Influence-Based Abstraction in Deep Reinforcement Learning

Suau de Castro, Miguel; Congeduti, Elena; Starre, Rolf; Czechowski, Aleksander; Oliehoek, Frans

Publication date
2019

Document Version
Final published version

Published in
AAMAS Workshop on Adaptive Learning Agents (ALA)

Influence-Based Abstraction in Deep Reinforcement Learning

Miguel Suau de Castro
Delft University of Technology
Delft, Netherlands
M.SuaudeCastro@tudelft.nl

Elena Congeduti*
Delft University of Technology
Delft, Netherlands
E.Congeduti@tudelft.nl

Rolf A.N. Starre*
Delft University of Technology
Delft, Netherlands
R.A.N.Starre@tudelft.nl

Aleksander Czechowski*
Delft University of Technology
Delft, Netherlands
A.T.Czechowski@tudelft.nl

Frans A. Oliehoek
Delft University of Technology
Delft, Netherlands
F.A.Oliehoek@tudelft.nl

ABSTRACT

Real-world systems are typically extremely complex, consisting of thousands, or even millions of state variables. Unfortunately, applying reinforcement learning algorithms to handle complex tasks becomes more and more challenging as the number of state variables increases. In this paper, we build on the concept of *influence-based abstraction* which tries to tackle such scalability issues by decomposing large systems into small regions. We explore this method in the context of deep reinforcement learning, showing that by keeping track of a small set of variables in the history of previous actions and observations we can learn policies that can effectively control a local region in the global system.

KEYWORDS

Reinforcement Learning; Dec-POMDP; Influence-based abstraction

1 INTRODUCTION

The strength of reinforcement learning (RL) methods in training agents to solve certain tasks depends very much on the size of the problem and the amount of computing power that is available. In some cases, the use of function approximators, instead of tables, allows generalizing between multiple states and reduces the complexity of the problem. Yet, learning models that can represent policies or value functions from extremely large input spaces still remains very challenging.

However, it seems plausible that certain systems can be broken down into small, simple pieces that can be studied separately and then combined together to give a solution to the original problem. Take traffic control for example. Due to the large number of state variables, directly applying deep learning methods to model a policy to control the traffic lights of an entire city is intractable. Moreover, this centralized approach would also require an expensive communication system that could collect sensor data from the entire city in real-time. Alternatively, we could try to find a policy that can direct traffic at a single intersection by only receiving local information. Unfortunately, in some situations, using only the current observation we would not be able to predict the consequences of our own actions, even if the policies of the agents controlling the other intersections remained fixed. Some of the external state variables that are abstracted away could influence the local region and affect the transition probabilities. In other words, although using only a fraction of all the information available makes learning an optimal policy computationally more tractable, it turns the problem into a *partially observable Markov Decision Process (POMDP)* [14].

Partial observability in RL has been extensively studied during past years [8]. The methods can be roughly divided in two broad categories. On the one hand, there are approaches that ignore the lack of Markov property and apply standard RL methods [15, 20, 29]. Issues like chattering and divergence, however, make the applicability of these algorithms unsuitable for situations in which the observation space contains insufficient information [7, 10]. On the other hand, we have methods that more explicitly deal with non-Markovianity: policy search [13, 22], learning POMDP-like models [6, 9], predictive state representations [36, 37], EM-based methods [16, 34], (Monte-Carlo) AIXI [12, 33]. More recent approaches use recurrent neural networks (RNN) to maintain a context of what has occurred up to the current time step [11, 35]. However failure to converge or converging to poor quality local optima are typical issues of all these methods in practice. Besides, in our case, keeping track of a complete sequence of actions and observations would only transfer the scalability issues from the state space to the temporal space, which could make this approach impractical in some cases.

Aiming at finding common ground between these two methodologies, we investigate if it is possible to predict how the system's non-local dynamics influence the small region we aim to control by using only a small fraction of local variables. Previous work on *influence based abstraction (IBA)* [25] demonstrates that the non-Markovian dependencies that affect the local problem can be fully monitored by keeping track of a subset of variables in the history of local actions and observations, while the rest of the local region can be assumed to be Markovian. The intention of this paper is to show with practical examples that a similar approach can also be applied to deep RL. In particular, we use the insights from IBA to propose a different way of organizing the neural network used for policy and value estimation: we place recurrent cells at the border regions of the abstracted representation thus imposing an inductive bias on the function. This method is tested on a traffic setting [17] and a version of the Atari breakout video game [3]. The results suggest that informed placement of memory can indeed facilitate learning. We see improved performance, in some scenarios, when compared to memoryless policies, and better or equal performance when compared to networks which do not carefully select the position of the recurrent cells but instead feed the RNN with the entire input image.

*These authors contributed equally to this research.

