Deep Multi-Agent Reinforcement Learning in a Homogeneous Open Population

Roxana Rădulescu[1], Manon Legrand *, Kyriakos Efthymiadis[1], Diederik M. Roijers[1,2], and Ann Nowé[1]

[1] Vrije Universiteit Brussel
Brussels, Belgium
{roxana.radulescu, kyriakos.efthymiadis, ann.nowe}@vub.be
[2] Vrije Universiteit Amsterdam
Amsterdam, The Nederlands
d.m.roijers@vu.nl

Abstract. Advances in reinforcement learning research have recently produced agents that are competent, or sometimes exceed human performance, in complex tasks. Most interesting real world problems however, are not restricted to one agent, but instead deal with multiple agents acting in the same environment and have proven to be challenging tasks to solve. In this work we present a study on a homogeneous open population of agents modelled as a multi-agent reinforcement learning (MARL) system. Using the SimuLane highway traffic simulator as a test-bed we show experimentally that using a single-agent learnt policy to initialise the multi-agent scenario, which we then fine-tune to the task, out-performs agents that learn in the multi-agent setting from scratch. Specifically we contribute an open population MARL configuration, how to transfer knowledge from single- to a multi-agent setting and a training procedure for a homogeneous open population of agents.

Keywords: multi-agent systems · reinforcement learning · open population · highway traffic

1 Introduction

Recently, a great surge in reinforcement learning research has led to competent artificial agents for increasingly complex tasks, such as playing Atari games and Go [19, 25], and robotics [27]. Through these advances, solutions for many real-world problems are now within reach.

A feature of many real-world learning problems is that they require interactions between agents [1, 3, 7, 22] — both human agents, and increasingly also other artificial agents. Such a multi-agent aspect makes these problems particularly challenging. Specifically, artificial agents need to learn to anticipate the behaviour of other agents; typically, failing to do so greatly diminishes the performance. Furthermore, if agents are not fully cooperative (or zero-sum) [15, 32], learning algorithms may have trouble identifying a suitable equilibrium policy.

* Contribution done during the master thesis studies at the Vrije Universiteit Brussel.

In this paper, we study the behaviour of a population of homogeneous learning agents that is continuously changing, as new agents are entering the system, possibly at different rates than the ones exiting. We characterize this problem as an *open population of homogeneous learning agents*. To tackle this scenario, the agents are sharing the same policy and are learning simultaneously, in the presence of other agents (e.g., human drivers) modelled as part of the environment. The policy-sharing aspect has the advantage that only one policy needs to be learned, and that the agents (both human and artificial) can more easily anticipate the behaviour of the other learning agents. Furthermore, we believe that in many situations humans will be able to more easily predict what the agents will do, which we believe to be a desirable aspect in many systems that also include human agents.

We situate our study in (simulated) traffic on a highway. Not only is this a suitable domain for studying homogeneous agents that share their policy — a car manufacturer probably will want to sell autonomous cars with one policy only — but it also illustrates why we would like to be able to learn around other agents, i.e., humans, and why predictable agents would be desirable. We propose a new simulator for highway traffic, which we briefly discuss in Section 3. For a more extensive discussion of the simulator, please refer to [11] and [12].

To learn a policy for our artificial agents, we mainly build upon the Deep Q-learning [18, 19] approach. First, we apply this algorithm to learn a policy for a single agent (Section 5.1), and test how suitable this policy is for the multi-agent setting. Second, we tackle the multi-agent scenario through the centralized learning decentralized execution paradigm (in which homogeneous learning agents sample simultaneously from an environment to train the same shared neural network that represents their policy) (Section 5.2). As expected, this training procedure is significantly slower than single-agent learning. Our key insight is that multi-agent learning can be sped up by first training a single-agent policy, and then using this single-agent policy as a starting point for a homogeneous set of agents. We integrate this key insight into our multi-agent learning algorithm, completing our main contribution. We show experimentally, in Section 5.3, that homogeneous multi-agent learning via reuse of a policy trained for a single agent yield better results and is much faster than learning from scratch.

