrically on the map. The units of the first, allied, group are controlled by the decentralised controllers. The enemy units are controlled by a built-in StarCraft II AI, which makes use of handcrafted heuristics. The initial placement of units within the groups varies across episodes. The difficulty of the computer AI controlling the enemy units is set to `medium`. At the beginning of each episode, the enemy units are ordered to attack the allies. We compare our results on a set of maps where each unit group consists of 3 Marines (3m), 5 Marines (5m), 8 Marines (8m), or 2 Stalkers and 3 Zealots (2s_3z).

Similar to the work of Foerster et al. (2018), the action space of agents consists of the following set of discrete actions: `move[direction]`, `attack[enemy_id]`, `stop`, and `noop`. Agents can only move in four directions: north, south, east, or west. A unit is allowed to perform the `attack[enemy_id]` action only if the enemy is within its *shooting range*. This facilitates the decentralisation of the problem and prohibits the usage of the *attack-move* macro-actions that are integrated into the game. Furthermore, we disable the following unit behaviour when idle: responding to enemy fire and attacking enemies if they are in range. By doing so, we force the agents to explore in order to find the optimal combat strategy themselves, rather than relying on built-in StarCraft II utilities.

Partial observability is achieved by the introduction of unit *sight range*, which restricts the agents to receive information about allied or enemy units who are out of range. Moreover, units can only observe others if they are alive. Therefore, they cannot distinguish between units that are dead or out of range. Since the sight range is bigger than the shooting range, agents need to make use of moving commands before starting to fire.

5.2. Environment Features

The local observations of individual agents are drawn within their field of view, which encompasses the circular area of the map surrounding units and has a radius equal to the sight range. Each agent receives as input a vector consisting of the following features for all units in its field of view (both allied and enemy): `distance`, `relative x`, `relative y` and `unit_type`.[2]

The global state, which is hidden from agents, is a vector comprised of features of units from the entire map. It does not contain the absolute distances between agents and stores only the coordinates of units relative to the centre of the map. In addition, the global state includes the `health`, `shield` and `cooldown` of all units.[3] Marines, Stalkers, and Zealots have 45, 80, and 100 hit points, respectively. In addition, Stalkers and Zealots have 80 and 50 shield points, respectively. All features, whether in local observations or global state, are normalised by their maximum values.

Similar to the work of Usunier et al. (2016); Foerster et al. (2018), the reward signal is shared between all agents. At each time step, the agents receive a joint reward equal to the total damage dealt on the enemy units. In addition, agents receive a bonus of 10 points after killing each opponent, and 200 points after killing all opponents.

5.3. Architecture and Training

The architecture of all agent networks is a DRQN with a recurrent layer comprised of a GRU with a 64-dimensional hidden state, with a fully-connected layer before and after. Exploration is performed during training using independent ϵ-greedy action selection, where each agent a performs ϵ-greedy action selection over its own Q_a. Throughout the training, we anneal ϵ linearly from 1.0 to 0.05 over $50k$ time steps and keep it constant for the rest of the learning.

We set $\gamma = 0.99$ for all experiments. The replay buffer contains the most recent 5000 episodes. We sample batches of 32 episodes uniformly from the replay buffer and train on fully unrolled episodes. The target networks are updated after every 200 training episodes.

[2]`unit_type` is only included in the 2s_3z map.

[3]A unit's `cooldown` is the time it must wait before firing again. Shields act as additional forms of hit points and are lost first. In contrast to health, shields regenerate over time after absorbing damage.

To speed up the learning, we share the parameters of the agent networks across all agents. Because of this, a one-hot encoding of the `agent_id` is concatenated onto each agent's observations. All neural networks are trained using RMSprop[4] with learning rate 5×10^{-4}.

During training and testing, we restrict each episode to have a length of 60 time steps for 3m and 5m maps and 120 time steps for 8m and 2s_3z maps. We only treat a transition as terminal if either of the armies has been defeated.

