Influence-aware Memory for Deep RL in POMDPs

Miguel Suau[*] TU Delft

Elena Congeduti TU Delft

Jinke He TU Delft

Rolf A. N. Starre TU Delft

Aleksander Czechowski TU Delft

Frans A. Oliehoek TU Delft

Abstract

Making the right decisions when some of the state variables are hidden, requires removing the uncertainty about the current state of the environment. An agent receiving only partial observations needs to infer the true values of these hidden variables based on the history of observations. Recent deep reinforcement learning methods use recurrent models to keep track of past information. However, these models are expensive to train and have convergence difficulties, especially when dealing with high dimensional input spaces. Inspired by theory from *influence-based abstraction*, which asserts that in order to predict the hidden state variables we may only need to remember about a small subset of observation variables, we propose *InfluenceNet*. This new neural architecture tries to overcome the training difficulties in high dimensional problems by restricting the input that goes into the recurrent layers to those variables carrying important information about the non-Markovian dynamics. Results indicate that, by forcing the agent's internal memory to focus on this subset rather than on the full observation, we can outperform ordinary recurrent architectures. This approach also reduces training time and obtains better scores than methods that stack multiple observations to remove partial observability.

1 Introduction

It is not always guaranteed that an agent will have access to a full description of the environment to solve a particular task. In fact, most real-world problems are by nature partially observable. This means that some of the variables that define the state space are hidden [19]. This type of problems can be modeled as *partially observable Markov decision processes (POMDP)* [14]. The model is an extension of the MDP framework [25], which assumes that states are only partially observable, and thus the Markov property is no longer satisfied. That is, future states do not solely depend on the most recent observation. Most POMDP methods try to extract information from the history of actions and observations to disambiguate the state in the underlying MDP. We argue however, that in many cases, memorizing all the observed variables is costly and requires unnecessary effort. Instead, we can exploit the structure of our problem and abstract away from the history those variables that have no predictive value over the hidden ones.

Previous work on *influence-based abstraction (IBA)* [23] demonstrates that, in certain POMDPs, the non-Markovian dependencies in the transition and reward functions can be fully monitored by keeping track of a subset of variables in the history of actions and observations. In this paper, we use the insights from IBA to tackle high dimensional reinforcement learning (RL) problems which show this structure but where no explicit model of the world is available. In particular, we propose a different way of organizing the recurrent neural network (RNN) used for policy and value

[*]Corresponding author: m.suaudecastro@tudelft.nl

Presented at Deep RL Workshop, NeurIPS 2020.

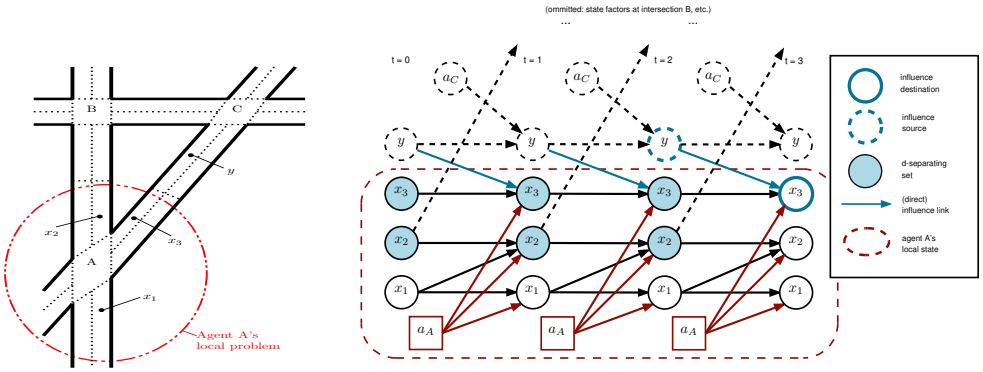

Figure 1: Traffic network (left). Dynamic Bayesian Network representing the structure of the traffic network. The arrows illustrate the dependencies between the variables. For simplicity, some of the external state variables as well as the actions taken at intersection B are omitted from the DBN (right).