Our contributions for this work can be summarized as follows: i) we successfully demonstrate how a single agent policy can be learned in our problem setting and then demonstrate the need for training a model while several autonomous agents are present in the environment, as the single agent one is unable to perform well in a multi-agent scenario; ii) we demonstrate two methods for sharing knowledge between agents: either by learning a shared policy using the experiences of all the agents in an open population, or by additionally transferring knowledge from a single-agent setting to a multi-agent one, in a complex problem setting.

2 Background

In Reinforcement Learning (RL) [28] an agent learns to solve a task by interacting with the environment, using a numerical reward signal as guidance. A common class of RL techniques are value-based algorithms. In value-based algorithms, the goal typically is to find an estimation of the action-value function Q defined as the expected sum of rewards discounted at each time step t by a factor γ, when acting according to a policy $\pi = P(a|s)$, defining the probability of any action a in a state s:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_0 = s, a_0 = a, \pi\right].$$

The optimal value function is then: $Q^*(s,a) = max_{\pi} Q^{\pi}(s,a)$.

Q-learning [31] is a popular value-based RL algorithm, in which the value function is iteratively updated to optimize this expected long-term reward, by bootstrapping the estimated value of the next state. Specifically, after a transition from state s to s', through action a, Q-learning performs the following update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \ ,$$

where α is the learning rate, γ is the discount factor and r is the immediate reward received from the environment.

In this paper, we build upon Deep Q-learning, which relies on a Deep Q-network (DQN) [18], i.e., a neural network as a function approximator to estimate the action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$, rather than a tabular representation of the value function. A key aspect for having a stable learning process when introducing this non-linear function approximator is keeping a secondary target network, $Q(s,a;\theta^-)$. In the case of DQN, it suffices to update the target network's parameters every τ steps with the online network [30]. The parameters θ are learned by performing one-step gradient descent updates according to the following loss function at iteration i:

$$L_i(\theta_i) = \mathbb{E}\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2$$

DQN is then used together with *experience replay memory* (ERM) as detailed in [19] to further stabilize and improve the learning process. While an agent interacts with its environment, it gathers experiences of the form $< s_t, a_t, s_{t+1}, r_t >$ at each time step t. In the original Q-learning algorithm [31], each experience is used only once to update the Q-value function. This can be considered wasteful, as certain events may occur with a low frequency. Reusing experiences is the key idea behind the experience replay mechanism [14]; the agent keeps all the past experiences in memory and can then reuse them to update its Q-value function. Additionally, for Deep Q-learning, experience replay plays an important role in breaking the bias induced by the sequentiality between learning samples.

For the action selection strategy we use ϵ-greedy, choosing a random action with a probability ϵ and the action with the highest Q-value for the rest of the time.

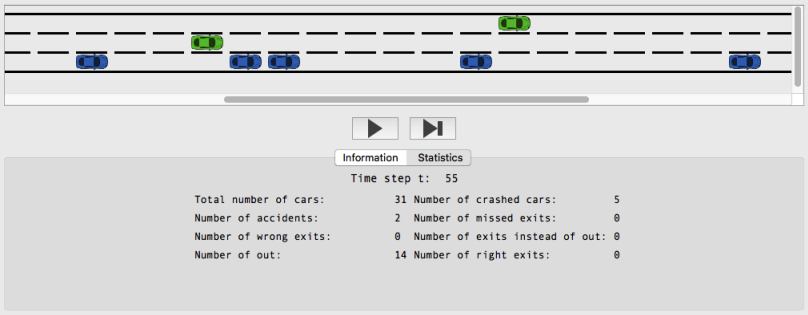

Fig. 1: Graphical Interface of SimuLane. The human drivers are represented in blue, while the autonomous agents are in green.

3 Problem Setting

SimuLane [12] is a highway traffic simulator modelling both a single- and multi-agent reinforcement learning environment. It offers a discrete representation of a highway section, where each lane has a preferred speed, reflecting the idea that cars' speed usually increases from right to left on highway lanes. Figure 1 presents a screen-shot of SimuLane's graphical user interface, in which both humans drivers (in blue) and autonomous agents (in green) are interacting in the environment.

