Spatial Action Decomposition Learning Applied to RTS Combat Games

Marius Stanescu and Michael Buro
Department of Computing Science
University of Alberta, Canada
{astanesc|mburo}ualberta.ca

Abstract

Learning good policies for multi-agent systems is a complex task. Existing methods are often limited to a small number of agents, as learning becomes intractable when the agent number increases considerably. In this paper we describe Spatial Action Decomposition Learning that tries to overcome inefficiencies of standard multi-agent Q-learning methods by exploiting existing spatial action correlations. We apply our method to real-time strategy (RTS) game combat scenarios and show that Spatial Action Decomposition Learning based systems can outperform handcrafted scripts and policies optimized by independent Q-learning.

Introduction

As multi-agent decision problems are ubiquitous, building better AI agents that work together has numerous real-world applications, such as city transport optimization, stock trading, advertisement bidding, multi-player online gaming, and coordinating robot swarms. Multi-agent reinforcement learning (MARL) (Panait and Luke 2005; Busoniu, Babuska, and De Schutter 2008) is a popular solution paradigm in this research area. Typically, a set of autonomous agents share a common environment and jointly optimize a single team's reward signal accumulated over time. While applying standard RL techniques like Q-learning to multi-agent settings seems straight-forward, difficulties arise from the combinatorial explosion of the joint action sets and non-trivial interactions with the environment and other agents – which can be cooperative or adversarial or mixed (e.g., opposing teams). In addition, partial observability and communication constraints require decentralised policies that only depend on local agent observations. This also helps dealing with the exponential growth of the joint action space when increasing the number of agents. Learning these decentralised policies in a centralised fashion has the added benefit of being able to use extra state information during learning that is hidden from agents at runtime (e.g., lifting the fog-of-war). This approach has become very popular lately (Jorge et al. 2016; Foerster et al. 2017a; Rashid et al. 2018).

A challenging aspect of decentralization is to find an effective representation of the centralized action-value function Q_{tot} that integrates the effects of all agents' actions. But such functions are difficult to learn in the presence of many agents and extracting the decentralised individual agent policy (one agent chooses one action based on individual/partial observation) is not straightforward.

In the next section we discuss in more detail related work that tackles multi-agent Q-learning in various ways – ranging from centralized learning, over simple Q-function decompositions, to learning expressive networks for mixing individual Q-functions. We then describe our novel learning approach which is based on the observation that agent actions are often spatially correlated, e.g., nearby agents frequently execute similar actions, such as moving in the same direction, or collaborating to achieve a local goal such as defending a choke point. We then present experimental results for our spatial action decomposition method — applying it to the popular multi-agent learning domain of 2-team combat in real-time strategy (RTS) games with up to 80 vs. 80 units. Finally, we conclude the paper with summarizing our findings and discussing future work.

Background and Related Work

Independent Q-Learning

Arguably the easiest and most commonly used learning method for multiple agents is Independent Q-Learning (IQL) (Tan 1993). It considers all other agents as part of the environment and decomposes the multi-agent learning task into simultaneous single-agent problems. IQL suffers from instability caused by non-stationarity introduced by learning and exploration of other agents (Laurent et al. 2011), and consequently loses the convergence guarantees of Q-learning. As a concrete example, (Claus and Boutilier 1998) show that independent Q-learners cannot distinguish team mates' exploration from stochasticity in the environment, and fail to solve even an apparently trivial, 2-agent, stateless, (3×3)-action problem.

While there are ways of improving IQL's stability (Foerster et al. 2017b), it usually requires individual reward functions as uniform team reward signals do not directly relate to individual agents' actions. Reward shaping is difficult and few methods guarantee to preserve optimality w.r.t. the initial objective (Devlin et al. 2014; Eck et al. 2016). A preferable, more general approach is to learn how to decompose the team reward. Still, in practice, IQL is an unexpectedly strong benchmark method, even in mixed cooperative-

competitive MARL problems (Leibo et al. 2017).

Centralised Learning

Alternatively, the joint action Q-function Q_{tot} can be learned directly. This avoids the non-stationarity problem and can lead to better coordination and results, at the cost of poor scaling performance. For example, experiments in (Usunier et al. 2016) are limited to 15 agents per side. Some methods are partially centralised, using one or more centralised critics to guide the optimisation of decentralised policies in an actor-critic paradigm (Foerster et al. 2017a; Gupta, Egorov, and Kochenderfer 2017; Leibo et al. 2017). To work well, these methods require additional information to be exchanged between agents, e.g., CommNet (Sukhbaatar, Szlam, and Fergus 2016) or BicNet (Peng et al. 2017). Furthermore, such on-policy methods are usually sample inefficient.

