Multi-agent reinforcement learning in a simulated harbour environment

Empirical investigation of cooperative and competitive behaviours using proximal policy optimization.

Karol Skalski
Supevised by Rico Möckel & Amir Ahangi
Bachelor Thesis
University College Maastricht
June 2019

Table of contents

Introduction

Reinforcement learning lays between supervised and unsupervised learning, namely there is no formula for the right output of the policy, yet the agents receive feedback on their actions. Optimizing parametrized policies without prior knowledge of the environment is a rich and complex challenge. The agents need not only optimize their output well towards the rewards, but also efficiently explore their environments in order to build their policies. Reinforcement learning opens an opportunity for unpredictable ingenuity to emerge. Training RL agents is computationally expensive as the algorithms need to explore the environment based on randomly initialized policy until they find out rewarding strategies.

Recent developments in RL research include Google's robot-arm plant, which brought scaled reinforcement learning outside of the realm of computer generated simulations by making the agents control dozens of robotic arms in parallel and collect feedback and data with cameras and sensors (Levine et. al, 2017). Furthermore, the Deep Minds competitive model of Alpha Go (Silver, et al., 2016) is one of the breakthrough algorithms of RL and an example of outstanding results achieved through algorithm self-play and learning to simulate human players. Significant body of research focuses on applications of RL in the industry. Kara, Dogan (2018) developed "cased based" reinforcement learning methods for inventory management and automated ordering, which financially outperformed event-based or demand-based systems. The authors indicate that reinforcement learning methods in industry are more advantageous, the more complex the environment. Sallab et. al (2018) tested an application of RL with recurrent neural networks to control an autonomous car simulator. Reinforcement learning was also used for the development of supply chain management systems for a maritime port logistics chain (Ascencio, et al., 2014). Al-Rawi, Ng, Yau (2015) explored possibilities of applying agent-based approach to make routing decisions for distributed wireless network routers. The autonomous routers tended to make better routing decisions than ones with predefined routing policies. Manion, Dougan and Howley (2016) describe an applications of multi-agent RL architecture to Adaptive Traffic Signal Control with dynamic information transfers between agents optimizing the traffic flows. Carrio et al. (2017) describe wide application of RL in control of autonomous drones, Schultz and Sokolow (2018) review the use cases for RL in solving public transportation challenges.

Reinforcement learning can successfully be applied in numerous problems not only in research. An area where reinforcement learning could be useful is optimization of complex industrial processes, such as management of a maritime unit. The role of optimization in the logistics sector is growing in importance as logistics costs have an increasing share in total costs of goods (Ascencio et al., 2014). Statistical modelling can therefore help large organizations understand, predict and solve their problems. Maritime units are special environments because of the multitude of stakeholders interacting with each other in order to achieve their economic objectives. Using agent based simulations can yield interesting insights into their dynamics and help the decision makers in strategic planning (Henesey, Notteboom, & Davidsson, 2004). Specifically, agent-based approaches can help to understand the incentive structures and their influence on the dynamics between the actors. For example, Irannezhad, Hickman and Prato (2017) used a multi-agent reinforcement learning to model the effect of communication between agents through a central information hub, the Port Community System. The agents managed to create communication protocol to coordinate the shipments and share vehicles and optimize the costs. This thesis will propose a multi-agent approach to modelling a simplified and abstracted simulation of a maritime port using reinforcement learning.

Currently, RL is becoming more and more of an experimental, empirical, rather than theoretical domain (Whiteson, 2009; Schulman et al., 2017; Minh et. al., 2016; Lowe et al., 2018). The complexity of environments in which the algorithms are tested does not allow for creating a differentiable models of them and theoretically proving superiority of a given variation of the algorithm. Instead the algorithms are tested in controlled, often simulated environments against objective performance measures. This thesis will take a similar approach of experimentally examining the performance and behaviour of RL algorithms in a multi-agent context.

Because reinforcement learning is a rich and complex discipline, there is a plethora of unsolved challenges within it, especially in multi-agents variants of the algorithms. Even experimental research in RL is based on theorems that guarantee convergence given a sufficient number of trials and adequately small update steps on the policy (Carden, 2014). Multi-agent scenarios pose significant challenges to proofs for convergence of reinforcement learning algorithms. In single agent scenarios, the environment of the agent does not change its behaviour as the agent learns. Therefore, given enough attempts and enough exploration of the environment, the agent is guaranteed to find a locally optimal strategy that will consistently work within a given environment (Bus et al, 2010). However, in contrast to single agent scenarios, the environment in multi-agent scenario does not remain static. If a few agents simultaneously evolve, they dynamically change the environment for each other. The behaviour of the agents then constitutes part of the environment for the other agents. Hence, there is no guarantee that as the agent explores the environment its rewards will improve, specifically in adversarial scenarios with conflicting rewards (Bus et al, 2010). The environment is constantly changing according to the policy that is being optimized, giving no guarantee for a stable system and thus convergence. The competitive dynamics between agents cause difficulties for the assessment and benchmarking of the algorithms (Lowe et al., 2018). Nonetheless, the problem with theoretical proof for convergence does not imply that the algorithms will not improve. Instead, the metrics for improvement need to be adjusted. For example, Contzer and Sandholm, (2007) defined the metrics for a satisfactory multiagent learning algorithm to be that the agents learn the optimal strategy against stationary opponents and converges to a Nash equilibrium in self-play.

Another difficulty with Reinforcement learning algorithms, especially in the context of modelling real life situations, is arising of unrealistic scenarios. The RL algorithms are optimizing towards the maximum cumulative rewards, which in some scenarios may lead to, for example, inequality between agents. Competing agents may develop strategies that promote disproportionate inequalities between each other because those strategies lead to higher overall reward. Even in cooperative scenarios, the objective of RL algorithms is the maximization of total expected value of accumulated reward. From this perspective strategies such as sacrificing one agent for the sake of other agents may turn out to be an optimal cooperation strategy. In order to simulate real world situations such as logistics units or actors in the financial markets, the reward structures need to reflect values relevant to the modelled situation, such as equality of rewards among agents.

Due to the above mentioned challenges, there is no standard, established way of modelling mixed cooperative and competitive environments, and the dynamics within them are highly complex. Therefore, the thesis will empirically investigate the cooperative and competitive dynamics between agents based on different policy setups and reward structures following them. The thesis considers the following question: What are the competitive and cooperative dynamics in the behaviour of agents trained with reinforcement learning in a single- vs. multi-agent environment of a simplified and abstracted modelled harbour?

The aim of the thesis is to propose an extension of known multi-agent reinforcement learning methods for use in modelling complex industrial processes. It is a proof of concept application of reinforcement learning to a multi-agent scenario simulating a simple abstracted harbour environment with vessel agents trying to dock and exit the port. The objective of the agents, similarly to a real harbour (Port of Rotterdam, 2016, p.10), is efficiency and safety. More precisely, the agents aim to arrive at the berth and exit the harbour within a short time while avoiding collisions with other agents and avoiding the unnecessary travel. The scenario entails both cooperative and competitive elements as the agents need to coordinate in order to make the traffic smooth, yet each of them is rewarded individually for unloading their own cargo. Competition and cooperation separately have proven to be effective strategies in reinforcement learning. For instance, DeepMind’s AlphaGO Go player used self play in order to improve its strategies (Silver,2016). This approach is very convenient for curriculum planning, as the algorithms always play exactly at their difficulty level when competing with themselves. It also has a long history. As far as In 1959, Arthur Samuel used self play to collect data for his limited depth tree search to learn checkers. Cooperative scenarios usually entail shared reward systems. An example of application of independent cooperative algorithms is OpenAI five – a team of heterogeneous cooperating agents playing a multiplayer battle-arena game Dota 2 with 20-dimensional action space and complex, partially observable state. Open five also uses self-play between teams of agents to optimize their performance. Despite the complexity of the game the algorithm defeated current Dota 2 champions (OpenAI, 2018).

The experimental design on which the analysis is based was drawn based on prior research. All experimental conditions used Proximal Policy Optimization (PPO) as a learning algorithm. The experiment compared the performance and reward distribution in three conditions - (1) All vessels were controlled by a single central agent, with three action probability distributions, one for ach agent. The reward for the policy was sum of rewards for all agents. In the second condition (2) all agents acted individually but shared the same policy network and were rewarded separately. In the third condition (3) every agent had their own individual policy network optimized according to individual reward. The thesis starts with the theoretical background and the formulation of the problem as a Markov Decision Process. This is followed by a review of the available reinforcement learning algorithms from value iteration algorithms, to policy iteration algorithms including PPO. Next, the methods section outlines the experimental set up of the environment and the conditions. The framework used in this study was found largely insufficient, yielding unstable results. However, the successful learning runs displayed coordination patterns and differences between the conditions. The results are discussed in light of their implications for the research question. Finally, the thesis concludes by providing an answer to the research question and suggestions for future research. Integral part of the thesis are implementations of all policies, scripts for loading the models and visualising the learned policies, the video documentation of both validation phase and experimental phase, the modified environment repository and logs of the experiment together with visualisations of computational graphs to be used with tensorboard.

Background

Formulation of the problem

The problem of vessels in the harbour can be formulated as a Markov Decision Process. This is a control task where a vessel agent interacts with the environment in discrete time t.Given the state of the environment s_t the agent takes action a_t based on a stochastic policy $\pi(s)$ parametrized by parameter vector θ. After each action, the state of the environment transitions to the new state s' according to a partially random transition function $P(s,a)$. This process forms a discretized trajectory $\tau = (s_{1,}a_1, s_2, a_2, s_3, a_3, \ldots)$. The current state of the environment contains all the information required for the agent to make decisions and for the environment to transition to the next state . After the transition, the agent receives a reward r_t and information about the new state. The rewards lose their relevance over time. The discounted value of the future rewards is calculated based on the discount rate $\gamma = 0.999$ applied after each passing time-step. The accumulated discounted value of the rewards for time t_0 within a time horizon $H$is denoted by:

$$R = \sum_{t=0}^{H} \gamma^t \cdot r_t$$

and the value V of the state s can be denoted as expected value of total discounted rewards.

$$V^\pi(s) = E\left[\sum_{i=0}^{H} \gamma^i \cdot r_i\right]$$

The agents aim is to maximizeV^π. The policy improves towards the accumulated discounted value of the rewards, awarded within a time horizon H from the action. The agent is estimating V^π based on past experiences of action-state-value sequences and adjusts the policy parameters θ with the learning rate α to maximize the loss function L after each learning episode. The goal of the agent is to approximate an optimal policy that would yield the highest expected rewards for all states given their probability.

$$\pi^* = argmax_\pi V^\pi(s) \ \forall s \in S$$

In the Markov decision process formulation, the environment is static, meaning that the transition function $P(s,a)$ does not change. In multi-agent environments, every agent can be considered separately, so the agents constitute parts of the environment for each other. As the agents learn, their influence on the environment changes. Given two agents a and b, policy π_b is part of the transition function for agent a $P(s,a \mid \pi_b)$ which makes the transition function dynamic as $P(s,a \mid \pi_b) \neq P(s,a \mid \pi'_b)$. To acknowledge the dynamic interaction between agents as separate from the environment, Littman (1994) proposed a framework of Markov Games, where the sets of agents perform sets of actions in an observable environment. Instead of optimizing towards the stationary $P(s,a)$ the agent's goal is to optimize for the worst case scenario of the movements of the opponent with a probabilistic policy. The algorithms implemented in this study optimize towards the current strategy of the other agents with an implicit assumption of them being stationary.

Algorithms

The algorithms for multi-agent reinforcement learning can be the same algorithms used for single agent learning problems. When optimizing the policy for one agent, the other agents' actions are treated as a part of the observable state based on which agents choose their actions. Because the environment consists of evolving agents, it is non-stationary and can lead to issues with convergence of learning. This may impede optimization methods that rely on a stationary nature of the environment. Consequently, these methods are less applicable to multi-agent problems, than to single-agent ones. An example of such a method is Deep-Q-Network (DQN).

Deep-Q-Networks

DQN is an off-policy value iteration algorithm. It is a variation of the Q-learning policy, where the Q-value (i.e. an expected value of the state-action pair) is estimated.

$$Q^\pi(s,a) = E[R \mid s = s_t, a = a_t] = E_{s'}\left[r(s,a) + \gamma E_{a' \sim \pi}[Q^\pi(s',a')]\right]$$

The algorithm iteratively improves its estimate of the Q-value, towards an optimal Q-value based on experience gathered by interacting with the environment:

$$Q^*(s,a) \ = \ r(s,a) + V^*(s') \ = \ r(s,a) \ + \ \gamma \max_{a'} Q^*(a',s')$$
$$Q'(s,a) \ = \ Q(s,a) + \alpha(\, r(s,a) \ + \ \gamma \max_{a'} x'(a',s') \ - Q(s,a)\,)$$

The agent chooses actions with$\epsilon \ greedy$algorithm based on the highest estimated Q-value. In order to allow the agent to explore the environment, oftentimes while learning agent can take a random action with a constant probability p (Touati et al, 2018)**.** DQN is a variation of the algorithm where the Q-function is approximated using deep neural networks and learns by minimizing the loss function

$$L(\theta) = E_{s,a,r,s'}[(Q^*(s,a|\theta) - y)^2], \quad where\ y \ = \ r \ + \ \gamma \max_{a'} \bar{Q}^*(s',a')$$

$\bar{Q}^*(s,a)$is a periodically updated target Q-function. The rewards, states and actions arise from random sampling from a replay buffer$D$of experience, which is gathered using an arbitrary policy π. One of the most important advantages of off-policy algorithms is their sample efficiency (Sutton et al., 2000). DQN allows for reusing the past experience of the agent. However, because the environment in multi-agent scenarios is non-stationary from the perspective of any single agent, the experience of the agent from the past rollouts loses its applicability as other agents evolve (Foerster et al., 2018).

Despite its shortcomings, Deep-Q-Networks served as one of the first methods to showcase the capabilities of reinforcement learning in complex control tasks. With raw pixel feed as input, the algorithm achieved better results in Atari games than an average human player (Mnih, et al., 2015).

