Deep Q-Network Memory Sharing

Inter-Agent Prioritised Experience Replay

by

Deniz Hofmeister

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday February 20, 2019 at 13:30.

Student number:	4084799	
Project duration:	June, 2018 – February, 2019	
Thesis committee:	Dr. Frans A. Oliehoek,	TU Delft, supervisor
	Prof. dr. Catholijn M. Jonker,	TU Delft
	Dr. Matthijs T. J. Spaan,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This master thesis report has been written to fulfil the graduation requirements of a Master of Science in Electrical Engineering at the Technical University of Delft and was done under the supervision of the department of Intelligent Systems, Delft University of Technology.

All the work done in this report was done by M.D. Hofmeister starting summer 2018 onward. All the code was writting using python and OpenAI was used to write a custom gym environment in python along with tensorflow as the backbone of the agents.

Deniz Hofmeister
Delft, February 2019

Abstract

Humans teach each other by recollecting one's own experiences and sharing them with others. The intention being that the person being taught, does not need to experience those things first-hand to be able to learn from them. A large portion of human learning is in some form derived from this concept. This has inspired this report.

Recent developments in Deep Q-Networks applied a so-called replay memory. This replay memory stores experiences in a buffer for it to learn from later. It is this replay memory that is now used as a source of information for other agents. The core principle being attempting to find important entries in the replay memory and sharing those memories with other agents could improve the learning process of an agent. This memory selection process is done by use of prioritised sampling. The desired result being an agent able to learn faster.

A memory entry consists of the tuple $[s_t, a_t, r_t, s_{t+1}]$ and one can calculate its temporal difference error by:

$$\delta_j = r_j + \gamma \, \hat{Q}(s_j, \underset{a}{\operatorname{argmax}} \, Q(s_j, a)) - Q(s_{j-1}, a_{j-1}) \tag{1}$$

This is used to determine the importance of memory samples by influencing how likely those memories are to be sampled for teaching one another:

$$\begin{aligned} P(i) &= \frac{p_i^\alpha}{\sum_k p_k^\alpha} \\ p_i &= |\delta_i| + \epsilon \end{aligned} \tag{2}$$

Temporal difference is chosen as a metric for importance as it represents a surprise value. This method is then compared to a baseline DQN-agent who does not communicate, a randomly messaging DQN-agent and some other variations. The benchmark tasks describe a simple gridworld with objectives with varying levels of difficulty.

The results of the experiments show that, initially, prioritisation results in rapid learning. This is due to the fact that agents are able to converge to identical policies without it having detrimental effects on the return of rewards. This converging, however, has proven to hamstring the learning process as a whole that prioritised messaging agents have. Very similar policies in benchmark tasks which stand to benefit from heterogeneous policies lead to agents not being able to find their personal optimal policy. The non-communicating agent was able to diverge in policies and therefore ultimately solve the problems more efficiently and with a higher return in rewards.

Acknowledgements

I would like to thank my supervisor Frans A. Oliehoek of the Technical University of Delft for giving me a lot of freedom in asking my own questions in this report and for allowing me to build this thesis around my own curiosity. Thank you for always thinking with me and asking critical questions.

I would also like to thank Miguel Suau de Castro for the discussions surrounding the topics of this report and sharing his views on all things related to artificial intelligence. These discussions have helped me to keep the report focused and precise.

I would also like to thank my friends, Ashish Sachdeva, Gerard Baquer and Guillermo Ortas for being there for me. I can truly say that my years spent as a master student have been the best years of my life, thank you for everything.

I am truly grateful for the support and flexibility that was offered by everyone involved, both professionally and personally. This is especially true for the moments where I have personally not lived up to expectations. Thank you for being understanding and accommodating.

Contents

1 Introduction

Artificial neural networks have been a focus of research for decades. The idea of being able to simulate neurons and, in the grander scheme, brains have always been one of the ultimate goals of this field of research. Being able to recreate human or animal intelligence artificially or having computers being able to solve complex and abstract tasks better than humans is something that has fascinated and motivated many. Going beyond simulating a single brain, tasks which require cooperation or competition, requires teamwork or rivalry, is another direction this research can go into and will be the focus of this report.

The first publication that sparked the research on artificial neural networks came from the biological perspective. Hebb [3] published a book on the workings of biological neural networks and it was theorized in the book that neurons would improve their connections with neighbouring neurons if they would activate together. He suggested that neurons might work by strengthening pathways in the brain each time they are used. A while later, the first artificial neural network was simulated by Rosenblatt [9] in 1957 with his research on perceptrons. In 1969, Minsky and Papert [7] published a book called Perceptrons. This book marked the center of Artificial Intelligence discussions for the coming years. This is largely due to the fact that Minsky and Papert [7] expressed heavy criticism on the topic and numerous attempts failed at refuting the criticism. After this research artificially recreating the behaviour of neurons was put on hold. Events in this period made this direction of research seem to be a dead end. Fast forwarding to recent decades, there have been a number of important breakthroughs. Back-propagation became the norm in solving the weights for neural networks and later on the paper by Mnih et al. [8] showed neural networks with a Deep Q-Network (DQN) architecture were able to play a select set of computer games better than humans and therefore making DQNs a proper basis on which the proposed information sharing method will be built on.

DQN's are used in this report as a platform on which a communication system is built. This report searches for ways on how multiple agents in the same environment can stand to benefit from each others' information and experiences. From a human perspective, an import method of learning is by being taught something from someone else. The person shares his/her experiences with the intention of having you learn from that information such that you don't need to experience those moments first hand. Sharing information should, intuitively speaking, lead to an improvement in learning speed.

This report explores through simulations different possibilities of information sharing with local agents. Limitations are applied to be more representative to what an agent would encounter as a physical robot, therefore steps out of the simulation realm : Agents cannot communicate over an infinite amount of distance and therefore a limit will be applied in the simulations. The second limitation that is applied is local observability. Each agent will only be able to observe its direct surroundings and not the entire map. An agent will need to solve simple tasks in a grid-world type of environment. The communicating agents will be compared to a baseline which consists of a single agent without any communication abilities.

1.1. Problem Statement

In a system where multiple agents are active, learning happens in parallel between all the agents. Each agent is learning on its own and therefore, possibly gaining knowledge which other agents might already have learned. In a system where all the agents have the same goal, they will possibly learn from similar experiences. This is redundant. Ideally there exists a collective pool of information that all the agents drop their experiences in and then agents can draw from this pool to learn. This would render every observation and action made by any agent in the system a potential source of information for any other agent in the system. This allows collective and symbiotic learning.

A caveat regarding learning the same task and therefore implying that all agents learn the same policy is that it might be possible that having heterogeneous policies will prove to be more effective than identical policies for every agent. As an example: If Agent 1 and Agent 2 both need to reach the same goal, but two routes are available, then it could be possible that it's better that the agents choose different routes to not get in each other's way. This means that the agents need to learn a different policy. These effects are also to be considered.

Limitations

The shared learning idea, however, could potentially require a large communication channel since this pool needs to be available to be read by every agent and this is not a reasonable assumption for robots working in the real-world like warehouse distribution systems. In such a system, all agents run around collecting packages and delivering such packages at the desired locations, infinite communication is not possible. One can also not assume that every agent is able to communicate with every other agent, potentially an agent can communicate with other agents within proximity.

Another limitation given to agents is limited vision. An agent in the real world cannot possibly see beyond a the range of its sensors. It is limited by the information giving by the sensors it has access to. This limitation is simulated in this report and thus each agent can only view its local surroundings. More details on this can be found in section 4: Methodology. These fundamental limitations are imposed in this report and these simulations to conform more to the limitations an actual robot in the real world would have.

The third limitation applied is complete independence and decentralisation. On the same principles of making agents only locally aware, no form of centralisation is applied. Agents do not get fed any form of information that would allow agents to know more than locally available information with respect to themselves.

Limitations are also applied to define the scope of this report. Since this report is based on existing agents learning in an environment it is a direct comparison between non-communicating agents versus agents using a limited communication channel. This will therefore show the effect of what there is to gain by sharing information. Any form of communication which could potentially require large packages of data to be transferred are unrealistic and therefore not considered. The exact type of information transfer chosen is discussed in section 4: Methodology.

Research Question

As to be shown in section 4: Methodology, DQN will lend itself well to expanding its algorithms to the multi-agent domain, leading to the following questions to be discussed and answered in this report:

- How do DQN agents that share information directly compare against agents that don't?
- How can an agent select what is important information to share?
- Which method of information sharing yields the best performance in terms of maximum average rewards?

2

Background

Firstly in section 2.1, a way of modelling some real world systems as an MDP is discussed. This will provide the basis of the learning algorithms. Then the concept of reinforcement learning and in particular Q-learning is discussed in 2.2.

One can interpret Q-learning as an estimator of the value of an action given a certain state. This estimator can also be built with neural networks at the base. This is done using Deep Q-Networks, which is explained in section 2.4. Finally, this deep Q-network is tweaked and modified by use of prioritised experience replay. This is discussed in 2.5.1.

2.1. Markov Decision Process

A Markov decision process (MDP) is the mathematical model of the full set of possible states, actions and rewards a closed system has. It also describes their relationship between the states, actions and rewards. An MDP will dictate which state transitions are able to occur given the available actions, how likely this transition is and how rewarding this transition is. It formally consists of the 4-tuple: (S, A, P_a, R_a) with the following properties:

- S: The set of states possible in the system
- A: The set of possible actions given the states
- $P(s'|s,a)$: The probability of transitioning to a next state given the current state and chosen action
- $R(s,s')$: The reward associated with transitioning from the current state:s to a new state:s' given a certain action

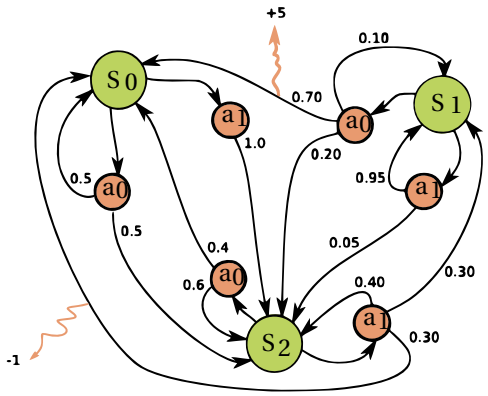


Figure 2.1: Markov Decision Process. This system contains 3 states: s_0, s_1 and s_2. 2 actions: a_0 and a_1 for each state, and two rewarding transitions, shown as curly arrows.

It provides a framework for an agent's decision making. An agent who is in a certain state (s) of this process, will have a set of actions (A_s) available to it. From there the agent chooses an action and observes the new state (s') and a potential reward ($r(s,s')$). In this report the transition probability component of a Markov Decision Process is removed by assuming that the probability of transitioning to a next state is guaranteed. Implying that for every action, there is a fixed, single next state.

2.1.1. Partially Observable Markov Decision Process

This report focuses on local observability. This implies that feeding the state information S directly to the agent is not allowed. Instead, this report feeds the agent local observations only. This creates possible ambiguity as there is now a split between underlying true state S and observations Ω. An agent cannot directly infer that it's observations directly correlate with a true hidden state. The formal definition of a Partially Observable MDP (POMDP) is given by a 6-tuple $(S, A, P_a, R_a, \Omega, O)$ and is an expansion on the first 4-tuple of an MDP by:

- Ω: The set of observations
- O: set of conditional observation probabilities

The set of conditional observation probabilities O describe the chances of receiving an observation from Ω given the states S and actions A. In this report these probabilities are fixed with the same logic as above. Every state only has one possible observation, thereby eliminating the conditional probabilities. This observation will be related to the underlying true state and will not have any form of noise or other perturbations. By describing the system as a POMDP, one can build an agent using this framework. This is described in the next section.

2.2. Reinforcement Learning

Reinforcement learning methods aim to learn the 'best' action given a certain state or observation. The definition of best may vary based on the application, however in general one aims to in some way maximise the amount of rewards the agent receives. For example, given the MDP in figure 2.1, one can see that to maximise the sum of discounted future rewards one would want to trigger the +5 reward associated with state s_1 by taking action a_0 as often as possible. Implying that maximisation would be taking action a_0 at state s_1 or taking the actions necessary to reach state s_1 as soon as possible. Reinforcement learning tries to algorithmically find such actions.

2.2.1. Discounted Future Rewards

For every step, an agent could receive a reward. Looking forward, one can sum all the future rewards an agent would receive. The goal of an agent is to find a set of actions that maximise the this sum. Formally:

$$\sum_{t=1}^{T} \gamma^{t-1} R_t = R_1 + \gamma R_2 + \gamma^2 R_3 ... \tag{2.1}$$

This function describes the sum of total discounted rewards, given that $t = 0$ refers to the current step of the agent, $t > 0$ referring to future steps. The discount factor γ is added to scale down the value of rewards which are a large amount of time-steps away. γ is picked as $\gamma \in [0,1]$, with $\gamma = 0$ leading to no value being added for rewards that are not immediate, and $\gamma = 1$ leading to the simple sum of all rewards regardless of distance. $\gamma \in (0,1)$ leads to a natural decay in value the larger t becomes and acts as a method of balancing the value of immediate versus (potentially higher) future rewards. the upper limit T of the sum signifies the look-ahead horizon.

2.2.2. Value-Function

The value-function is a representation of the value of a given state. The value of a given state is inferred by the discounted sum of future rewards. Which future rewards the agent will receive will be dependent on the choice of actions determined by a policy π. This results in the value function, given policy π:

$$V_\pi(s) = \mathbf{E}_\pi \left[\sum_{t=1}^{T} \gamma^{t-1} R_t \mid S_t = s \right] \tag{2.2}$$

For any MDP, there exists an optimal value function which has higher values than other functions for all states. This is directly dependent on the optimal policy.

$$V_*(s) = \max_\pi V_\pi(s) \tag{2.3}$$

Equation 2.3 shows that there has to exist a certain policy π such that the value-function $V(s)$ is maximised. It is however not initially known which policy will lead to this maximisation. An example of a value-function applied to a system where states are represented by X and Y coordinates is shown in figure 2.2.

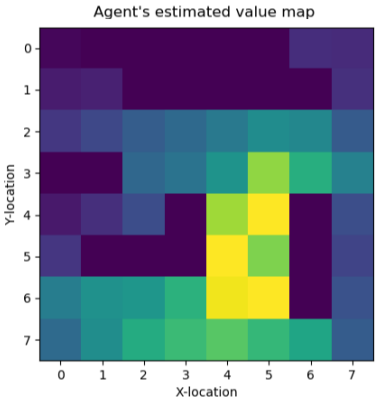


Figure 2.2: Value estimation example for locations of this map. Brighter colours indicate states of higher estimated value. One can see that locations closer to $[X, Y] = [5, 5]$ return higher values. This is due to the map only containing a reward at that location. The decay in value is visible as a gradient the more steps the location is away from the reward. Dark patches indicate untraversable terrain.

Initially, this optimal value-function is unknown. To iteratively solve the value-function one needs to apply value iterations. These are given by:

initialise $V(s)$ *arbitrarily*
Repeat
for $s \in S$ **do**
  for $a \in A$ **do**
    Q(s,a) $\leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$
  end
  V(s) $\leftarrow \max_a Q(s,a)$
end
until V(s) converged
With:

- $Q(s,a)$: state-action value-function, described in subsection 2.2.4.
- r: immediate reward
- γ: discount factor
- $P(s'|s,a)$: state transition probabilities

Algorithm 1 shows that to determine V(s) one needs to know the state transition probabilities $P(s'|s,a)$. This is known as a model-based learning algorithm. There exists however, value approximators that do not require such knowledge, such as Q-learning. This is discussed in the following subsections.