estimation: we restrict the input of the recurrent cells to those regions that we believe contain sufficient information to estimate the influence of the hidden variables. That is, we impose an inductive bias [9] by selecting the areas in the observation input that need to be memorized and processing the rest with a feedforward neural network (FNN). This intends to mitigate the training difficulties that RNNs face in high dimensional problems. We test our method on a traffic control task and five different Atari video games [1]. The model obtains equal performance to networks that try to overcome the partial observability issues by feeding multiple past frames, in environments where short-term memory is sufficient, and better performance in domains where the agent needs to remember events that are distant in the past. Our experiments show that, in the second case, stacking a history of frames is no longer practical and can soon deteriorate the algorithm's efficiency. Additionally, the results indicate that, by constraining the dimensions of the input to the agent's internal memory, we can speed up the learning process and find better policies than those obtained using full recurrent models. Further analysis suggests that as a result of implementing two separate processing streams in our network: one for variables that we can treat as Markovian (FNN) and another for variables that we need to remember about (RNN), we facilitate the learning process and allow the network to form internal representations that are much better at predicting the relevant hidden quantities.

2 Background

The model presented in Section 4 builds on the POMDP framework and the concept of *influence-based abstraction*. For the sake of completeness, we briefly introduce each of them here and refer interested readers to [14, 22].

Definition 1 (POMDP) *A POMDP is a tuple* $\langle S, A, T, R, \Omega, O \rangle$ *where* S *is the state space,* A *is the set of actions,* T *is the transition probability function, with* $T(s_t, a_t, s_{t+1}) = Pr(s_{t+1}|a_t, s_t)$. $R(s_t, a_t)$ *defines the reward for taking action* a_t *in state* s_t*,* Ω *is the observation space,* O *is the observation probability function,* $O(a, s_{t+1}, o_{t+1}) = Pr(o_{t+1}|a, s_{t+1})$*, the probability of observing* o_{t+1} *after taking action* a *and ending up in state* s_{t+1}.

In the POMDP setting, the task consists in finding the policy π that maximizes the expected discounted sum of rewards [29]. Since the agent receives only a partial observation of the true state s, a policy that is based only on the most recent information can be sub-optimal. In general, the agent is required to keep track of its past experiences to make the right action choices. Policies are therefore mappings from the history of past actions and observations $h_t = \langle o_0, a_0 ..., a_{t-1}, o_t \rangle$ to actions.

We illustrate the concept of *influence-based abstraction* using the example in Figure 1, which shows a small traffic network with three intersections. The task consists of optimizing the flow of vehicles at intersection A. The agent can only observe 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 observation space. These variables determine the traffic density at each road segment. The dynamics of the problem can be represented with a dynamic Bayesian network (DBN) [24, 3] (Figure 1, right). The network shows how the local observations are affected by both the local variables x and the external variables y that are outside of the agents observable region. The actions taken by 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 DBN.

To find the optimal policy, the agent needs to be able to predict future observations. By inspecting the Bayesian network in Figure 1 we see that if we want to predict the value of x_1 at the next timestep it is enough to know its current value and the action taken. However, estimating future values of x_3 is much more complicated, because it depends on external variables y. In particular, we see that a_c influences x_3 at the following timestep 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. We then say that y is an *influence source* for x_3, the *influence destination*.

Our local observation variables can be decomposed as $o = \langle x, \bar{x} \rangle$ to distinguish the influence destinations $\bar{x}$, such as x_3, from those which are only directly affected by local variables, like x_1 and x_2. In order to predict future values of $\bar{x}$ we first need to infer y_t, based on our local history h_t, $Pr(y_t|h_t)$. On the other hand, x_{t+1} can be estimated using only o_t. Hence we can write:

$$Pr(o_{t+1} = \langle x_{t+1}, \bar{x}_{t+1} \rangle | h_t, a_t) = Pr(x_{t+1}|o_t, a_t) Pr(\bar{x}_{t+1}|y_t, o_t, a_t) Pr(y_t|h_t). \quad (1)$$

That is, we need to condition on our local history, to be able to determine the effect of the influence sources y on the influence destinations $\bar{x}$ while the rest of the variables x can be treated as Markovian.

We can further exploit the spatial structure of our traffic network, and ignore certain nodes that are conditionally independent from y. The blue circles in the DBN, indicate the variables in h_t that we need to condition on to estimate the effect of the influence source y on our local variable x_3. This group of variables is known as the d-separating set (d-set).