Policy gradient algorithms

Another family of algorithms, the on-policy algorithms, use the optimized policy in the learning episodes. For example, the policy gradient algorithms use gradient ascent to gradually move towards locally optimal policies (Sutton et al., 2000). In contrary to Q-learning, most policy gradient algorithms use stochastic policy functions. The policy takes the representation of the state as input and outputs probabilities of choosing each of the available actions. The agent chooses an action by sampling one from the distribution. The policy gradient methods are surprisingly simple, as the gradient of the loss function in respect to the policy parametersθ can be formulated in terms of an expectation and does not depend on the gradient of the state-distribution $p(s'|s,a)$.

$$\nabla_\theta J(\theta) \ = \ E\ [\nabla_\theta \ log\pi_\theta(a|s) Q^{\ \pi}(s,a)]$$

In the simplest case $Q^{\pi} \ \ (s,a)$ is substituted with the sum of discounter rewards $R = \sum_{t=0}^{H} \gamma^t \cdot r_t$ commonly referred to as REINFORCE algorithm (Williams, 1992). Similarly to value iteration algorithms policy gradient algorithms are guaranteed to converge given infinite visitation of all states and sufficiently small update steps (Sutton et al., 2000). An advantage of this approach is that exploration of the environment is embedded in the learning process, whereas in DQN the exploration is induced externally by making the agent choose a random action with fixed probability. Because of using a stochastic policy, small changes to θhave small effects on $\pi$and on the trajectory. In DQN on the other hand, when deterministic greedy policy is used, small changes in parameters of Q-function can have discontinuous effect on the outcome of actions (ibid.) A disadvantage of policy gradient methods is their instability. Their gradients tend to display high variability in the learning process and so do their estimates of expected rewards (Schulman, 2017). The problem is even more prominent in multi-agents setting, where the rewards of the agent is highly dependent on the actions of others. To mitigate variability in the gradients, the actor-critic family of algorithms was developed, where the value function $V^{\pi}(s)$ is approximated alongside the policy function. The loss function in advantage actor-critic algorithms is an advantage of an action formulated as the difference between the estimated and actual discounted accumulated rewards:

$$A(a,s) = Q(s,a) - V(s) = A(s) = r + \gamma V(s') - V(s)$$
$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) A(s,a)]]$$

The actor (the policy network) maximizes this difference trying to outperform critics expectations, whereas the critic tries to minimize the mean square of this difference and in order to assess the value of the state as accurately as possible. This adversarial setup balances the gradient by setting the level of expectations approximately at the level of performance (Mnih, et al., 2016). At first, actor-critic algorithms were developed in an asynchronous variant, A3C. Later the synchronous variant was found to be equal in performance and in certain cases more computationally efficient. Foerster et al. (2017) have used A2C algorithms to control a player in star-craft game. The critic network had parameters of other agents' policies as input, which indirectly allowed the agents to learn the behavioural model of other agents, without using it in the execution.

Despite the improvements to gradient stability, policy gradient algorithms still suffer from relative instability of learning. A proposed solution to that was mitigating the size of the update steps. Trust Region Policy Optimization (TRPO) (Schulman et. al., 2016) realized this principle by introducing a KL-divergence penalty, to limit change to parameters that an agent can make within one update step. A disadvantage of TRPO was use of second order approximations for policy updates. A year later Proximal Policy Optimization (PPO) was developed, using the same principle of limiting the update step by introducing a clipped surrogate objective to the loss function.

$$L^{CLIP}(\theta) = E[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t] \ where\ r_t(\theta) = \frac{\pi_{\theta}}{\pi_{\theta_{old}}}$$

PPO limits the upper bound of the policy updates based on positive rewards. In one update step the policy cannot increase the probability of an action by more than a constant ε determined empirically to be 0.2. The justification of PPO is empirical (Schulman et. al, 2017). The learner is pessimistic, as it limits its positive reward, but does not limit the reaction to negative rewards, that is decreasing the probability of given actions. The main advantage of PPO is that it uses the gradient to determine the direction of updates, which makes it more computationally efficient and easy to implement, yet its performance and stability is comparable to the one of TRPO. PPO is a version of the advantage actor-critic algorithm. There are two separate function approximators optimized in the process of learning. Critic approximates the value function V^{π} (s) , optimized according to loss as a mean of squared errors, and the actor, trying to maximize the difference between the expected and actual discounted rewards from the state. The actors' policy uses the clipped surrogate objectiva in the loss function to limit the update steps increasing the probability of positively rewarded actions.

As of now PPO is a state of the art policy gradient algorithm (Schulman et. al, 2017). It has been shown to perform better, or comparable to other more complex benchmark algorithms (Schulman, et al., 2017). PPO is a learning algorithm used in all experimental conditions in this thesis.

Other algorithms

There are a few other noteworthy algorithms that were not considered. An example of an off-policy policy gradient method using actor-critic approach is a deterministic policy gradient, (DPG) or deep deterministic policy gradient (DDPG). The main idea of DDPG is using an arbitrary stochastic behavior policy for collecting the data in the learning episodes and a deterministic target policy for execution. DDPG is especially suited for multidimensional continuous action spaces. (Silver et al., 2014) and has been successfully applied in a multi-agent setting (Lowe et al., 2018). In simple benchmark multi-agent scenarios DDPG outperformed other commonly used algorithms. Similarity to A2C**,** the multi-agent application of DDPG augmented input of the critic network with the actions of the other agents. This allowed the agents to use the knowledge about others' policies for learning, yet left them independent of each other at execution as the policy network had only the representation of the state as its input. Another interesting alternative, especially for scaled parallelized learning are evolutionary algorithms (Salimans et al., 2017).

All approaches listed above do not require the learner to create an explicit representation of the environment. Agents construct their strategy based on the expected outcomes of available actions. A completely different approach are model-based algorithms. Model based learning methods require a differentiable model of the environment. Use of those methods must thus be limited to very simple environments and are especially difficult to use in multi-agent scenarios, as every agent must keep approximating the model of every other agent. For this reason the model based algorithms were excluded from the comparison.

Methodology

The environment

The environment was built using OpenAI's particle environment (Mordatch & Abbeel, 2017). Multi-agent particle environment is an open source framework containing simple benchmark environments with agents, and passive environment elements - the landmarks. The framework was written in python using pyglet and openGL for visualisation. In the environment, agents and landmarks functioned in a two-dimensional space. Their behaviour was based on a simulation of basic physics, namely agents had a mass, velocity and inertia, and could take action by increasing their speed in either of the four directions (north, east, south and west). When two entities collide they bounce of each. In order to simulate land and berth areas the environment was extended by square shapes and their basic physics. Vessel agents, for the sake of simplicity, were circles with their mass set to one (figure 1) They could increase their speed in any direction and the environment provided passive impedance to their movement. The environment was based on a widely used OpenAI gym framework for reinforcement learning algorithms (Brockman, 2016). Using an already existing environment framework required significant modifications, yet the standard API of the environment allows for comparing other already implemented algorithms on the framework, and testing dedicated algorithms on other, more simple scenarios. The latter was used for validation in the current study.

Similar to the standard gym environment, the simulated port environment included the following methods:

- Reset → This was the initialization of the environment with random parameters. In this case, 3 vessel agents were created and placed in a random position within the bay. All the landmarks had static positions within the harbour.
- Step → This entailed moving the environment forward by one time step with each actor taking action a_t , the environment transforming based on the previous state s_{t-1} and set of actions a_{t-1}. The action space was discrete and the agents could take one of the four actions (i.e. move towards any direction), or stay in one place. The environment could change because all the forces that affected agents were acting and altering their velocity. The agents' positions changed according to their velocity. If a collision with other agents or landmarks occured, the opposing force arised to push the agents away from the other objects. Furthermore, the new positions of the agents lead to rewards or punishments. The Step method returned a sum of rewards and punishments per agent. The agents received new parameters of the state [s] and rewards R_t. The step method also informed whether the learning episode had finished.
- Render → This visualised the environment at the current state [s].

The vessels moved within the environment, aiming to dock at the berths. The harbour was an enclosed environment, where the agents could bounce off the walls as from other static objects. The state space was continuous and the action space, for the sake of simplicity, was discrete. Using a simplified and abstracted model was necessary, as the framework was a proof-of-concept focused on interaction between agents, not simulating an actual port.

The vessel-agent life cycle

Each vessel agent followed the same life-cycle. The life-cycle started by entering the harbour. Then, the agent moved within the harbour until it found its way to the berth. When it docked, the agent occupied one of the spots at the berth for 1 time steps. After it had docked, the agent could move again until it found the exit. Then, emulating a new ship the agent was marked as loaded, and could go to the berth again.

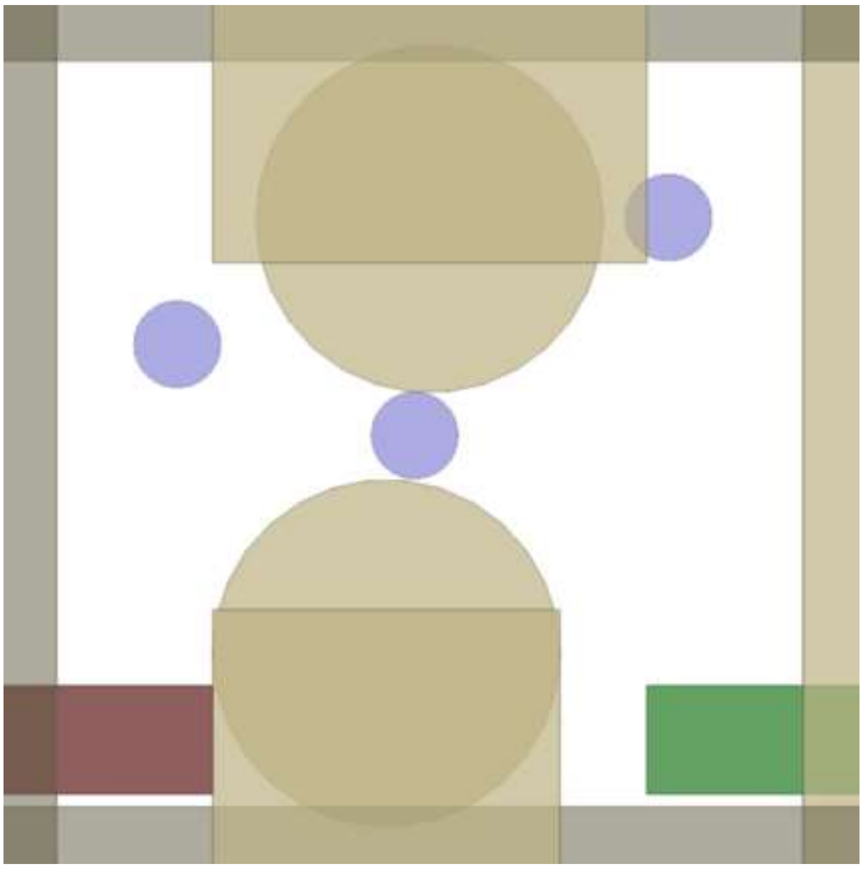


Figure 1. A simple environment model: a vessel agent traveling from the exit (brown) towards the berth(green) before the narrow channel.

Observability of the state

All agents could observe their own position and velocity, the position and velocity of all other agents, and a binary information whether the other agents had docked. All the information about the state was equally visible to all agents.

Rewards structure

Almost every action of the vessel agents got rewarded or punished immediately. The most important reward was the reward for docking. Namely, the vessel agents received a reward of +5 for docking at the berth. It was essential that the vessels tried to dock and exit the harbour as soon as possible. Consequently, the vessel agents received a small negative reward of -0.001 for being in a harbour area. In order to simulate fuel consumption, the agents needed to be punished for movement as opposed to remaining static. Thus, the vessel agents received a small negative reward of – 0.001 for each movement, which disincentivized unnecessary travel. Moving within the port area with other agents could have caused collisions. Therefore, the agents were punished for contact with land or contact with other agents. Each ship had a safe-zone around its centre. When another ship entered the safe zone, both agents received a negative reward of -0.1 for each timestep when the safe zones of the vessels overlapped.

$$r_t = 0.001 * velocity - 0.001 - 0.1 * (is\ in\ collision) + 5 * ((docked\ if\ loaded)\ or\ (exited\ if\ unloaded))$$

Experimental Conditions

The experiment consisted of three policy conditions. In one policy condition (1) one policy network simultaneously governed all vessel agents. This policy condition was a de-facto single agent approach. The general policy was rewarded based on the sum of the agents' rewards in each timestep. The policy network consisted of one hidden layer of 60 neurons and the output layer of 3*5 neurons with softmax activation per each 5 neurons corresponding to 5 choices of actions. In the second condition (2) each agent had the same policy as other agents, but acted autonomously. The agents' rewards were based on individual actions of the agent, but policy updates were based on one agent per episode. Since all agents had the same action capabilities and randomly initialized starting positions, experience of one agent generalized to all others. Gathering observations only for one agent simplified the implementation. In the third condition (3) every agent had their own policy network that updated according to the experience and the rewards of this particular agent. The policy networks for last two conditions had 2 hidden layers with respective sizes of 40 and 35. All hidden layers used relu as activation function. In all cases the input size was 15 corresponding to position (x,y), velocity (x,y) and a boolean "is docked" information about each agent. Since all agents saw all of the environment, single agent policy network (1) did not need to have a different input size than others.

Validation of the implementations

Implementations of all algorithms were validated in a basic environment ("2agents_demo" in environments repository). In this scenario, two agents tried to occupy two landmarks with randomly initialized positions. Agents got rewards for decreasing their distance to a closest landmarks and increasing the distance to another agent.

$$reward_a = -2 * (distance\ to\ closest\ landmark) + 0.1 * (distance\ to\ agent\ b) - 0.5 * (is\ colliding\ with\ agent\ b)$$

It is a variation of "simple_spread" scenario from Lowe (2016), with a simplified reward system and only two agents. An expected response of the agents was that both of them would occupy different landmarks.

Criteria for the success of the algorithm

Performance of the algorithms was measured with running average of rewards of all agents over 10 timesteps during one learning episode. The results were compared to arbitrary benchmarks based on manually executed strategy of moving the agents with moderate speed. Both the validation scenario and the experimental scenario have predefined target strategies. An important measure was whether and how regularly the algorithms developed the target strategy.

The chosen algorithm: Proximal Policy Optimization

Among the reinforcement learning algorithms discussed above, Proximal Policy Optimization seemed the most appropriate due to its relative simplicity in implementation and high performance. The policy is stochastic, meaning the output of the policy network is a conditional probability distribution over a discrete set of actions. The actions taken by the agents were sampled from that distribution using softmax.