2.2.3. Model-Free Learning

In most systems where agents need to solve a given task, the internal dynamics of the system are not known to the agent. The agent will not be able to predict which actions leads to which state transitions nor does it know which actions trigger rewards. This means that in the MDP above, $P(s'|s,a)$ and $R(s,a)$ are unknown for the agent. Therefore, a direct solution cannot be computed in the form of solving an equation. Yet, such systems are still solvable using model-free value functions that iteratively learn the value of the states in the system. The Q-Function is an example of model-free learning.

2.2.4. Q-Function

This value-function can be expanded to also be a function of actions. As said earlier, different policies can lead to different future rewards. To disambiguate the fact that the same state can lead to different future rewards given different actions, the Q-function is introduced and builds on the value function by adding the action space as a dimension to the value-function.

$$Q(s,a) \;=\; \mathbf{E}_{\pi}\left[\sum_{t=1}^{T}\gamma^{t-1}R_t \;\mid\; S_t = s\;,\; A_t \;=\; a\right] \tag{2.4}$$

This function is able to return values for each given state-action pair. After the action has been chosen, it assumes that the agent will continue following policy π.

2.2.5. ϵ-greedy Policy

A policy is a function which takes current state or observations as an input and gives an action as an output. Although there are numerous different methods available, this report uses the ϵ-greedy policy. This policy, can be described as follows:

$$\pi(a|s) \;=\; \begin{cases} b & \text{if p} \;<\; \epsilon, \\ \underset{a}{\text{argmax}}\;\; Q(s,a) & \text{otherwise} \end{cases}$$

With:

- p: a uniformly randomly drawn value $\in [0,1]$
- b: a random action drawn from the available action space.
- ϵ: usually set close to 1 initially. Over time, this is reduced to a value approaching zero.

Exploration vs. Exploitation The variable ϵ determines the chance of occurrence that a random action b is picked. This random walk is described as exploration. The usefulness of exploration is that it explores the (PO)MDP graph. Possibly encountering shorter paths or more rewarding paths. The usefulness of this exploration will be clear in the next subsection.

When the agent picks the action related to the highest $Q(s,a)$-value, then this is known as exploitation. As the agent uses already known information to maximise its discounted future sum of rewards.

2.2.6. Q-Learning

The Q-Function has to be iteratively found. Initially, the Q-function can be set with random values, zeros or even starting off in any other arbitrary way. The intention is to iterative update this Q-function such that it convergences to the true Q-function.

Starting from this, the Q-function updates its entries based on the following:

$$\hat{Q}_{t+1}(s,a) \leftarrow (1-\alpha)\hat{Q}_t(s,a) + \alpha\left[r + \gamma \underset{a'}{\text{max}}\hat{Q}_t(s',a') - \hat{Q}_t(s,a)\right] \tag{2.5}$$

where:

- $\hat{Q}(s,a)$: is the current estimation of the true, unknown Q-values: $Q(s,a)$
- α: Learning rate. Determines how much of the previous estimated $\hat{Q}$-value carries over to the new estimation. Usually fixed at $\alpha = 0.1 \sim 0.3$
- r: immediate reward of current state-action.
- γ: discount factor. Determines the influence of the next state-action value of Q on the current state-action value. High values lead to high importance given to future state-action values. Usually fixed at $\gamma = 0.8 \sim 0.95$.
- s', s: next state and current state, respectively.
- a', a: next action and current action, respectively.

this method of iteratively updating the Q-function is called Q-learning.

Combining this with the previously discussed ϵ-greedy policy, gives an agent the ability to choose actions based on state-action estimations. These estimations will over time convergence to the true Q-function as the agent is iterating and learning. An agent will, once converged, be able to consistently make the optimal choice. Optimal is defined here as choosing the action which returns the highest discounted future rewards.

2.3. Fully-Connected Neural Networks

The previously discussed Q-function tries to tag a value to states and actions. This can be seen as a look-up table. This look-up table can also be estimated by a neural network. Why this is a useful thing will be clear later. There are many variations to neural networks and in this report only the fully connected neural network will be discussed and used.

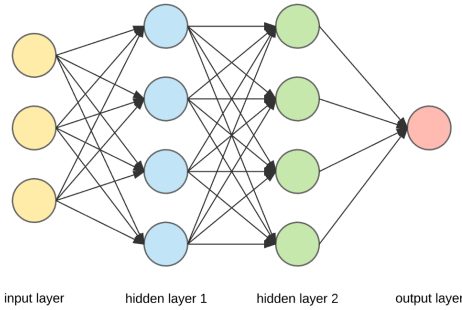
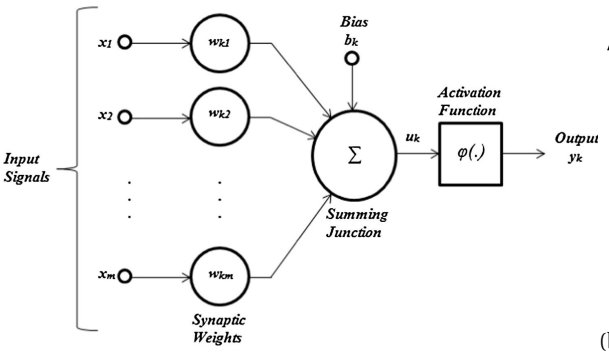
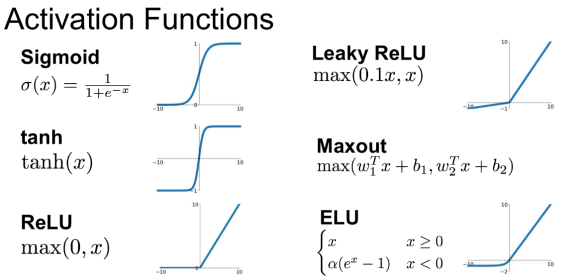


Figure 2.3: Basic layout of a fully connected neural network consisting of two hidden layers.

A fully connected neural network connects an input layer to an output layer with possible layers in between, see figure 2.3. All nodes on a lower layer are connected to all the nodes of the following layer. A neural network's task is to fit some form of function such that given inputs will produce the desired output. To do so, every node is able to perform mathematical operations on its own respective inputs.



(a) Exploded view of a single node.



(b) Nonlinear functions to use in the activation function.

Figure 2.4a shows one of the nodes in exploded view. The outputs of the previous layer are first weighted then summed, with a possible bias added to the summation. This result is now passed through a nonlinear function to generate the output. The most commonly used nonlinear functions are shown in figure 2.4b. The nonlinear function is an important part of the neural network as it allows a nonlinear relationship between the input and the output. This allows neural networks to estimate many real world relationships between two metrics, which are very often nonlinear in nature.

2.3.1. Loss Function

One can assign a metric as to how correct a neural network is. Given that the input and the output is known, one can calculate the difference between the neural network's output and the desired output. Given desired output vector $\mathbf{y}'$ (also named training samples) and network output vector $\mathbf{y}$, one can formulate a squared euclidean distance loss function as:

$$\mathbf{L}(\mathbf{y},\mathbf{y}') = \frac{1}{2}||y-y'||^2 \tag{2.6}$$

The repeated minimisation of this loss function by tuning the weights and biases will hopefully lead to a properly fitted neural network able establish a (nonlinear) relation between the input and output.

2.3.2. Backpropagation & Gradient Descent

The minimisation is done through back propagation and gradient descent techniques. Backpropogation is the method of using the loss function $\mathbf{L}(\mathbf{y}, \mathbf{y}')$ and finding the partial derivatives of $\mathbf{y}$ with respect to every weight and bias in the network. This derivative can then be used to increment the value of each individual weight or bias such that the loss function's output is reduced. Performing the partial derivative leads to:

$$\nabla\mathbf{L}(\mathbf{y},\mathbf{y}') = \nabla\frac{1}{2}||y-y'||^2 = \left[\frac{\partial\mathbf{L}}{\partial w_1},...,\frac{\partial\mathbf{L}}{\partial w_n},\frac{\partial\mathbf{L}}{\partial b_1},...,\frac{\partial\mathbf{L}}{\partial b_n}\right] \tag{2.7}$$

To take the partial derivative of $\frac{\partial\mathbf{L}}{\partial w_1}$ one first needs to write $\mathbf{L}$ as a function of w_1. Using a simple neural network node as an example and taking $tanh(x)$ as an activation function, like figure 2.4a:

$$\begin{aligned}\mathbf{L} &= \frac{1}{2}e^2\\ e &= y-y'\\ y &= \tanh(b+\sum_i w_i x_i)\end{aligned} \tag{2.8}$$

Now using the chain rule the partial derivative reads:

$$\frac{\partial\mathbf{L}}{\partial w_1} = \frac{\partial\mathbf{L}}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial w_1} = (e)(-1)(1-\tanh(w_1)) = -e+e\tanh(w_1) \tag{2.9}$$

This defines the gradient of the loss function with respect to w_1.

One can now use this gradient to update the weights and biases of y such that the resulting loss is lower. This is what is called a gradient descent. For the example of w_1 this follows:

$$w_{1,k+1} = w_{1,k} - \alpha\frac{\partial\mathbf{L}}{\partial w_{1,k}} \tag{2.10}$$

with the k-index signifying the difference between old and new weights and α being the step size. This process can be done for all weights and biases to reduce loss.

2.4. Deep Q-Network

In the paper by Mnih et al. [8] the deep Q-network uses a neural network as a Q-value approximator but retains most of the Q-learning methods to solve an (PO)MDP problem. In this report, a fully connected neural network is used instead of a convolution neural network.

The main differences between the Q-function explained before and the Deep Q-Network will be explained in the following subsections.

2.4.1. Target Network

A neural network based function approximator has shown to be unstable in its learning process. To combat this, the paper uses a target network θ^-. This target network fully copies the currently active neural network to its own. This copying happens every C-steps. This target network is now used when the loss function is computed and a gradient is calculated instead of the current neural network's weights and biases.

2.4.2. Replay Memory

A second addition involves not immediately training the network on the current transition. Instead, all the required information to perform a training step is stored in the replay memory. Equation 2.5 shows the required variables to compute the update step: $[s, a, r, s']$. These tuples are stored in the replay memory by FIFO-principle. Starting Empty and then increasing in size until max size N. This replay memory allows the training to happen without subsequent updates being heavily correlated, since one can now randomly sample this replay memory in any order desired. Correlated samples used in training have proven to have detrimental effects on the DQN's performance.

2.4.3. Loss-Function

The loss function as described in the paper is written as:

$$\mathbf{L}_i(\theta_i) = \mathbf{E}_{(s,a,r,s')\sim U(D)}\left[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2\right] \tag{2.11}$$

with

- $(s, a, r, s') \sim U(D)$: uniformly distributed random batch sampling from the replay buffer.
- θ_i^-: target network.
- θ_i: current network.

Now applying the sample principles of partial derivatives with respect to the weights on this loss function leads to:

$$\nabla_{\theta_i}\mathbf{L}_i(\theta_i) = \mathbf{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right] \tag{2.12}$$

The same method of updating the neural network weights is used as discussed previously in the section regarding backpropagation, namely the gradient descend iterations, shown in equation 2.13.

$$\theta_{i+1} = \theta_i - \alpha \nabla \mathbf{L}_i(\theta_i) \tag{2.13}$$

2.4.4. Full Algorithm

Combining the additions described above into the full method of training a DQN leads to the following algorithm:

initialise replay memory D to capacity N
Initialise action-value function Q with random weights θ
initialise target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode 1,M* **do**
  Observe s_1
  for $t = 1, T$ **do**
    Pick action a_t using the ϵ-greedy policy $\pi(a|s_t)$
    Execute a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random mini-batch (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to θ
    Every C steps, set $\hat{Q} = Q$
  end
end

2.5. Prioritised Experience Replay

Schaul et al. [10] builds on this Q-network by arguing that the replay memory could potentially be used to distinguish important entries from unimportant memories with the intention to assist the training speed of a DQN-based agent. Prioritised experience replay modifies the DQN algorithm by considering non-uniform methods of sampling the replay memory. Instead, it determines which memories are more important than others by computing the temporal difference error(TD-Error).

2.5.1. TD-Error

A replay memory sample j contains $[s_{j-1}, a_{j-1}, r_{j-1}, s_j]$. The temporal difference error δ for this memory sample is defined in Schaul et al. [10] as:

$$\text{TD-error: } \delta_j = r_j + \gamma\, \hat{Q}(s_j, \underset{a}{\text{argmax}}\, Q(s_j, a)) - Q(s_{j-1}, a_{j-1}) \tag{2.14}$$

Using this definition one can now link the sample priority p_j to the TD-error δ. Next the sample probability can be calculated for every transition i in the replay memory and is defined as:

$$\begin{aligned} P(i) &= \frac{p_i^\alpha}{\sum_k p_k^\alpha} \\ p_i &= |\delta_i| + \epsilon \end{aligned} \tag{2.15}$$

with α a scalar to determine the prioritisation amount and if $\alpha = 0$, this reduces to the uniform sampling case and ϵ to provide a priority floor for samples with very low TD-error, to still allow them to be trained on from time to time. By default, $\alpha = 1$.

2.5.2. Full Algorithm

This leads to the full algorithm of prioritised DQN:

initialise replay memory D to capacity N
Initialise action-value function Q with random weights θ
initialise target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode 1,M* **do**
  Observe s_1
  for $t = 1,T$ **do**
    Pick action a_t using the ϵ-greedy policy $\pi(a|s_t)$
    Execute a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample mini-batch (s_j, a_j, r_j, s_{j+1}) from D $\sim\ P(j) = \frac{p_j}{\sum_i p_i}$
    Compute TD-error: $\delta_j = r_j + \gamma\, \hat{Q}(s_j, \underset{a}{\text{argmax}}\, Q(s_j, a)) - Q(s_{j-1}, a_{j-1})$
    Update transition priority $p_j \leftarrow |\delta_j| + \epsilon$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at } j+1 \\ r_j + \gamma \underset{a'}{\max}\, \hat{Q}(s_{t+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to θ
    Every C steps, set $\hat{Q} = Q$
  end
end

This is the final algorithm that will be used in this report. The priority sampling of equation 2.15 will also be used to send out memory samples to other agents. The method of picking a sample is identical. More on this in 4: Methodology.

3

Related Work

In the paper by Sukhbaatar et al. [11] they expand the neural network to also control the communication channel. In this paper, they introduce a network called CommNet $\Phi(s)$. It describes modules $f^1, .., f^i$ which form a multilayer fully connected neural network. Every agent carries the same identical neural networks. The performance for a similar testing environment simulating a discrete traffic junction shows that Commnet consistently outperforms independent, fully connected and discrete agents. Fully connected implying that the connections between agents does not follow the layout given by CommNet and discrete that agents do not communicate gradients but symbols from a small set. CommNet relies on training in a centralised fashion. Even though, it shows robustness for adding and removing agents (the modules are identical for every agent, therefore agents can be added and removed at will), it does require pre-training. This report attempts to train decentralised.

Foerster et al. [1] train two independent Q-networks, one who handles the actions and one who handles the communication. The messages are discrete symbols. The Q-networks are interconnected where an agent's communication output is another agent's state input. They achieved remarkable performance through this method. Melo and Veloso [6] Attempted similar methods, but instead of training the second Q-network on messages, it was set as a coordination critic, trying to balance the personal optimal policy over the value of not following one's own optimal policy. This proved to not provide the desired results.

another attempt at symbolic messaging was from Kasai et al. [4]. There the message type was hardcoded. In the hunter-prey simulation, each communication message would consist of observed other agent, observed goal, observed walls and an auxilliary signal. It showed that limited communication immediately performed better over no communication.