This paper is organized as follows. First, we present a simple traffic example that we will use to clarify some of the ideas that are discussed throughout the paper. Then we briefly outline the concepts of POMDP and IBA which are the basis of our work. In section 4 we explain how to adapt IBA to deep RL and describe the Influence network structure that we use in our experiments. Finally, we discuss the results and set the direction of future work.

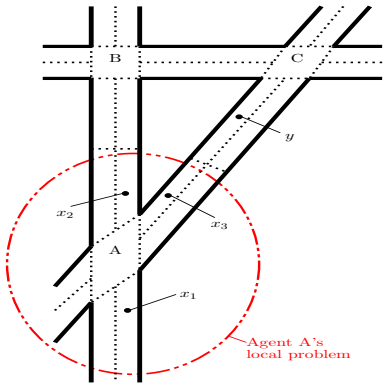

Figure 1: The local state of intersection A might depend in non-trivial ways on its past.

2 EXAMPLE SCENARIO

Figure 1 shows a small traffic network with three intersections. The task consists of optimizing the flow of vehicles at intersection A. The agent can only access the local region delimited by the red circle. Variables denoted by x correspond to state features that are local to the agent while y is assigned to those that are external and therefore, not part of the agent's input space. These variables determine the traffic density at each of the road segments.

The intuition is that a memoryless agent controlling the traffic lights in A could make instantaneous decisions based on the cars that happen to be inside the red circle at the current time step. Yet, if this same agent could also memorize the number of cars that had left the intersection in the direction of B or C, it could use this extra information to estimate a possible increase in the incoming traffic at the local region. For instance, if the traffic density at the road segment that connects A and B increases, the lights at B could switch to green and let the vehicles coming from A continue straight and leave the intersection. This would, in turn, reduce the amount of traffic going from B to C, which could make C send more vehicles towards A.

3 BACKGROUND

In this section we introduce the required background on which we build in the remainder of the paper. This consists of a concise overview of some of the formal frameworks for decision making under uncertainty, and the concept of influence-based abstraction.

3.1 Decision Making Frameworks

The main focus of this paper will be on partially observable single agent problems. However, we envision that these would typically be embedded in a multi-agent context. For instance, in a large traffic control problem, we would want to learn a local best response to the policies used at the other intersections. To clarify the embedding of a single decision maker in a multi-agent scenario, we introduce the more general Dec-POMDP framework [24].

Definition 3.1 (Dec-POMDP). A Dec-POMDP is a tuple $\mathcal{M} = \langle n, S, A, T, R, \Omega, O, h, b^0 \rangle$ consisting of:

- a set of n agents;
- S is the (finite) state space;
- $A = A_1 \times \cdots \times A_n$ is the space of joint actions $a = (a_1, \ldots, a_n)$, where A_i is the set of actions for agent i;
- T is the transition probability function, $T(s_t, a_t, s_{t+1}) = Pr(s_t | a_t, s_{t+1})$. That is, the probability of s_{t+1} being the next state given that the joint action a_t is taken in state s_t;
- $R(s_t, a_t)$ is the immediate reward function received by every agent for taking the joint action a_t in state s_t;
- $\Omega = \Omega_1 \times \cdots \times \Omega_n$ is the set of the joint observations $o = (o_1, \ldots, o_n)$, where Ω_i is the set of observation for agent i;
- O is the observation probability function which specifies $O(a_t, s_{t+1}, o_{t+1}) = Pr(o_{t+1} | a_t, s_{t+1})$, the probability of receiving a joint observation o_{t+1} after taking the joint action a_t and ending up in state s_{t+1};
- h is the horizon;
- b^0 is the initial state distribution;

Given a full Dec-POMDP model, the task consists of finding the set of policies $\pi_1, \ldots, \pi_n$ that maximizes the expected discounted sum of rewards [31]. Since each of the n agents receives only a partial observation of the true state s, a policy that is based only on the most recent information would be sub-optimal. Agents are required to keep track of their past experiences to make the right action choices. Policies are therefore mappings from the history of individual observations to actions.

A convenient way of representing the states in a Dec-POMDP is by using a set of state variables, often called factors:

Definition 3.2 (factored Dec-POMDP). A factored Dec-POMDP is a Dec-POMDP whose state space S is the product of some state variables, or factors. That is, a state $s = \langle x_1, ..., x_k \rangle$ is specified by some number k of state variables X_i which take values x_i.

In the traffic scenario in figure 1, a state is fully specified by a set of variables that measure the traffic density at each of the road segments. We use x for factors that belong to an agent's local observation space and y to denote external state variables. In the next section, we show how using this representation we can more closely study how transition, observation and reward functions affect our system by focusing on how the different factors are influenced by each other.

As mentioned before, we will focus on single agent scenarios. Given a Dec-POMDP $\mathcal{M}$ and a set of fixed policies π_{-i} for all the agents except agent i, we can treat them as part of the environment dynamics and define a POMDP for agent i's decision making problem [21]:

Definition 3.3 (POMDP). A POMDP is a Dec-POMDP with $n = 1$ decision making agents.