Hyperparameters for both actor and critic networks were chosen by trial and error. The learning rate for actor was 0.001, for the critic 0.005. The episode length was 600 timesteps.

Shorter episodes led to the agents settling for local minima very early and did not lead them to develop an expected strategy of reaching the berth and coming back to the exit. The learning rate assumed was 0.001 and learning rates of 0.0002 also yielded similar results. The loss function was clipped according to the rate $\epsilon = 0.2$ based on the original PPO paper (Schulman et al, 2017). Due to the comparative nature of the analysis, all of the hyperparameters where kept similar across the conditions, apart from the size of the network. Tensorflow was used to create the computational graphs together with optimization algorithms. The visualisation of the computational graphs can be found in the appendices.

The policy function approximator

Both actor and critic use neural networks architecture with densely connected hidden layers as function approximators. The possibilities for other types of neural networks included Recurrent Neural Networks (RNN). Those are especially attractive in partially observable environments where the agents need to selectively retain information. An example of the use of RNN is Long-Term-Short-Term Memory Networks used in a 3D labyrinth simulation game (Mnih et al., 2016). In the case of full observability, simplicity of the feed-forward networks makes them preferable. A possible limitation for feed-forward neural networks is that all information about the state needs to be fitted at each time frame as input. For instance, in DeepMind’s Atari games control task the input consisted of 4 consecutive screen frames in order to capture the direction of the motions in the state (the approach used in DQN). In order to avoid computational overhead complexity in the current study, the velocity of agents was included in the observation space.

Optimization algorithm

As opposed to supervised learning, the optimization method in reinforcement learning affects the dataset itself, as it influences the actions of all of the agents and thus the learning environment for each of them. Two main factors in the choice of the optimization algorithm were performance and the and popularity in research. The most standard and simple option was using the Stochastic Gradient Descent (SGD). The current study, however, used the algorithm based on SGD, Adam optimizer for reinforcement learning due to its high performance and robustness in respect to hyperparameters (Kingma & Ba, 2014; Schulman et al., 2017).

Noise injection

Injecting extra noise is a common practice for increasing exploration. The noise injected on the policy parameter level seems to improve performance more than action-level noise injection (Plappert et al., 2017). This experiment used very moderate for noise - entropy was one of the substrates of the loss function of the actor, with the coefficient of 0.02 for condition (2) and (3).

Implementation and hardware details

The algorithms were modifications of existing implementations of PPO for single agent settings modified to suit multi-agent problems. The implementations are flexible, and can be automatically adapt to any number of homogenous agents with continuous observation space and discrete actions space. All implementations used the tensorflow framework. The algorithms were executed using both CPUs and GPUs. The single-agent (1) algorithm was implemented using parallelized data collection and the two other conditions were single-thread implementations. The algorithms were executed on a personal PC and using a virtual private server. The server operated on one GPU Nvidia Tesla P100 and 8 CPUs on google cloud with debian 9 and python 3.5. PC had 8 CPUs and run Ubuntu 18.04 with python 3.6. Unfortunately, hardware accelerations were not very effective, as the most time consuming part of the process was the inefficiently implemented computations of state transitions. One up to three episodes were used per each policy update, so parallelizing the rollouts was limited. The speed of learning was still higher on a PC than on a cloud machine. Each condition took about 5-7h for 20 000 Episodes of 600 timesteps. All implementations may be found in thesis_2 repository with prefix "train_".

Results

In a validation environment "2_agents_demo" all of the algorithms managed to increase their rewards over time, though only the single policy for all (1) and separate policies (3) developed visibly towards the expected strategy. Agents which shared the policy (2) were attempting to occupy the same landmarks in most of the runs.

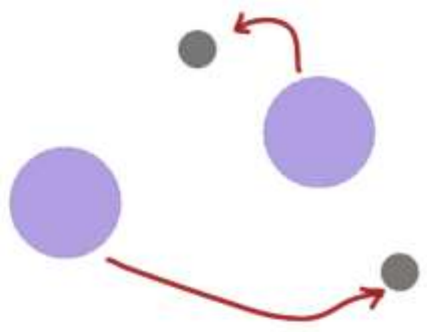


Figure 2. Expected strategy in validation environment: each agent aims to occupy their own landmarks

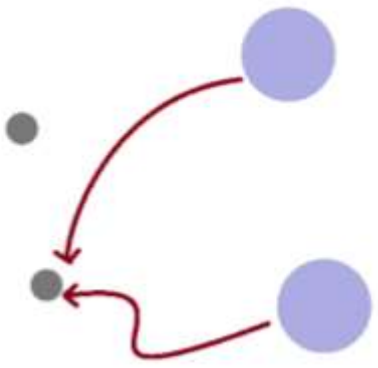


Figure 3. Suboptimal strategy for shared policy parameters: agents aim to occupy the same landmark.

In the harbour environment, the agents were sometimes able to find out the optimal strategy of going towards the berth when they needed to unload the cargo, and afterwards going from the berth to the exit. The results, however were not stable and the agents often stopped in the learning process and did not move at all. The rewards for correct docking with the manually executed strategy were about 15 points per agent per 600 time-steps. When the agents did not find the way to the berth, the rewards were on average -1.6 per 600 timesteps when they did not move. If the agents developed meaningful strategies, the rewards were around 5-15 points per agent. Agents sharing policy parameters (2) were either all seeking to dock and exit, or all were stuck not moving at all. The shared policy network seemed to have the least collisions in comparison to all other conditions, however the successful runs were too infrequent to reliably compare the quality of cooperation between the conditions.

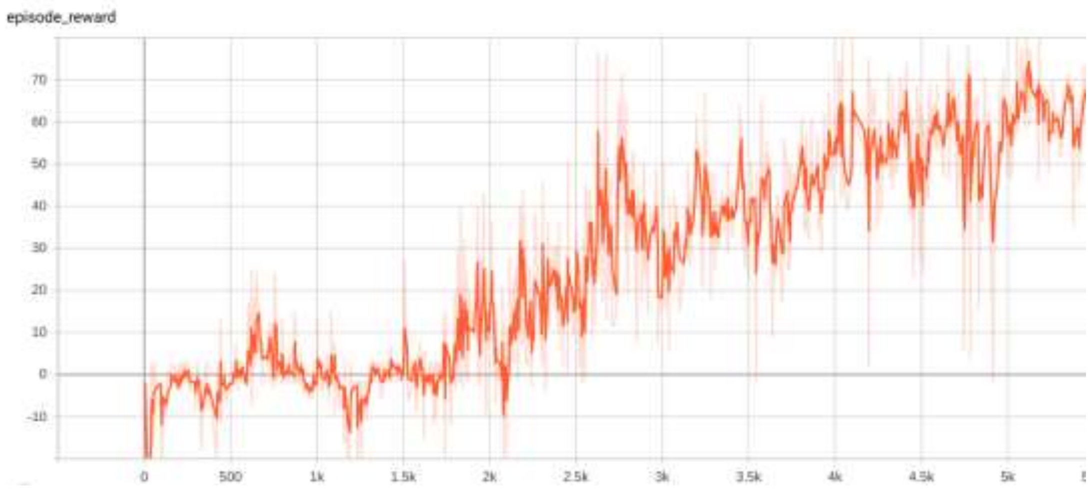


Figure 4. Successful run with all agents sharing one policy: the agents are steadily developing their strategy and increasing rewards

When the algorithms developed strategies of docking and exiting the port, which for the experimental trials was only the shared policy condition (2) and single agent condition (1), a few cooperation patterns could be identified. One pattern was that the agents formed a queue and followed each other in both directions towards the berth and back towards the exit.

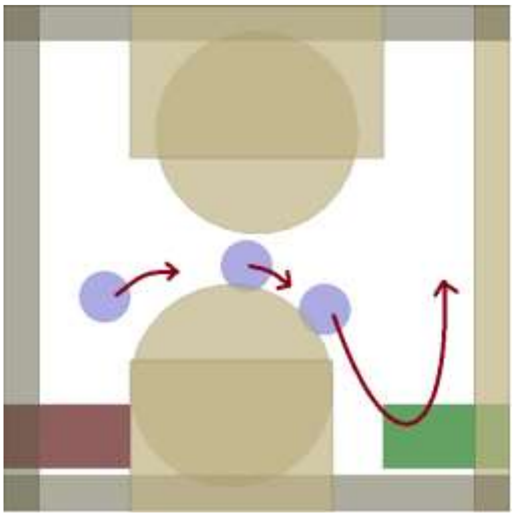


Figure 5. Agents form a queue and approach the berth

When going back to the exit, the agents who shared the policy often formed a circular motion both clockwise and counter-clockwise and queued back again to pass the narrow channel.

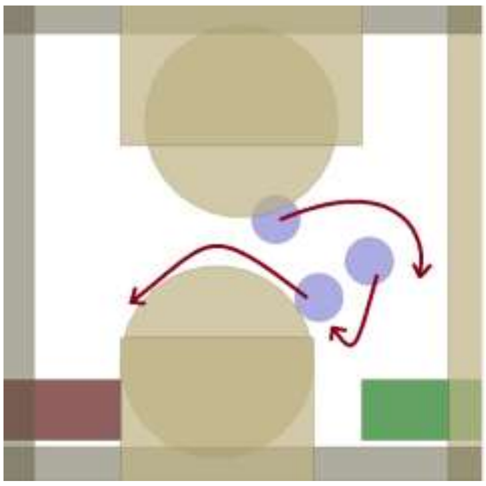


Figure 6. Agents circulate on the berth side.

A main difficulty was when the agents tried to travel through the narrow passage from both sides simultaneously. In this situation, there were two common scenarios. First, one of the agents would recognize the movement from the other side and retreat in order to give way to the coming agents. The other scenario was that both agents attempted to pass, which led to one of the agents being pushed away and the other agent passing. In this case, usually two agents on one side would push an agent on the opposite side. The latter scenario is considered a more primitive and less beneficial strategy, as both agents lose time and get punished for consecutive timestamps of the overlap.

For a further investigation of the results, reward plots and loss function plots can be viewed in Appendix A and B.

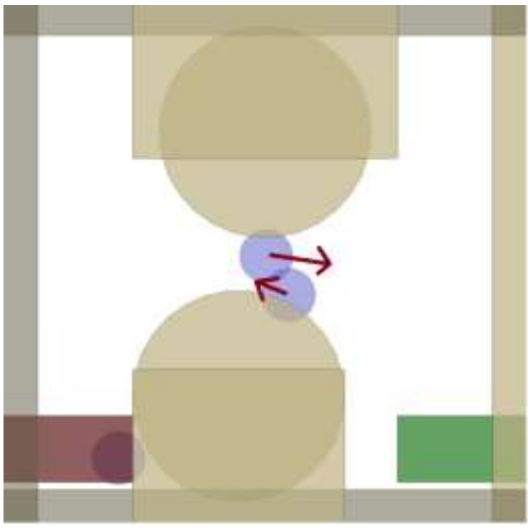


Figure 7. Collision of agents in the narrow passage

The collisions in the narrow passage happened more often in the single agent case (1) than in the shared policy case (2). In the single agent case (1) one network was updated based on summed rewards from all the vessels. Competition in such a case was the least expected, especially in comparison to the individual reward system of the shared policy condition (2). Vessels controlled by the same policies gave way to other vessels more often, even in the cases where the agent needed to visibly turn back in order to avoid collision (figure 8)

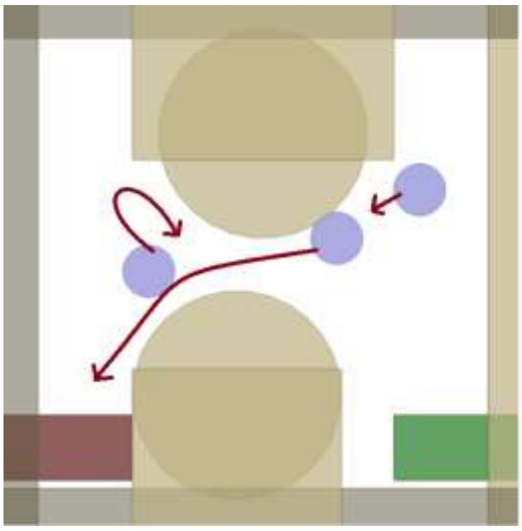


Figure 8. One agent recognizes two agents approaching and retreats to let them pass

An interesting strategy was developed by the single agent network (1), where one or two of the agents headed to the corners immediately as they were placed on the map, and only one agent was moving around freely, unloading the cargo, exiting, and repeating the cycle.

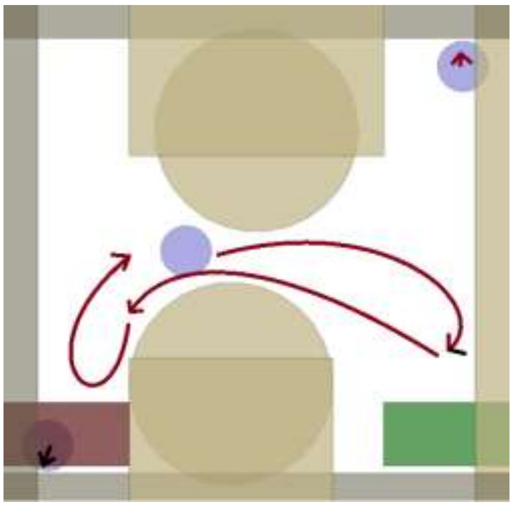


Figure 9. Two agents stay in the corners, while, one agent actively moves within the environment

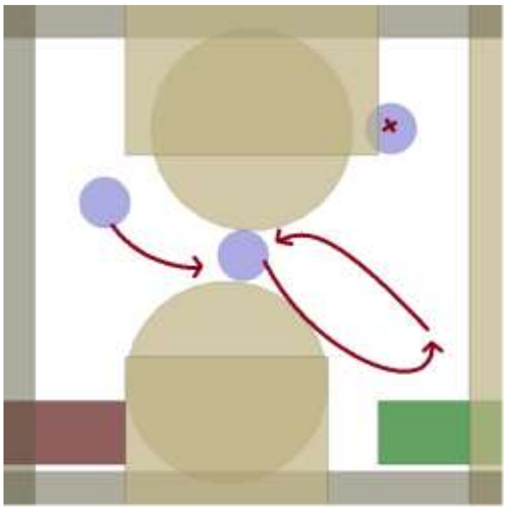


Figure 10. One agent (sacrifice) stays still while the others use the free space to find the berth and exit.