The mixing network consists of a single hidden layer of 32 units, utilising an ELU non-linearity. The weights of the network are produced by two hypernetworks. Each hypernetwork receives the global state s and performs an affine transformation, followed by an absolute function. The output is then reshaped appropriately. We performed minimal hyperparameter tuning on the 5m map and used those hyperparameters across all methods.

5.4. Ablations

We perform ablation experiments in order to investigate the influence of the inclusion of extra state information and the necessity of nonlinear transformations in the mixing network.

First, we analyse the significance of extra state information by comparing against QMIX with no hypernetworks. Thus, the weights of the mixing network are learned in the standard way, without conditioning on the state. We refer to this method as QMIX-NS.

Second, we investigate the necessity of nonlinear mixing by removing the hidden layer of the mixing network. This method can be thought of an extension of VDN that uses the state s to perform a weighted sum over Q_a values. We call this method QMIX-Linear.

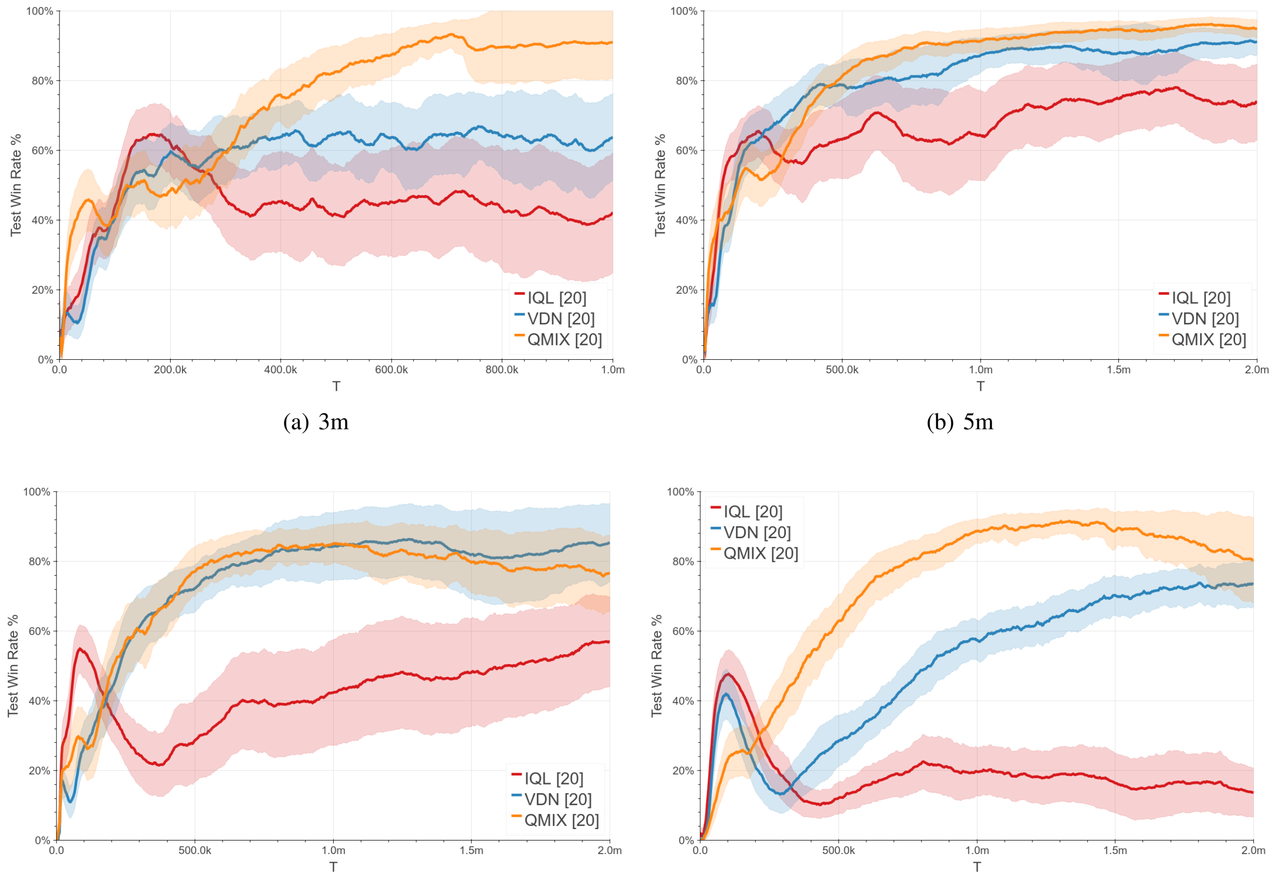
6. Results

In order to evaluate each method's performance, we adopt the following evaluation procedure: for each run of a method, we cease training every 50 episodes and run 20 independent episodes with each agent performing greedy decentralised action selection. The percentage of these episodes in which the method defeats all enemy units within the time limit is referred to as the test win rate.

Each graph in Figure 3 plots the mean test win rate across all runs of a particular method, together with the 95% confidence interval.

In all scenarios, IQL fails to learn a policy that consistently defeats the enemy. In addition, the training is highly unstable due to the non-stationarity of the environment which

[4]We set $\alpha = 0.99$ and do not use weight decay or momentum.