A big concern in multi-agent systems is non-stationarity. Foerster et al. [2] apply a sampling technique to estimate the priority of the sample based on policy differences between the time of receiving a replay memory entry and the time of sampling. An samples naturally become obsolete because old samples describe a system in which agents behave according to a different policy, this sampling technique assigns a lower weight to the sample.

Leibo et al. [5] Simply disabled replay memory for their DQN agents to prevent problems regarding non-stationarity.

4

Methodology

A way to address these questions posed in section 1.1 regarding the usefulness of information sharing compared to no sharing of information at all, one needs to create both a simulation environment and an agent that together are capable of answering those questions. These will provide a measurable and repeatable method of answering the questions. These are the two core components of the system. The environment supplies agents with information (observations or directly the state), the agent uses this information to choose an action and then the environment computes the consequences of the action the agent chose. The consequences consist of new observations (or a new state) and possibly a reward, these are then fed again to the agent. This is one loop of the environment-agent simulation cycle. This back and forth exchange of observations and decisions is the core method of training an agent. The agent's task is to find a way of choosing the actions which correspond to the highest return of average rewards as described by the Bellman equations. Basically attempting to find a set of rules to transform the input information (observations/state and rewards) to an action to return the maximum rewards. More details on the functioning of this decision process is discussed in 2.2.6: Q-Learning and 2.2.5: ϵ-greedy Policy. The high-level information flow is shown in figure 4.1.

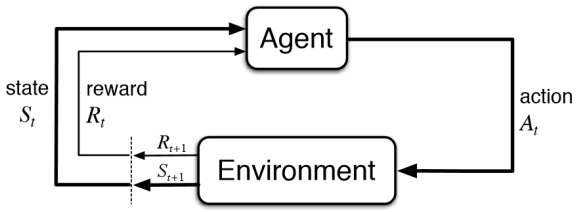


Figure 4.1: Interaction between the environment and agent

Multiple variations of the environment and multiple agent types are considered. This is done to test the different agents in different types of tasks to find out under which conditions communication could be useful. So each task will be tested with a baseline DQN agent without communication versus a number of different type of agents that communicate differently. Each task will be based on a similar grid world environment but will vary in terms of the task given to the agent, either it's a single-agent task or a competitive task. Three different tasks will be considered. As one should not assume that communication will have the same performance advantage/disadvantage for every benchmark task posed.

Firstly, let's discuss the information flow in greater detail. Figure 4.2 shows an information flow diagram with the intend to dig deeper into the figure shown in 4.1. The agent and simulation blocks are opened to show the exchange of information on a lower level. Regardless of task or chosen agent type, they all follow the same principles of information flow. Therefore, this is discussed first. The choice of this lower level layout comes from the basic requirements which will now be discussed further. The default system only has an action input scheme to the environment which does not allow any form of messages to be sent by an agent. It also does not allow agents to receive messages as the default environments don't support such a concept. To support communication another input in added to the default layout of a training system. The full information flow of the system used in this report is shown in figure 4.2. There the flow is discussed in detail.

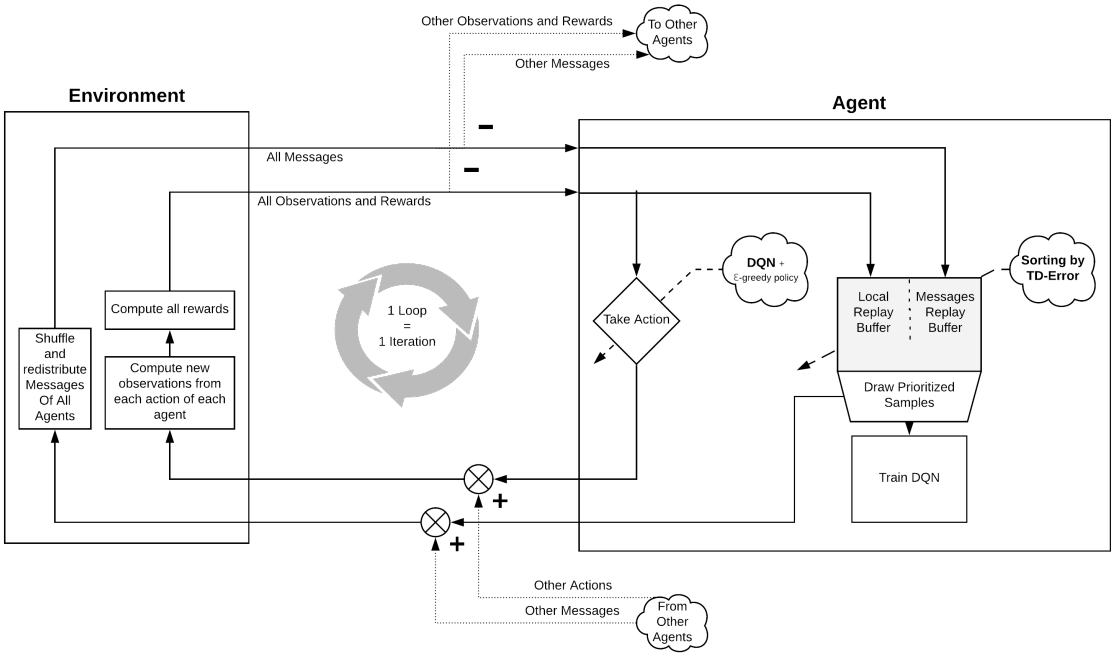


Figure 4.2: Information Flow. Starting at the environment's output. The output contains multiple collections, each containing a message, a new observation and the reward. The number of collections is equal to the number of agents. There it is processed and split by an auxiliary script to make sure the correct agents receive their respective collection. The information now enters the agent. There the observations are used to compute the next action. This choice is determined by the trained model then the action is sent out to be made into a collection again, this collection containing all other agents' actions and messages too. The entire input collection is also stored in the replay buffer where it is sorted by TD-error and used to train the model. Once in the replay buffer, no distinction is made between own observations versus observations contained in the messages. The output collections There the messages are shuffled (randomly re-assigned to other agents), new observations and rewards are computed and the environment's internal state is updated.

Now that the basics of the interactions between the agent and environment have been discussed it is time to go into detail with each in the following sections. Firstly, the buildup of the environment is discussed in section 4.1: Simulation Environment, followed by the agent's design in section 4.2: Agent. In these sections also the specific type of information that is exchanged is explained.

4.1. Simulation Environment

4.1.1. Method

Even though three different tasks will be considered, they share the same base layout of a grid-world. Figure 4.3 shows the layout. This is chosen to provide a reasonable simple environment which would still be capable of having the intended functionality as limited vision and limited communication. This environment contains walls and traversable terrain and is built with OpenAI compatibility in mind.

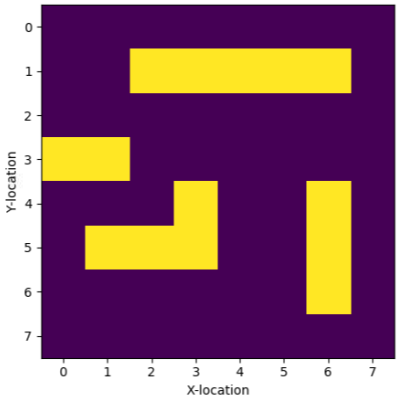


Figure 4.3: grid-world. Yellow: Walls, Blue: Traversable terrain

The design of this map is based on a few principles. Firstly, the grid world should not be unnecessarily complex. Any form of redundant complexity should be avoided, as it would not help answering the research questions. Secondly, the map should contain a unique layout. Every location of the map should in some sense have a recognisable footprint. This stems from the fact that agents with limited vision are used. If one would imagine a map with a layout which contain multiple identical sections and an agent is only able to view within the boundaries of this section of the map, then an agent will not be able to localise itself in the greater map due to the fact that multiple locations are identically observed by the agent. This is to support the side-question as to how much communication is able to help to solve this local observability problem, or how much communication is detrimental to an agent's ability to perform localisation. Figure 4.3 shows that the map is relatively small, as it's only an 8x8 grid, but does try to keep uniqueness for every location on the map. This uniqueness only breaks down when very small ranges of vision are used.

More on this vision system will be described in in paragraph Vision System. However, firstly the internal agent tracking system and movement system of the environment is discussed in States and Movement System.

Figure 4.4: 4 agents, in green, reward in yellow, walls in dark blue, the rest of the map are traversable tiles and also legal spawning locations.

States and Movement System Initially, every agent is spawned on the map on a free location. Meaning that agents cannot spawn on top of each other and cannot spawn inside walls. All locations which comply to this requirement are from now on called legal spawning locations. This location corresponds to the internal state of the agent. This state is not shown to the agent, rather, an observation which is derived from this hidden state is fed to the agent, more on this in 4.1.1: Vision System. The environment keeps track of these agents' locations across iteration cycles. The internal states hidden in the environment are based on the X-Y coordinates of each agent active on the map. From there each agent has the ability to choose 1 out of 4 actions (the action space): (1)Up, (2)Down, (3)Left or (4)Right. Note that an agent cannot choose to stand still. After an agent chooses an action, its location is updated by increments of 1 in either the X or Y direction, with $[0,0]$ being top left of the map. Each agent has to choose 1 action at each cycle of the loop in figure 4.1. This is the most common state transition in the system and this transition is applied after the agents have chosen their move. For example: Agent 2 currently located at $[x,y] = [4,2]$ chooses action (2)Down which corresponds to a translation of $[0,+1]$, thus its new location is $[4,3]$. Movement costs energy in this environment. It is implemented in the form of a (small) penalty per action. This is an unavoidable penalty.

There are a few exceptions to this movement system. These exceptions come from either a trigger coming from the collision system or the reward system. In those cases a different state transition is applied after an agent has chosen an action. They are discussed in their respective paragraphs.

Collision System There are two types of collisions possible in this system. Either an agent collides with a wall or an agent collides with another agent. The internal algorithm which does this check is different, but the results are the same. After such a collision is detected, the agent(s) receive(s) a negative reward (a penalty)

and they are reset. The reset implies that the agent is removed from its collision location and respawned on a random legal spawning location. The intent of this system is to train agents not to hit walls or each other.

This collision system takes precedent over the reward system. Meaning that two or more agents choosing to move onto the reward location at the same time will be penalised instead of rewarded, due to the collision trigger happening before the reward trigger.

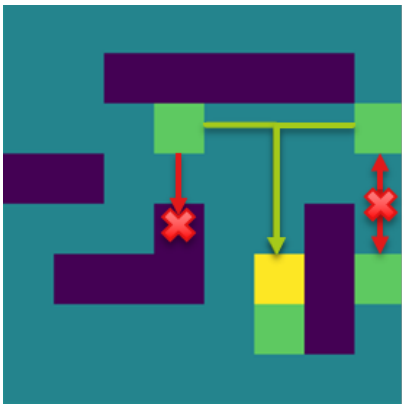


Figure 4.5: Collisions in red, ideal path in green.

Reward System Every task will have a goal. this goal can be different for every task, but every task has a goal nonetheless. This goal on the map corresponds with reaching a certain location. When an agent is located directly next to the goal and that agent picks the action which results in it's new location being the same as the reward location, then it is rewarded and reset. A reset meaning that it is respawned in a legal location and its (historic) observations are cleared.

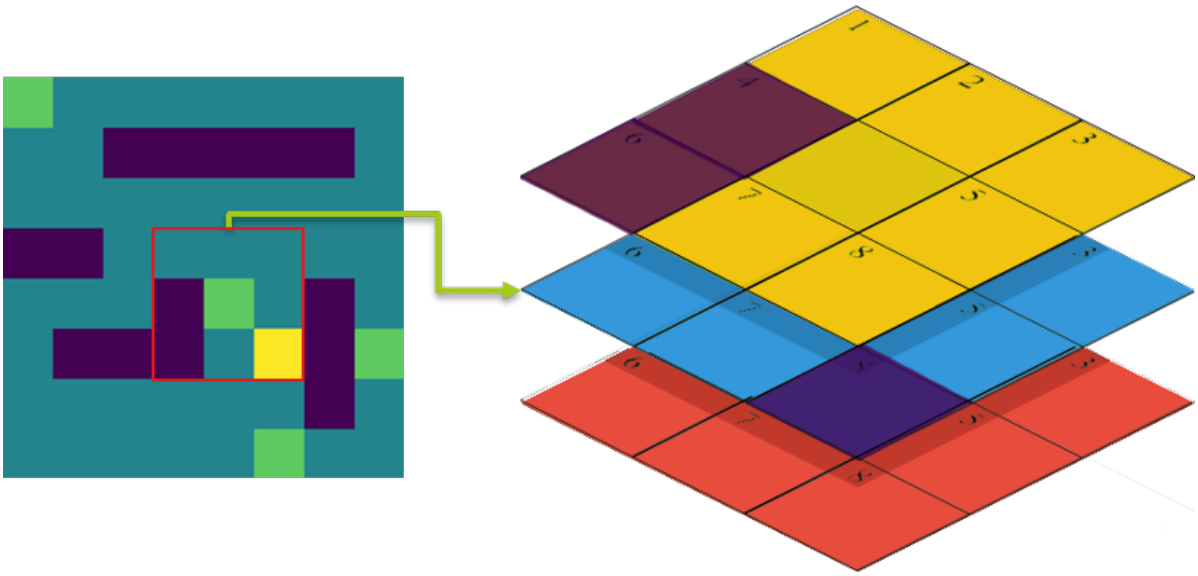


Figure 4.6: Agent's vision: in a 3 × 3 vision grid, the agent sees three layers. Top yellow layer shows walls, middle blue layer shows rewards, bottom red layer shows other agents. The agent itself is always centred in this observation grid.

Vision System To make the limited vision system concrete on this map context agents are fed observations derived from their own respective locations. Firstly, every agent receives three layers of observations. Each containing different information. Layer 1 contains observations regarding the presence of walls in its local field of view. Layer 2 contains possible rewards in view and layer 3 contains other agents. Based on the principle of having only local vision, the vision is created by returning every layer as a square, of for example a 3 × 3 pixel size, and centre this square around the agent, then this would imply that a 3 × 3 observation size would yield a view range of 1 pixel in either direction, since the agent itself is located in the middle pixel. A 5 × 5 Observation size would yield a 2-pixel view range is either direction, and so on. Naturally, one cannot have an even number for the observation shape as an agent can then not occupy the centre pixel. This therefore creates a local vision system for each agent respectively.

Another aspect of the vision system is that it is capable of returning observations of past steps, showing the historic observations. This is done to allow agents to act based on a simple form of memory. The full observation input is therefore of size: observation space = $L \times N \times N \times H$. With L being the amount of layers,

which remains fixed at 3, N being the local observability window size and H the history depth, how many historic observations are fed back to the agent. N and H are hyper-parameters which will be tuned during the simulations. To be specific, an agent sees objects in their respective layers as an array containing 1's and 0's. 1's to signify objects and 0's for traversable and free terrain. The outer perimeter of the map is an 'infinitely' thick wall. For example: an agent located at the edge that has a view range of 4 pixels, will see a 4 pixel thick wall in that direction. This preserves the input dimension size such that it can remain constant, regardless of location.