Discussion

The aim of the study was to propose a proof-of-concept application of a multi-agent reinforcement learning algorithm - Proximal Policy Optimization - to a simplified and abstracted simulation of a harbour environment. The focus of the study was on cooperative and competitive tendencies that arose in the learning process. Three conditions were tested in order to analyze the strategies. The first condition (1) was a single agent approach, with one policy network rewarded based on the sum of all rewards. The second condition (2) implemented the same policy network for each agent, with updates based on the experience of one agent per episode, and rewarded agents individually. The third condition (3) was a separate policy network for each agent, also rewarded individually. Overall, the most important result was the instability of learning and irregular development of strategies. This suggests that the algorithms need refinement, before being able to be systematically applied to multi-agent problems.

The validation phase in a simplified environment showed the first differences between the algorithms. The expected strategy was that the agents would occupy different landmarks. The conditions (1) and (3) developed this expected strategy, whereas agents sharing the policy parameters (2) increased their rewards over time, but failed to develop the expected strategy. Instead, both vessels attempted to occupy the same landmark from two sides (figure 3). This was not optimal as the agents did not manage to increase the distance from each other and gather the full reward. Instead, the agents which aimed at the same landmark were effectively competing for it and mutually decreasing each other’s rewards by getting closer to it.

The difference in strategy in the validation phase may be explained by the environment favouring differences between agents’ policies. In condition (1) and (3) the policy parameters per agent differed. Namely, in condition (3) the entire policy networks were different and in the case of the single agent approach (1), the last layer’s parameters were different for each agent. Therefore, agents in those conditions could be expected to diverge more in their learning than agents in condition (2). Different policies may have lead to choosing different preferences for landmarks. The environment favours the agents choosing different landmarks, therefore if initial differences in the policies arose in the separate policies condition, this was likely reinforced. The agents choosing different landmarks due to differences in policies could have been advantageous. Specifically, allowing for specialization of the agents seems to make it easier for the agents to divide limited resources by adopting different strategies.

Another possible explanation for suboptimal behaviour of the agents sharing a policy (2) are local optima. The environment favours proximity to the landmarks more than distance between agents (**equation for reward)**. When the closest landmark is occupied, shifting away from it temporarily decreases the rewards for the agents, until the second, unoccupied landmark is closer. It is thus possible that in conditions of insufficient exploration the agents would settle for this non-optimal strategy.

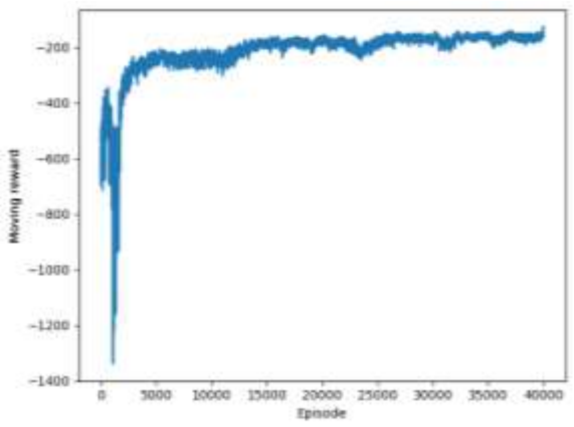


Figure 11. Moving average of rewards of a single agent in the validation scenario

```
EP_MAX = 40000
EP_LEN = 100
GAMMA = 0.999  # reward discount factor
A_LR = 0.001  # learning rate for actor
C_LR = 0.001  # learning rate for critic
MIN_BATCH_SIZE = 100  # minimum batch size for updating
PPO
UPDATE_STEP = 15  # loop update operation n-steps
EPSILON = 0.2  # for clipping surrogate objective
HIDDEN_LAYER_SIZE = 50
GAME = '2_agents_demo'
```

In the experimental runs, it was evident that the implementations proposed in this thesis were not fully suitable to solve the given challenge. The agents often settled for suboptimal behaviours and did not fully develop the expected strategy, which was to dock at the berth, go to the exit, and go to the berth again. Although the results of learning were unstable and difficult to replicate, the successful runs still yielded interesting insights into cooperative and competitive tendencies arising during the learning process. Once the agents developed an expected pattern of docking and exiting, their movements seemed strategic and formed recognizable behavioral patterns. The agents displayed coordination by either giving way to one another in the narrow passage between the berth and the exit, or by forming a queue and following each other. Their coordination patterns sometimes failed, causing collisions when both agents tried to enter the narrow passage at the same time. Contrary to expectation, more competitive behaviours and collisions were observed in the single agent condition (1) than in the shared policy condition (2). These results were unexpected as the single agent policy network was reinforced based on the sum of the rewards for individual agents, thus incentivising cooperation. The shared policy network, in contrast, was reinforced based on individual performance of the agents, which is expected to lead to more selfish behaviour of the agents. It is possible that the single agent network was less developed in comparison to the shared policy network, as the complexity of the single agent network was higher, and the rewards of each agent affected all outputs equally. Whereas in the shared policy condition the rewards were related to the specific outputs. The difference in complexity could have slowed down the single agent algorithm.

The agents often got stuck in the strategy of remaining stationary, which could have been a primitive, yet persistent local optimum. All agents used stochastic policies. Therefore, if the agents' policy was to move too little to reach the berth, the updates would decrease the probability of movement even further as it led to additional negative rewards for fuel consumption, and no positive rewards. In this way, the tendency to move would gradually decrease, the learning would converge and the system would become stable. Another factor that contributes to the stability of the non-moving policy, is that no movement leads to less exploration of the environment. The agents moving less are thus less likely to discover the high rewards coming from docking at the berth.

It is unlikely that the stationary optimum arose due to the agents' interaction, as very similar behaviour was found when only one agent was allowed to navigate the port, learning with the same algorithms as in the multi-agent setting. Given enough successful trajectories at the beginning of learning, the agent would find out and "remember" that docking and exiting the port is rewarded highly. Additionally, post-experiment trials without punishment for fuel seemed to be more successful in terms of building a full strategy.

Another suboptimal strategy that the agents settled on, was spreading out into corners. This may be attributed to avoidance of collisions learned too early. Decreasing the negative rewards for every timestep of overlap of safe-zones of vessels from 0.5 to 0.1 decreased the occurrences of stabilizing into this strategy. Interestingly enough, one or two agents moving towards corners can be considered a sacrificing approach, as discussed above. It is possible that immobilizing one or two agents may have been optimal for the remaining agent to move around, and decreased likelihood of punishment for collisions. Local optima had a different effect on conditions (1) and (3) in comparison to condition (2). In the former, if one agent settled for a local optimum, other agents could still develop more advanced strategies. In condition (2), on the other hand, all agents either fell into local optima or all agents displayed more advanced strategies.

There are a few ways to mitigate the problem of insufficient exploration. One way is through curriculum-based learning, where the agents learn small action-chunks in a simple environment, and move to more complex, aggregate tasks after achieving a satisfactory performance level (Sutton et al., 1999). Furthermore, OpenAI five (OpenAI, 2018) and dexterous robot arm (Andrychowicz et al., 2019) use randomisation of the starting states and noise injection. OpenAI five also uses "lane assignment" which in essence forces the agent to explore areas of the map by penalizing the agent for leaving them.

Because the local optima traps occur randomly and consistently, it may be possible to account for them by using a hybrid evolutionary and gradient strategy, where after every update step 10 parallel rollouts would be learning their next policy updates with only 5 highest performing ones being able to continue. In this way, the strategies that end up in dead ends would be systematically pruned and the successful strategies could branch out further. In evolutionary algorithms changes to the parameters are generated randomly according to gaussian distribution with current parameters as means (Salimans, 2018). In case of reinforcement learning, especially in the case of complex environments with stochastic policies, the changes to the policies could be generated by policy updates along the gradients as each trajectory leads to its own policy updates. This method would be applicable in cases were the environment is complex enough to offer many possible successful strategies, or, like in this case, where there are many local optima where the agent can get stuck and does not develop sophisticated strategies.

The issues with local optima may have been an effect of insufficient tuning of hyperparameters. For example, in trial runs, where rewards where of an order of -100 and -800, the losses of the critic were of an order of 1e4 at the beginning. The purpose of actor-critic architecture is to stabilize the loss functions. For the actor, which was compromised due to unstable critic's loss (see figure 12).

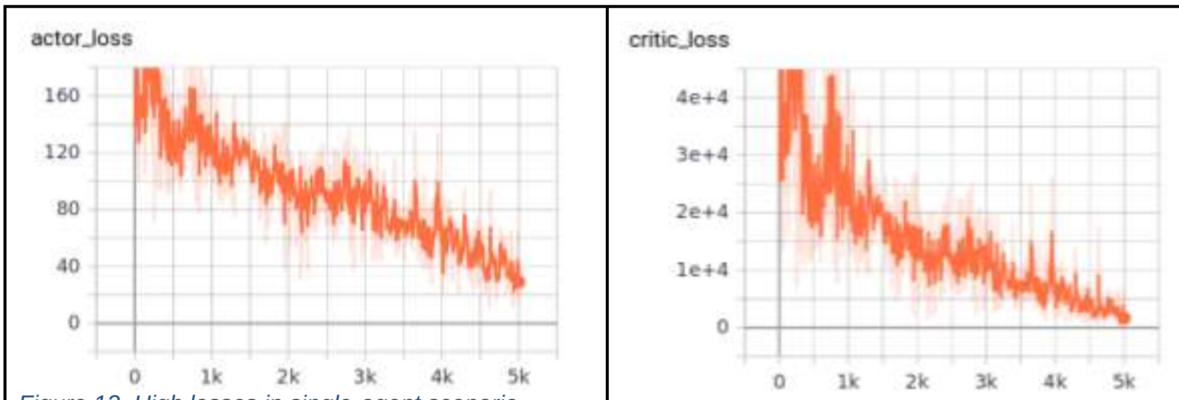


Figure 12. High losses in single-agent scenario.

Another noticeable issue with hyper-parameters was a steady rising of the critics loss after stabilisation (see figure 13).

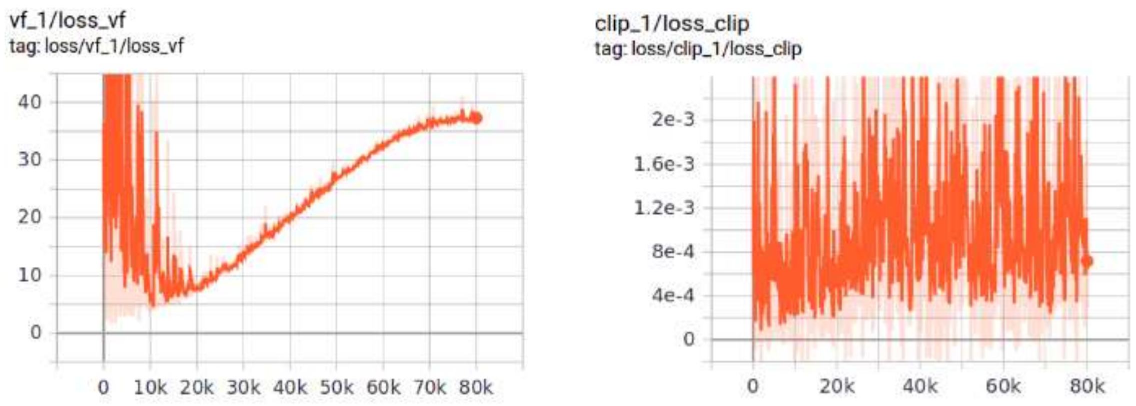


Figure 13. Growing critic's losses in separate policies scenario.

It was likely caused by underdevelopment of the critic in relation to the actor. The critic should learn faster than the actor, so it can stabilize actor's loss. It can also be done by setting a slightly higher learning rate for the critic than for the actor, which stabilized learning (Lowe et al., 2018).

Conclusion

This thesis was an exploratory study, seeking to test an application of reinforcement learning to a multi-agent problem, with the possible future application to modelling real world problems as a long term goal. The problem was formulated in terms of a Markov Decision Process. From the range of algorithms including DQN, actor-critic and DPG, Proximal Policy Optimization was chosen as the most suitable. The experimental setup consisted of three conditions, namely the single agent condition (1), the shared policy network condition (2) and the separate policies condition (3). Implementations of every condition was validated in the demo environment and tested in the experimental environment. The application of PPO itself needs refinement to stabilize learning. It is possible that a solution for limited exploration is necessary for the algorithms to produce consistent and stable results. Most likely, hyper-parameter tuning is also fruitful for stabilizing the results. Interesting avenues for future research are using evolutionary algorithms or deterministic policy to control the agents. Given the recent advances in scaling of applications of PPO, once the learning is stabilized, adding complexity to the environment may require only the addition of the computational power and more refined parallelization solutions.