In this setting, agent i's policy can be represented as a function that takes as input the belief $b(s)$ over the possible states s and outputs the action to be taken.

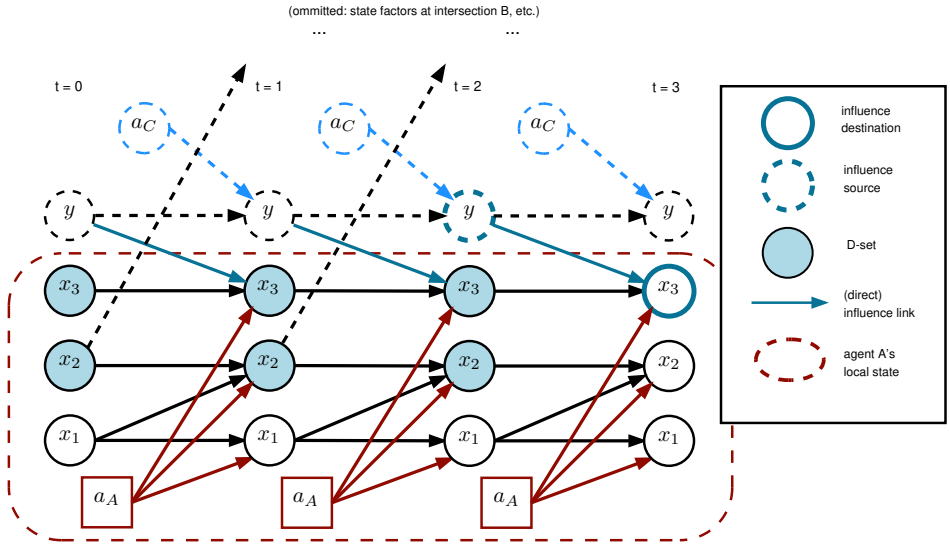

Figure 2: Simplified representation of the traffic example as a DBN unrolled over time. We use x for factors that belong to the agent's local observation and y to denote external state variables. The arrows reveal the dependencies between the factors shown in the diagram. For simplicity, some of the external state variables as well as the actions taken at intersection B are omitted from the Bayesian network.

Definition 3.4 (Belief). A belief of an agent in a POMDP is a probablity distribution over states:

$$b_t(s_t) = Pr(s_t|o_t, a_{t-1}, ..., o_1, a_0). \tag{3.1}$$

After taking an action and receiving a new observation, the belief state can be sequentially updated using Bayes' rule:

$$b_{t+1}(s_{t+1}) = \frac{1}{\eta} \sum_{s_t} T(s_t, a_t, s_{t+1}) O(a_t, s_{t+1}, o_{t+1}) b_t(s_t), \tag{3.2}$$

where η is a normalization constant,

$$\eta = \sum_{s_t} T(s_t, a_t, s_{t+1}) \sum_{o_{t+1}} O(a_t, s_{t+1}, o_{t+1}) b_t(s_t). \tag{3.3}$$

We can apply the update formula to maintain a belief over the states and try to obtain a policy that can map from beliefs to actions.

Finding the exact solution of the POMPD is equivalent to computing a best response to the set of fixed policies π_{-i} in the multi-agent context. Note that the variable a_t in 3.2 refers only to the action taken by a single agent, while the other agents' actions are given by π_{-i}. The reason why π_{-i} does not appear in equation 3.2 is because the policies are considered part of the environment dynamics and therefore modeled by $T(s_t, a_t, s_{t+1})$ [1].

The RL formulation, however, does not assume complete knowledge of the environment dynamics. We can only estimate observation and transition functions by interacting with the environment. This means that the exact belief cannot be computed. Instead, the agent needs to learn a representation that provides a Markovian signal to learn over, using the past history of actions and observations [18].

In section 4 we explain how we try to solve the POMDP RL problem by applying spatial abstraction to simplify our local model. Our intention is not to compare this solution with the one we could obtain if we had access to all the state factors since we consider that modeling the full set of variables present in our system is computationally intractable.

3.2 Influence Based Abstraction

Here, we briefly describe the framework of influence-based abstraction, aiming at getting across the intuition and core concepts, for an extensive discussion see [25].

Conceptually the idea of IBA is (1) build a smaller local model for one or a few agents given the policies of the others, then (2) compute an abstract representation of the influence exerted by the rest of the system on the local model, and finally (3) use this *influence-augmented local model* to compute a best response policy.