Definition 2 (D-separating set) *The d-separating set is a subset of variables d_t from the local history h_t, such that the influence sources are conditionally independent from the remaining parts of the local history $h_t \setminus d_t$ given d_t: $\Pr(y_t|h_t) = P(y_t|d_t, h_t \setminus d_t) = Pr(y_t|d_t)$. This conditional independence can be tested using the notion of d-separation [2, Ch. 8].*

The upshot of IBA is twofold: (1) it effectively splits observations into Markovian x and non-Markovian variables $\bar{x}$, and (2) it makes use of d-separation to reduce the amount of information the agent needs to memorize in order to estimate the effect of the influence sources and predict future observations.

3 Influence-aware Memory

While IBA offers a perspective on how to reduce the memory dependencies in POMDPs, it requires a fully specified Bayesian network in order to determine the d-sets. In this paper, however, we tackle the deep RL setting in which such a model is usually not available, making the straightforward IBA approach inapplicable. Instead, we aim to use the basic insights from IBA and apply them as inductive bias to the model of the value function by proposing a suitable network architecture.

3.1 An IBA perspective on RL for POMDPs

The value function in a POMDP can be expressed in terms of the history of actions and observations $h_t = \langle o_0, a_0 ..., a_{t-1}, o_t \rangle$ as

$$Q(h_t, a_t) = R(h_t, a_t) + \sum_{o_{t+1}} \Pr(o_{t+1}|h_t, a_t) \max_{a_{t+1}} Q(h_{t+1}, a_{t+1}), \quad (2)$$

where $R(h_t, a_t) = \sum_{s_t} \Pr(s_t|h_t) R(s_t, a_t)$ is the expected immediate reward at time t over the set of possible states s_t given a particular history h_t.

According to IBA, we can replace the dependence on the full history of actions and observations h_t by a dependence on the d-set d_t (Definition 2),

$$Q(\langle d_t, o_t \rangle, a_t) = R(\langle d_t, o_t \rangle, a_t) + \sum_{o_{t+1}} \Pr(o_{t+1}|\langle d_t, o_t \rangle, a_t) \max_{a_{t+1}} Q(\langle d_{t+1}, o_{t+1} \rangle, a_{t+1}), \quad (3)$$

and $d_{t+1} \triangleq \langle d_t, D(o_{t+1}) \rangle$, where $D(\cdot)$ is the d-set selection operator, which chooses the variables in o_{t+1} that are added to d_{t+1}. Then, since d_t contains enough information to predict the distribution over future states (Equation 1 and Definition 2), we can write

$$Q(h_t, a_t) = Q(\langle d_t, o_t \rangle, a_t), \quad (4)$$

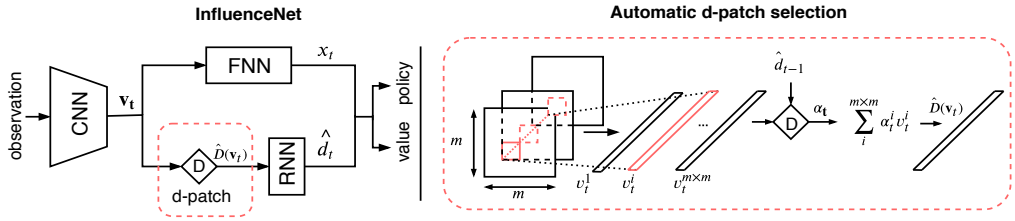

Figure 2: InfluenceNet architecture (left). Automatic d-patch selection (right)

3.2 IBA as an Inductive Bias

Even though we can not perfectly determine the exact d-set in deep RL settings, in problems like our traffic example it is not difficult to make an informed guess about the region(s) containing sufficient information to predict the influence sources. Therefore, we can introduce an inductive bias in the value function by selecting small regions of the observation, which we call *d-patches*, and feeding them into the recurrent layers. We use $\hat{D}(\cdot)$ to refer to the d-patch selection operator, such that the history of d-patches is given by $\hat{d}_{t+1} \triangleq \langle \hat{d}_t, \hat{D}(o_{t+1}) \rangle$. However, in practice, we process them with an RNN, and thus, with slight abuse of the notation, we define $\hat{d}_{t+1} \triangleq F_{\text{rnn}}(\hat{d}_t, \hat{D}(o_{t+1}))$. Then, if the important information in the d-set d can be approximately recovered from $\hat{d}$, we can ignore some variables in the history and still be able to estimate the *optimal value*,

$$Q(\langle d_t, o_t \rangle, a_t) \approx Q(\langle \hat{d}_t, o_t \rangle, a_t). \tag{5}$$