Another important component of the simulator are the human drivers. They act according to a behavioural model that can be summarized by three rules: i) "drivers must not crash", ii) "drivers must reach their desired speed" and iii) "drivers must respect the lanes' speeds".

3.1 State and Action Space

Each agent perceives at each time step a state containing only local information, i.e., in accordance to its field of view. The agent is able to observe the presence of other cars, their speed, its own speed and location (i.e., current lane), together with its goal. Our agents' goal is to safely traverse the simulated highway segment. The speed of the cars can take integer values in the interval $[0, 3]$. A visual representation of this car-centric input space can be seen in Figure 2.

An action is a tuple of the form $<$*acceleration, direction*$<$, where the acceleration value is an integer in the range $[-2, 2]$, while the direction can take values from the set {*forward, left, right*}. Our action space is formed by all the possible $<$*acceleration, direction*$>$ combinations and thus has a size of 15.

3.2 Parameters

Traffic density defines a per lane probability for a car to enter the highway at each time step. The first cell of each lane is thus updated according to the traffic

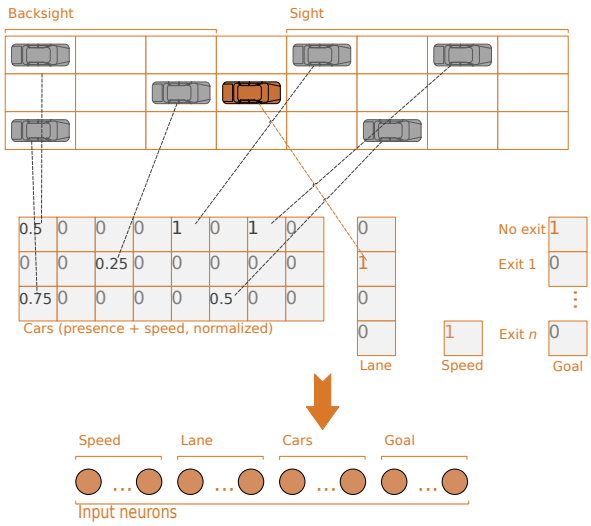

Fig. 2: State space and network inputs for each learning agent in the SimuLane environment. Everything is normalized between $[0, 1]$.

density at every step. For the multi-agent scenario, the *ratio* of autonomous cars is the probability that a new driver entering the highway is an autonomous car.

The size of the highway section can also be configured by setting the *number of lanes* and *cells*. Additionally, the *irrationality* defines the probability for a human driver to choose a random action, allowing one to simulate the fact that a driver can make mistakes.

For the single-agent case we define the maximum number of *steps* the agent can spend on the highway in order to avoid an infinite episode. If the agent is still present on the highway beyond this number of time-steps his outcome will be set to overtime and the episode is terminated. For the multi-agent scenario we fix each episode to a certain number of steps. The *reward function* is also fully configurable for each possible outcome (i.e., crash, overtime and achieved goal).

4 Methods

We are considering here three learning settings: *single-agent*, where only one autonomous car is present on the highway at all times; *multi-agent learning from scratch*, where multiple autonomous cars sharing the same learning model are trained from scratch; and *multi-agent initialized with a single-agent network* as a starting point for the training. We note that for each of the three settings, the human drivers are also present in the environment at all times.

4.1 From Single to Multi-Agent Learning

In a single-agent setting an agent has to learn to respond well to the environment. While this environment may contain other agents, these can generally be assumed to behave according to stationary rules. In our problem setting, we generate such non-learning agents on-the-fly, following a stationary distribution over policies such agents may have.

Our goal for the multi-agent setting is to learn one agent-centric policy, uniform over the entire population. As our environment consists in a fixed highway segment through which different agents pass at various rates, we are dealing with an open population. We are learning a policy, by having all agents that pass through the environment contribute, with the experiences they gather, to a central experience replay memory. To this end we employ Deep Q-learning coupled with a *centralized training with decentralized execution* framework, i.e., during training the agents add their experiences to the same experience replay buffer and only one network is trained for the entire population. At execution time, each agent receives its own network copy and acts according to its own local input. This approach has two main potential advantages: i) speeding up the training process due to the information sharing between all the agents, ii) the policy learned will be uniform across the entire population outputting a more predictable behaviour for all the agents. This second point is highly important, as we aim to be able to learn in non-stationary environments.