We will try to illustrate this using the traffic example in figure 1. As we already mentioned, the agent's local model consists of those features that lie within the red circle. These correspond to the nodes labeled with x in the dynamic Bayesian network [26] depicted in figure 2 where we unrolled 3 time steps. The arrows reveal some of the dependencies between variables. For instance, according to the diagram, x_2 depends on the value of x_1 and x_2 at the previous time step. The network also shows how the local model is affected by the external variables y that are outside the agents observable region. The actions taken by each of agents at the intersections A and C are denoted by a_A and a_C respectively. For simplicity, some of the external state variables as well as the actions taken at intersection B are omitted from the Bayesian network.

By inspecting both figure 1 and the Bayesian network 2 we see that a_c influences x_3 at the following time step via y. That is, the actions taken at intersection C will affect the traffic density at the road segment y which in turn will affect x_3. In particular, we say that y is an influence source for x_3, which is called influence destination.

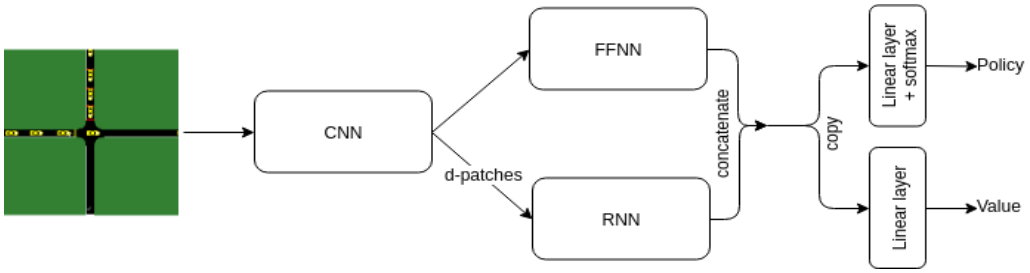

Figure 3: InfluenceNet architecture. The input image is first processed by a CNN, which finds a compact representation of the local observations. The whole encoded image is fed into a FFNN while the RNN receives only small regions, the d-patches, combines them with its internal state and outputs the influence prediction. Finally the output of the FFNN is concatenated with the influence prediction and passed through two separate linear layers which compute values and action probabilities.

To compute a best response policy, we need to estimate future states of our global system. If we want to predict the probability over the possible values of x_1 at the next time step it is enough to know its current value and the action taken. However, estimating future values of x_3 turns out to be much more complicated because it depends on external variables. Hence, x_3 does not follow the Markov property since it does not only depend on the present observation. Yet, if we consider the policies of the agents at intersections B and C to be fixed, we can treat them as part of the environment dynamics, calculate the probability distribution of y, $Pr(y|h)$, based on our local history h, and use it to predict x_3.

Moreover, we can exploit the spatial properties of our traffic network and ignore certain nodes that are conditionally independent from y . We define the d-separating set d as the smallest subset of variables in our local history that we need to condition on to compute the probability of the influence sources. In other words, the d-separating set is the group of variables that d-separates ([4], chap. 8) the influence sources from the rest of the local region. Adding any other local variables to the d-separating set would not bring any extra evidence, $Pr(y|h) = Pr(y|d)$. The blue circles in the Bayesian network correspond to the nodes that form the d-separating set in the traffic example.

The upshot of IBA is that one can replace the spatial dependence of the local region on the non-local part by a time dependence on the d-separating set and create models without any loss in value. Unfortunately, this history dependence leads to large computational costs when approached exactly. In this paper, we investigate if it is possible to adapt this perspective to function approximation and exploit it in the context of deep RL.

4 INFLUENCE-BASED ABSTRACTION IN DEEP RL

One of the advantages of using neural networks as function approximators is that we can directly work on a high dimensional sensory input without having to extract more descriptive features. For instance, in our traffic scenario, rather than using state variables, we can use images [23, 27] centered around intersection A to capture the traffic density. This implies that we need to redesign the influence model so we can safely apply spatial abstraction. In this section, we first explain conceptually how the ideas of IBA can be translated into an inductive bias for deep RL, then we give some details on the resulting 'InfluenceNet'.

4.1 Influence-based Inductive Bias

As previously mentioned, by abstracting away those parts of the system that have a weaker impact on the local region we care about, we converted our problem into that of a POMDP [2]. Some of the state variables are no longer Markovian when using only local information to control our system, which makes the usual RL methods inappropriate.

We could still train an agent to control the system by only using our most recent observation, but this might lead to making suboptimal decisions due to the fact that we lack some information about the past history. If we want to make good action choices in this new setting, it is essential that we treat the problem as a POMDP, and keep track of the past actions and observations. The obvious way to do so would be to store a finite history of experiences and feed these to the network. However, we could soon run into computational issues if the observation space is too big or the history is too long. Instead, we could train an RNN to maintain and update a context of what has occurred up to the current timestep [11, 35]

Training an RNN on memorizing information about a high dimensional input space, like an image, is particularly expensive. Especially if the information needs to be retained for a long period of time. Ideally, we would like our model to focus on remembering only the essential pieces. The theory of IBA demonstrates that in order to capture the non-Markovian dependencies introduced by the abstraction of the global system, it suffices to condition on the history of a subset of local variables as long as that subset d-separates the influence sources from the remainder of the local problem. In our case, since the input to our network is an image, explicitly identifying an exact d-separating set and computing the influences is cumbersome. Instead, we make use of the spatial structure of the system to introduce an inductive bias in the model of our policy. That is, we define 'd-patches', which are small regions in the input space, and feed these into recurrent layers. As such, we effectively discriminate the local state variables between those that we need to remember about and those that we can treat as Markovian.