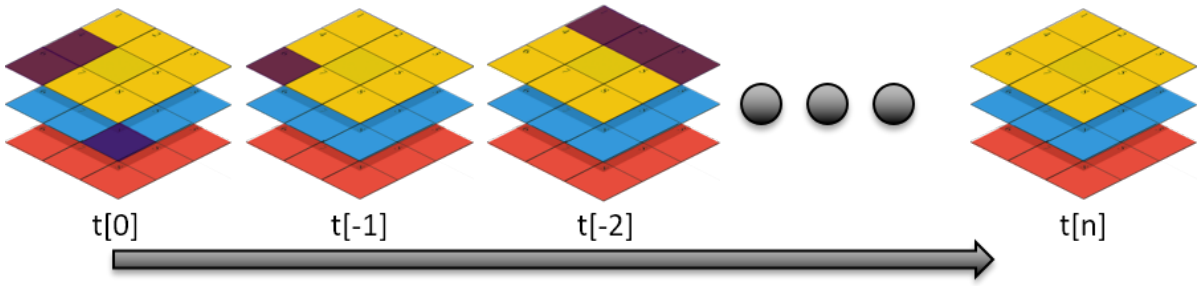


Figure 4.7

Messaging System The environment supports any input into this messaging channel. It can be a piece of text, a number or anything. Its sole requirement is that the amount of messages received is equal to the amount of agents active in the system. Once the system received the list of messages, it will shuffle the order of these messages and send them back. This implies that agents have no control over who receives their messages and that an agent may receive its own message back. This system is designed based on the principle of simplicity. One message per agent to send and one message per agent to receive. Which message is received is fully random.

The content of the messages is therefore chosen by the agents that use them. It is discussed in 4.2: Agent.

Benchmark Tasks

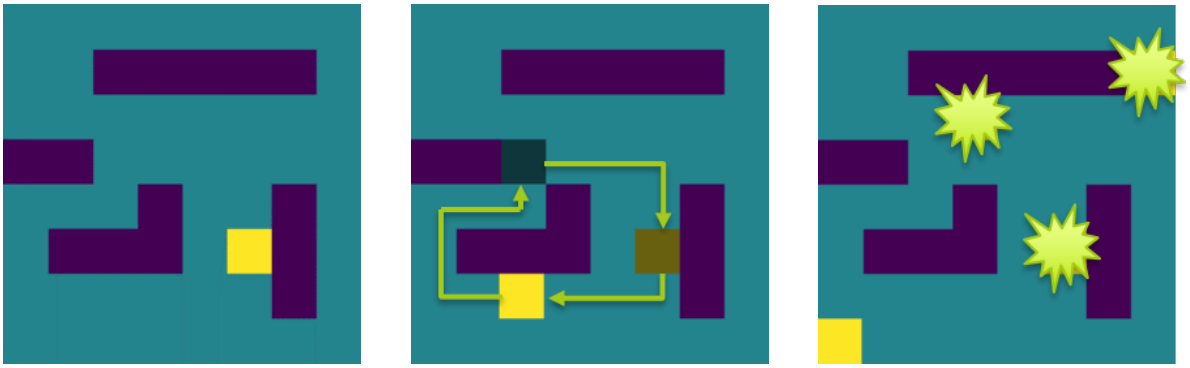


Figure 4.8: from left to right: Task 1, 2 and 3. Yellow box indicates the reward location. Off-colour boxes indicate previous or future reward locations. Arrows are drawn to signify a predictable rotating pattern of reward locations in task 2. Task 3's flashes indicate that the reward can disappear and reappear anywhere on the map. Note that for Task 2 and 3, the reward only changes location once an agent has reached it and triggered the reward sub system.

Benchmark Task 1 The first assignment will consist of simply reaching a goal. All agent have the same goal. the only interaction between the agents will be that the agents are potentially within vision of each other and that these agents can possibly collide. This is to see if the proposed algorithm in section 2.5 has any positive effect in the simple case.

Benchmark Task 2 The second task involves all agent in the field trying to reach a goal. The initial position of the goal is the same as task 1, however once any agent reaches the goal, the goal changes location. There are three locations where the goal can spawn. This change of location happens in a rotating, predictable fashion. This creates a situation where agent have to race against each other and the task becomes competitive.

Benchmark Task 3 The third and final option considered is random spawn locations of the goal where agents compete to reach the location first. This is a similar situation to environment 2, however it is different in that the rewards' change of location is unpredictable. It may be located anywhere and does not follow a set pattern.

4.1.2. Environment Parameters

An overview of all the values used on the map. This is to centralise all the values used in the environment and provide a clear overview of all the parameters. These tables are explained in their respective paragraphs.

Rewards and Penalties The chosen values are based on the same principles as most of this the report: simplicity. Each type of feedback is an order of magnitude higher or lower than the next. This is done to allow a good differentiation between reward types, making a TD-error prioritisation based approach able to distinguish learning different types of rewards and more easily prioritise learning rewarding transitions.

Rewards and Penalties		
Type	Value	Trigger
Reward	+10	Reward System
Collision	-1	Collision System
Movement	-0.1	Every Iteration

Table 4.1: Tables are described in their respective paragraphs: 1) 4.1.1: Collision System & Reward System and 2) Vision System.

Reward Locations The location of the reward for Task 1 was chosen to be near the centre, but no more thought was put into this process. Task 2's reward locations were chosen to be within vision of a 9×9 agent if the agent would be at the location of one of the other possible reward locations. Thereby allowing agents to know here to go. Table 4.2 shows reward locations on the $[X, Y]$-coordinate system starting top left of map 4.3, starting at 0's.

Rewards Locations	
Benchmark Task	Coordinates
1	[5,5]
2	[5,5], [2,6], [2,3]
3	Random

Table 4.2: Chosen locations for reward placement.

Vision Given the definitions by 4.1.1: Vision System, the considered observation space and therefore the DQN input layer size is shown in table 4.3.

Layers	View Range	Memory	Observation Space	Action Space
3	3×3	1	27	4
3	3×3	5	135	4
3	9×9	1	243	4
3	9×9	5	1215	4

Table 4.3: Resulting Observation and Action Spaces of the environment.

4.2. Agent

4.2.1. Method

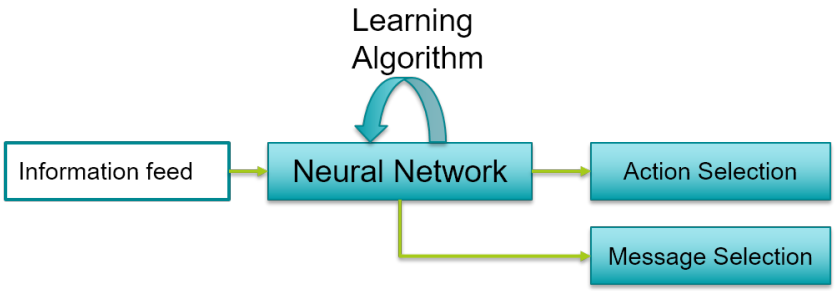


Figure 4.9

The basic flow of information in shown in figure 4.9. At the very centre of the agent lies a Deep Q-Network(DQN). The functioning of an DQN agent is described in 2.4: Deep Q-Network. On top of this a prioritised queue is established to provide a sense of sorting important replay memory entries from others. This sorting is described in 2.5: Prioritised Experience Replay. It is this concept of sorting that is now used for the communication channel and to create variations of the baseline prioritised DQN agent. In this section, previously discussed algorithms are combined to form a complete agent. As shown in figure 4.9, there are 4 main parts in an agent. The central processor (A neural network in this case), a learning algorithm, an action selector and for this report's purpose: a message selector.

Figure 4.10 shows the chosen algorithms for each part of the agent. Firstly, information is directly fed into the DQN. Based on the output of the network, an action is selected through the ϵ-greedy policy. The message selection happens through prioritisation. All of which is described in 2: Background.

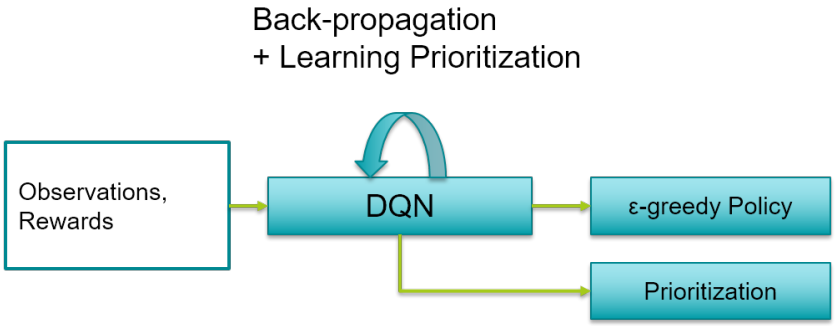


Figure 4.10

DQN As stated in 4.1.1: Vision System the observation space is determined by the chosen vision size, which are in turn determined by the hyper-parameters N and H. The size of this observation space directly fixes the size of the DQN's input and is therefor always the same size as the observation space. The internal layout of the Deep Q-network has 2 fully connected hidden layers, which are fixed in table 4.5, and an output dimension equal to the action space of the environment, which in this system is 4, see 4.1.1: States and Movement System. The output layer does not use biases.

Message Sampling Agents that communicate will do so by selecting transitions stored in the replay buffer. a [$state$, $action$, $next\_state$, $reward$]-set is transmitted through a selection procedure. The difference between each type of agent is discussed in the following paragraph.

Type	**Method of Learning**	**Method of Communication**	**Acronym**
0	Prioritised Sampling	None	PS-None
1	Prioritised Sampling	Uniform Sampling	PS-US
2	Prioritised Sampling	Prioritised Sampling	PS-PS
3	Prioritised Sampling	Prioritised to Uniform Sampling over time	PS-PUS
4	Prioritised to Uniform Sampling over time	Prioritised Sampling	PUS-PS

Agent Variations There are 5 different types of agents tested in this report. Firstly a baseline agent which has no messaging functionality. After this, 4 other messaging agents are considered.

Type 3 and 4 modify type 2 by linearly reducing the chance of prioritised sampling of either messaging or learning, respectively, as shown in figure 4.11b. This is with respect to amount of steps made during a simulation.

4.2.2. Agent Parameters

To centralise all the values used in this report, they are all put in this subsection.

Deep Q-Network		**Initialisation**				
Layer	nodes	Weights		Biases		Activation Function
		μ	σ	μ	σ	
Input	*Observation Space*					
Hidden 1	64	0	0.0001	0	0.0001	Leaky relu ($\alpha = 0.2$)
Hidden 2	12	0	0.0001	0	0.0001	Leaky relu ($\alpha = 0.2$)
Output	*Action Space*	0	0.0001	*None*		*None*
		Truncated Normal Distribution				

Table 4.4: Deep Q-Network Connections and initialisation. All based on fully-connected layers.

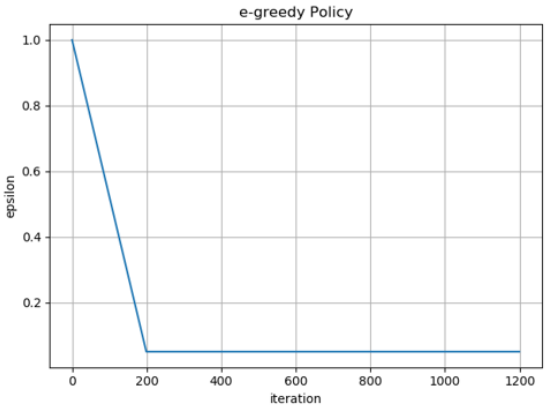
The initialisation of the weights and biases is based on a zero-mean(μ) truncated normal distribution with a very small standard deviation(σ). The standard deviation is chosen as such to be able to create unique agents but not predetermine agents too much by their initialisation.

Hyper-parameter	-
Replay Buffer	50
Learning Rate	0.0003
Discount Factor (γ)	0.9
Temporal Difference (ϵ)	0.1

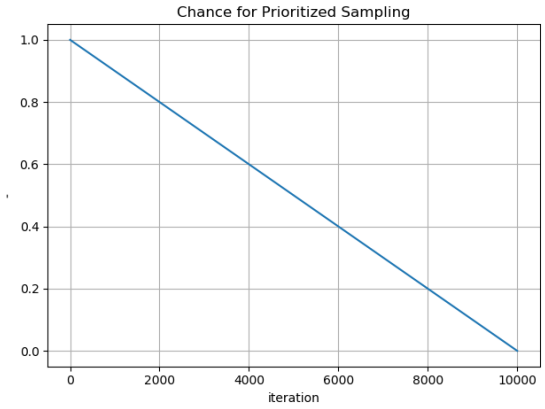
Rewards Locations	
Benchmark Task	Coordinates
1	[5,5]
2	[5,5], [2,6], [2,3]
3	Random

Table 4.5: Agent hyper-parameters. For an explanation of what these parameters affect, see 2: Background.

A small replay buffer to allow for the prioritisation sampling to take place, but not too big for information to become outdated. In a small gridworld with a size of 8×8, 50 will be plenty. The learning rate and other values of this table were chosen empirically while conducting this research. The ϵ-greedy policy starts at an ϵ of 1, which linearly decays to 0.05 within 200 steps and then stays constant at 0.05. The prioritised sampling chances decay linearly from 1 to 0 over 10.000 steps. The decay function is shown in figure 4.11b, along with the ϵ-greedy policy's decay in figure 4.11a.



(a) decay of ϵ in the ϵ-greedy policy.



(b) decay of prioritisation

4.3. Performance Metrics

To judge the performance of the proposed communication system, there needs to be some criteria set. The chosen metrics are based on two core principles.

Firstly, the average performance of the group will be measured. This will be done after 50 simulation runs each containing 50.000 steps with each simulation. These numbers were based on maximising both values but had to stay within computational constraints. There is no other reason not to increase either number.

The average performance will be defined as the average reward of across all agents for the last 200 steps. Please note that to avoid boundary effects, the moving average only starts being calculated once an agent has made at least 200 steps.

Secondly, the inter-agent variability will also be measured. This is defined as the difference between the best and the worst performing agents. Best and worst are chosen as the first of the active agents to achieve an average reward that's higher than a threshold, worst being the last agent of the group to reach that threshold. This is defined as the reward threshold. Another way of measuring inter-agent variability is to look into possible differences in behaviour of agents. A possibility is that due to the divergent nature of non-communicating agents, that these will exhibit different behaviour if compared with each other internally, as communicating agents might convergence to the same policy as they share the same memories. 5000 steps were chosen with the same argument as other metrics: The more the better, but one has to constrain it due to limited computational power. Given that the average distance an agent can be from the reward is around 8, giving 5000 steps would allow sufficient averaging of behaviour.

All hyper-parameters regarding performance metrics are shown in table 4.6.

Metrics	
simulations	50
agents per simulation	4
steps per simulation	50000
Reward Threshold	0.1
Moving Average Window size	200
Heatmap steps	5000

Table 4.6: Measurement method

To be able to explain the performance differences between agents with little vision versus agents with a lot of vision, heatmaps are generated from fully trained agents. After the set of 4 agents have been trained for the full duration of the simulation, they are then tasked with solving the tasks with the learning part of the algorithm turned off. This will then show if agents tend to get stuck in the event that they are provided too little information to solve a problem, or if the agents are able to solve the task. A proper heatmap should show higher values the closer the location is to the goal, as agents tend to funnel towards the goal, leading to a higher prevalence at locations close to it.

4.4. Expectations

It is expected that the average performance metric shows that communication helps the learning by a significant, measurable amount. As important information gets passed around, agents will be able to learn from more valuable information. The average reward would attain higher values along the course of learning.

The inter-agent differences should show a more homogeneous set of agents if they share information. Since agents that communicate learn from similar experiences, it is assumed that they will tend to converge to each other in terms of behaviour. The non-communicating agents should not have such behaviour and might diverge, leading to noticeable differences in behaviour with respect to each other.