The multi-agent setting is thus considerably harder, i.e., an agent not only has to respond well to the environment, but also to other learning agents – which in our case are running the same, but evolving, policy. This means that the environment is non-stationary, due to these concurrently learning agents. This induces two important challenges: 1) the experience tuples in the experience replay buffer may not resemble the current situation well enough any more, as the learning agent's policies have changed and 2) as a result of this the agents may learn a policy that responds to overly erratic agents, as in the beginning, the learning agents do behave erratically. As an initial approach to mitigate this situation we shrink the experience replay memory [8] until we obtained a stable learning process. As an additional helping factor for learning in a multi-agent environment we employ a dropout mask [26] before the output layer. While we expect this approach to mitigate part of the above-stated problems, we still expect the performance of the agents trained in this manner to be significantly worse than in the single-agent case.

4.2 Single to Multi-Agent Knowledge Transfer

In our highway traffic problem setting there are two main aspects that have to be learned: i) driving in a highway environment populated with human drivers, ii) dealing with the behaviour of other autonomous driving agents. Transfer from the single-agent case to the multi-agent case leverages knowledge about the former from a model trained in a single agent setting, which takes less time to train, and then further tunes this network in order to include in the policy a

behaviour adapted for the latter. As we show empirically in Section 5, this both reduces the overall training time until convergence with respect to multi-agent network trained from scratch, while the performance is significantly higher.

The fine-tuning procedure we adopt for this work consists in freezing all the weights of the single-agent network model we are transferring to the multi-agent scenario, with the exception of the ones between the last hidden layer and the output. This procedure ensures sufficient exploration in the beginning to escape the risk of running into a local optimum early, while keeping the useful information about the environment encoded in the earlier layers [21]. Additionally, we decrease the learning rate by a factor of 10 in order to stabilize the learning process and avoid unlearning the transferred knowledge.

5 Experiments

In order to test the performance of our algorithms for multi-agent reinforcement learning in open homogeneous populations, we run them on a highway traffic setting, using the settings for our algorithms and our SimuLane simulator stated below. We measure performance of our algorithms as the fraction of times the agents have successfully traversed the highway segment, and thus reached their goal, until each respective episode. We additionally report the fraction of crashes and overtimes.

DQN settings – We use a DQN function approximator by means of a feed forward network composed of two fully connected layers of size 100 (with a *relu* activation function) and 50 (followed by a *tanh* activation). There are 42 inputs and 15 outputs. This network architecture is kept the same throughout all our experiments in order to ensure a proper comparison between all the cases (we argue that we look from a policy maker's perspective and aim to learn a car-centred policy in all the settings). The optimizer used is *adagrad* [5] with a learning rate of 0.01. We set two different update intervals for the online and target networks, 10 and 50 respectively, while the batch size for the online network update is set to 16. The experience replay memory has a size of 10 000.

SimuLane settings – Throughout all our experiments we keep the *traffic density* at a constant 0.25 value, while the highway segment we consider has 3 lanes, each consisting of 40 cells. For the single agent scenario, the learner has 70 time steps to traverse the highway, otherwise the output is set to overtime. For the multi-agent experiments we keep the length of an episode at 160 steps.

Q-learning settings – The Q-learning parameters are set as follows: ϵ is 0.1, γ 0.9, while the reward function is set to: goal 1, crash -1, no-speed -0.01, overtime -0.4. The small negative reward for each time step in which the car speed is zero was introduced to encourage agents to avoid the overtime outcome.