Using a team reward signal makes credit assignment challenging, even for simple problems. For example, in a 2-player soccer game with the number of scored goals being the team reward, the agent who is worse at scoring sometimes ends up failing to learn to shoot at all, as its exploration would impede its teammate (Hausknecht 2016).

Coordination graphs have been used to decompose the global reward into a sum of local agent rewards (Guestrin, Koller, and Parr 2002), but the method requires solving a linear program and message passing between agents at runtime. COMA (Foerster et al. 2017a) is an actor-critic method that uses a counterfactual baseline to marginalise out a single agent's action, while keeping the other agents' actions fixed. Another idea is to transform multiple agent interactions into interactions between two entities: a single agent and a distribution of the other agents (Yang et al. 2018).

Value Decomposition

A more elegant way of solving the credit assignment problem is to use a value decomposition network (VDN) to represent Q_{tot} as sum of individual Q-functions Q_i which depend only on agent-local observations (Sunehag et al. 2017) . The network learns how to assign the team reward signal to Q_is implicitly, without shaping or global state information.

One disadvantage is that the VDN representation of Q_{tot} is limited by the addition, because agents interactions are usually more complex. QMIX addresses this issue by replacing the sum operation with a mixing network that combines all individual Q_i into Q_{tot} in a complex, monotone, and non-linear fashion informed by the global state information during training (Rashid et al. 2018). Access to the global state is not required after training because due to monotonicity, the arg-max performed on Q_{tot} yields the same result as individual arg-max operations on each Q_i.

While more natural, these methods still suffer when handling larger numbers of agents as Q-learning becomes infeasible due to noise accumulation caused by many exploratory actions (Colby et al. 2015).

Abstractions and Hierarchies

To address the scaling issue and improve sample efficiency the value decomposition networks can be combined with hierarchical decomposition. This can be done by considering temporal abstraction layers (Levy, Platt, and Saenko 2017). Multiple policies can represent a diverse set of behaviors, and learning is sped up because the environment can be explored at higher levels more effectively. Alternatively, in suitable domains spatial abstractions can be used to speed up learning. In feudal reinforcement learning (Dayan and Hinton 1993) the state space is hierarchically subdivided into increasingly smaller regions at each level of abstraction – similar to quad-trees in the grid world pathfinding example they discuss. Each level has an associated Q-function that is trained by giving lower levels credit for achieving higher level goals. The experimental results indicate that feudal reinforcement learning outperforms classic non-hierarchical Q-learning in their domain. The method we present in the next section is also based on spatial decomposition. However, its objective is to generate concurrent actions for multiple agents and to effectively learn agent policies despite huge combinatorial action spaces.

Spatial Action Decomposition Learning

In this work we focus on the MARL challenge of increasing the number of agents and the resulting combinatorial explosion of the joint action sets and interactions with other agents present in the environment, friend or foe. As previously mentioned, IQL is a quick and simple solution, but does not guarantee convergence and is poorly suited for modelling all interactions between agents. On the other hand, fully combinatorial approaches are better at handling coordination but don't scale well with the number of agents.

One approach that showed good results is to estimate the join action-value of a team of agents as a linear (Sunehag et al. 2017) or non-linear (Rashid et al. 2018) combination of individual, per-agent action-values. These per-agent values condition on agent-local observations, which is required in a partial observable environment. (Rashid et al. 2018) use hypernetworks to integrate the full, global state information into the joint action-value during training.

Spatial Action Decomposition Using Sectors

We build on the value decomposition idea, but as we focus on the number of agents as the main challenge, we choose to work in a fully observable environment. As such we don't have to condition on agent-local observations. We decompose Q_{tot} into individual values similarly to VDN and QMIX, but use the full, global state both during training and evaluation. To handle the large number of agents and the difficulty introduced by their exploratory actions, we use a grid based spatial decomposition. We add a *sector* abstraction, making the assumption that the joint action value can be decomposed into value functions across sectors – disjoint areas of the map – instead of individual agents. We use a simple sum and leave more complex decompositions such as QMIX to future work, as experiments in (Rashid et al. 2018) show that non-linear value function factorisation is often not required for scenarios with homogeneous agent types.