5 Results

The results of each individual challenge are shown in their separate sections. In each section the differences in performance between the agent types are discussed along with other relevant metrics as discussed previously.

5.1. Benchmark Task 1

Challenge 1 was the static challenge with a mutual non-competing goal with only collisions making it a multi-agent problem. The results of all the options of hyper-parameters are now discussed in the following subsections. For more on the challenge itself see 4.1.1: Benchmark Tasks.

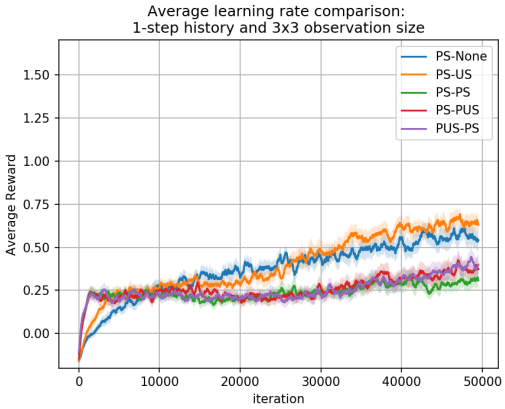
To keep the description and the figure in the same page for convenience, it is moved to the next page.

5.1.1. Average Performance

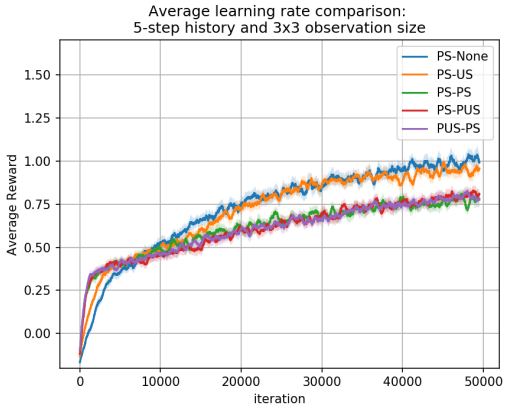
The average rewards over all 4 agents over the course of training is shown in figure 5.1. Let's break the figure down into it's sub-components and discuss each graph individually when needed. Firstly, figure 5.1a. This can be further broken down into the transient phase and the steady-state phase.

In the transient phases of figure 5.1a one can see that agents who share messages in a prioritised fashion quickly reach an average return of 0.25, but then flattens out. The non-communicating agent and the randomly communicating agent (type-0 and 1 respectively) are slower in this transient phase.

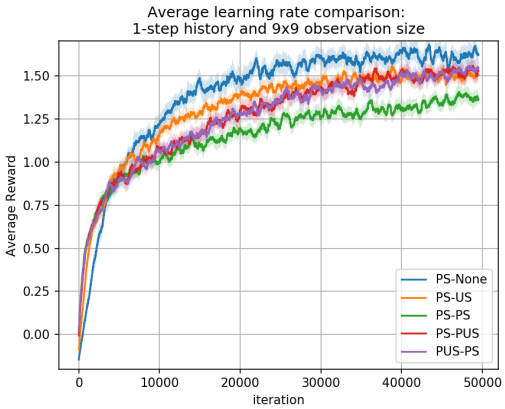
Some iterations later these slower types catch up and overtake the prioritising agents. Ultimately reaching a higher average return than prioritising at the end of the learning cycle. Also an observation that can be made is that the double prioritisation agent (both learning as messaging) tends to perform worse than any other agent for large observation spaces. Figure 5.1c and 5.1d show this. Otherwise the double prioritisation agent performs similarly as the annealing agents. These trends hold true for all graphs in figure 5.1 but are the most clear in figure 5.1a, however figure 5.1c and 5.1d show that even a randomly messaging agent (type-1) ends with a lower average return of reward than an agent who does not communicate at all.



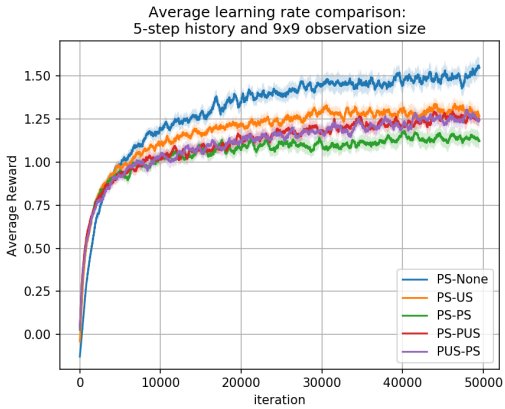
(a) Performance comparison with limited view range.



(b) Performance comparison with limited view range and historic observations.



(c) Performance comparison with nearly full view range.



(d) Performance comparison with nearly full view range and historic observations.

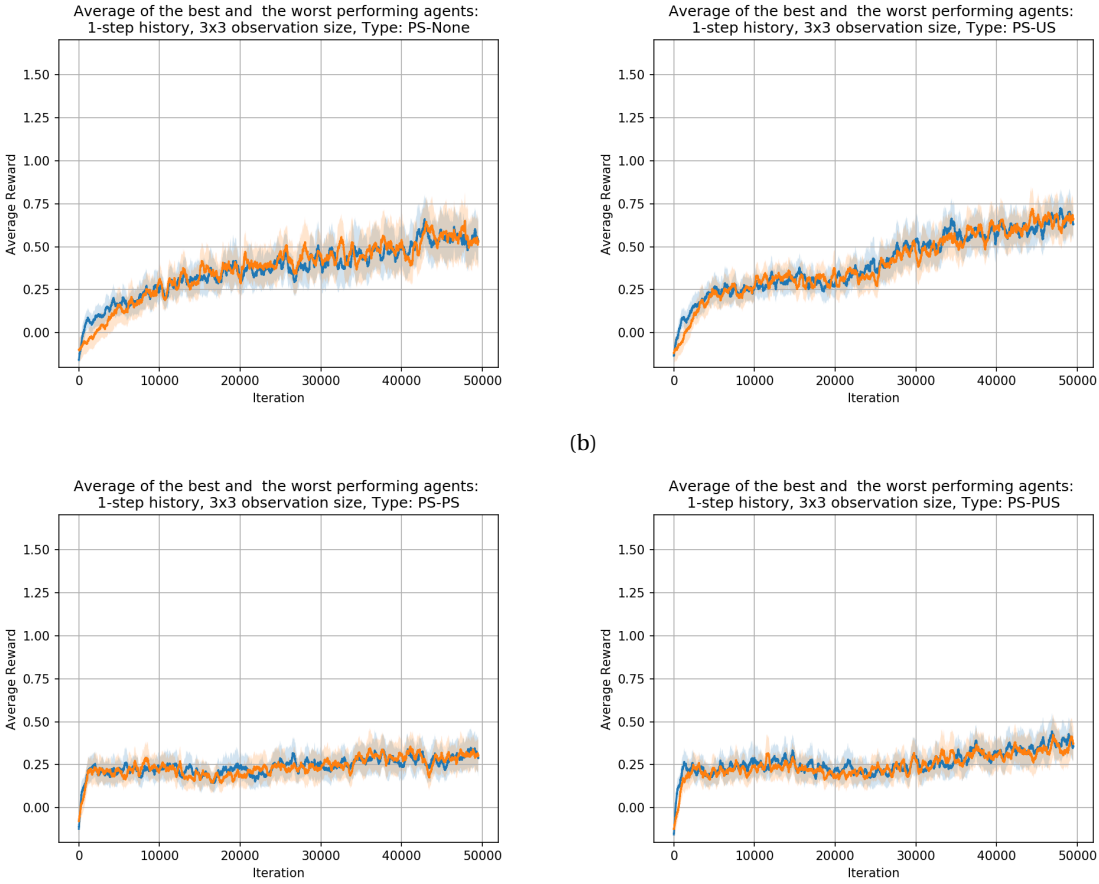
Figure 5.1

5.1.2. Inter-Agent Performance

To compare the performance between agents, the threshold criterion is used, as described in 4.6: Measurement method. Not all the graphs are shown in this subsection due to the fact of there being too many, leading to clutter. Every vision option tests 5 different agents. This means that with 4 vision options there are $4 \times 5 = 20$ graphs to be shown for Challenge 1. The remarkable ones are shown here and the rest can be found in 8: Appendix. In the appendix one can see that the behaviour of the discussed graphs are similar to the ones shown in the appendix. The chosen graphs to be shown here are concerning the $3 \times 3 \times 1$-case as an example.

In figure 5.2 one can see that there is a difference in learning speeds for the non-communicating and the randomly communicating agents (type-0 and 1) compared with the agents that communicate important messages. For those agents, they learn at approximately equal speed, leading to a more uniform learning rate across the board for all agents. For the other graphs, the difference becomes marginal. However, only differences at the start can be seen. After some learning has been done, the two agents converge and learn at approximately equal rate leading to no distinguishable difference using this proposed metric, perhaps if other metrics were used could behavioural differences be detected.

Please note that these graphs show the best and worst agent of a set of 4 agents that were active in a simulation. Information regarding the two agents who performed average, according to the criterion, is discarded. After this filter, the average of 50 best and 50 worst agents is computed, leading to the two lines.



(a) (b)

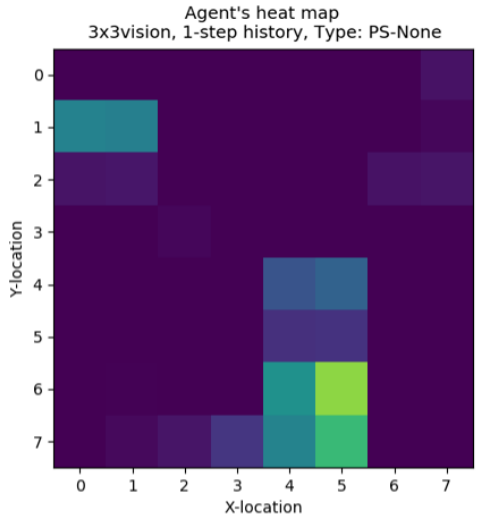
(c) (d)

Figure 5.2

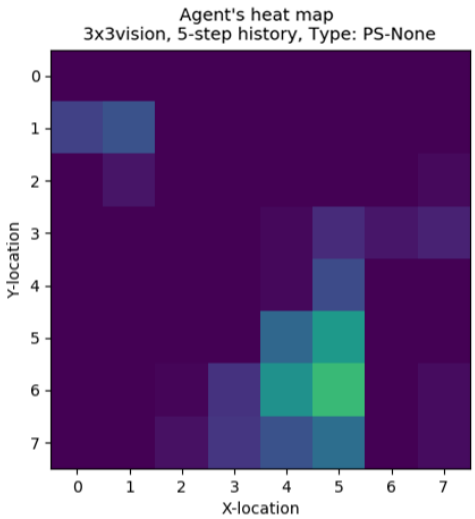
5.1.3. Agent Routing

The heatmaps are shown in figure 5.3. The chosen heatmaps are based on PS-None heatmaps for the same reason as the paragraph before. PS-None was selected as it was consistently the best performing agent. Other heatmaps are shown in the appendix.

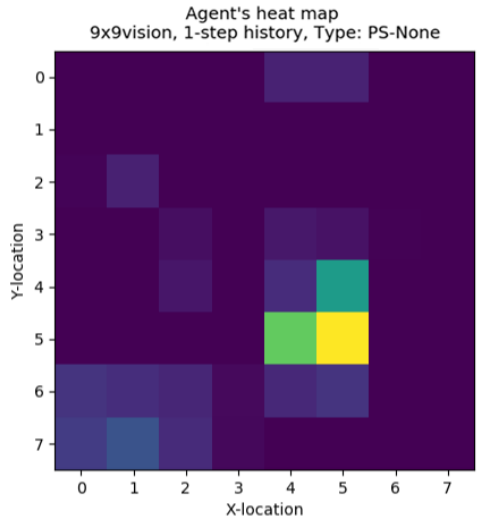
Except for small artefacts of location pairs where agents could get stuck, one can clearly see that agents focus around the location of the reward. Even small vision agents produced heatmaps which shows that they managed to find their way to the location of the goal. One can attribute a higher return for the higher view range agents based on the fact that these high vision agents can more easily find shortest paths.



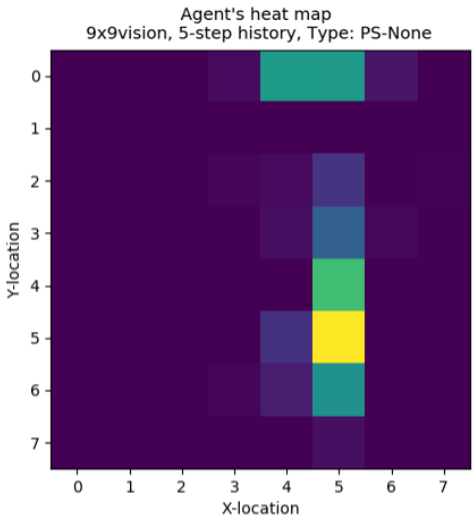
(a)



(b)



(c)



(d)

Figure 5.3

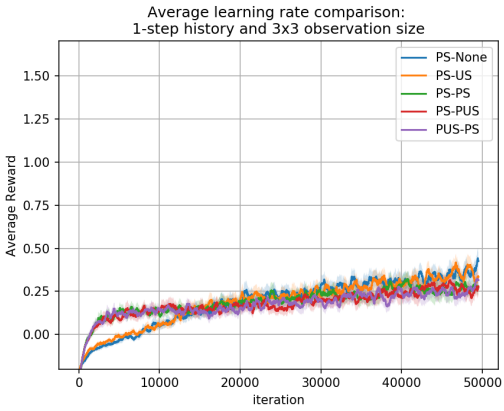
5.2. Benchmark Task 2

Challenge 2 was the rotating reward challenge where agents race to reach the goal. This challenge is competitive. For more information see 4.1.1: Benchmark Tasks.

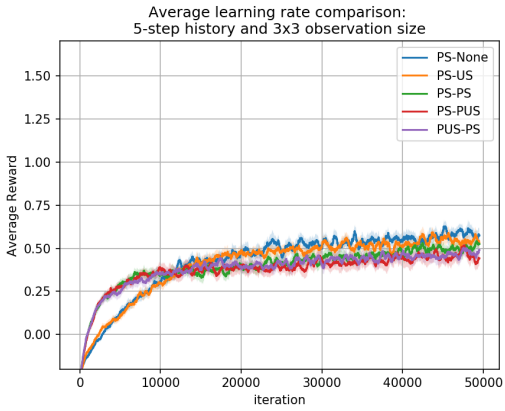
5.2.1. Average Performance

The discrepancy between on one side non-communicating or randomly communicating agents and prioritised communicating agents became larger in this challenge for the transient phase. One can clearly see that for a very limited view range, $3\times3\times1$ and $3\times3\times5$, prioritisation of communication helps speed up the learning up to a plateau. This effect is also visible for the larger observation space agents, but is less profound.

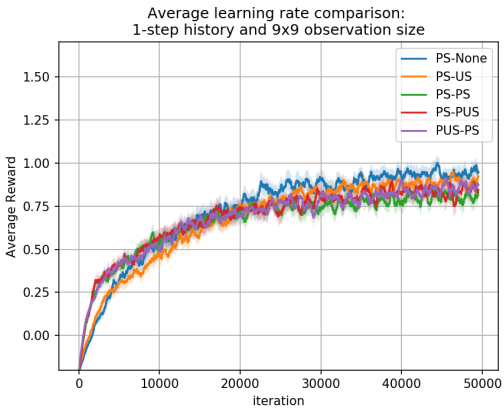
For the final phase of the learning curve one can again see that an agent that does not communicate at all will out-perform all other agents, but to a lesser extend.