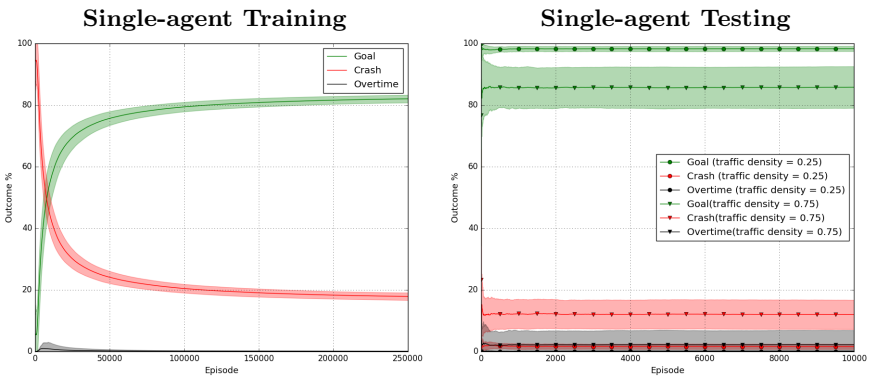

Fig. 3: Single-agent model performance in training (left) and testing (right). The agent keeps a constant ϵ exploration rate of 0.1 for the entire training process. The ϵ is set to 0 during testing. In the testing scenarios, we notice that the further we go away from the value the model was trained under (i.e., 0.25), the bigger the drop in performance gets.

5.1 Single-Agent

As a baseline, we first train a policy using only a single agent in the environment, and subsequently test how well that policy performs if it is employed by multiple agents in a multi-agent scenario.

Figure 3 (left) presents the training for the single agent case over 250 000 episodes. The results are averages over 30 runs and we plot the mean and standard deviation for each outcome. We note that for the entire training period the agent keeps a constant ϵ exploration factor. We notice that in training, the agent manages to converge to a performance of over 80%. Additionally, the training curves do not exhibit a high variance, outputting a stable expected performance.

Now that we have a trained network model for the single agent setting, we can test this policy in various scenarios. We begin by looking at how the model behaves in different *traffic densities*. We illustrate the outcomes of our simulations in Figure 3 (right), under two different traffic density values: 0.25 (the value used during training) and 0.75[1]. We average the results over the 30 trained models. One can notice that the further we go away from the value the model was trained under, the bigger the drop in performance will get. This result indicates that, in order to have a better performing model, one should also look at training in a multi-task fashion [6, 23], to allow the model to adapt to a bigger variety of environments (at least in terms of traffic density).

A first step towards our goal of having a policy that is able to handle a multi-agent highway scenario is to see how our single agent model performs when multiple learners are present in the environment. We run simulations varying

[1] We note that we run simulations for other intermediate values, but only show here the two extremes, for the sake of graph legibility.

Fig. 4: Performance of the single agent model in a multi-agent setting with various ratios of autonomous cars present on the highway. The drop in performance is a clear indication for the need of a model trained in a multi-agent setting, that allows the policy to also incorporate a response to the behaviour of other autonomous agents.

the ratio of autonomous drivers on the highway between 0 and 1, with steps of 0.01. Every agent entering the highway is using the model from the single agent setting in order to act in the environment. The simulations for each ratio last for 500 steps and are repeated 100 times. We plot the mean and standard deviation for each result.

Figure 4 presents the results of these simulations. Notice the downwards trend along the ratio axis in the left subplot, which clearly indicates that a *simple transfer of the single agent policy to a multi-agent setting is not sufficient*. The more autonomous agents are on the highway, the more difficult it is to cope and behave in the environment. On the right side we also take a look at the number of cars entering the highway segment versus the number of cars exiting. We notice also a steep decrease in these values, signalling an increased presence of crashes and stopped cars on the highway, bringing the traffic close to a standstill towards the higher values of the ratio.

5.2 Multi-Agent From Scratch

In addition to our single-agent-trained policy as a baseline for multi-agent settings, we also try to train a multi-agent policy from scratch. Please note that for the multi-agent setting there are two important changes in the parameters mentioned above. The ERM has now a size of 20, and we introduce a dropout mask before the output layer, as explained before, with a value of 0.5. Figure 5 presents the training outcome. The results are averaged over 18 runs and we again run the training process for 250 000 episodes. Notice that in comparison to the single agent case, the expected performance is lower (ending up at a bit over 60% in training), while the variance is higher, showing uncertainty in the final training outcome.