3.3 Influence Network

We design the Influence Network (InfluenceNet) architecture so that it can learn the optimal value function more effectively by encoding the ideas of the IBA framework as an inductive bias. The architecture is depicted in Figure 2. The input image o_t is first processed by a convolutional neural network (CNN), which finds a compact representation of the local observations. The whole encoded image $\mathbf{v}_t$ is fed into an FNN while the RNN receives only small preselected regions, the d-patches, and uses them to update its internal state $\hat{d}_t$ Finally, the output of the FNN x_t is concatenated with $\hat{d}_t$ and passed through two separate linear layers which compute values and action probabilities. The benefit of connecting the FNN in parallel is that it can process the information that does not need to be stored in memory but that is required for estimating the current state value, thus freeing up space in the RNN's internal memory. Note that, as opposed to IBA, our end goal is not to find a minimal d-set but to aid the network in approximating the optimal value function (equation 5). Below we describe two different alternatives for determining the d-patches that are fed into the RNN.

3.3.1 Manual selection of d-patches

In the first version, we exploit domain knowledge to manually specify the d-patch selection operator $\hat{D}(\cdot)$. For instance, going back to the traffic example in Section 2, if we wanted to control the traffic lights in A using images rather than state variables, we would place d-patches at the regions of the image that correspond to the two road segments that connect A with B and C (Figure 1) so that the RNN can maintain an internal memory of what happens at the edges of the intersection and be able to estimate the traffic density in B and C. The idea is to make the rest of the pixels in the observation conditionally independent from the hidden variables so that they can be processed by an FNN.

3.3.2 Automatic selection of d-patches

Manually selecting the regions that need to be fed into the RNN requires some amount of prior knowledge of the structure of the problem. Therefore, rather than choosing the d-patches beforehand, we would like to *learn* the d-set selection operator $D(\cdot)$ from experience. Some of the Atari games, contain moving sprites whose speed and direction cannot be measured using only the most recent frame, and thus, conceptually, they can be seen as hidden variables. In Breakout, for example, it is not necessary to memorize the whole set of pixels in the game screen to predict where the ball will be next, but only the ones that are around the ball. Intuitively, these pixels constitute a d-set. However, handcrafting $\hat{D}(\cdot)$ such that it can track the position of the bouncing ball at every timestep is not trivial. To overcome this potential limitation we introduce a method to automatically select the regions that need to be fed into the RNN. This method can be described as follows: first, the image is processed by a CNN $F_{\text{cnn}}(o) = \mathbf{v}$ which produces $m \times m$ vectors v of size N, where N is the number

of filters in the last convolutional layer and $m \times m$ the dimensions of the 2D output array of each filter. Each of these vectors corresponds to a particular region in the input image. The d-patch is formed by taking a weighted average of them, $\hat{D}_{\alpha_t}(v_t) = \sum_{i=0}^{m \times m} \alpha_t^i v_t^i$. We propose two different ways of computing the weights α_t^i for each of the regions:

Static d-patch selection: The values for α, which remain constant after training, are obtained by taking the softmax of a set of weights that we train together with the rest of the network. The same weights are applied to every observation. This d-patch selection method suits environments where d-sets are static and do not depend on observations or histories, like in the traffic control task.

Dynamic d-patch selection: The second approach incorporates a spatial attention mechanism similar to the one proposed by [37] and uses it to determine the d-patches given the agent's internal memory $\hat{d}$. The advantage is that the position of the d-patches can now change from one state to another. This is meant to correct for the fact that, in some environments like in Breakout, the d-separating variables might differ depending on the state. Here α_t^i is computed using a two-layer fully connected FNN that takes as input the i-th vector v_t^i and the previous internal memory $\hat{d}_{t-1}$, followed by a softmax operator that calculates the weight for each of the regions (see Figure 2). In order to preserve the location-specific information, we also concatenate α to the output of the selection operator.