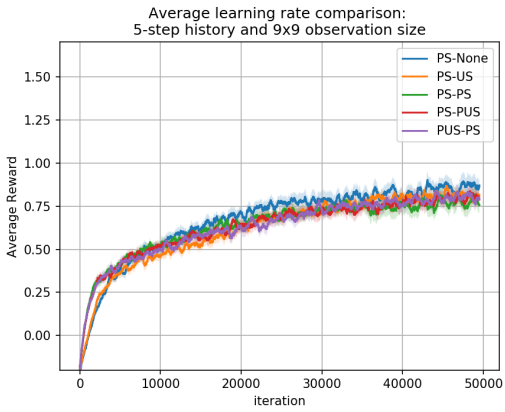
(a) Performance comparison with limited view range.



(b) Performance comparison with limited view range and historic observations.



(c) Performance comparison with nearly full view range.

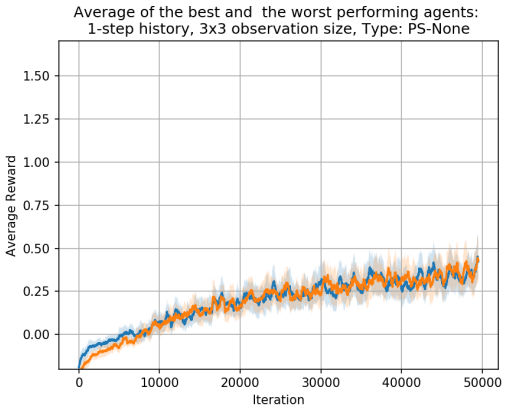


(d) Performance comparison with nearly full view range and historic observations.

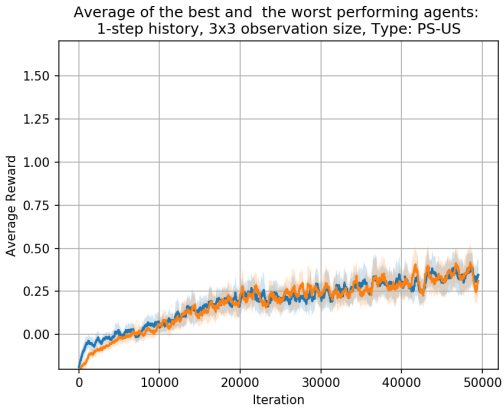
Figure 5.4

5.2.2. Inter-Agent Performance

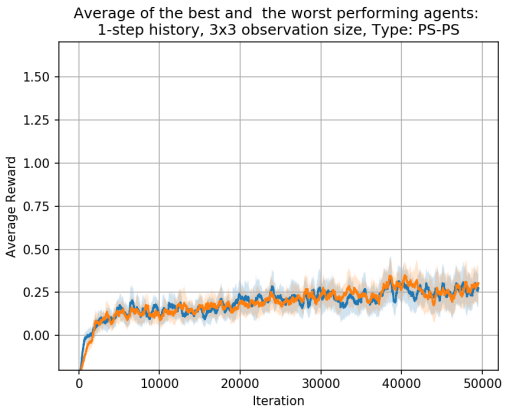
A marginal difference can be seen when applying the threshold criterion to separate slow and fast learners. Figure 5.5a shows that difference in the transient phase. To provide consistency, again only a $3 \times 3 \times 1$ view range agent is shown and for the other 4 sets of 4 graphs one can see 8: Appendix. Other observations are not different from challenge 1's observations.



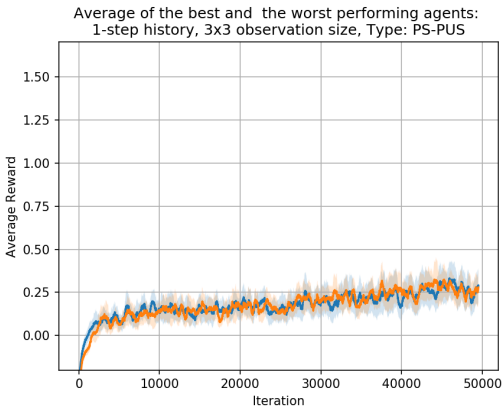
(a) Performance comparison with limited view range.



(b) Performance comparison with limited view range and historic observations.



(c) Performance comparison with nearly full view range.



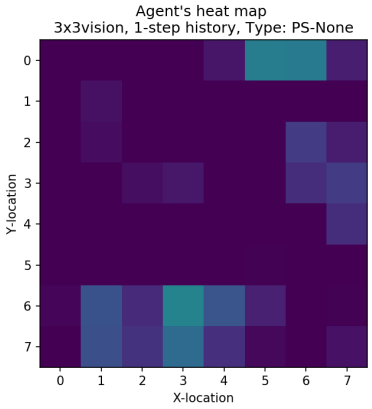
(d) Performance comparison with nearly full view range and historic observations.

Figure 5.5

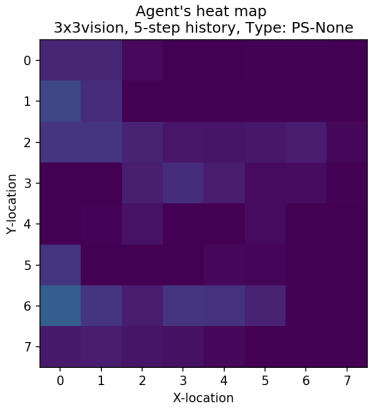
5.2.3. Agent Routing

Figure 5.6 shows heatmaps generated for a PS-None agent. PS-None agents was chosen to be represented here as it has proven consistently to out-perform other algorithms. The other heatmaps are shown in 8: Appendix. For figure 5.6a one can clearly see that the agents get stuck. Since agents are not allowed to stand still in this environment, a signature of agents getting stuck is isolated pairs of pixels. In those pairs, agents oscillate back and forth and never manage to leave the area. This can be due to under-training or due to not having enough information to make an optimal decision, with optimal being defined as the decision which is part of the shortest path to the goal.

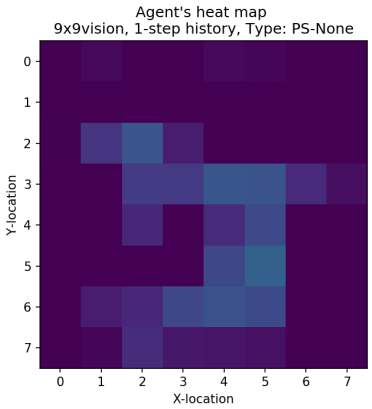
This observation does not hold for the other three figures. There one can see relatively smooth maps that tend to be more focused on the centre. Figure 5.6b shows that although agents get stuck less, it does not seem to converge toward goal locations. It seems to wander around without a sense of understanding which locations are more important. This is on the other hand clearly visible in figure 5.6c and figure 5.6d. There it clearly shows that agents move towards the centre, spend as little time possible in the outskirts of the map, since the rewards never spawn there, and focus on locations close to the rewards.



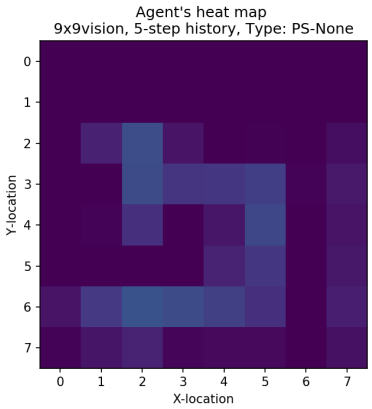
(a)



(b)



(c)



(d)

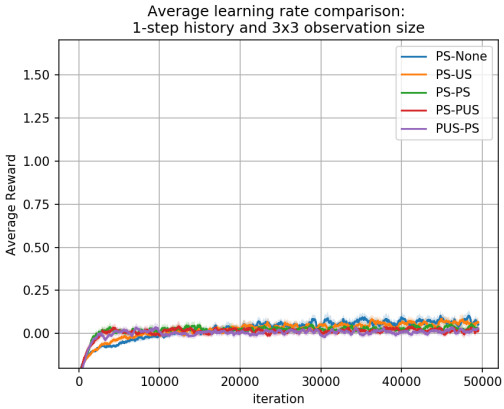
Figure 5.6

5.3. Benchmark Task 3

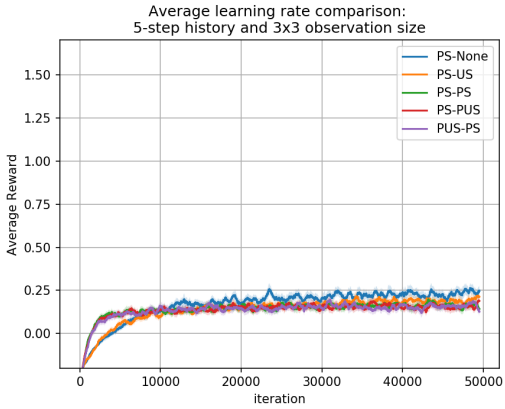
Challenge 3 was the randomly spawning rewards. This challenge is competitive. For more information see 4.1.1: Benchmark Tasks.

5.3.1. Average Performance

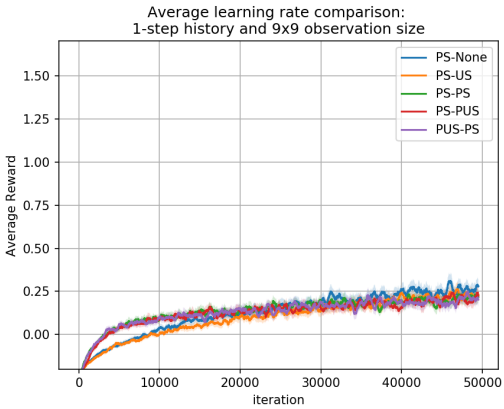
it is immediately clear that the performance of all the agents in this challenge is remarkably lower than in other challenges. The average performance across all agents for every option of vision is shown in figure 5.7. The transient phase of these agents show a clear distinction visible, again, for prioritising agents to learn faster initially. This advantage is lost halfway and by the end of the learning communication-less agents outperform the other agents by a small margin. This trend holds true for all observation space sizes.



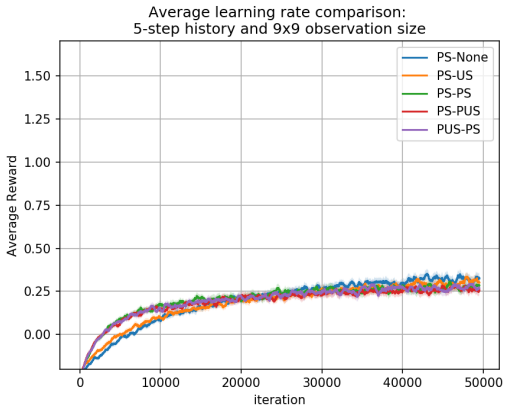
(a) Performance comparison with limited view range.



(b) Performance comparison with limited view range and historic observations.



(c) Performance comparison with nearly full view range.

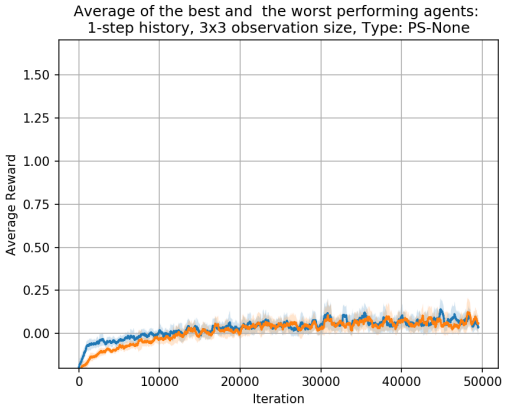


(d) Performance comparison with nearly full view range and historic observations.

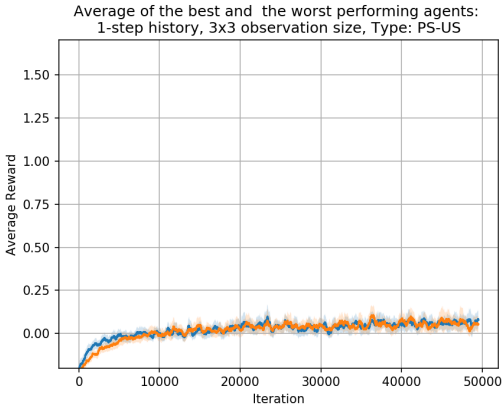
Figure 5.7

5.3.2. Inter-Agent Performance

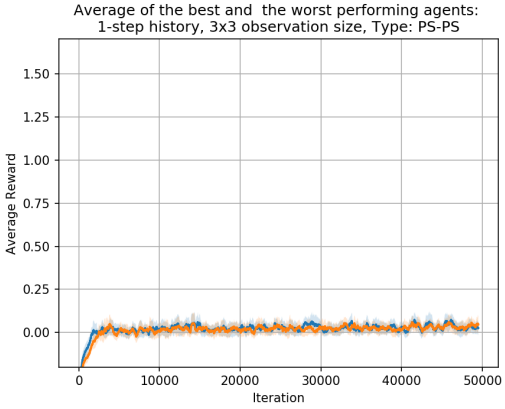
The same observations also hold true for challenge 3 as with challenge 1 and 2. The differences between learning rates of the defined "best" and "worst" agents are visible in the transient phase of the learning curve, after which, it converges and the agents' performances become indistinguishable from one another.



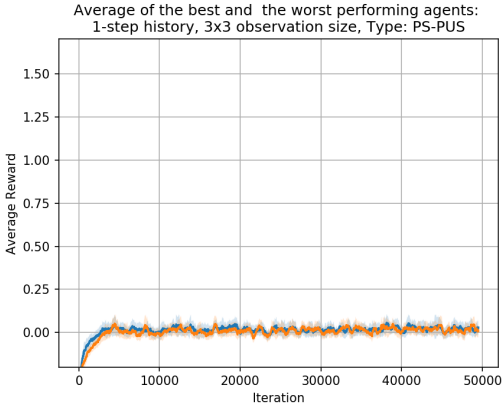
(a) Performance comparison with limited view range.



(b) Performance comparison with limited view range and historic observations.



(c) Performance comparison with nearly full view range.



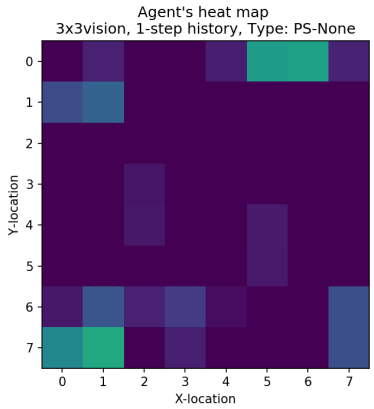
(d) Performance comparison with nearly full view range and historic observations.

Figure 5.8

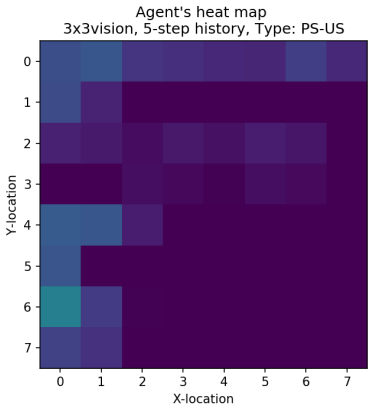
5.3.3. Agent Routing

One would expect a smooth heatmap without any peaks if agents were able to solve randomly located reward locations. This due to the fact that agents would trigger a reward, then the reward would move to a new place and an agent would then in turn trigger that reward. If this happens sufficiently often, it would lead to a smooth heatmap. It may have a slight focus on the centre as that would either be a corridor towards the goal or the goal itself. Side lines are less likely to be a corridor for the shortest path towards the reward.