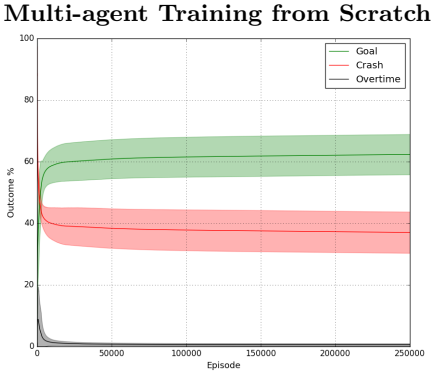

Fig. 5: Performance of the multi-agent from scratch model in training. The agents keeps a constant ϵ exploration rate of 0.1 for the entire process.

Fig. 6: Performance of the multi-agent model in a setting with various ratios of autonomous cars present on the highway. The drop in performance is not as steep as the single agent case, however there is still much variance in the second half of the graphs.

Another important aspect we should note regarding the multi-agent from scratch training procedure is about the training duration. The time for training a multi-agent model from scratch is about 4 times higher than the one for the single agent case (i.e., it took about 4 days, compared to around 20 hours).

Moving on to the testing phase, Figure 6 illustrates the performance of our multi-agent models when various ratios of autonomous cars are present on the highway. Keeping in mind that the ratio under which we trained the policy was 0.5, we can then extract from the left graph the average performance of the models during execution for this scenario, i.e., around 75%. In comparison to Figure 4, we can definitely notice that the multi-agent models handle better the increase in the autonomous agent ratios, however there is still a higher variance observed towards the second half of the axis. Regarding the number of cars

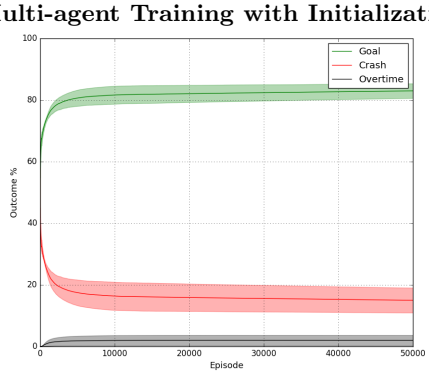

Fig. 7: Performance of the multi-agent with single agent initialization models in training. The agents keeps a constant ϵ exploration rate of 0.1 for the entire process.

entering the highway, we notice a linear drop, signalling a general decrease in the speed of the cars. Looking also at the number of cars exiting, we notice a slight diverging tendency towards the end, a sign for cars coming to a stand still or getting involved in more crashes.

5.3 Multi-Agent with Single-Agent Initialization

Finally, we test our main contribution: using a single-agent policy as an initialization to train a multi-agent policy in a homogeneous open population of learners. For this experiment we initialize the network with a model learned in the single agent scenario. We keep the same parameters as in the multi-agent from scratch case, with the exception of the learning rate, which is lowered to 0.001 and the dropout before the output layer, lowered to 0.25. The results are averaged over 8 independent runs.

The results of the fine-tuning process are shown in Figure 7. We can notice how the goal curve starts now much higher (i.e., at approximately 50%), due to the model initialization, while the final performance level is significantly better compared to the multi-agent learning from zero. The variance is lower and notice how 50 000 additional episodes were already enough to match the single-agent performance. We should also note that the total training time for the single agent model plus the multi-agent initialized from single agent one is still only half of the time required to train the multi-agent network from scratch, while the performance is significantly better.

Our final simulation results (Figure 8) illustrate how the multi-agent with single agent initialization models perform under various ratios of autonomous cars on the highway. The average performance level for the 0.5 ratio is a bit over 90%. We also notice that the performance has no longer such a steep downwards trend, but always remains above 80%. For the number of cars exiting and entering