Going back to the traffic example, we could assume the dynamics inside the intersection to be Markovian and process them directly by a feedforward neural network (FNN). However, if we want to

be able to predict the aforementioned incoming influences, it is important to place d-patches at the two road segments that connect A with B and C so that our RNN can maintain an internal context of what happens there.

4.2 InfluenceNet

In this section we describe the specific structure of the neural network that we employ for both policy and value function. We call the network InfluenceNet because it ties up well with the concept of IBA, where a small set of variables in our local history is used to determine how the system will react to our previous decisions.

The InfluenceNet architecture is depicted in figure 3. The input image is first processed by a convolutional neural network (CNN), which finds a compact representation of the local observations. The whole encoded image is fed into a FNN while the RNN receives only small preselected regions, the d-patches, combines them with its internal state and outputs the influence prediction. Finally the output of the FNN is concatenated with the influence prediction and passed through two separate linear layers which compute values and action probabilities.

As shown in the diagram, both RNN and FNN share the same CNN. Hence, instead of making the split between Markovian and non-Markovian variables on the input image and training two separate networks, we extract the d-patches after the image has been processed by the CNN, which significantly reduces the total number of parameters in our network. This algorithmic choice allows us to make better use of the information since the convolutional filters that are connected to the RNN are trained on a full image rather than on a small patch. Thus, more data is being used in every update to optimize the convolutional filters. This is only possible because the output of the CNN maintains the same spatial configuration as the input image.

Although the network architecture also suits any value based or policy gradient method, in our experiments we combine it with Proximal Policy Optimization (PPO) [28].

5 EXPERIMENTS

The InfluenceNet model was tested on two different environments: a traffic control task, where we made obvious the effect of incoming influences, and a modified version of the popular Atari Breakout video game that we call Myopic Breakout. Both environments are designed to be partially observable so that the agent can only master the tasks if it keeps track of previous actions and observations.

We compare the performance of memoryless policies and recurrent models to show that an agent can only reach a bounded level of performance when using only the most recent information. Furthermore, we also test whether the influence model, where the RNN receives only selected regions of the encoded image, improves over a regular RNN model [1].

5.1 Traffic Control

In this environment, the agent must optimize the traffic flow at the intersection in Figure 4. We restrict the observation space to the non-shaded area shown in the image. The agent can take two different actions, either switching the traffic light on the top to

[1]Videos showing the results of our experiments can be found at https://bit.ly/2UmKsY7

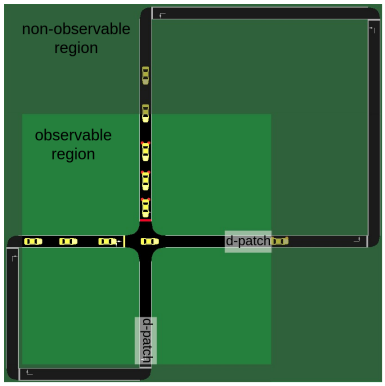

Figure 4: Traffic control environment. The observable region corresponds to the non-shaded box centered at the intersection. The d-patches are the grey areas at the left and bottom parts of the local region.

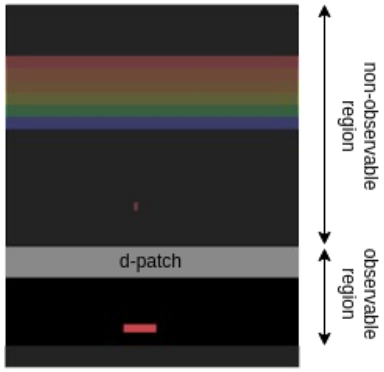

Figure 5: Myopic Breakout environment. The obscured region is invisible to the agent. The grey stripe represents d-patch that is fed into the RNN.

green, which automatically turns the other to red, or vice versa. There is a 6 seconds delay between the moment an action is taken and the time the lights actually switch. During this period the green light turns yellow and no cars are allowed to cross the road. In every episode, a total of 6 cars start either at the top right or bottom left corner. The episode ends when all 6 cars have completed 4 loops.

The local reward function is defined as follows:

$$r_t = \sum_{i=1}^{N} w_i, \tag{5.4}$$

where N is the total number of vehicles in the local region and w_i is a penalty for the i-th car if it is not moving.