After the sector actions are chosen by the network, each sector is responsible for emitting low-level actions for each agent present within, through a separate mechanism. The sector action can be as simple as selecting which action script to use to generate actions for the agents within that

particular sector. Besides simple but resource inexpensive methods, other algorithms that don't scale as well could be used due to the smaller problem size, for example search algorithms if there is a forward model.

Each sector's Q-function Q_{S_i} will be learned implicitly by the network and will not benefit from any reward specifically given to sector S_i or to any individual agent. Strictly speaking Q_{S_i} is more an utility function than a value function because it does not estimate an expected return by itself. For simplicity, however, we will continue to call both Q_{tot} and Q_{S_i} value functions.

Each sector chooses its action greedily with respect to its own Q_{S_i}, and the joint action Q_{tot} is chosen by maximizing $\sum_i Q_{S_i}$. The maximization of Q_{tot} can now be performed in time linear in the number of sectors, which will usually be much smaller than the number of agents. For this method to work well, the sectors dimension should be large enough such that at least a few agents are present in most sectors.

Network Architecture

To reduce the number of learnable parameters it is common to use a neural network for each agent and to share its weights between the agents. This approach can be taken with sectors as well, but since we have access to the global state it is more natural to use convolutions to keep the number of parameters low.

A deep ConvNet computes higher and higher level features layer by layer, with the same or possibly different spatial resolutions. It's a very good tool for a sector-based approach: it starts with higher resolution maps with low-level features and produces lower and lower resolution maps but with increasingly relevant features until reaching the target sector granularity.

An example network used for a grid-like 2-player combat scenario is shown in Figure 1. There are 5 input feature planes for a 64×64 grid map. Three of them are Boolean values indicating unaccessible terrain (e.g. walls), own and enemy unit presence on each square. Two planes represent the hit-points of the two armies, each cell containing at most one agent. More details about the environment used for experiments can be found in the next section.

Four convolutional layers with 4×4 stride-2 filters, each followed by a ReLU nonlinearity and using batch normalisation, are used to extract features and gradually halve the map resolution, obtaining activations C_{1-3} and S_1. For simplicity, at a given resolution level we considered only square sector configurations of the type $W \times W$, resulting in W^2 total sectors. Using a granularity of 2×2 sectors – each 32×32 cells – was not enough to execute complex maneuvers. Consequently, we start with the 4×4 sector representation of S_1, each sector being responsible for agents within 16×16 cells on the original map.

In certain situations it might be beneficial to have even more precise control. For this purpose we can apply the standard process of deconvolution, also known as transposed or fractionally-strided convolution (Dumoulin and Visin 2016), to generate features for more, smaller sectors. For example, in Figure 1 S_1 is upscaled by a factor of 2 to the 8×8 representation of S_2 using 2×2 stride-2 filters.

Either S_1 or S_2 can be used to obtain Q_{S_i} values, depending on the desired sector granularity. The Q functions are represented using value and advantage functions $Q(s, a) = V(s) + A(s, a)$ as recommended by the dueling architecture (Wang et al. 2016b). More robust estimates of the state values can be obtained by decoupling them from the necessity of being attached to specific actions. For S_1 the value function denotes how good the current state is for each of the $4 \times 4 = 16$ sectors, and a corresponding $4 \times 4 \times 1$ block V_1 can be obtained from S_1 using a convolutional layer with 1 filter of 3×3, stride-1. The advantage function tells how much better taking a certain action for any given sector would be compared to the other actions. Using a number of filters equal to the number of possible actions A for a sector results in the corresponding $4 \times 4 \times A$ block A_1. The argmax over the last dimension gives the action with the highest advantage value for each of the 16 sectors.

The same output head filters can be applied to any resolution S_i layer to obtain Q-values for a desired sector granularity. Actions can be emitted based on the values from any of these layers, either by choosing a specific resolution beforehand, by taking the max across all resolutions, or even by adding a separate network component responsible for learning which sector granularity to pick given the current state.

Increasing the number of sectors provides more accurate control over the agents, but also increases the difficulty of the credit assignment and slows the learning process. Using fewer sectors and the resulting ability to specify large scale actions should make the exploration for actions leading to large rewards more efficient, and unnecessary micromanagement could be avoided for map areas that are largely empty or that do not require complex behaviors.

In this work we focus on spatial decomposition, and opt for the simple fully convolutional design described above. Although more sophisticated network blocks could be designed, and even though temporal abstractions are compatible with our method, they are not the focus of this paper. Stacking multiple input frames along the depth dimension, using gated architectures such as LSTM and GRU, or learning agent policies that implement the provided sector actions are left for future work.