(a) 3m (b) 5m

(c) 8m (d) 2s_3z

Figure 3. Win rates for IQL, VDN, and QMIX on four different combat maps.

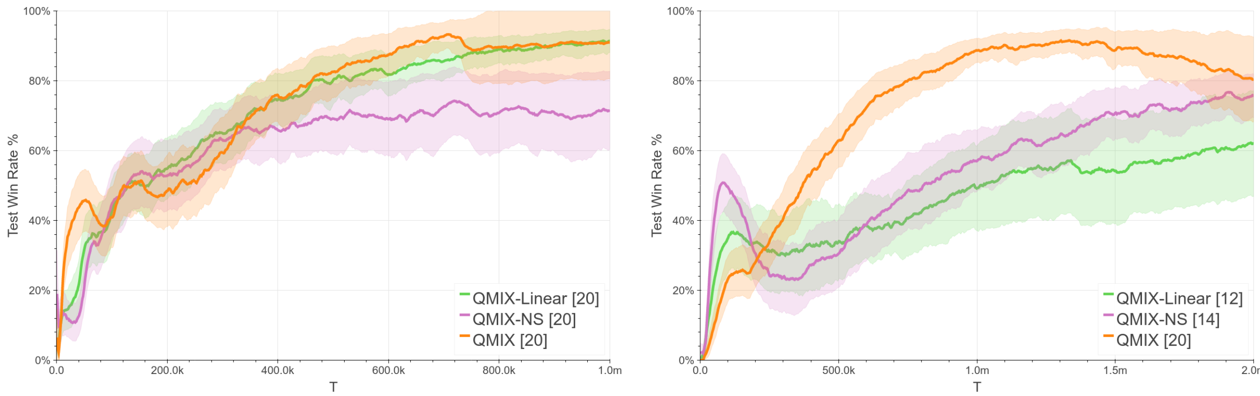
arises due to the other agents changing their behaviour during training.

The benefits of learning the joint action-value function can be demonstrated by VDN's superior performance over IQL in all scenarios. VDN is able to more consistently learn basic coordinated behaviour, in the form of *focus firing* which allows it to win the majority of its encounters on the 5m and 8m maps. On the 8m map, this simple strategy is sufficient for good performance and explains the performance parity with QMIX. However, on the 3m task, which requires more fine-grained control, it is unable to learn to consistently defeat the enemy.

QMIX is noticeably the strongest performer on 3 out of the 4 maps. In particular, Figure 3d shows that QMIX not only achieves higher performance than VDN but also does so significantly faster. In this scenario with heterogeneous agent types, the superior representational capacity of QMIX combined with the state information presents a clear benefit over a more restricted linear decomposition.

A qualitative analysis of individual policies learnt by QMIX reveals that it is able to learn particularly sophisticated strategies which exhibit extended coordinated behaviour. Since Zealots are especially effective against Stalkers, QMIX adopts to dispatch enemy Zealots first and to position allied Stalkers far from the enemy. If the Zealots notice that allied Stalkers are under attack by an enemy Zealot, they will choose to attack it over an enemy Stalker. Additionally, if a Stalker is attacking an enemy Zealot it will choose to move and fire as opposed to staying in one position. This strategy, known as *kiting*, allows it to triumph over a Zealot one-on-one. The learning of these complex strategies allows QMIX to consistently defeat the enemy units.