We built the environment using SUMO (Simulator of Urban Mobility) [17]. The traffic network was designed so that the agent's local observations are clearly non-Markovian. Cars leaving the observable region from the right-hand side will appear again after some time at the top part. Hence, the agent is forced to memorize that information in order to be able to make the right decisions. Knowing this, we place the d-patches at the end of the outgoing

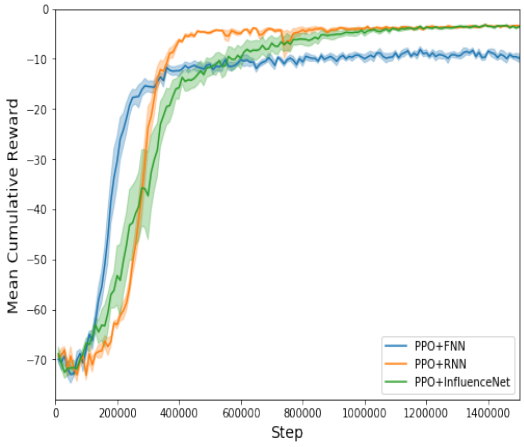

Figure 6: Traffic control. Mean cumulative episodic reward at every 10K time steps. The curves are the average of 5 runs. Shaded areas indicate the one standard deviation of the mean.

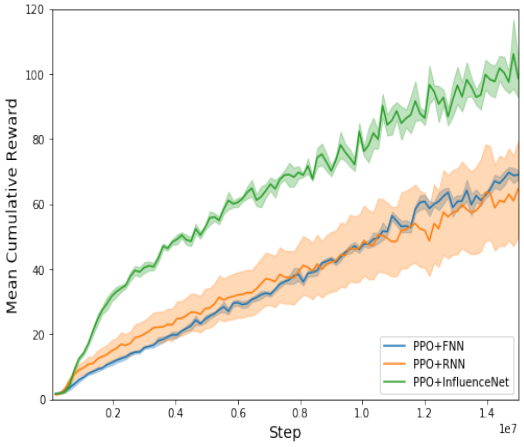

Figure 7: Myopic Breakout. Mean cumulative episodic reward at every 100K time steps. The curves are the average of 5 runs. Shaded areas indicate the one standard deviation of the mean.

road segments and treat the rest of the hidden features in the local observation as Markovian, see grey boxes in Figure 4.

5.2 Myopic Breakout

We want to show that an agent with limited vision can learn the global dynamics of a given system and be able to anticipate the long term effect of its own actions by only keeping track of the history of a subset of variables. To that end, we created Myopic Breakout using OpenAI Gym [5]. In this version of the Atari video game the agent only receives as input a small region at the bottom of the screen, Figure 5.

The intuition is that, because the dynamics of the scenario are relatively stationary, the agent should be able to make a reasonable prediction about where the ball will hit next according to the direction and speed at which it left the local region. Therefore, we place the d-patch at the top part of the observable region (grey stripe Figure 5).

5.3 Experimental setup

The InfluenceNet model (PPO+InfluenceNet) was tested against two other different network configurations: a model with no memory (PPO+FNN) and a full recurrent model (PPO+RNN) where the RNN is fed with the whole output of the CNN. Note that all three models have only access to the small observable region shown in Figures 4 and 5. We ran our experiments 5 times with different random seeds and using the hyperparameters given in the appendix. We used the cumulative reward obtained in every episode as performance measure. In each run the values are averaged every 10K and 100K steps for Traffic Control and Myopic Breakout respectively. For more details about hyperparameter tuning see the appendix.

5.4 Results

The results of running PPO using the three different network configurations, PPO+FNN, PPO+RNN and PPO+InfluenceNet on the traffic control task and the Myopic Breakout environment are depicted in Figures 6 and 7 respectively. The curves show the mean cumulative reward averaged over 5 runs with different random seeds.

In the traffic control task, the size of the observable region and the time delay between actions and traffic light responses were chosen so that if an agent tries to switch the light at the exact time when a car enters the local region, the car will have to stop for a few seconds before continuing. This enforces the recurrent models to remember the location and the time at which cars left the intersection and limits the performance of agents with no memory, see Figure 6. Agents need to anticipate to cars appearing at the incoming lanes and switch the lights in time for the cars to continue without stopping. Although the InfluenceNet model takes more steps to converge to the optimal policy, we obtained speedups of around 1.5 with respect to the full RNN model when comparing wall-clock times.

On the other hand, agents trained using InfluenceNet on Myopic Breakout are able to outperform both the full recurrent model and the feedforward model. By feeding only small d-patches, we reduce the number of parameters in the RNN and ease the task of the recurrent cells. We hypothesize that the InfluenceNet agent is able to predict the location at which the ball will appear based on the direction and speed at which it left the observable region [2].