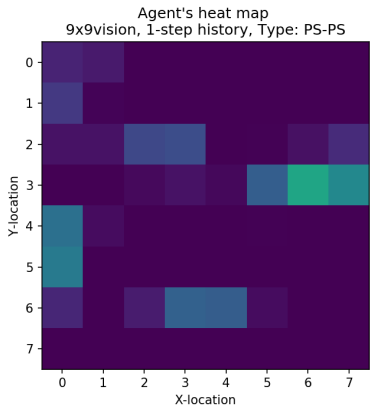
It does not seem to be the case that such a predicted heatmap was generated. The heatmaps for the best performing agents (type-0) are shown in sub-figures of figure 5.9. The agents seem to desire to stay in certain locations. Figure 5.9d seems to do decently smooth, relatively speaking to the other sub-figures, however a glance at figure 5.7b shows that this agent still did not receive a high return rate of rewards.



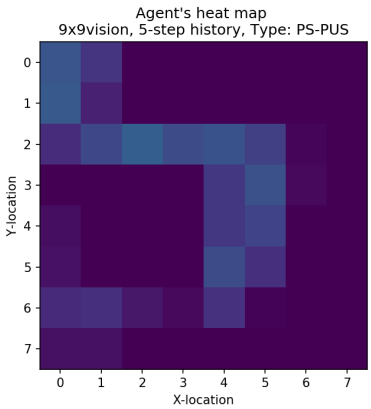
(a)



(b)



(c)



(d)

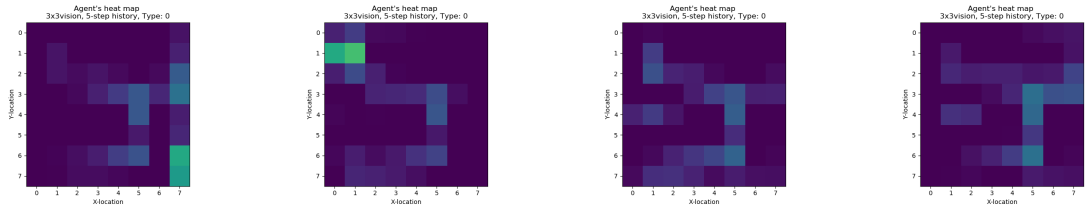
Figure 5.9

6

Discussion

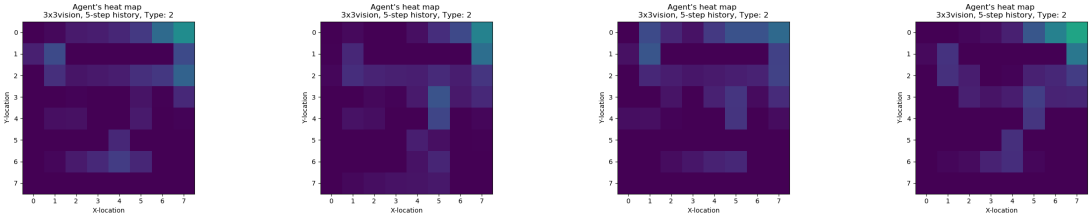
6.1. Performance Gap

The average performance difference between the agents in figure 5.1 shows a clear distinction between the non-communicating type and other types of agents. This could indicate an intrinsic problem to sharing information. An important issue with sharing information is that it promotes convergence. Agents will have less chance to diversify due to the fact that communicating agents will learn from each other and thus learn from similar memories. In the case of a simple non-competitive goal, there is still a multi-agent element that could lead to a higher average return if agents work on different policies. Since collisions are possible, one can imagine that agents with different policies regarding routing and collision avoidance (taking priority or backing off) could work better together.



(a) Agent 0 (b) Agent 1 (c) Agent 2 (d) Agent 3

Figure 6.1: Separate heatmaps for each agent that was active in the environment.
These sub-figures concern Non-communicating (PS-None) agents



(a) Agent 0 (b) Agent 1 (c) Agent 2 (d) Agent 3

Figure 6.2: These sub-figures concern prioritised communicating (PS-PS) agents

Figure 6.1 and figure 6.2 show the differences in behaviour for each individual agent in the first benchmark for the $3 \times 3 \times 5$-vision case between non-communicating(PS-None) and prioritised communicating(PS-PS) agents. One can see that the PS-None-agents have diversified more than the PS-PS agents. The difference in policy is visible in the shown heatmaps. The agents who prioritise their messages end up converging to the same policy. One can see that in figure 6.2, the heatmaps are more similar. One can now argue that this

convergence now inhibits the learning speed of the agents, since the agents will get in each other's way and due to the fact that they share their experiences, will not diverge in their policies.

As regarding the early learning of prioritised messaging agents, one can argue that prioritisation leads to agents quickly learning every transition which leads to a penalty, since unexpected penalties lead to high priority in learning. These surprising transitions will get transmitted, received and transmitted again, leading to an echo chamber of the same memories going back and forth between agents until those memories have successfully been learned by the agents and no longer produce a high TD-Error. At which point it will slide out of the collective memory buffer since other memories which have now become more important will occupy the space. This is a useful effect initially, however proves detrimental to the final performance.

The convergence of policies shows that the do-not-hit-wall part of the policy, which is the most abundantly triggered reward initially, can be identically learned for all agents. This can be a single-agent based policy. Prioritised learning seems to greatly assist this part of the learning process.

However, once diverging policies are needed for better performance, prioritisation of communication leads to agents not being allowed to diverge in policies since such agents tend to learn from similar transitions.

This performance gap is not so prevalent in other challenges as one can see in figure 5.4 and 5.7. The heatmaps also do not differ in such a clear way as in the first benchmark, these are shown in the appendix. From this one can say that policy heterogeneity is not as important of a feature to solve the other benchmarks.

6.2. PS-PUS vs PUS-PS Differences

In no benchmark does linearly reducing either the learning sampling or the messaging sampling make a difference with respect to each other. This implies that prioritised-to-uniform sampling can be applied in either case, weather it is selecting samples to learn or to transmit, the performance will be the same.

Its performance with respect to the other agent types lies in between complete random sampling of messages and fully prioritised sampling of messages. This is expected as these two form some concession of the two.

6.3. Computational Load

These agents run on different algorithms, therefore their computational load are different for the same number of iterations. There have not been a proper research conducted on this specifically, however, empirically I observed that the non-communicating agent is on average around 20% faster than other algorithms. Given this notion, not only does the PS-None offer a better learning pattern, it also does so faster. If one would allow this agent to train for the same amount of time as the others, then there would have been no contest in which agents perform better.

7

Future Work

7.1. Prioritise Only Common Policies

As described in the conclusion, training each other to converge to common policy is only detrimental if identical policies are not optimal. In the case where there exists sub-policies which can be shared along agents without obstructing the development of the main heterogeneous policy, then there could possibly be a benefit of prioritised memory sharing. As shown in the report, prioritised learning will very rapidly learn the homogeneous sub-policy of not colliding with a wall. This motivates a limit on communication.

7.2. Limiting Communication

One can argue that there exists a certain optimum that merges the best of both worlds. The initial rapid learning of prioritising agents combined with the higher average reward at the final phase of learning of non-communicating agents. This however leads to problem in terms of resulting behaviour of the agents. The initial rapid learning seems to coincide with convergence of agent policies, this exact convergence is detrimental in the long run where divergence is proven to be more successful at attaining higher rewards.

A possible solution to this is to limit the communication such that agents still do share important memories, but limit it in such a way that they do not end up cloning each other's policy. Some options to be considered:

Threshold Apply a TD-Error screening for every received message. Only store messages when they are sufficiently surprising, otherwise do not store the message. For a well chosen threshold this would imply that initially there will be communication since newly initialised neural networks will produce transitions with high TD-Errors, but over time this will decay and ultimately lead to a complete silence in communication as no transition becomes surprising anymore, therefore providing room for divergence in the later phase of the learning process. A suggested threshold function could be based on interpreting the replay buffer as a stochastic process and thereby fitting a continuous probability distribution to an agent's own buffer's TD-Errors. Then one can interpret an incoming message based on how unlikely this message is given the distribution fitted. This will lead to an agent screening the incoming messages before it decides to learn from them, thereby reducing the convergence rate, but hopefully still learn from important messages if they get received. The downside of this method is an increase in computational complexity.

Message Critic Since there seems to be a positive correlation between communication and convergence, one can limit the communication based on a metric that indirectly would reflect convergence. For example, assuming that the currently active agents apply prioritised messaging, this implies that the received message was likely to be surprising from the sender, however this says nothing about the surprise factor for the receiver (a high relative TD-Error value is defined as surprising). Since the receiver will calculate the message's TD-error himself, it can judge the value of the received message in relation to the average value of his buffer. This comparison can now be used to judge the value of receiving messages. For example, one can tune the ratio between received messages and own experiences to learn from based on the ratio of average TD-errors produced by messages versus the average TD-errors of own experiences. The choice between the two is then a chance of selection based on the ratio. This creates a natural filter if other agents continuously send messages which are of no value to you.

On/Off-Switch The simplest approach would be to simply switch off the possibility of communication after the initial phase of learning. This is a more direct approach of the same concept of applying threshold, with the same intentions.

7.3. Master - Student Hierarchy

This report has been based on randomly initialised neural networks (DQNs specifically) who learn simultaneously. However, this communication system that was discussed in this report could also prove to be helpful in training a new neural network if there already exists another neural network. Since trained DQNs will have mostly memory samples closely resembling the optimal policy and also contain more memory samples which are rewarding, it can be fed to the student DQN-agent. Again care has to be taken not to create a clone of the master agent. This could lead to methods allowing hot-swapping agents in an already active environment. For example, in systems where there are already agents active, such as a warehouse distribution system, one could then have the possibility to simply add more agents in the system. Since copying the neural network of an existing agent proved to be non-optimal, one would desire divergence in policies and the new agents "fill the gaps" for the other agents. By implementing a master - student system, the student could learn faster and, if proper care is taken to prevent convergence, could learn policies which are adapted to other agents.

8

Conclusion

Prioritised replay memory sharing has detrimental effects for multi-agent systems where inter-agent policy heterogeneity is beneficial.

The positive effects of prioritised replay memory sharing shown in the initial phases of the learning curves will seem good, better performance than the regular DQN-Algorithm is to be expected, however they only show due to the fact that identical policies in the case of obstacle avoidance work. The only thing the agent learns at the start of the learning phase is to not hit a wall. The policy needed to not hit walls can perfectly be identical across the agents without large detrimental effects on the optimal policy. Starting from here, it becomes beneficial to diverge in policies. Prioritised learning's problem is not allowing agents to diverge and find a better agent-specific optimal policies. This report has shown that prioritisation of messages forces agents to converge to the same final policy. Even though, due to the competing conventions problem, might still have entirely different neural network weights. This convergence to the same behaviour is detrimental for the final goal of training an agent, namely achieving the best performing agent at the end of the training.

The rapid initial learning of agents using prioritised experience replay sharing is a bonus that one would not easily want to discard, a method should be possible to allow inter-agent teaching and yet preserve policy differences.

Final Thoughts

The core idea of sharing important information sounds good. Similar to how humans share information by teaching, one would like to find a mathematical analogy. It is with this motivation that we should keep going with finding mathematical methods of implementing this idea, as the method attempted in this report has proven to not be fruitful. However, let's make conclusions in the opposite direction. Let's try to understand our own behaviour after having seen the results of information sharing in this report. These local optimums and convergence 'problems' discussed in this report do find an analogy in the human world, namely through culture. In this society we tend to learn from each other in case there is no other source of information. We will copy each other's behaviour when we are unable to find our own ways. This is most noticeable in the way we treat each other. In a world where cultures can differ greatly from each other, we learn from the people directly around us, we imitate their behaviour and converge to a similar mindset, much like the agents in this report.

Bibliography

[1] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *CoRR*, abs/1605.06676, 2016. URL http://arxiv.org/abs/1605.06676.

[2] Jakob N. Foerster, Nantas Nardelli, Gregory Farquhar, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *CoRR*, abs/1702.08887, 2017. URL http://arxiv.org/abs/1702.08887.

[3] Donald Olding. Hebb. *The Organization of Behavior. A neuropsychological theory*. John Wiley Sons, 1949.

[4] Tatsuya Kasai, Hiroshi Tenmoto, and Akimoto Kamiya. Learning of communication codes in multi-agent reinforcement learning problem. *2008 IEEE Conference on Soft Computing in Industrial Applications*, pages 1–6, 2008.

[5] Joel Z. Leibo, Vinícius Flores Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *CoRR*, abs/1702.03037, 2017. URL http://arxiv.org/abs/1702.03037.

[6] Francisco S. Melo and Manuela Veloso. Learning of coordination: Exploiting sparse interactions in multiagent systems. pages 773–780, 2009. URL http://dl.acm.org/citation.cfm?id=1558109.1558118.

[7] Marvin L. Minsky and Seymour Papert. Perceptrons and pattern recognition. Jan 1967. doi: 10.21236/ada078863.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

[9] Frank Rosenblatt. The perceptron: A perceiving and recognizing automaton (project para). report no. 85-460-1 by rosenblatt, frank - january, 1957, Jan 1957. URL https://www.biblio.com/book/perceptron-perceiving-recognizing-automaton-project-para/d/308182718.

[10] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015. URL http://arxiv.org/abs/1511.05952.

[11] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016. URL http://arxiv.org/abs/1605.07736.