4 Experiments

The goal of these experiments is: (1) evaluate whether our approach improves over full recurrent models, (2) see if it represents a better choice for handling partial observability than using a fixed-sized window of past frames, (3) compare it to a similar architecture that does not preselect what regions of the image are fed into the RNN, and (4) analyze the effect of our network configuration on the hidden activation patterns to make sure it is aligned with the ideas of IBA. To that end, the InfluenceNet model was tested on a traffic control task and the flickering version of the Atari games [8] against three other network configurations[2]: a model with no internal memory (FNN), a full recurrent model with no d-patch selection (LSTM) and a model that also implements an FNN and an RNN in parallel (FNN+LSTM). Note that the FNN+LSTM baseline resembles InfluenceNet in the sense that it can still choose whether to store certain input variables in memory or simply process them with the FNN. However, this architecture does not impose any constraints on the size of the RNN's input. For a fair comparison, and in order to ensure that both types of memory (frame stacking and RNN) have access to the same amount of information, the sequence length parameter in the recurrent models (number of time steps the network is unrolled when updating the model) is chosen to be equal to the number of frames that are fed into the FNN baseline. Recurrent models receive only 1 frame as input.

4.1 Traffic Control

In this environment, the agent must optimize the traffic flow at the intersection in Figure 3. The observation space is restricted 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 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. 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. 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. Agents need to anticipate cars appearing in the incoming lanes and switch the lights in time for the cars to continue without stopping. Knowing this, in the manual version of InfluenceNet, we place the d-patches at the end of the outgoing road segments and treat the rest of the hidden features in the local observation as Markovian (see grey boxes in Figure 3).

[2] Although the network architecture also suits any value based or policy gradient method, in our experiments we combine it with Proximal Policy Optimization (PPO) [26].

Table 1: Average score per episode and standard deviation over the last 200K timesteps after 50 hours of training (6 hours on the traffic domain). The scores are also averaged over three trials. Bold numbers indicate the best results on each environment. Multiple results are highlighted when the differences are not statistically significant.

	FNN	LSTM	FNN+LSTM	InfluenceNet aut.(man.)
Traffic Control	-6.72 ± 0.31	-3.93 ± 0.20	-5.51 ± 0.23	3.88 ± 0.08($\mathbf{-3.30 \pm 0.02}$)
Breakout	26.57 ± 1.51	21.32 ± 0.45	26.17 ± 0.82	$\mathbf{83.10 \pm 5.29}$
Pong	18.07 ± 0.06	-20.25 ± 0.03	$\mathbf{19.72 \pm 0.27}$	$\mathbf{20.07 \pm 0.11}$
Space Invaders	$\mathbf{854.93 \pm 11.64}$	520.44 ± 9.41	583.12 ± 8.76	$\mathbf{834.66 \pm 21.23}$
Asteroids	1393.75 ± 11.28	1424.87 ± 5.23	1841.87 ± 26.09	$\mathbf{2281.63 \pm 63.92}$
MsPacman	$\mathbf{2388.03 \pm 167.03}$	1081.11 ± 293.79	2097.97 ± 34.71	$\mathbf{2326.04 \pm 31.53}$

The results in Table 1 reveal that both the manual and automatic (static) versions of the InfluenceNet model outperform a FNN model that receives a fixed memory of 32 frames. This implies that memorizing the information in the selected d-patches is sufficient to estimate the effect of the hidden state variables on future transitions and rewards. Moreover, while providing the last 32 frames makes the environment fully observable (cars take a maximum of 32 steps to complete the big loop) the network is still unable to learn the optimal behavior. On the other hand, although the ordinary recurrent network can also reach the same level of performance it takes longer to converge due to its larger size. Apart from the score improvements, and as a result of shrinking the recurrent module by feeding only the selected d-patches, the total execution time is reduced by a factor of 2. Table 1 in the supplementary material shows a detailed runtime performance comparison of the four model architectures. Finally, the inductive bias that InfluenceNet imposes by preselecting the d-patches facilitates the job of the RNN component, which needs to deal with a lower dimensional input than the recurrent module in the FNN+LSTM baseline[3].

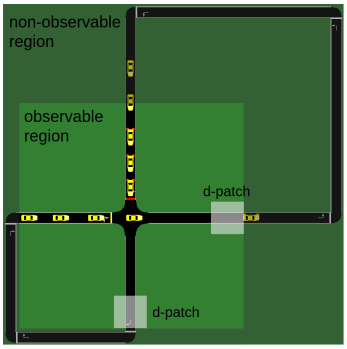

Figure 3: Traffic control task: The observable region corresponds to the non-shaded box centered at the intersection. The d-patches are placed at the edges of the intersection.