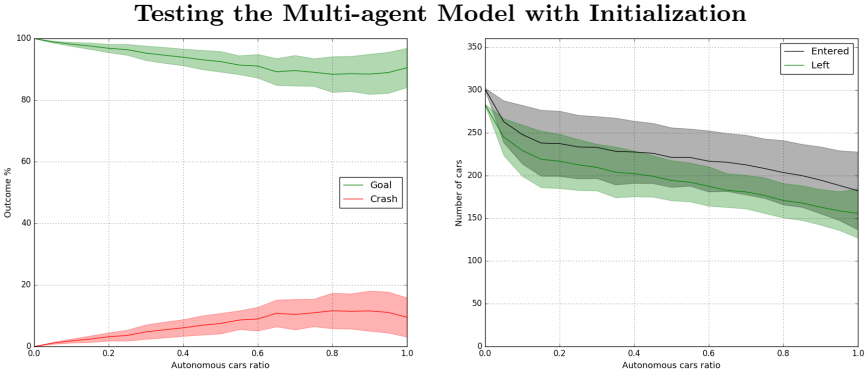

Fig. 8: Performance of multi-agent with single agent initialization models in a setting with various ratios of autonomous cars present on the highway. The drop in performance is only showing a slight decrease, demonstrating that these models are better adapted to a multi-agent scenario.

the highway, even though the variance is fairly high, we can notice the two curves do not diverge from each other, signalling that there is only a decrease in the cars' speed. We can conclude from these results that *models trained in a multi-agent setting, having an initial knowledge transfer from the single agent case exhibit the best performance* in our problem setting.

6 Related Work

Our work takes inspiration from the A3C algorithm [17], in which multiple single-agent environments are run in parallel, and share their experience through pushing gradient updates to a centrally maintained actor and critic. The environment studied in this work however, is vitally different on two important points: in this paper multiple agents act in the *same* environment, and as the population is open the number of agents varies over time.

Our multi-agent training setting is additionally related to the *parameter sharing* training procedure described by [9]. The idea is to leverage the homogeneity of the population of agents and allow them to share the parameters of a common policy. Our open population characteristic, however, prevents us from adding an index for each agent within the model, as the number of agents, as well as the agents themselves, do not remain constant in the population.

In the work of [4] we can also find the idea of an interplay between single and multi-agent scenarios. Although not directly related to our setting, we do find there an interesting procedure for allowing agents to learn as independent learners until there is an explicit requirement for interacting and cooperating with other learners.

The knowledge transfer approach from a single to a multi-agent task presented in this work resembles one of the study cases of [2], where they perform

transfer learning from a single-agent predator-prey setting to a multi-agent one (with two or three agents). However, due to our goal of learning a uniform behaviour for each agent in the population, we do not need to establish any mapping between agents for the transfer procedure. We are additionally dealing with a more complex problem setting and a larger number of agents that can enter and exit the system at varying time rates and that are not necessarily performing a cooperative task.

7 Conclusions

In this paper, we looked at the problem of learning in a homogeneous open population of agents, applied in the SimuLane highway traffic simulator. We started by successfully training a single agent in the environment, however that proved to be insufficient in order to cope with a multi-agent setting.

We then examined two basic ways of sharing knowledge between agents: sharing experience from multiple agents in an open population by learning a shared policy, and additionally transferring knowledge from a single-agent setting to a multi-agent setting.

We have shown that in our setting homogeneous multi-agent learning via policy reuse from a single agent yields better results and is much faster than learning from scratch. We thus conclude that transferring from a single-agent policy to a multi-agent policy in homogeneous open-population multi-agent reinforcement learning is both key to keeping learning tractable, and significantly increases performance.

We note that the direct parameter sharing we used for both types of knowledge transfer is only possible due to the homogeneous population. When the set of agents would be heterogeneous, other transfer methods [29] would be required.

In future work, we aim to examine how the learning can be decentralised and still share experience via social learning [10] or by developing a grounded communication system [20]. Additionally, it holds great interest for us to evaluate different (multi-agent) reinforcement learning approaches such as [13, 16, 24] in this problem setting. Furthermore, another possible study aspect is to remove the homogeneity constraint from the agents and learn in a heterogeneous population. One can then start identifying clusters of similar agents [33] and apply the principle of *learn-from-whom-to-learn* in either a centralized or decentralized manner. Finally, the complexity of the environment can also be increased by considering dynamic traffic densities (simulating daily traffic patters).

Acknowledgements

This work is supported by Flanders Innovation & Entrepreneurship (VLAIO), SBO project 140047: Stable MultI-agent LEarnIng for neTworks (SMILE-IT), and the European Union FET Proactive Initiative project 64089: Deferred Restructuring of Experience in Autonomous Machines (DREAM) and the Security-Driven Engineering of Cloud-Based Applications (SeCLOUD).