Appendix

.1. Challenge 1

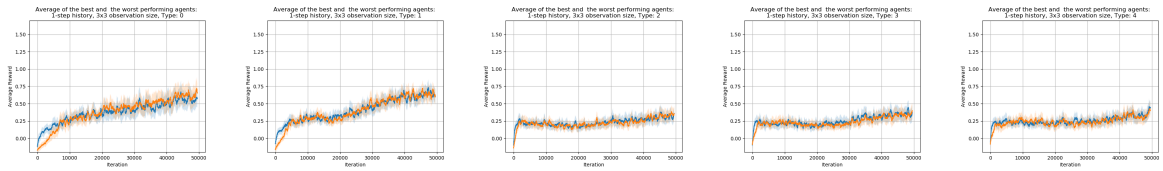


Figure 1: $3 \times 3 \times 1$ Vision

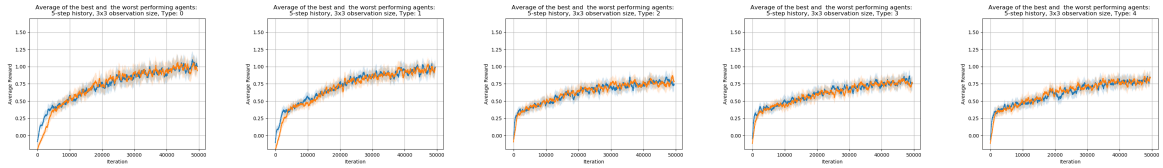


Figure 2: $3 \times 3 \times 5$ Vision

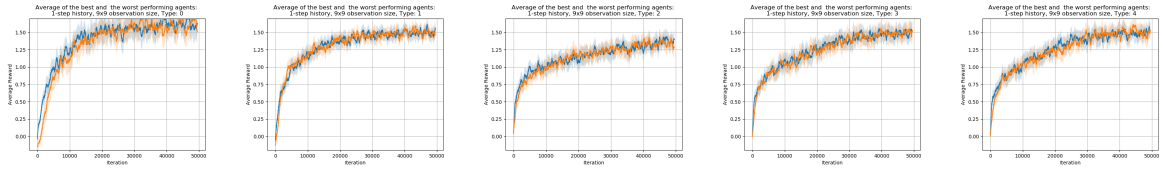


Figure 3: $9 \times 9 \times 1$ Vision

Figure 4: $9 \times 9 \times 5$ Vision

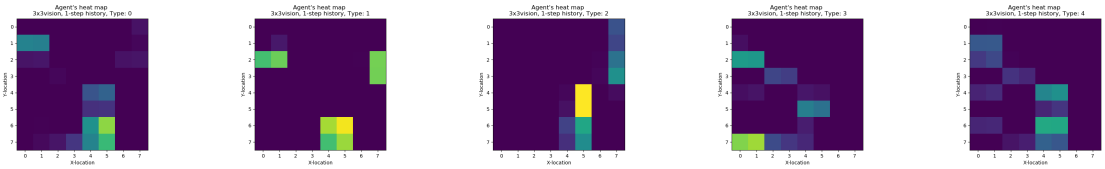


Figure 5: $3 \times 3 \times 1$ Vision

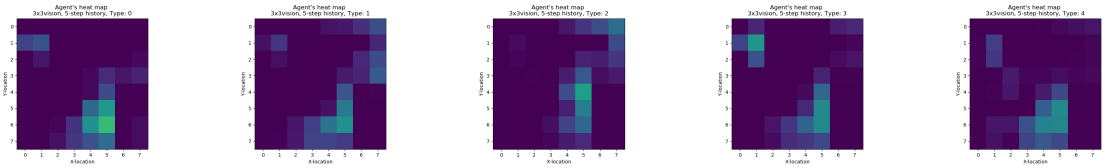


Figure 6: $3 \times 3 \times 5$ Vision

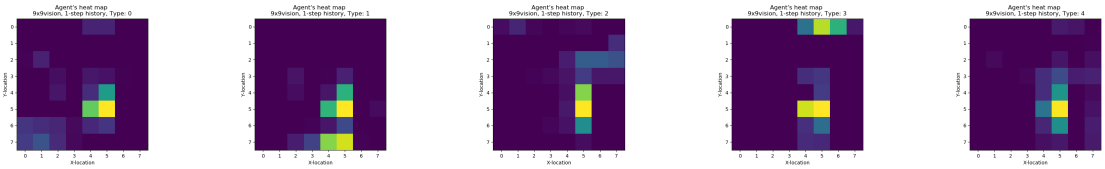


Figure 7: $9 \times 9 \times 1$ Vision

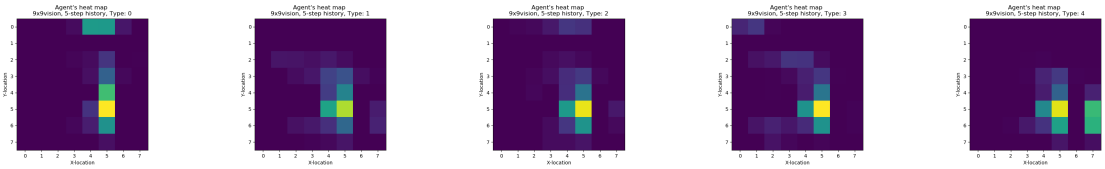


Figure 8: $9 \times 9 \times 5$ Vision

.2. Challenge 2

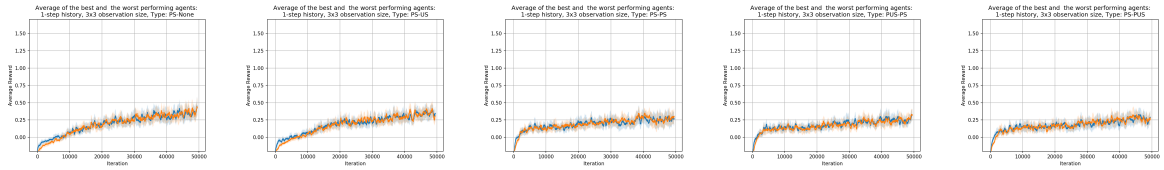


Figure 9: $3 \times 3 \times 1$ Vision

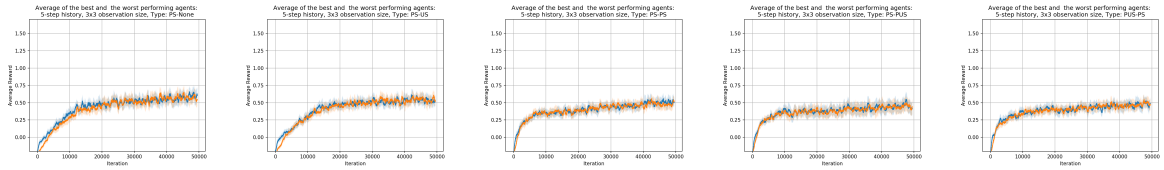


Figure 10: $3 \times 3 \times 5$ Vision

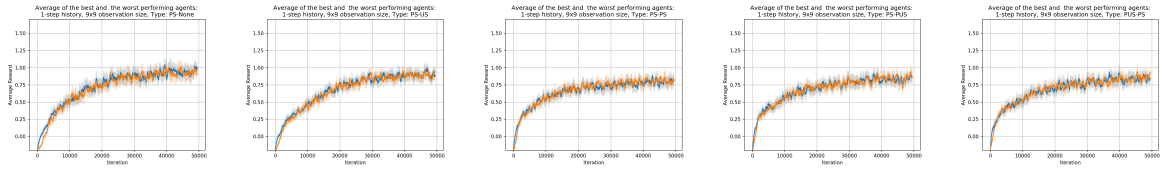


Figure 11: $9 \times 9 \times 1$ Vision

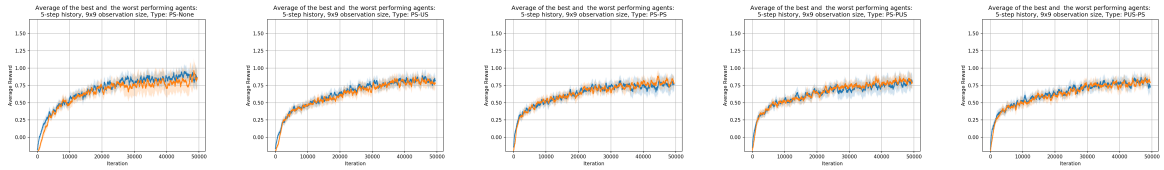


Figure 12: $9 \times 9 \times 5$ Vision

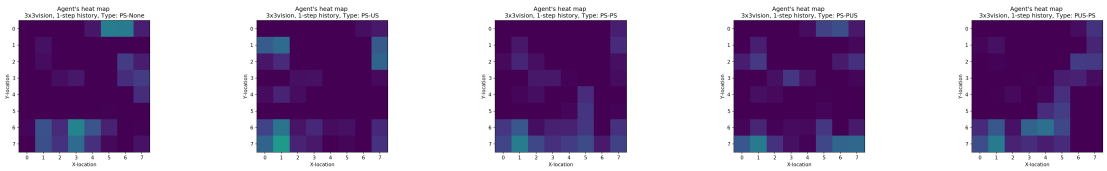


Figure 13: $3 \times 3 \times 1$ Vision

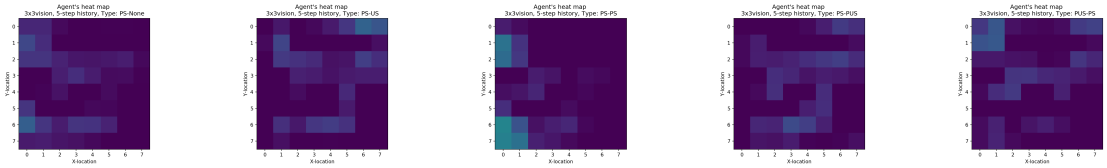
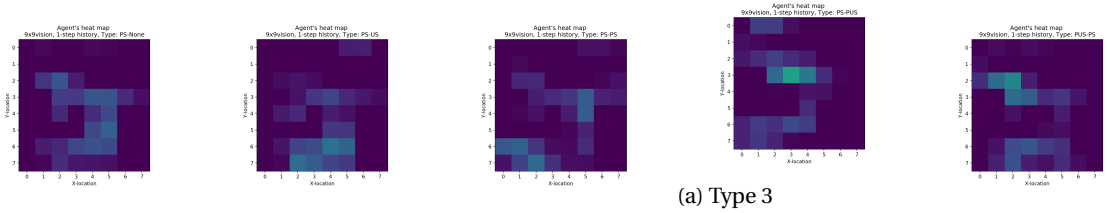


Figure 14: $3 \times 3 \times 5$ Vision



(a) Type 3

Figure 15: $9 \times 9 \times 5$ Vision

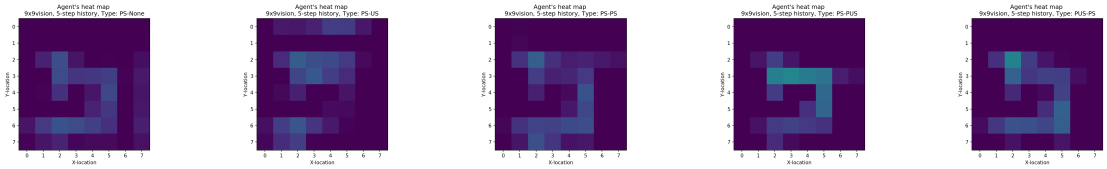


Figure 16: $9 \times 9 \times 5$ Vision

.3. Challenge 3

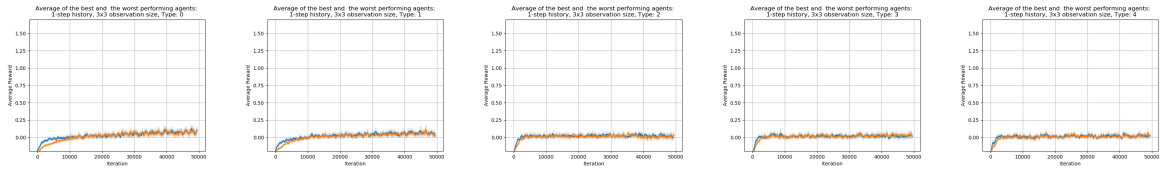


Figure 17: $3 \times 3 \times 1$ Vision

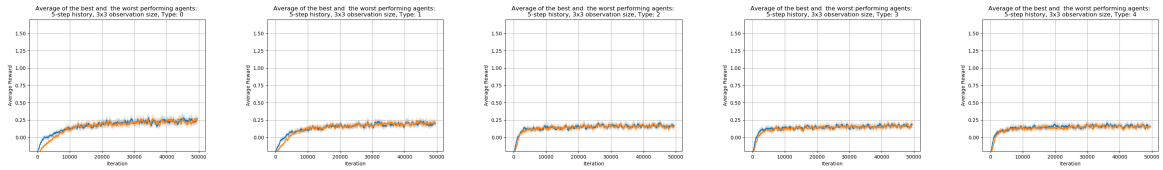


Figure 18: $3 \times 3 \times 5$ Vision

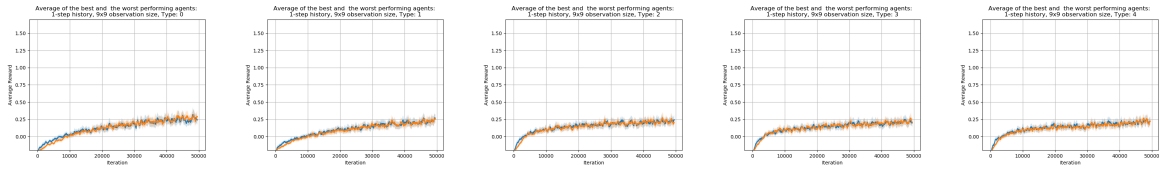


Figure 19: $9 \times 9 \times 1$ Vision

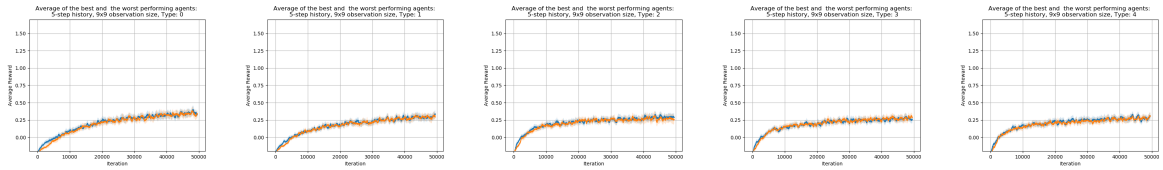


Figure 20: $9 \times 9 \times 5$ Vision

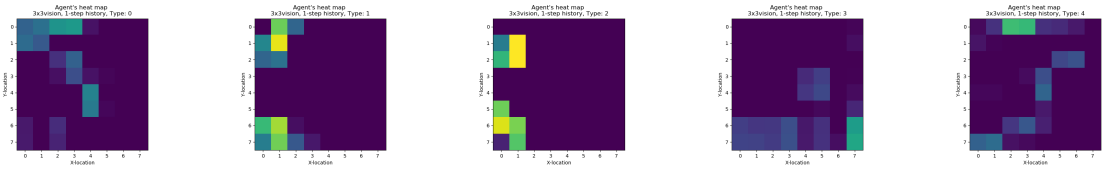


Figure 21: $3 \times 3 \times 1$ Vision

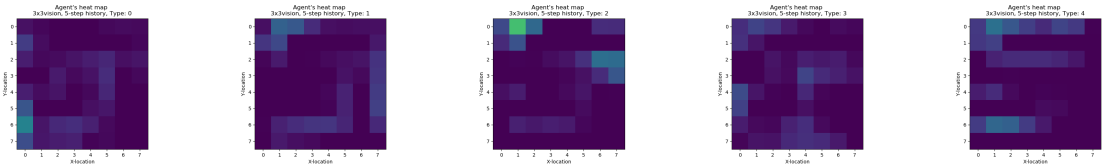


Figure 22: $3 \times 3 \times 5$ Vision

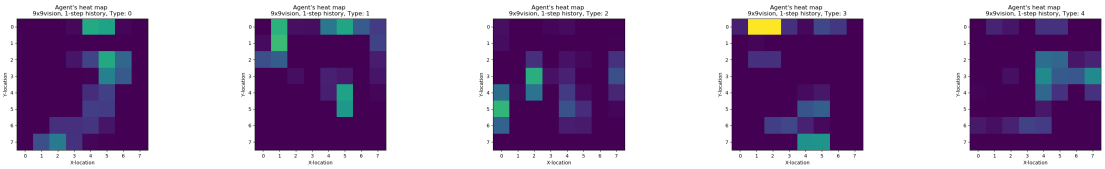


Figure 23: $9 \times 9 \times 5$ Vision

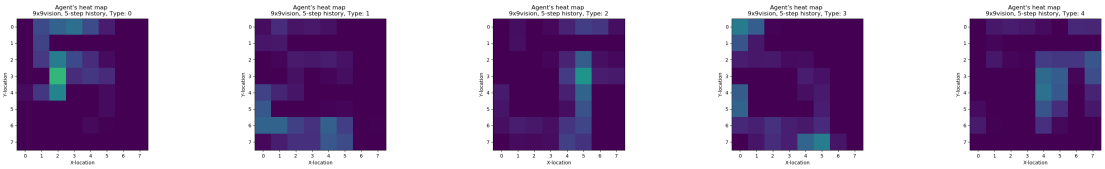


Figure 24: $9 \times 9 \times 5$ Vision