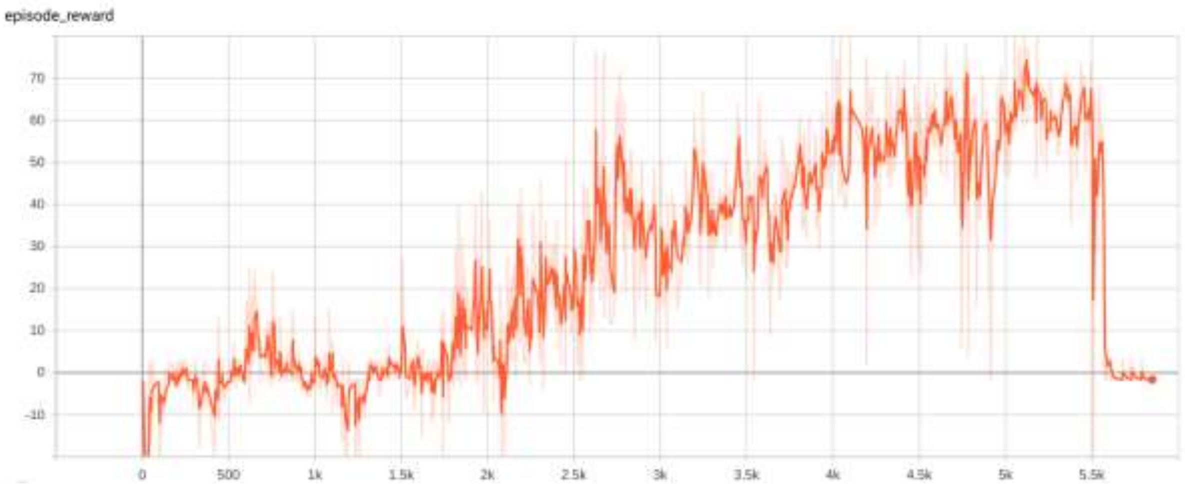
The thesis considered the question: What are the competitive and cooperative dynamics in the behaviour of agents trained with reinforcement learning in a single- vs. multi-agent environment of a simplified and abstracted modelled harbour? The competition and cooperation tendencies found were counterintuitive and did not provide significant ground for firm conclusions. More competition-like behaviours arose among agents controlled by a single network compared to when the agents were autonomous. The coordination between agents in the algorithm validation phase was seemingly helped by differences in network parameters, as specialization was favoured by the environment. As predicted, sacrifice-like behaviours arose, but only in single agent conditions. It needs to be validated, whether adjusting the reward system to penalize inequality would decrease the tendency of the algorithm to immobilize one agent and keep others moving. Overall, when coordinated behaviour patterns arose, the recognized patterns were: queuing, circulating movement and retreating to give way to other agents. The coordination patterns arose around the most problematic part of the environment, namely the narrow passage. The competitive behaviours were mostly agents pushing each other through the narrow passage, which happened mostly in the single agent scenario.

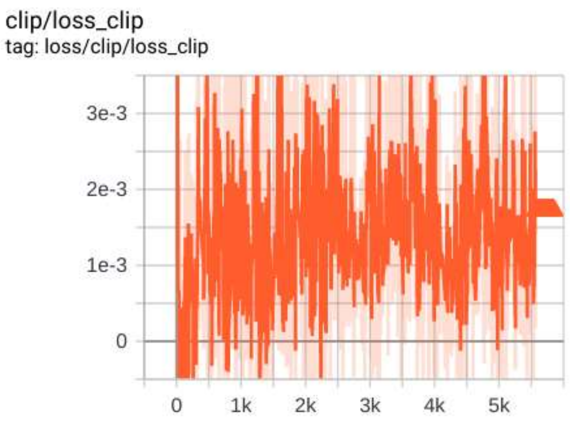
Part of the objective of the thesis was to build a boiler-plate for the future modelling of multi-agent problems in the context of complex industrial processes with multiple stake-holders. The environment and implementations of the algorithms were built in a way to realize this objective. Because the environment was built as an extension of an existing framework, it can easily by expanded upon or altered. Also, other implementations of multi-agent algorithms can be “plugged in” with generic gym API methods. The implementations of the algorithms themselves are also extendable to any environment built according to the OpenAI gym standards, given that the agents are homogenous , the observation space is continuous, the action space is discrete, and the environment is completely visible to all agents.

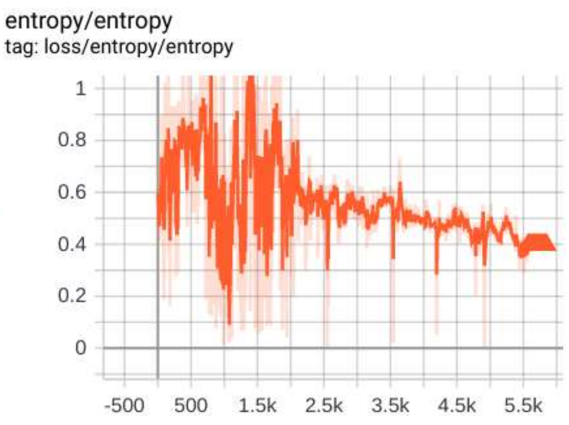
With the flexible basics of the framework, parts of a complex ecosystem, such as a maritime port, could possibly be abstracted and modelled, with agents showing recognizable behavioural patterns and interacting with one another in an understandable manner. In this way, reinforcement learning can help the decision making process in the industry and can aid in understanding the tendencies of the agents in the real world. Furthermore, given the RL agents’ tendency to come up with surprising strategies to deal with the constraints of the environment, given enough precision in modelling, they can possibly serve as an inspiration for people to come up with innovative solutions.

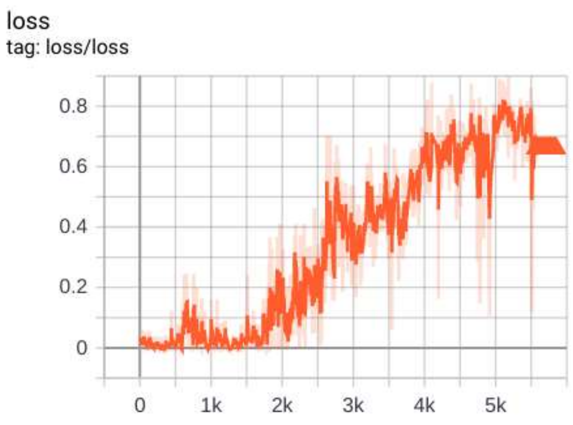
Appendix A Results graphs

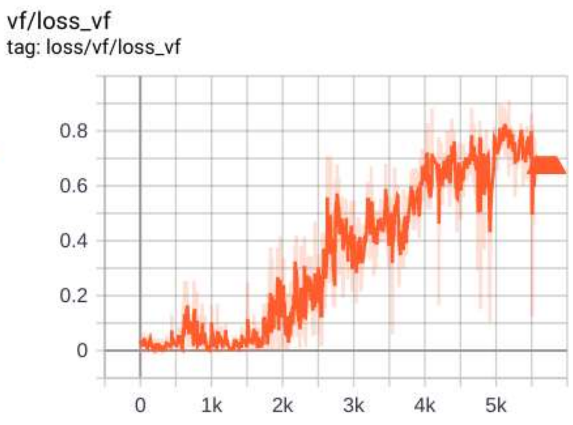
All agents sharing strategies



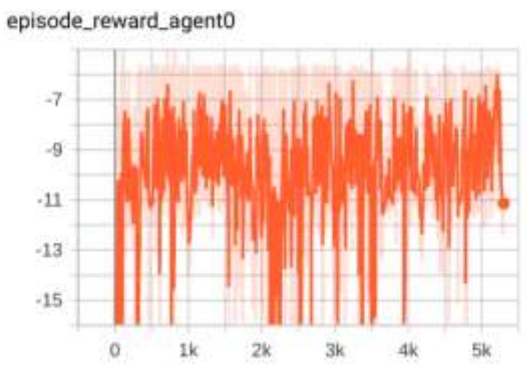


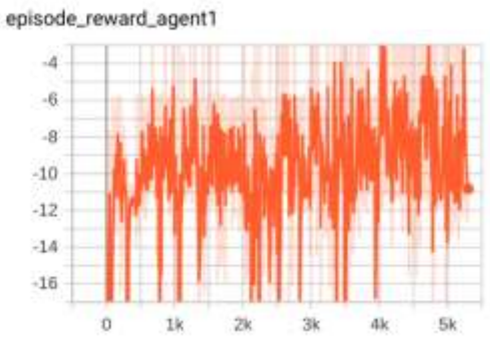


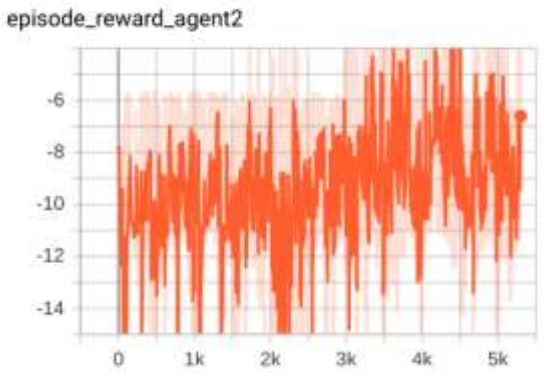


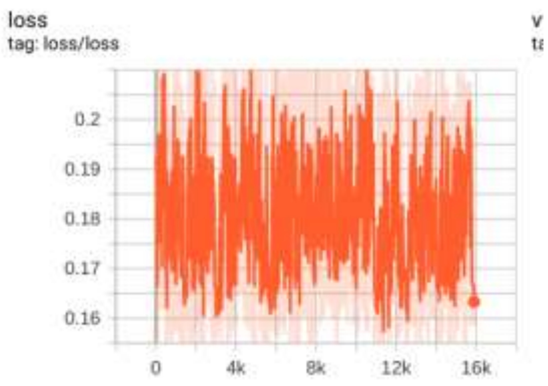


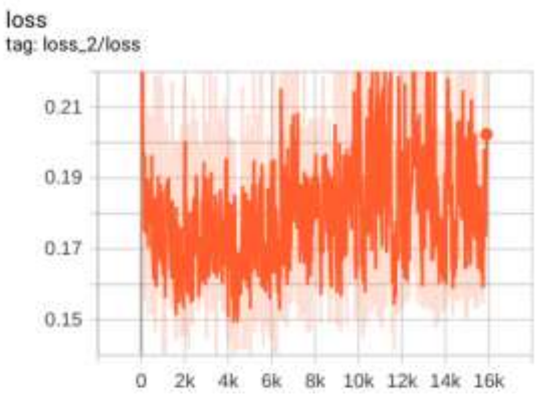
All agents with separate strategies: stuck in a local optimum

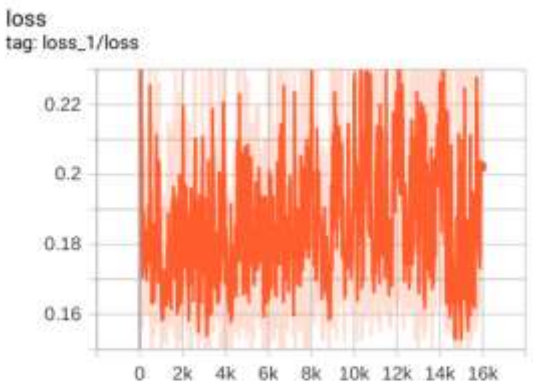




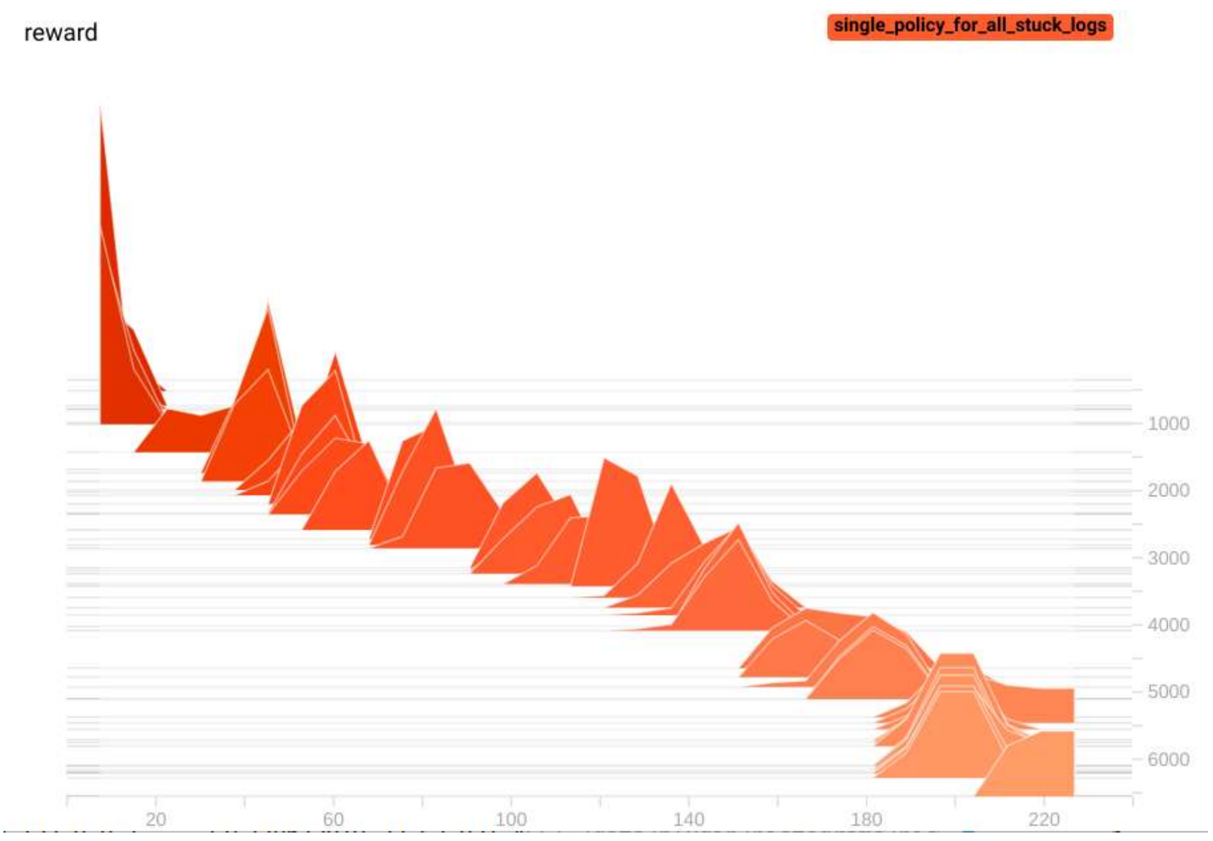


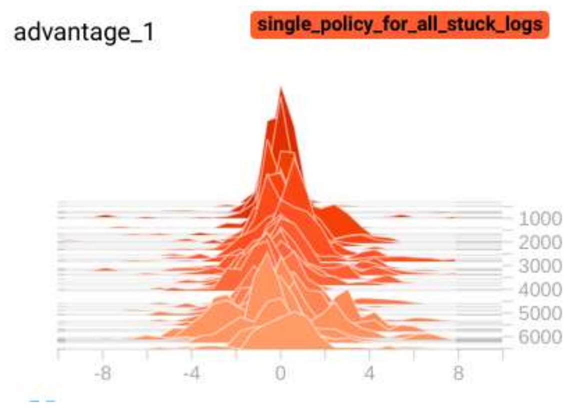


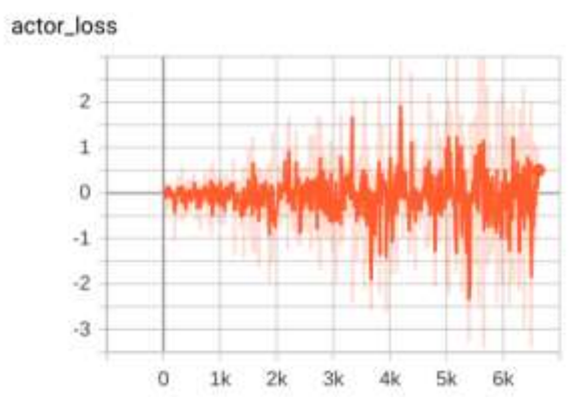




Single policy for all two agents stuck in local optima, one found a developed strategy