Our additional ablation experiments reveal that QMIX outperforms, or is competitive with, all of its ablations discussed in Section 5.4. Figure 4a shows that non-linear value function factorisation is not always required on a map with homogeneous agent types. However, the additional complexity introduced through the extra hidden layer does not slow down the learning. In contrast, Figure 4b shows that



(a) 3m

(b) 2s_3z

Figure 4. Win rates for QMIX and ablations on 3m and 2s_3z maps.

on a map with heterogeneous agent types a combination of both central state information and non-linear value function factorisation is required to achieve good performance.

QMIX-NS performs on par with VDN in both scenarios, which suggests that a non-linear decomposition is not always beneficial when not conditioning on the central state. This is expected since the representational capacity of VDN and QMIX-NS is the same.

7. Conclusion

This paper presented QMIX, a deep multi-agent RL method that allows end-to-end learning of decentralised policies in a centralised setting which makes efficient use of extra state information. QMIX allows the learning of a rich joint action-value function, which admits tractable decompositions into per-agent action-value functions. This is achieved by imposing a monotonicity constraint on the mixing network.

Our results in decentralised unit micromanagement tasks in StarCraft II show that QMIX improves the final performance over other value-based multi-agent methods that employ less sophisticated joint state-value function factorisation, as well as independent Q-learning.

In the near future, we wish to conduct additional experiments in order to compare the methods across tasks with a greater diversity of units. In the longer term, we aim to make use of maximising bootstrapping qualities of QMIX for off-policy learning in the presence of a centralised critic. We also wish to complement QMIX with more coordinated exploration schemes for settings with many learning agents. Ultimately, we strive to develop more sample-efficient methods for real-world applications, such as self-driving cars and autonomous drones.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement number 637713). It was also supported by the Oxford-Google DeepMind Graduate Scholarship, the UK EPSRC CDT in Autonomous Intelligent Machines and Systems, Chevening Scholarship, Luys Scholarship and an EPSRC grant (EP/N509711/1). This work is linked to and partly funded by the project Free the Drones (FreeD) under the Innovation Fund Denmark and Microsoft. The experiments were made possible by a generous equipment grant from NVIDIA.

We would like to thank Frans Oliehoek and Wendelin Boehmer for helpful comments and discussion. We also thank Oriol Vinyals, Kevin Calderone, and the rest of the SC2LE team at DeepMind and Blizzard for their work on the interface.

References

Busoniu, Lucian, Babuska, Robert, and De Schutter, Bart. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews*, 38(2):156, 2008.

Cao, Yongcan, Yu, Wenwu, Ren, Wei, and Chen, Guanrong. An Overview of Recent Progress in the Study of Distributed Multi-agent Coordination. *arXiv preprint arXiv:1207.3231*, 2012.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Dugas, Charles, Bengio, Yoshua, Blisle, Franois, Nadeau, Claude, and Garcia, Ren. Incorporating functional knowledge in neural networks. *Journal of Machine Learning Research*, 10, 2009.

Foerster, Jakob, Nardelli, Nantas, Farquhar, Gregory, Torr, Philip, Kohli, Pushmeet, Whiteson, Shimon, and others. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *Proceedings of The 34th International Conference on Machine Learning*, 2017.

Foerster, Jakob, Farquhar, Gregory, Afouras, Triantafyllos, Nardelli, Nantas, and Whiteson, Shimon. Counterfactual multi-agent policy gradients. *AAAI 2018: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. To appear.

Guestrin, Carlos, Koller, Daphne, and Parr, Ronald. Multiagent Planning with Factored MDPs. In *Advances in Neural Information Processing Systems 14*, pp. 1523–1530. MIT Press, 2002.

Gupta, Jayesh K., Egorov, Maxim, and Kochenderfer, Mykel. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In *Autonomous Agents and Multiagent Systems*, pp. 66–83. Springer, 2017.