Experimental Setting

In this section we describe the combat scenario problem to which we apply Spatial Action Decomposition Learning. We provide details of the state features and training methodology and evaluate our method's performance in comparison to independent Q-learning.

Environment

Real-time strategy (RTS) games are now a well established benchmark for the RL community. They offer more difficult multi-agent challenges compared to previous environments such as Atari games. In particular, RTS game combat scenarios are a popular evaluation method for MARL algorithms, offering mixed cooperative and competitive multi-agent environments. Combat scenarios have been traditionally used to test algorithms' ability to scale, from simple search based methods (Churchill, Saffidine, and Buro 2012;

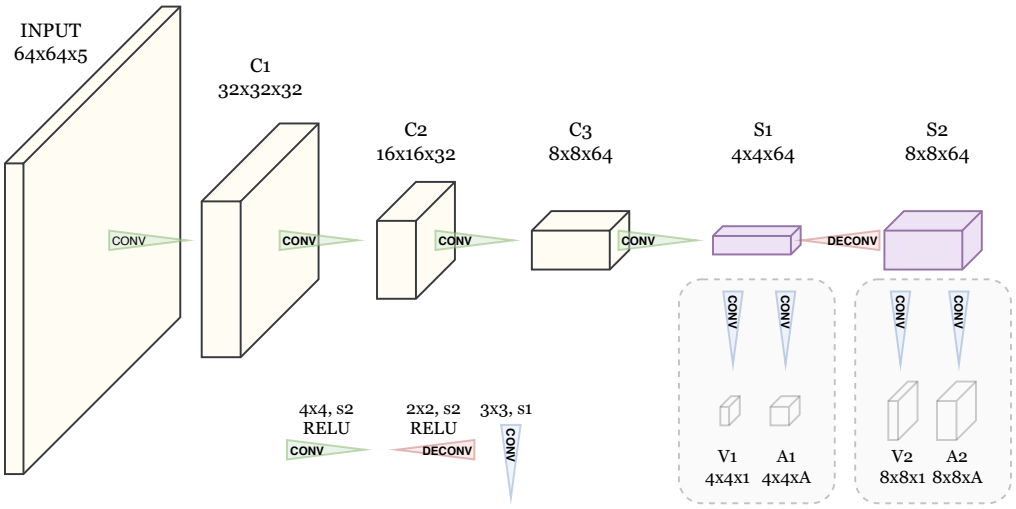

Figure 1: Layout of the network architecture. More deconvolutional layers can be added to increase the sector granularity.

Ontañón 2017) to script and portfolio blended variants (Churchill and Buro 2013; Wang et al. 2016a; Lelis 2017).

The StarCraft II Learning Environment (SC2LE) (Vinyals et al. 2017) is one such popular platform for RL experiments. However, for our experiments we chose the simpler but faster many-agent (MAgent) environment (Zheng et al. 2017). It supports up to a million agents on the same map while being much less computationally intensive compared to SC2LE. Moreover, it provides environment/agent configurations as well as a reward description language that enables flexible environment design.

In this work we focus on a centralised micro-management scenario in which two groups of identical units fight on a grid-world map. The units of the first army are controlled by the learning method. The opposing units are controlled by a scripted algorithm that moves towards and focuses fire on its closest enemies. The initial placement of the units for each army is randomized at the start of each game: around two thirds are split randomly among a random number of 2 to 5 orderly groups, while the remaining units are randomly scattered on the map. An example scenario is shown in Figure 2. This setup is designed to mimic combat scenarios commonly occurring in popular RTS games such as StarCraft II. The small attack range compared to the size of the map and the number of units and clusters promote smart group maneuvering which is notoriously hard for agent-centric learning.

We adopt MAgent's default combat settings: at each time step agents can either move to or attack any cell within a radius of two. When attacking a unit, the attacker's attack value is subtracted from the attacked unit's hit-point value. Units with hit-point value ≤ 0 are removed. The goal is to accumulate rewards by eliminating all opponents. The rewards are also closely following the default settings: 5 for killing an enemy unit, -0.1 for attacking an empty grid cell, 0.2 for attacking an enemy unit, and -1 for being killed (increased from the -0.1 default value). Additionally, there is a reward of 100 for killing all enemies. Each game is restricted to have a length of at most 300 time steps.

We use the full observability settings: there is no unit vision limitation or fog of war.