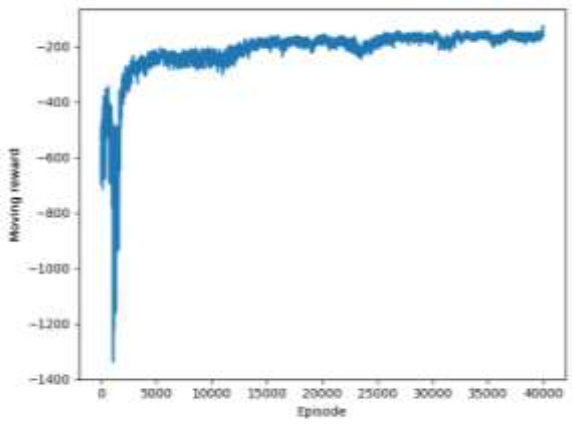






Appendix B

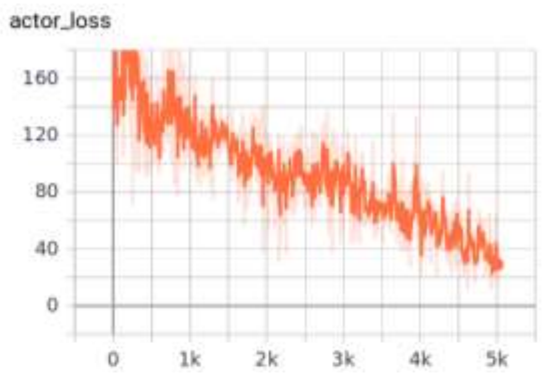
Visibly convergent results on 2_agents_demo scenario

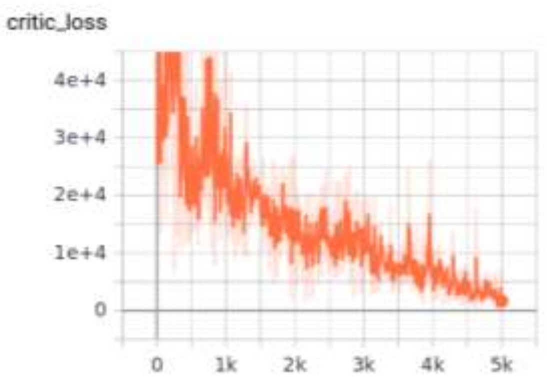


EP_MAX = 40000
EP_LEN = 100
GAMMA = 0.999 # reward discount factor
A_LR = 0.001 # learning rate for actor
C_LR = 0.001 # learning rate for critic
MIN_BATCH_SIZE = 100 # minimum batch size for updating PPO
UPDATE_STEP = 15 # loop update operation n-steps
EPSILON = 0.2 # for clipping surrogate objective
HIDDEN_LAYER_SIZE = 50
GAME = '2_agents_demo'

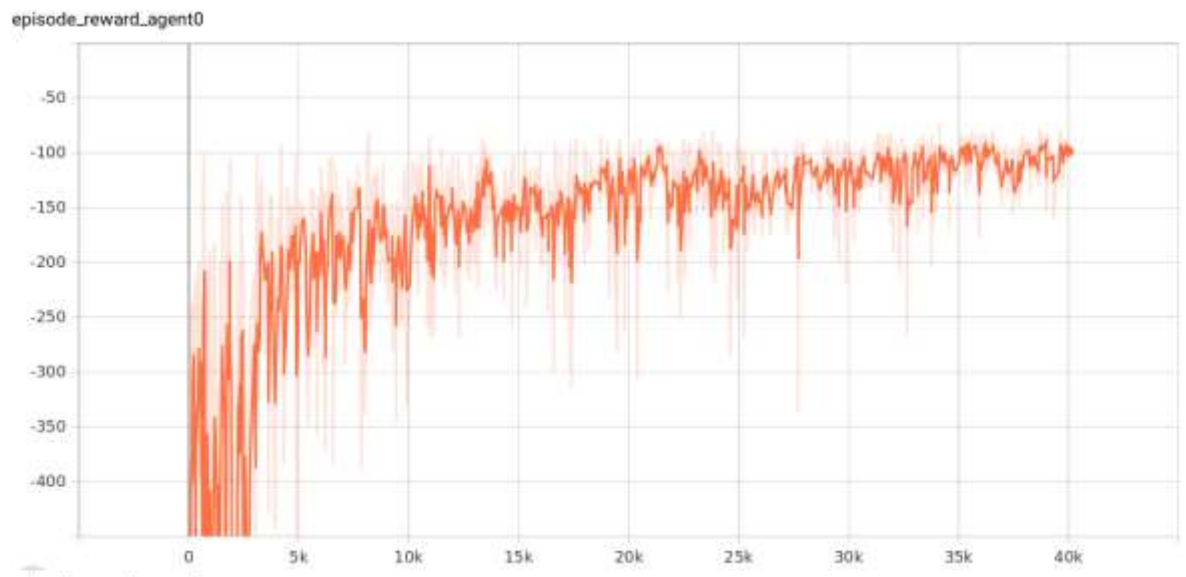
Discrete single PPO (film attached)
one agent with one output layer softmax on every 5 outputs (corresponding to an agent)

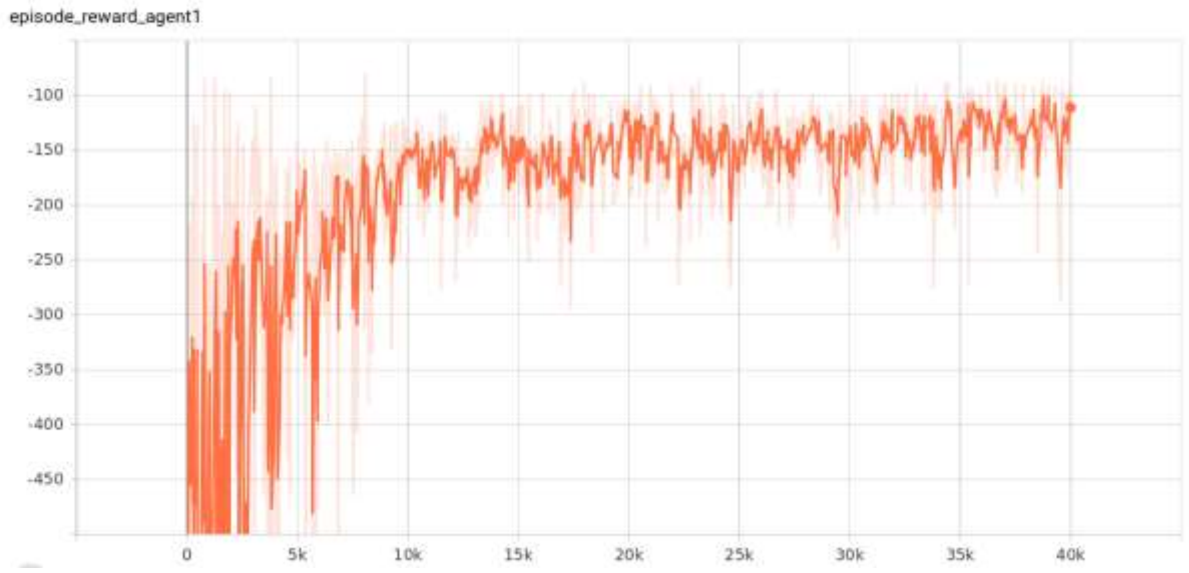
The agents moved smoothly towards their respective landmarks (movie attached)





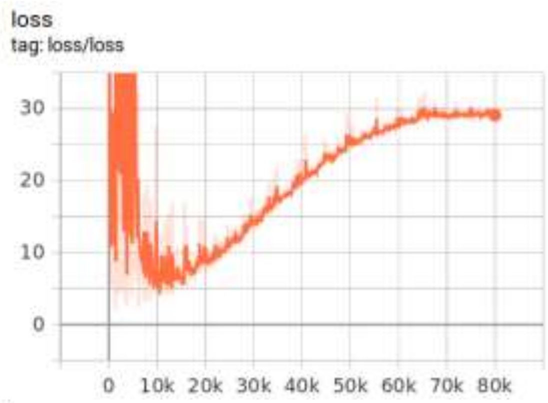
Separate policy for each agent

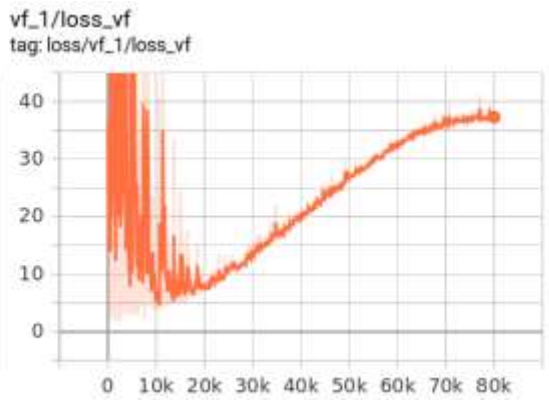


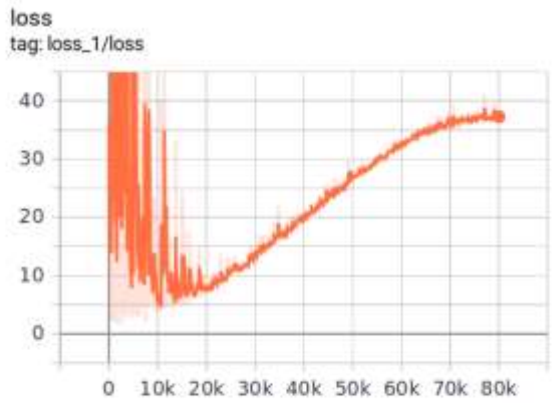


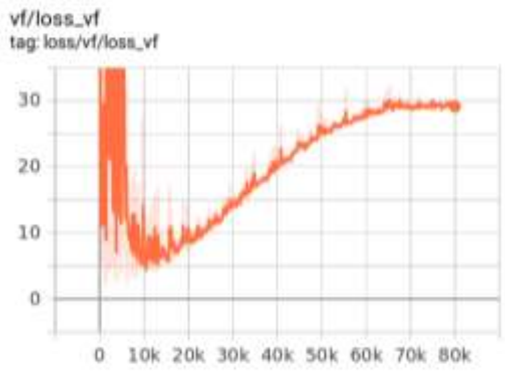
Agents are moving around and still have a lot of randomness to their movement but they clearly aim at dividing the target points between each other. See the video attached.

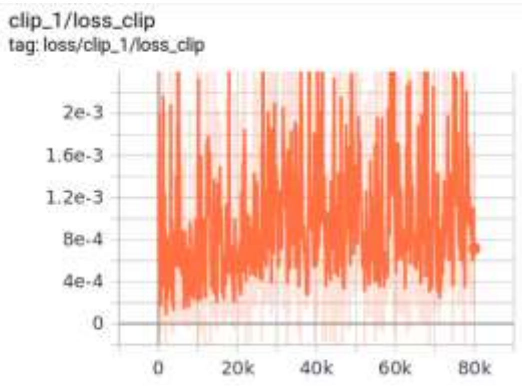
ITERATION = int(4e4)
GAMMA = 0.99
LR = 2e-5
EPISODE_LEN = 100movie
ENV_NAME = '2_agents_demo'