4.2 Atari

We also evaluated the InfluenceNet model on flickering Atari. This is a modified version of the standard games introduced by [8], where the observations are replaced by black frames with probability $p = 0.5$. This adds uncertainty to the environment and makes it more difficult for the agent to keep track of the ball. Agents trained using InfluenceNet on flickering Atari are able to outperform the full recurrent models (LSTM and FNN+LSTM). By feeding the RNN the vector computed by the d-patch selection operator $\hat{D}_{\alpha_t}(v_t)$, which contains only the most critical information, we effectively reduce the dimensionality of the input and ease the task of the recurrent cells (see Table 1). On the other hand, the model achieves similar or better scores than the FNN baseline which receives 8 frames as input. This is consistent with the results obtained on the traffic control task and brings more evidence to confirm that stacking a history of frames is not a viable option in POMDP domains with long time dependencies. Table 1 in the supplementary material shows the runtime performance comparison on the Atari games. Again, we see a runtime performance improvement with respect to the full RNN architectures (LSTM and FNN+LSTM) and when compared to the FNN model. Full learning curves and results on the standard Atari games are provided in the supplementary material

4.3 Architecture Analysis

This section contains a series of sanity checks that intend to confirm that the performance boost revealed by our previous results agrees with our initial hypotheses and it is not just an artifact of the network configuration. We studied the behavior of the agent after training and analyzed the internal representations of the neural network to make sure the effect of the inductive biases we imposed in the network architecture is aligned with the IBA perspective.

[3]Videos showing the results of the traffic control experiment can be found at https://tinyurl.com/wc3jpf4. Learning curves and more details about this environment are provided in the supplementary material.

4.3.1 Automatic d-patch selection maps

To assess whether our two automatic d-patch selection mechanisms can capture the regions in the observation space that intuitively have higher capability for predicting the hidden variables, we created d-patch selection maps. We placed a grey-scale heatmap on top of the game screen to highlight the regions that are given more weight. The two examples at the top in Figure 4 indicate that the method is able to track the bouncing ball and thus feeds into the RNN sufficient information to be able to estimate its velocity. Although we do not train the model on predicting next observations, the mechanism is able to discover what variables carry information about the non-Markovian dependencies. D-patch selection maps for the rest of the games can be found in the supplementary material.

4.3.2 Decoding the agent's internal memory

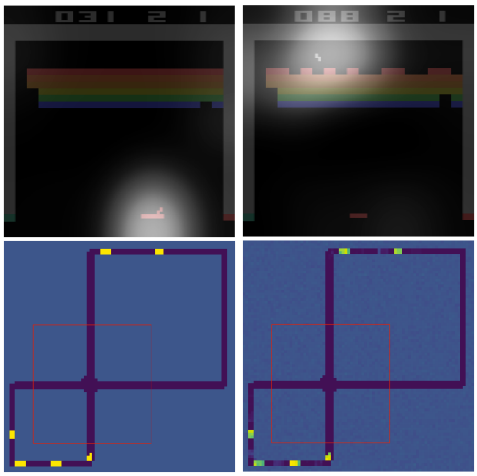

Figure 4: Examples of the regions chosen by the attention mechanism (top). Example of a full game screen (bottom left) and the reconstruction made by the memory decoder (bottom right)[3].

We also evaluated if the information stored in the agent's internal memory after selecting the d-patches and discarding the rest of the observation was sufficient to estimate the hidden state variables. We trained a decoder on predicting the full game screen given the encoded observation x_t and the internal memory of d-patches $\hat{d}_t$, using a dataset of images and neural activations collected after training the policy. The image at the bottom left of Figure 4 shows an example of the full game screen, from which the agent only receives the region delimited by the red box. The image at the bottom right shows the prediction made by the decoder. Note that although everything outside the red box is invisible to the agent, the decoder is able to make a fair reconstruction of the entire game screen based on the d-patch history encoded in the agent's internal memory $\hat{d}$. This implies that InfluenceNet can capture the necessary information and remember how many cars left the intersection and when without being explicitly trained to do so[4].