Ha, David, Dai, Andrew, and Le, Quoc V. HyperNetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Hausknecht, Matthew and Stone, Peter. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hüttenrauch, Maximilian, Šošić, Adrian, and Neumann, Gerhard. Guided Deep Reinforcement Learning for Swarm Systems. *arXiv preprint arXiv:1709.06011*, 2017.

Jorge, Emilio, Kågebäck, Mikael, and Gustavsson, Emil. Learning to Play Guess Who? and Inventing a Grounded Language as a Consequence. *arXiv preprint arXiv:1611.03218*, 2016.

Kok, Jelle R. and Vlassis, Nikos. Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *Journal of Machine Learning Research*, 7:1789–1828, 2006.

Kraemer, Landon and Banerjee, Bikramjit. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Leibo, Joel Z, Zambaldi, Vinicius, Lanctot, Marc, Marecki, Janusz, and Graepel, Thore. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pp. 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

Lowe, Ryan, Wu, Yi, Tamar, Aviv, Harb, Jean, Abbeel, OpenAI Pieter, and Mordatch, Igor. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6382–6393, 2017.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, and others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, De Maria, Alessandro, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, Legg, Shane, Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David. Massively Parallel Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1507.04296*, 2015.

Oliehoek, Frans A. and Amato, Christopher. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer, 2016.

Oliehoek, Frans A., Spaan, Matthijs T. J., and Vlassis, Nikos. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32(1):289–353, 2008.

Omidshafiei, Shayegan, Pazis, Jason, Amato, Christopher, How, Jonathan P, and Vian, John. Deep Decentralized Multi-task Multi-Agent RL under Partial Observability. *arXiv preprint arXiv:1703.06182*, 2017.

Peng, Peng, Wen, Ying, Yang, Yaodong, Yuan, Quan, Tang, Zhenkun, Long, Haitao, and Wang, Jun. Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games. *arXiv preprint arXiv:1703.10069*, 2017.

Sukhbaatar, Sainbayar, Fergus, Rob, and others. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pp. 2244–2252, 2016.

Sunehag, Peter, Lever, Guy, Gruslys, Audrunas, Czarnecki, Wojciech Marian, Zambaldi, Vinicius, Jaderberg, Max, Lanctot, Marc, Sonnerat, Nicolas, Leibo, Joel Z., Tuyls, Karl, and Graepel, Thore. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv preprint arXiv:1706.05296*, 2017.

Synnaeve, Gabriel, Nardelli, Nantas, Auvolat, Alex, Chintala, Soumith, Lacroix, Timothe, Lin, Zeming, Richoux, Florian, and Usunier, Nicolas. TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. *arXiv preprint arXiv:1611.00625*, 2016.

Tampuu, Ardi, Matiisen, Tambet, Kodelja, Dorian, Kuzovkin, Ilya, Korjus, Kristjan, Aru, Juhan, Aru, Jaan, and Vicente, Raul. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 2017.

Tan, Ming. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337, 1993.

Usunier, Nicolas, Synnaeve, Gabriel, Lin, Zeming, and Chintala, Soumith. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Vinyals, Oriol, Ewalds, Timo, Bartunov, Sergey, Georgiev, Petko, Vezhnevets, Alexander Sasha, Yeo, Michelle, Makhzani, Alireza, Küttler, Heinrich, Agapiou, John, Schrittwieser, Julian, Quan, John, Gaffney, Stephen, Petersen, Stig, Simonyan, Karen, Schaul, Tom, van Hasselt, Hado, Silver, David, Lillicrap, Timothy, Calderone, Kevin, Keet, Paul, Brunasso, Anthony, Lawrence, David, Ekermo, Anders, Repp, Jacob, and Tsing, Rodney. StarCraft II: A New Challenge for Reinforcement Learning. *arXiv preprint arXiv:1708.04782*, 2017.

Watkins, Christopher. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

Yang, Erfu and Gu, Dongbing. Multiagent reinforcement learning for multi-robot systems: A survey. Technical report, 2004.