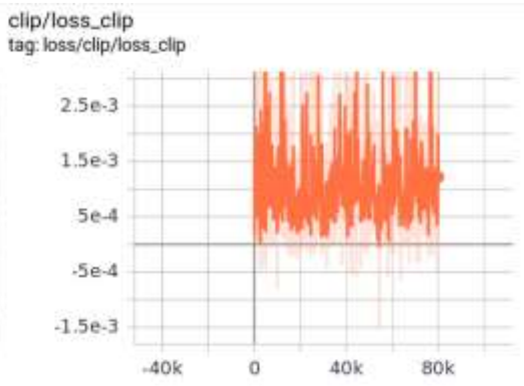


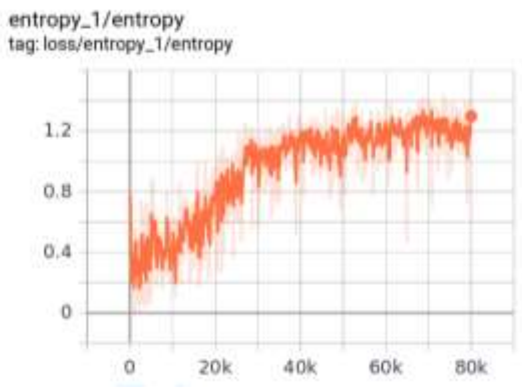


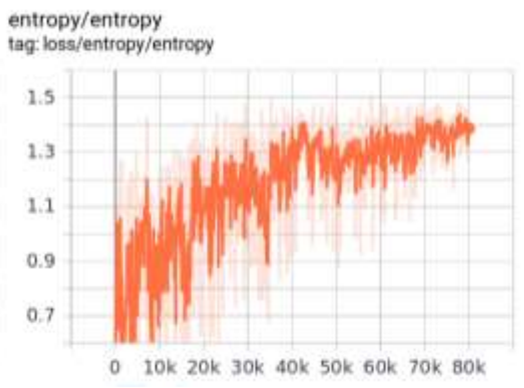








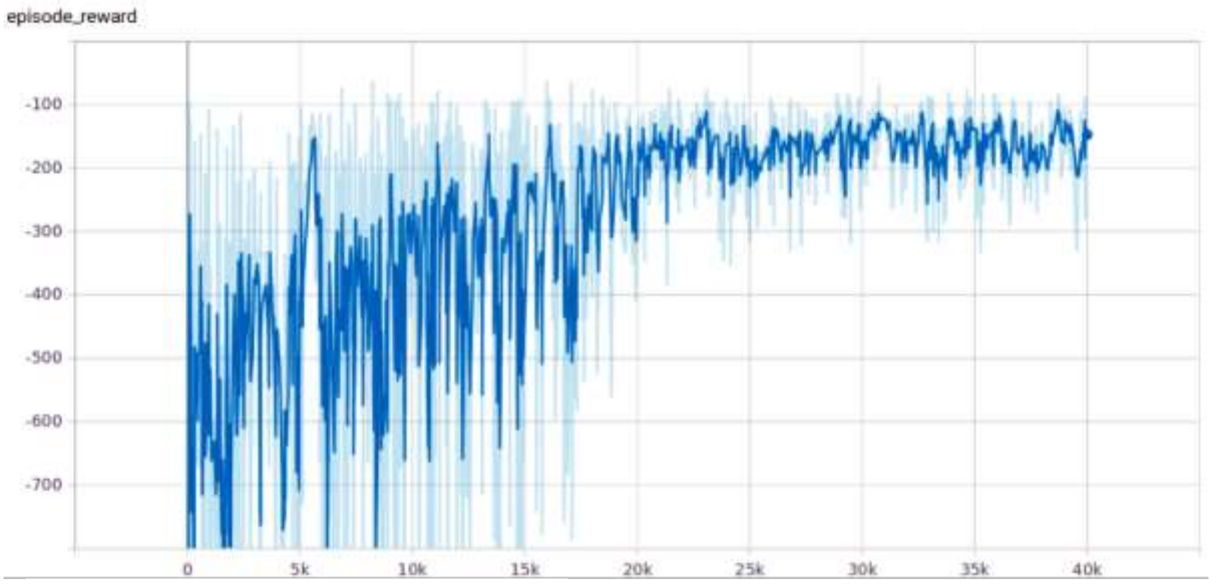


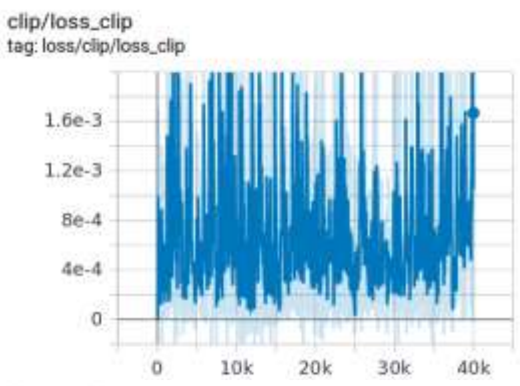


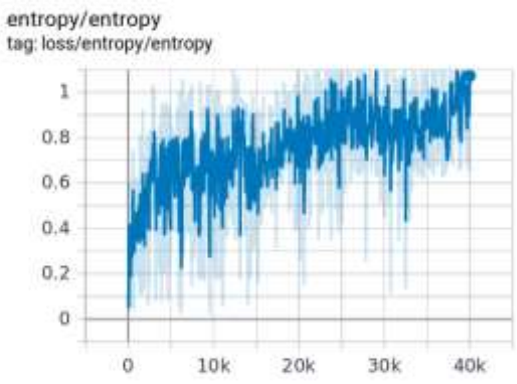
2 agents sharing a policy, updates are based on the experience on one agent per round:

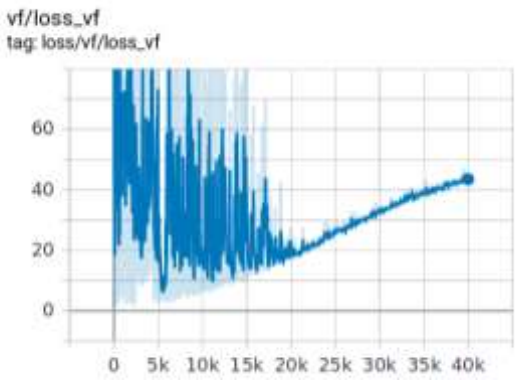
ITERATION = 4e4

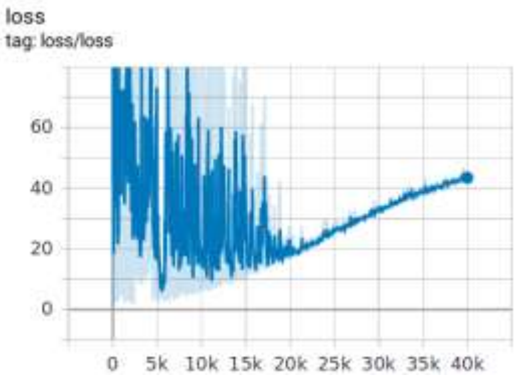
GAMMA = 0.999
EPISODE_LEN = 100
ENV_NAME = '2_agents_demo'





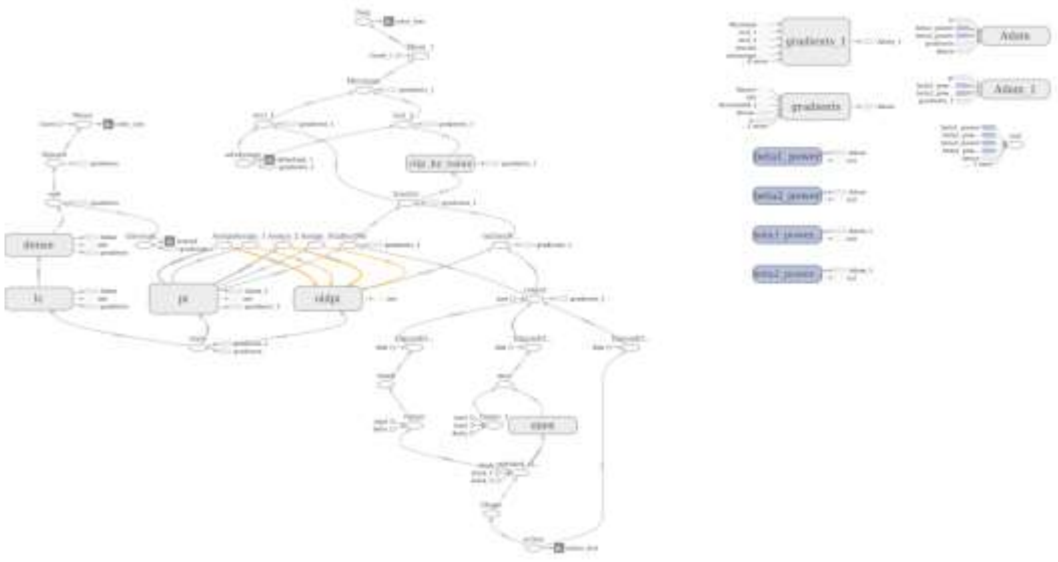






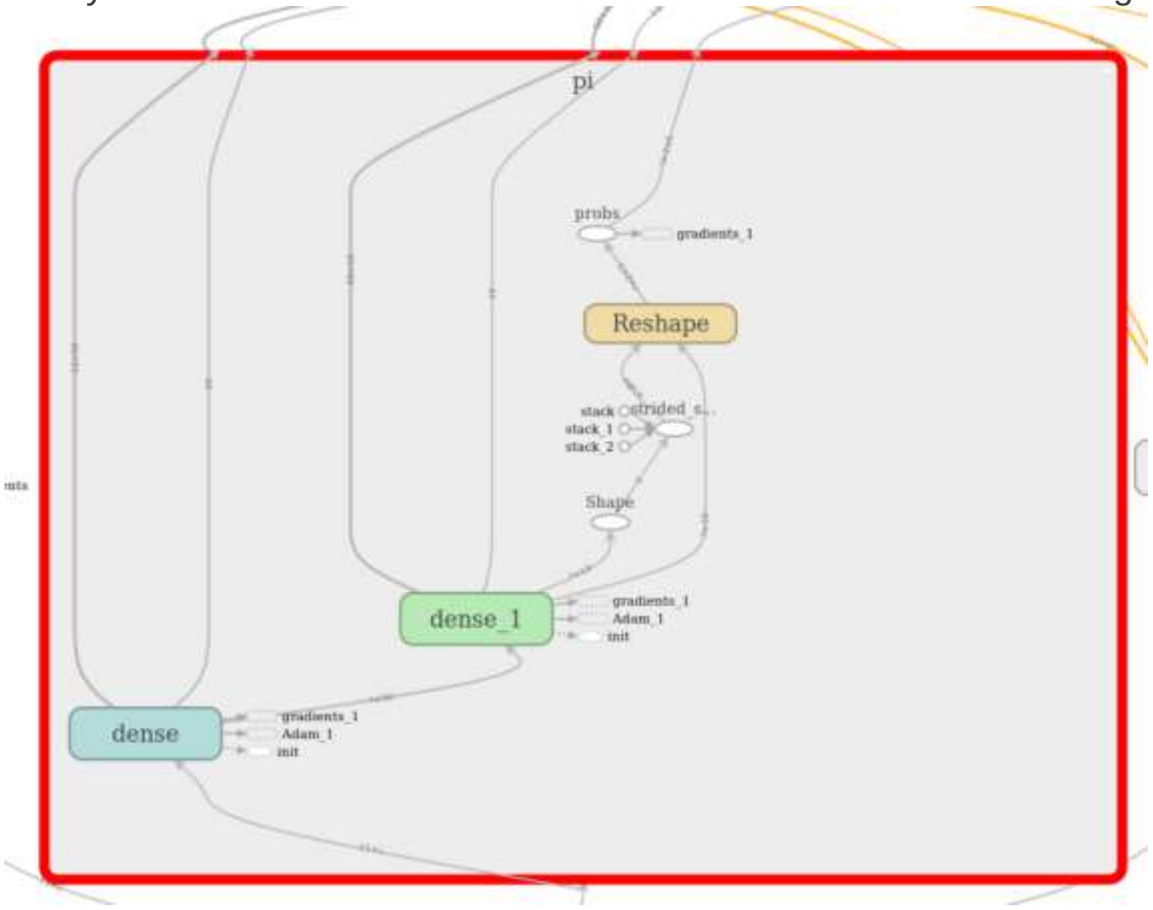
Appendix C, Computational graphs

The graphs are also available in tensorboard for all setups in attached repositories.

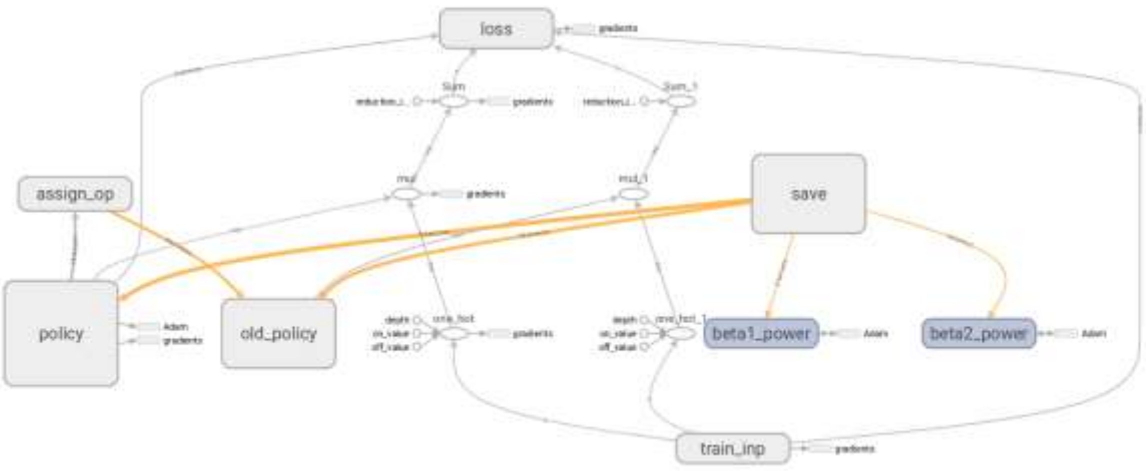


ı (1)

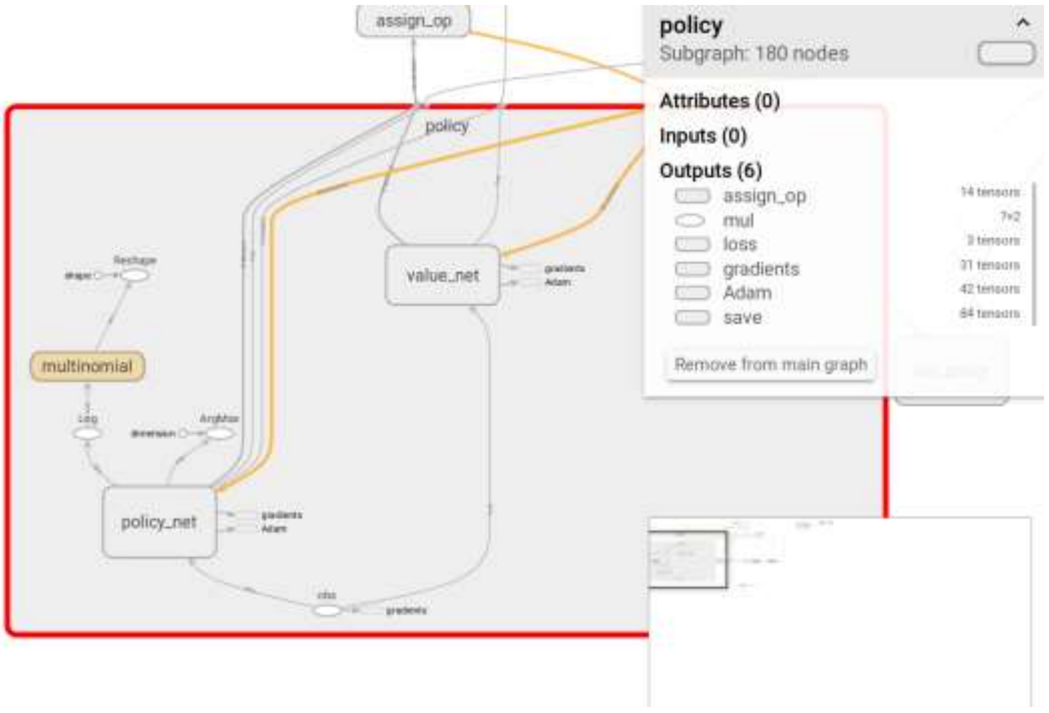
Policy graph



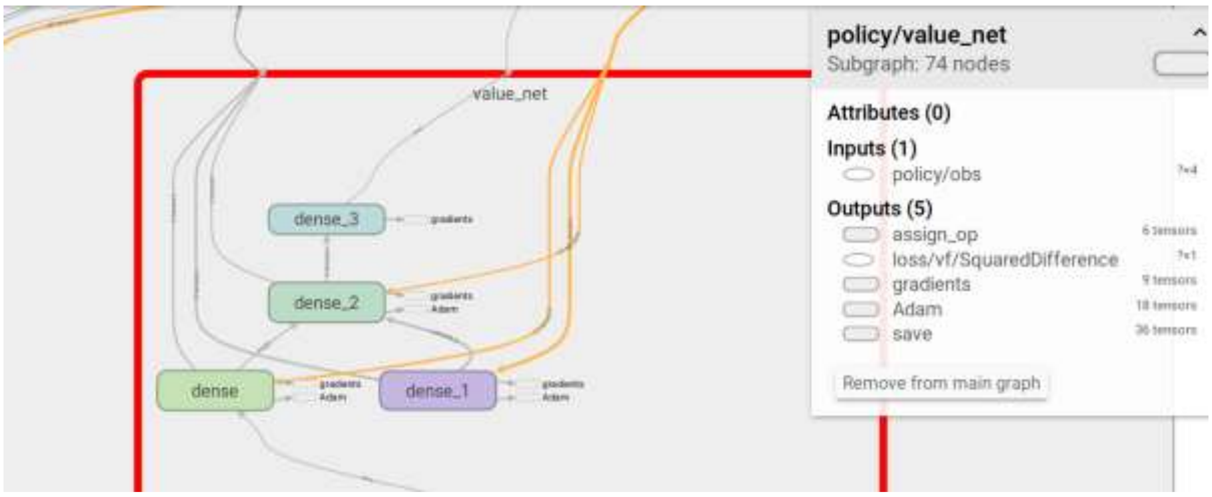
High level overview for shared policy (2) and separate policies (3) conditions