Necessarily, sector actions should translate into low-level actions for the agents located within. Here we choose to use 5 scripted algorithms to do so: four that move units in each of the cardinal directions and one for approaching and attacking enemy units, the same attacking script used by the enemy. In future work these scripts can be replaced with other fixed or even learnable agent policies, and the sector action can simply be which of these policies the agents within should use, similar to a meta-learning approach (Frans et al. 2017). To avoid switching behaviors too often and to make exploration easier via temporal abstraction, the sector action is only changed every k environment time steps. A value of $k = 5$ was chosen based on a brief tuning process. As a note, keeping the sector action fixed for a number of steps is recommended in conjunction with moderately intelligent scripts or policies. Otherwise, extremely basic scripts might move left for 1 step and then keep colliding into another unit or a wall for the remaining $n-1$ steps.

The network input is the global state, a 64×64 grid with 5 feature planes: obstacles, own and enemy units and own

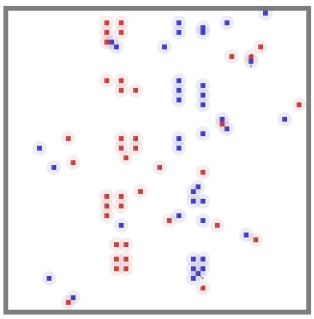

Figure 2: Randomly generated MAgent scenario, 64×64 map with 40 units on each side, loosely split into groups.

and enemy hit-points. All features are normalised by their maximum possible values.

Model and Training Settings

The network used for the sector abstraction is shown in Figure 1, with actions being emitted on a 4×4 sector granularity, unless otherwise specified. For IQL, the input is a 13×13 observation centered around the agent's position, with 7 feature planes. In addition to the 5 already mentioned, there are two more layers containing minimap information for both armies. For these layers, the original map unit information is scaled down to 13×13, and normalised such that each cell value is equal to the number of units within divided by total number of alive units. A value of 1 is added to the cell containing the acting agent, in the allied minimap. These minimap layers were included to provide the agents information about the global state.

The IQL network consists of one 4×4 stride-2 convolutional layer followed by two 3×3 stride-1 convolutional layers, all three with 32 filters and followed by ReLU activations and batch normalisation. A linear layer of 256 units follows, and two output heads for the value and advantage functions which are summed to extract the final Q-values.

The learning algorithm is based on DQN (Mnih et al. 2015), with the dueling architecture update and N-step returns, with $N = 3$ across all experiments. The replay buffer contains the most recent 300k experiences. Training starts after the buffer is populated with 10k experiences. Batches of 128 experiences are sampled uniformly from the replay buffer every 128 steps played, and the network parameters are updated. The target network is updated every 3000 time steps. All experiments use $\gamma = 0.99$.

All networks are trained using the Adam learning algorithm (Kingma and Ba 2014) with the learning rate initialized with 0.000625 for our method and with 0.0001 for IQL, both chosen after brief tuning on a 40×40 scenario with 10 units per army. Exploration during training is performed using independent ε-greedy action selection, in which each sector chooses its action greedily using only its own Q_{S_i}. The ε value is annealed from 1.0 to 0.02 over the first 50k games and kept constant afterwards.

Results

We train all models with 700k games against the handcrafted script mentioned in the previous section. After every 1000 training games 100 test games are played independently with actions chosen greedily in evaluation mode – i.e., ε for exploration is set to 0. In what follows the proportion of these games in which all enemy units are defeated within the target time limit is called *test win rate*.

Figures 3 and 4 show the test win rate for three single runs of IQL and Spatial Action Decomposition Learning, on 64×64 grid scenarios with 20, 40, and 80 units for each side. For the chosen parameter settings IQL fails to learn how to defeat the script consistently — not even achieving a 50% win rate. The training also appears to be unstable at times. As described in the introduction, this may be caused by all agents exploring simultaneously. We speculate that using VDNs could mitigate this problem, and isolating the

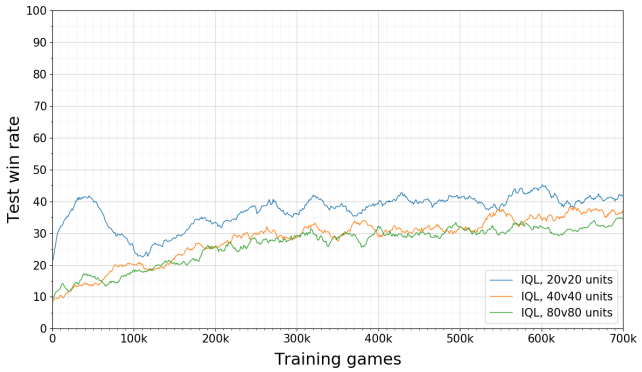

Figure 3: Test win rate for IQL and different army sizes measured every 1000 games during training, A sliding window of length 20 is used for smoothing