References

1. Amato, C., Oliehoek, F.A.: Scalable planning and learning for multiagent POMDPs. In: AAAI. pp. 1995–2002 (2015)
2. Boutsioukis, G., Partalas, I., Vlahavas, I.: Transfer learning in multi-agent reinforcement learning domains. In: European Workshop on Reinforcement Learning. pp. 249–260. Springer (2011)
3. Busoniu, L., Babuska, R., Schutter, B.D.: A comprehensive survey of multiagent reinforcement learning. IEEE Trans. Systems, Man, and Cybernetics, Part C **38**(2), 156–172 (2008)
4. De Hauwere, Y.M.: Sparse interactions in multi-agent reinforcement learning. Ph.D. thesis, Vrije Universiteit Brussel (2011)
5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research **12**(Jul), 2121–2159 (2011)
6. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., et al.: Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. arXiv preprint arXiv:1802.01561 (2018)
7. Foerster, J., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 2137–2145 (2016)
8. Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., Whiteson, S., et al.: Stabilising experience replay for deep multi-agent reinforcement learning. arXiv preprint arXiv:1702.08887 (2017)
9. Gupta, J.K., Egorov, M., Kochenderfer, M.: Cooperative multi-agent control using deep reinforcement learning. In: International Conference on Autonomous Agents and Multiagent Systems. pp. 66–83. Springer (2017)
10. Heinerman, J., Rango, M., Eiben, A.E.: Evolution, individual learning, and social learning in a swarm of real robots. In: Computational Intelligence, 2015 IEEE Symposium Series on. pp. 1055–1062. IEEE (2015)
11. Legrand, M.: Deep Reinforcement Learning for Autonomous Vehicle Control among Human Drivers. Master dissertation, Vrije Universiteit Brussel (2017), `http://ai.vub.ac.be/sites/default/files/thesis_legrand.pdf`
12. Legrand, M., Rădulescu, R., Roijers, D.M., Nowé, A.: The SimuLane highway traffic simulator for multi-agent reinforcement learning. In: BNAIC 2017. pp. 394–395 (2017)
13. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
14. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning **8**(3-4), 293–321 (1992)
15. Littman, M.L.: Value-function reinforcement learning in markov games. Cognitive Systems Research **2**(1), 55–66 (2001)
16. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O.P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems. pp. 6382–6393 (2017)
17. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. CoRR **abs/1602.01783** (2016)

18. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing Atari with deep reinforcement learning. CoRR **abs/1312.5602** (2013)
19. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (feb 2015)
20. Mordatch, I., Abbeel, P.: Emergence of grounded compositional language in multi-agent populations. arXiv preprint arXiv:1703.04908 (2017)
21. Mossalam, H., Assael, Y., Roijers, D., Whiteson, S.: Multi-objective deep reinforcement learning. In: NIPS workshop on Deep RL (2016)
22. Nowé, A., Vrancx, P., De Hauwere, Y.M.: Game theory and multi-agent reinforcement learning. In: Reinforcement Learning: State of the Art, pp. 441–470. Springer (2012)
23. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016)
24. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897 (2015)
25. 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.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
26. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**, 1929–1958 (2014)
27. Steckelmacher, D., Roijers, D.M., Harutyunyan, A., Vrancx, P., Plisnier, H., Nowé, A.: Reinforcement learning in POMDPs with memoryless options and option-observation initiation sets. In: AAAI 2018. pp. 4099–4106 (2018)
28. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
29. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research **10**(Jul), 1633–1685 (2009)
30. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: AAAI. vol. 16, pp. 2094–2100 (2016)
31. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, University of Cambridge England (1989)
32. Wiggers, A.J., Oliehoek, F.A., Roijers, D.M.: Structure in the value function of two-player zero-sum games of incomplete information. In: ECAI16. pp. 1628–1629 (2016)
33. Zhang, C., Lesser, V.: Coordinating multi-agent reinforcement learning with limited communication. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. pp. 1101–1108 (2013)