6 DISCUSSION

Although we believe the results obtained are very promising, for the sake of transparency, we would like to mention that before reaching the reported levels of performance we had to test different sizes and configurations for the observable region and the d-patches. Besides, in the traffic control task, the reward function used in [23] had to be modified to facilitate the learning task to the RNN. The travel delay penalty, which is the ratio between a car's speed and

[2]Videos showing the results of our experiments can be found at https://bit.ly/2UmKsY7

the maximum allowed speed, induced a noisy reward signal and made it difficult for both the full RNN and the InfluenceNet model to associate the event of cars entering or leaving the local region with future penalties.

On the other hand, we are also aware that pinpointing the exact regions that are fed to the RNN is not always an option since it requires some amount of prior domain knowledge. Besides, in some situations the d-patches might differ from one state to another.

7 CONCLUSION AND FUTURE WORK

In this paper, we investigated if the concept of IBA can be effectively translated into the context of deep RL. In particular, we focused on problems where the agent can only observe a local region of a larger system and tested whether, by keeping track of only a small fraction of the information available, we could estimate the influence that the non-local system exerts on the agent's local neighborhood.

The experiments reveal that agents equipped with internal memory can anticipate the response of the system to earlier actions and react accordingly. We showed that, by making the split between Markovian and non-Markovian regions in the local observation, we were able to simplify the neural network architecture and considerably reduce training times. Moreover, the InfluenceNet structure seems to facilitate the learning process and obtains better performance than a regular RNN model on one of the two tasks.

In future work, we would like to study the actual benefits of our approach when applied to more general scenarios. We believe that the influence framework will enable important simplifications in large systems so that we can solve complex problems with reasonable computing power.

Our intention is also to investigate how to make the agent learn what information is important to memorize at each particular state. In this way, the agent would be able to automatically select the d-patches so that they do not need to be specified beforehand. A similar idea was explored in [30] using attention mechanisms [32, 38]

8 ACKNOWLEDGMENTS

This research made use of a GPU donated by NVIDIA. This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 758824 —INFLUENCE).

A HYPERPARAMETERS

The hyperparameters used in the two experiments are provided in tables 1 and 2. We used the same hyperparameter configuration reported by [19] as a starting point for all our different models, and hand-tuned those parameters that were added in the InfluenceNet architecture. These are the sequence length, which determines how many steps the RNN is unrolled when computing policy and value losses, and the size and position of the d-patches. If the sequence length is too short, we might lose valuable information about the past history, while longer sequences increase the computational cost. In our traffic example, for instance, we tested different values around the number of timesteps it takes for a car to complete the loop since our expectation was that this should be enough to capture all the history-dependent dynamics. On the other hand, we would like our d-patch to be as small as possible while still allowing the network to estimate the effect of the influence sources. Given our computational constraints, we were only able to run each configuration once. While additional runs would be needed to be more conclusive about what the best configuration is, we still expect to have found reasonable hyperparameter settings.

Table 1: Observation boxes and hyperparameter settings for the traffic control scenario.

Traffic control						
Observation	full		local		d-patch 1	d-patch 2
box_height	112		56		1	1
box_width	112		56		1	1
box_topleft_corner	[0,0]		[65,9]		[4,2]	[2,4]
frame_height	-		14		-	-
frame_width	-		14		-	-
Algorithm	PPO		PPO+RNN		PPO+InfluenceNet	
num_frames	1		1		1	
beta	1e-2		1e-2		1e-2	
epsilon	0.2		0.2		0.2	
time_horizon	128		128		128	
CNN	layer 1	layer 2	layer 1	layer 2	layer 1	layer 2
input	local	-	local	-	local	-
filters	16	32	16	32	16	32
kernel_size	4	2	4	2	4	2
strides	2	1	2	1	2	1
FFNN	layer 1		None		layer 1	
units	256		-		128	
LSTM	None		layer 1		layer 1	
input	-		local		d-patch 1 ∪ d-patch 2	
rec_units	-		256		128	
seq_length	-		128		128	

Table 2: Observation boxes and hyperparameter settings for the myopic breakout scenario.

Myopic breakout						
Observation	full		local		d-patch	
box_height	84		25		7	
box_width	84		84		19	
box_topleft_corner	[0,0]		[54,0]		[0,0]	
frame_height	-		42		-	
frame_width	-		42		-	
Algorithm	PPO		PPO+RNN		PPO+InfluenceNet	
num_frames	4		1		1	
beta	1e-2		1e-2		1e-2	
epsilon	0.2		0.2		0.2	
time_horizon	128		128		128	
CNN	layer 1	layer 2	layer 1	layer 2	layer 1	layer 2
input	local	-	local	-	local	-
filters	16	32	16	32	16	32
kernel_size	4	2	4	2	4	2
strides	2	1	2	1	2	1
FFNN	layer 1		None		layer 1	
units	256		-		128	
LSTM	None		layer 1		layer 1	
input	-		local		d-patch	
rec_units	-		256		128	
seq_length	-		32		32	

REFERENCES

[1] Christopher Amato and Frans Adriaan Oliehoek. 2013. Best Response Bayesian Reinforcement Learning for Multiagent Systems with State Uncertainty.