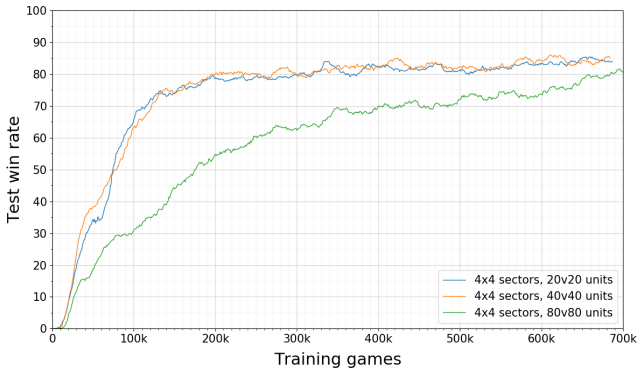

Figure 4: Test win rate for the 4×4 sector multi-agent action network and different army sizes. The same methodology was used as in Figure 3.

effect of the sector abstraction, comparison to an IQL+VDN baseline is on our todo-list.

Our method shows a more stable learning behavior and can consistently defeat the opponent in more than 80% of the games after training. Controlling 80 units well seems significantly more difficult than 20 or 40, as shown by the slower learning process. We think this is because there are more units placed randomly behind the enemy lines. Grouping allied units while dealing with the enemy's requires more complex maneuvering, and often makes consolidation of one's army more difficult.

From a qualitative point of view, our method displays forms of coordination such as surrounding or attacking the enemy on a wider front and fleeing from a stronger group of enemies until regrouping with the rest of the army. Snapshots from a sample game are shown in Figure 7. We observed our system often using a small number of units as bait to split the enemy forces and fighting them one-by-one.

Learning with a model that uses 8×8 sectors is more challenging. Firstly, it gets stuck more often in local optima of predominantly choosing to attack and avoiding move actions. This is likely due to the more difficult exploration induced by the 64 sectors. Using mostly attack actions leads to test win rates around 50%, as expected. After the con-

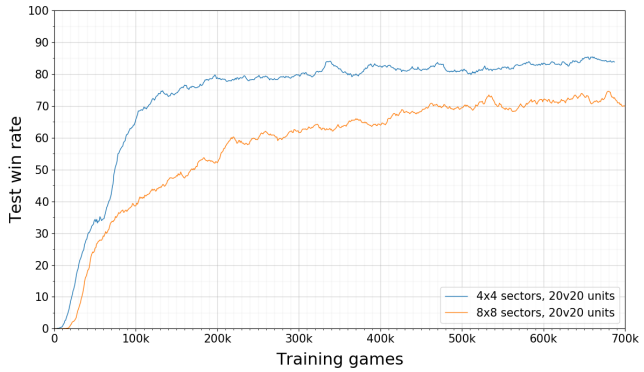

Figure 5: Test win rate for 4×4 and 8×8 sector decomposition, for models trained on scenarios with 20 units per army.

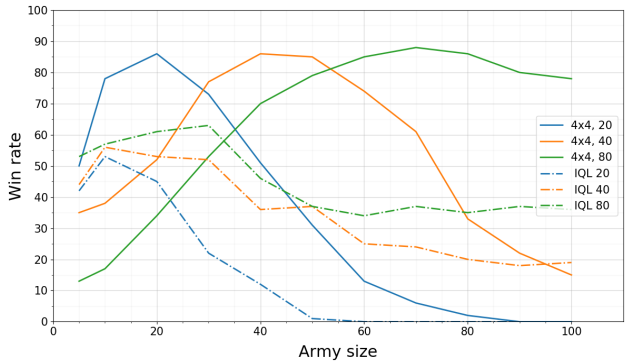

Figure 6: Results after $700k$ games of learning, for methods trained with scenarios of a specific army size (20, 40 or 80 units). Win rate is shown for evaluation in scenarios with a range of different army sizes, from 5 to 100.

volutional layers and S1 were initialised with weights from a trained 4×4 sectors model, learning proceeded smoothly and 75% win rates were reached, as seen in Figure 5. Building on representations and features learned by the 4×4 model helps exploration in the more difficult 8×8 scenario. Assigning credit for four times as many sectors is more difficult and as expected, learning converges more slowly with the maximum win rate being 10% lower.