Actor-critic



Value network graph



policy/value_net
Subgraph: 74 nodes

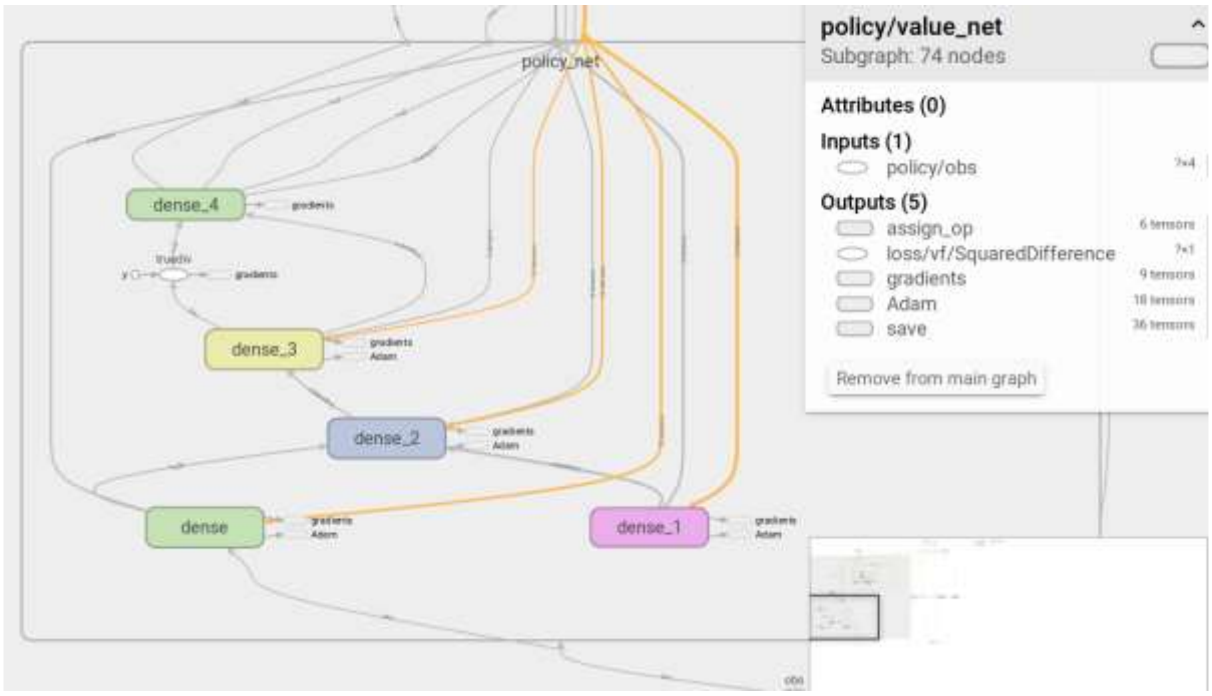
Attributes (0)

Inputs (1)
policy/obs ?×4

Outputs (5)
assign_op 6 tensors
loss/vf/SquaredDifference ?×1
gradients 9 tensors
Adam 18 tensors
save 36 tensors

Remove from main graph

Policy Network graph



policy/value_net
Subgraph: 74 nodes

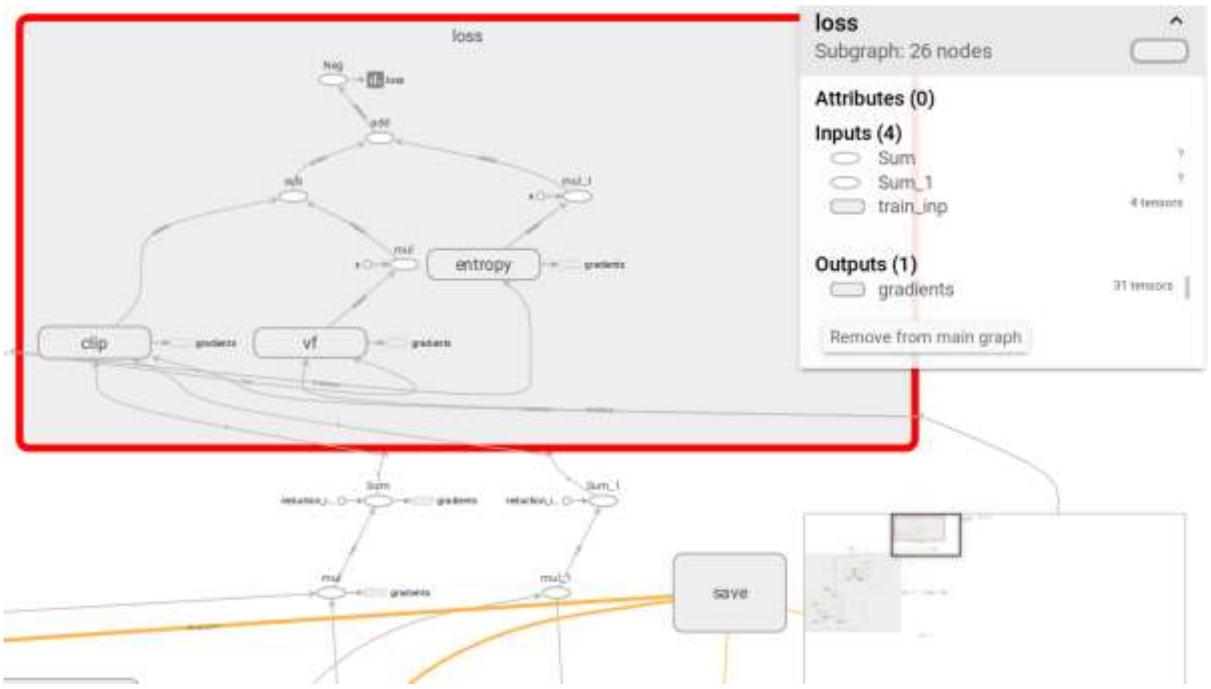
Attributes (0)

Inputs (1)
policy/obs ?×4

Outputs (5)
assign_op 6 tensors
loss/vf/SquaredDifference ?×1
gradients 9 tensors
Adam 18 tensors
save 36 tensors

Remove from main graph

Loss graph


loss
Neg
loss
add
sub
mul_1
mul
entropy
gradients
clip
gradients
vf
gradients
Sum
Sum_1
gradients
mul
mul_1
gradients
save
loss
Subgraph: 26 nodes
Attributes (0)
Inputs (4)
Sum
Sum_1
train_inp
4 tensors
Outputs (1)
gradients
31 tensors
Remove from main graph

Bibliography

Al-Rawi, H., Ng, M., & Yau, K.-L. (2015, 3 8). Application of reinforcement learning to routing in distributed wireless networks: a review. *Artificial Intelligence Review, 43*(3), 381-416.

Ascencio, L., González-Ramírez, R., Bearzotti, L., Smith, N., & Camacho-Vallejo, J. (2014). A collaborative supply chain management system for a maritime port logistics chain. *Journal of Applied Research and Technology, 12*(3), 444-458.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Openai, W. (2016). *OpenAI Gym.*

Bus¸oniu, L., Babuška, R., De Schutter, B., Bus¸oniu, L., Babuška, R., De Schutter, B., & Transac, I. (2010). *Multi-agent reinforcement learning: An overview * Multi-Agent Reinforcement Learning: An Overview Portions reprinted, with permission, from [20], 'A Comprehensive Survey of Multiagent Rein-forcement Learning', by.* Springer.

Carden, S. (2014). *Convergence of a Reinforcement Learning Algorithm in Continuous Domains Recommended Citation.*

Carrio, A., Sampedro, C., Rodriguez-Ramos, A., & Campoy, P. (2017, 8 14). A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. *Journal of Sensors, 2017*, 1-13.

Chiu, M., & Lin, G. (2004). Collaborative supply chain planning using the artificial neural network approach. *Journal of Manufacturing Technology Management, 15*(8), 787-796.

Conitzer, V., & Sandholm, T. (2007). AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Mach Learn, 67*, 23-43.

Devillé, S. (2011). Port of Rotterdam anchorages study. An occupancy evaluation using simulation. *TU Delft (Master's Thesis)*(October).

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2017). *Counterfactual Multi-Agent Policy Gradients.*

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P., Kohli, P., & Whiteson, S. (2017). *Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning.*

François-lavet, V., Henderson, P., Islam, R., Bellemare, M., François-lavet, V., Pineau, J., & Bellemare, M. (2018). An Introduction to Deep Reinforcement Learning.

Gao, L., & Cheng, Z. (2019). Learning with Collaborative Neural Network Group by Reflection.

Henesey, L., Notteboom, T., & Davidsson, P. (2004). *Agent-based simulation of stakeholders relations: An approach to sustainable port terminal management.*

Irannezhad, E., Hickman, M., & Prato, C. (2017). Modeling the Efficiency of a Port Community System as an Agent-based Process. *Procedia Computer Science, 109*, 917-922.

Kara, A., & Dogan, I. (2018, 1 1). Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications, 91*, 150-158.

Kingma, D., & Lei Ba, J. (2017). *Adam: a method for stochastic optimization.*

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., & Quillen, D. (2018, 4 12). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research, 37*(4-5), 421-436.

Li, Y. (2018). *DEEP REINFORCEMENT LEARNING: AN OVERVIEW.*

Littman, M. (1994). *Markov games as a framework for multi-agent reinforcement learning.*

Liviu Panait, & Sean Luke. (2005). Cooperative Muli-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems, 11*, 387-434.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Uc, P., Openai, B., & Openai, I. (2018). *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.*

Mannion, P., Duggan, J., & Howley, E. (2016). An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control. In P. Mannion, J. Duggan, & E. Howley, *Autonomic Road Transport Support Systems* (pp. 47-66). Cham: Springer International Publishing.

Mnih, V., Badia, A., Mirza, M., Graves, A., Lillicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016, 2 4). Asynchronous Methods for Deep Reinforcement Learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., . . . Hassabis, D. (2015). Human-level control through deep reinforcement learning.

Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Harley, T., Lillicrap, T., . . . Kavukcuoglu, K. (2016). *Asynchronous Methods for Deep Reinforcement Learning.*

Mordatch, I., & Abbeel, P. (2017). Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv preprint arXiv:1703.04908*.

Online, E., & Union, E. (2016). Information Guide Kosovo. (October), 1-8.

OpenAI. (2018). OpenAI Five.

Parolas, I. (2016). ETA prediction for containerships at the Port of Rotterdam using Machine Learning Techniques. 1-100.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R., Chen, X., . . . Andrychowicz, M. (2017, 6 6). Parameter Space Noise for Exploration.

Salimans, T., Ho, J., Chen, X., Sidor, S., & Openai, I. (2017). *Evolution Strategies as a Scalable Alternative to Reinforcement Learning.*

Sallab, A., Abdou, M., Perot, E., & Yogamani, S. (2017, 1 29). Deep Reinforcement Learning framework for Autonomous Driving. *Electronic Imaging, 2017*(19), 70-76.

Samuel, A. (1959, 7). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development, 3*(3), 210-229.

Schulman, J., Levine, S., Moritz, P., Jordan, M., & Abbeel, P. (2017). *Trust Region Policy Optimization.*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017, 7 20). Proximal Policy Optimization Algorithms.

Silver, D., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). *Deterministic Policy Gradient Algorithms.*

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., . . . Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*, 484-503.

Stratford, M. (2018). Welcome to Mendeley: Quick Start Guide.

Sutton, R., Mcallester, D., Singh, S., & Mansour, Y. (2000). *Policy Gradient Methods for Reinforcement Learning with Function Approximation.*

Sutton, R., Precup, D., & Singh, S. (1999). *Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning.*

Touati, A., Satija, H., Romoff, J., Pineau, J., & Vincent, P. (2018). *Randomized Value Functions via Multiplicative Normalizing Flows.*

Whiteson, S., Nl, A., Tanner, B., & Taylor, M. (2007). *Generalized Domains for Empirical Evaluations in Reinforcement Learning.*

Yi, D., Kim, S., & Kim, N. (2002). Combined modeling with multi-agent systems and simulation: Its application to harbor supply chain management. *Proceedings of the Annual Hawaii International Conference on System Sciences, 2002-Janua*(c), 1615-1624.