[2] Aijun Bai, Siddharth Srivastava, and Stuart J Russell. 2016. Markovian State and Action Abstractions for MDPs via Hierarchical MCTS.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47 (jun 2013), 253–279.

[4] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. (2016). arXiv:arXiv:1606.01540

[6] Finale Doshi-Velez. 2009. The Infinite Partially Observable Markov Decision Process. In *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta (Eds.). 477–485.

[7] Michael Fairbank and Eduardo Alonso. 2012. The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[8] Zoubin Ghahramani. 2001. An introduction to hidden Markov models and Bayesian networks. In *Hidden Markov models: applications in computer vision*. World Scientific, 9–41.

[9] Zoubin Ghahramani. 2001. An introduction to hidden Markov models and Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence* 15, 01 (2001), 9–42.

[10] Geoffrey Gordon. 1995. Stable Function Approximation in Dynamic Programming. In *Proc. of the Twelfth International Conference on Machine Learning*.

[11] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable MDPs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[12] Marcus Hutter. 2005. *Universal Artificial Intellegence - Sequential Decisions Based on Algorithmic Probability*. Springer. https://doi.org/10.1007/b138233

[13] Leslie Pack Kaelbling, Michael Littman, and Andrew Moore. 1996. Reinforcement Learning: A Survey. *Journal of AI Research* 4 (1996), 237–285.

[14] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1-2 (1998), 99–134.

[15] Michael L. Littman. 1994. Memoryless Policies: Theoretical Limitations and Practical Results. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3*. 238–245.

[16] Yun-En Liu, Travis Mandel, Eric Butler, Erik Andersen, Eleanor O'Rourke, Emma Brunskill, and Zoran Popovic. 2013. Predicting Player Moves in an Educational Game: A Hybrid Approach. In *Proceedings of the 6th International Conference on Educational Data Mining, Memphis, Tennessee, USA, July 6-9, 2013*. 106–113.

[17] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. Microscopic Traffic Simulation using SUMO, In The 21st IEEE International Conference on Intelligent Transportation Systems. *IEEE Intelligent Transportation Systems Conference (ITSC)*. https://elib.dlr.de/124092/

[18] Andrew Kachites Mccallum. 1996. *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. Dissertation. AAI9618237.

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

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (26 02 2015), 529–533. https://doi.org/10.1038/nature14236

[21] Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. 2003. Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. In *Proc. of the International Joint Conference on Artificial Intelligence*. 705–711.

[22] Andrew Y. Ng and Michael I. Jordan. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*. 406–415.

[23] Elise van der Pol and Frans A. Oliehoek. 2016. Coordinated Deep Reinforcement Learners for Traffic Light Control. Submitted to NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems. (2016).

[24] Frans A. Oliehoek and Christopher Amato. 2016. *A Concise Introduction to Decentralized POMDPs*. Springer. https://doi.org/10.1007/978-3-319-28929-8

[25] Frans Adriaan Oliehoek, Stefan J Witwicki, and Leslie Pack Kaelbling. [n. d.]. Influence-based abstraction for multiagent systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[26] Judea Pearl. 2014. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

[27] Tobias Rijken. 2015. *DeepLight: Deep reinforcement learning for signalised trac control*. Master's thesis. University College London.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[29] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. 1994. Learning Without State-Estimation in Partially Observable Markovian Decision Processes. In *Proc. of the International Conference on Machine Learning*. Morgan Kaufmann, 284–292.

[30] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. 2015. Deep attention recurrent Q-network. *arXiv preprint arXiv:1512.01693* (2015).

[31] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 17*. 5998–6008.

[33] Joel Veness, Kee Siong Ng, Marcus Hutter, William T. B. Uther, and David Silver. [n. d.]. A Monte-Carlo AIXI Approximation. *Journal of AI Research* ([n. d.]).

[34] Nikos Vlassis and Marc Toussaint. 2009. Model-free reinforcement learning as mixture learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 1081–1088.

[35] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. 2007. Solving deep memory POMDPs with recurrent policy gradients. In *International Conference on Artificial Neural Networks*. Springer, 697–706.

[36] David Wingate. 2012. Predictively Defined Representations of State. In *Reinforcement Learning: State-of-the-Art*, Marco Wiering and Martijn van Otterlo (Eds.). Springer Berlin Heidelberg, 415–439.

[37] Britton D. Wolfe. 2009. *Modeling Dynamical Systems with Structured Predictive State Representations*. Ph.D. Dissertation. University of Michigan.

[38] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of the 32nd International Conference on Machine learning*. 2048–2057.