Finally, Figure 6 shows the win rate of the 6 trained models from Figures 3 and 4 on scenarios with a range of different starting army sizes, instead of the only one used throughout the training. Models trained with fewer units do not generalize well, and both models that learned using a maximum of 20 units fail to win a single match when controlling more than 100 units. Sector models trained with more units do not perform too well in smaller battles either, mostly because although the network eventually sees states with few units, they are very differently positioned from the starting states of small scenarios. Better performance could be obtained by designing a mixed curriculum of battles. IQL has worse performance, but using larger armies results in stronger policies even on the smaller scenarios. That might be because with IQL, more agents translate into more experience samples.

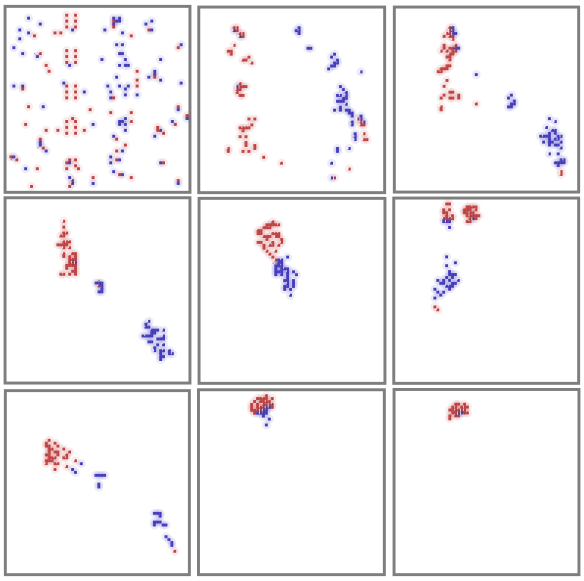

Figure 7: Snapshots from an 80v80 game played after 700k training games by the 4×4 sector algorithm (red units) against the scripted player (blue units). Baiting behavior to split the enemy forces can be observed multiple times.

Conclusions and Future Work

In this paper we study cooperative-competitive multi-agent RL with agents learning from a single team reward signal. Both individual as well as centralized learners fail to scale successfully with growing team sizes, and decomposing the joint action-value function into per-agent action-value functions has previously shown great promise. Here we present a method that exploits existing spatial action correlations to decompose the joint action-value function on a per-sector basis instead, and show its effectiveness in both small and large scale scenarios. Results for combat scenarios with 20 to 80 units per side show improved performance over simple scripting and independent Q-learning.

Without a doubt the presented method can be improved in various ways. For instance, QMIX can be used to facilitate non-linear Q-value mixing, and low-level sector policies could be learned rather than using fixed scripts. Also, skip-connections can be used to enrich high-level features with low-level map details – providing high-fidelity context to lower-level action generation. Another interesting research avenue is a deeper integration of hierarchical action-value functions operating at different temporal scales (Kulkarni et al. 2016), as temporally-abstracted exploration should help alleviate some of the problems that arise when decomposing the map in more and more sectors. Another remark is that while moving agents at a higher level is spatially correlated, often professional players control close units very distinctively based on their abilities and the tactical situation. To allow for such fine-grained control an option would be the use of very small sectors, or of more complex tactical algorithms as sector actions. Lastly, one can imagine that based on the learned Q-functions it may be possible to focus Monte Carlo tree search similar to AlphaGo (Silver et al. 2016) to construct much stronger multi-agent systems.

References

Busoniu, L.; Babuska, R.; and De Schutter, B. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Systems, Man, and Cybernetics, Part C* 38(2):156–172.

Churchill, D., and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *IEEE Conference on Computational Intelligence in Games (CIG)*, 1–8. IEEE.

Churchill, D.; Saffidine, A.; and Buro, M. 2012. Fast heuristic search for RTS game combat scenarios. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'12, 112–117.

Claus, C., and Boutilier, C. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI* 746–752.

Colby, M. K.; Kharaghani, S.; HolmesParker, C.; and Tumer, K. 2015. Counterfactual exploration for improving multiagent learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 171–179. International Foundation for Autonomous Agents and Multiagent Systems.

Dayan, P., and Hinton, G. E. 1993. Feudal reinforcement learning. In Hanson, S. J.; Cowan, J. D.; and Giles, C. L., eds., *Advances in Neural Information Processing Systems 5*. Morgan-Kaufmann. 271–278.

Devlin, S.; Yliniemi, L.; Kudenko, D.; and Tumer, K. 2014. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 165–172. International Foundation for Autonomous Agents and Multiagent Systems.

Dumoulin, V., and Visin, F. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Eck, A.; Soh, L.-K.; Devlin, S.; and Kudenko, D. 2016. Potential-based reward shaping for finite horizon online pomdp planning. *Autonomous Agents and Multi-Agent Systems* 30(3):403–445.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2017a. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*.

Foerster, J.; Nardelli, N.; Farquhar, G.; Torr, P. H. S.; Kohli, P.; and Whiteson, S. 2017b. Stabilising experience replay for deep Multi-Agent reinforcement learning. *arXiv preprint arXiv:1702.08887*.

Frans, K.; Ho, J.; Chen, X.; Abbeel, P.; and Schulman, J. 2017. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*.

Guestrin, C.; Koller, D.; and Parr, R. 2002. Multiagent planning with factored MDPs. In *Advances in neural information processing systems*, 1523–1530.

Gupta, J. K.; Egorov, M.; and Kochenderfer, M. 2017. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems*, 66–83. Springer International Publishing.

Hausknecht, M. J. 2016. *Cooperation and communication in multiagent deep reinforcement learning*. Ph.D. Dissertation, The University of Texas at Austin.

Jorge, E.; Kågebäck, M.; Johansson, F. D.; and Gustavsson, E. 2016. Learning to play guess who? and inventing a grounded language as a consequence. *arXiv preprint arXiv:1611.03218*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, 3675–3683.

Laurent, G. J.; Matignon, L.; Fort-Piat, L.; and Others. 2011. The world of independent learners is not Markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems* 15(1):55–64.

Leibo, J. Z.; Zambaldi, V.; Lanctot, M.; Marecki, J.; and Graepel, T. 2017. Multi-agent reinforcement learning in sequential social dilemmas. 464–473.

Lelis, L. H. 2017. Stratified strategy selection for unit control in real-time strategy games. In *International Joint Conference on Artificial Intelligence*, 3735–3741.

Levy, A.; Platt, R.; and Saenko, K. 2017. Hierarchical actor-critic. *arXiv preprint arXiv:1712.00948*.

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.

Ontañón, S. 2017. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research* 58:665–702.

Panait, L., and Luke, S. 2005. Cooperative multi-agent learning: The state of the art. *Auton. Agent. Multi. Agent. Syst.* 11(3):387–434.

Peng, P.; Yuan, Q.; Wen, Y.; Yang, Y.; Tang, Z.; Long, H.; and Wang, J. 2017. Multiagent Bidirectionally-Coordinated nets for learning to play StarCraft combat games. *arXiv preprint arXiv:1703.10069*.

Rashid, T.; Samvelyan, M.; de Witt, C. S.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. QMIX: Monotonic value function factorisation for deep Multi-Agent reinforcement learning. *arXiv preprint arXiv:1803.11485*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sukhbaatar, S.; Szlam, A.; and Fergus, R. 2016. Learning multiagent communication with backpropagation. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc. 2244–2252.

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.;

Leibo, J. Z.; Tuyls, K.; and Graepel, T. 2017. Value-Decomposition networks for cooperative Multi-Agent learning. *arXiv preprint arXiv:1706.05296*.

Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, 330–337.

Usunier, N.; Synnaeve, G.; Lin, Z.; and Chintala, S. 2016. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*.

Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A. S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; Quan, J.; Gaffney, S.; Petersen, S.; Simonyan, K.; Schaul, T.; van Hasselt, H.; Silver, D.; Lillicrap, T.; Calderone, K.; Keet, P.; Brunasso, A.; Lawrence, D.; Ekermo, A.; Repp, J.; and Tsing, R. 2017. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

Wang, C.; Chen, P.; Li, Y.; Holmgård, C.; and Togelius, J. 2016a. Portfolio online evolution in starcraft. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 114–120.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016b. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, 1995–2003.

Yang, Y.; Luo, R.; Li, M.; Zhou, M.; Zhang, W.; and Wang, J. 2018. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*.

Zheng, L.; Yang, J.; Cai, H.; Zhang, W.; Wang, J.; and Yu, Y. 2017. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *arXiv preprint arXiv:1712.00600*.