4.3.3 Analysis of the hidden activations

Finally, we used Canonical Correlation Analysis (CCA) [10] to measure the associations between the network hidden activations when playing Breakout and two important features in the game: ball velocity and number of bricks destroyed. We first computed the canonical variates that maximize the correlation between the RNN's hidden states $\hat{d}_t$ and the two components of the ball velocity vector. This yielded correlation coefficients of $\mathbf{0.98}$ and $\mathbf{0.90}$ with the first and second coordinates of the velocity vector. On the other hand, the same procedure carried out on the outputs of the FNN x_t found correlations of just $\mathbf{0.18}$ and $\mathbf{0.15}$. This suggests that the recurrent block is detaching the FNN from the task of estimating the influence of hidden variables so that it can focus on capturing useful information in the game screen that doesn't need to be memorized (see Definition 2). For instance, when we use CCA to compare x_t with the number of bricks destroyed in each frame we obtain a correlation coefficient of $\mathbf{0.96}$. These results are consistent with the observation that memorizing all information is unnecessary, and give more reasons to believe that connecting an FNN and an RNN in parallel can facilitate learning Markov representations of the form $\langle x_t, \hat{d}_t \rangle$. The projections of the hidden activations onto the space spanned by the canonical variates are depicted in Figure 5. The scatter plot on the left shows four distinct clusters of similar hidden states $\hat{d}_t$ which relate directly to the four possible directions of the velocity vector. The plot on the right, shows a clear uptrend. High values of the first canonical component of x_t correspond to frames with many missing bricks. More details about this experiment are given in the supplementary material.

[4] A video of this experiment where we use the decoder to reconstruct an entire episode can be found at https://tinyurl.com/y9cvuz7l. More examples are also provided in the supplementary material.

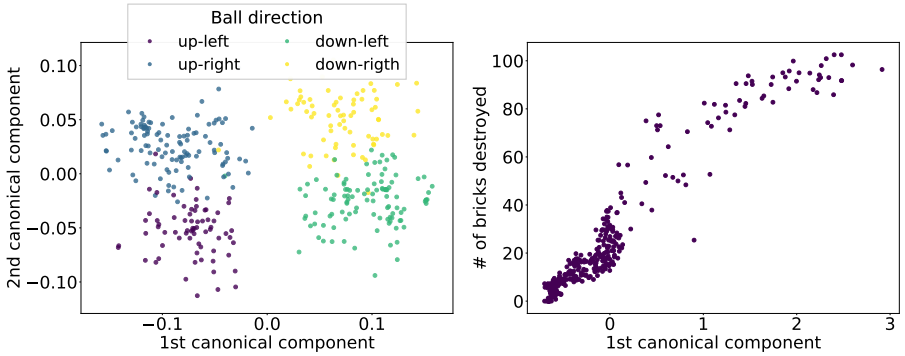

Figure 5: RNN's internal states $\hat{d}$ projected onto the two first canonical components, colors indicate the direction of the velocity vector (left). FNN's outputs x projected onto the first canonical component against the number of bricks destroyed (right).

5 Related Work

POMDP methods can be roughly divided into two broad categories. On the one hand, there are approaches that ignore the lack of the Markov property [27, 15]. Issues like chattering and divergence, however, make the applicability of these algorithms unsuitable for situations in which the observation space contains insufficient information [7, 5]. On the other hand, we have methods that more explicitly deal with non-Markovianity: policy search [14, 21], learning POMDP-like models [6, 4], predictive state representations [35, 36], EM-based methods [33, 16], AIXI [11, 32], recurrent policies with internal state [34, 8, 12]. However failure to converge or converging to poor quality local optima are typical issues of all these methods in practice. The InfluenceNet model lies in between these two methodologies in the sense that only those features of the observation input that are needed for predicting the hidden influence sources are fed into the network to update the internal memory while the rest are simply processed by an FNN.

The dynamic d-patch selection method we propose implements a spatial attention mechanism similar to the one used by [37] to automatically generate captions for images. This form of attention differs from the temporal attention mechanisms that are used in seq2seq models [18, 31] to condition on multiple past internal states. The architecture proposed by [28] also uses attention to select the important regions in the game screen. However, as opposed to InfluenceNet, their model does not connect an FNN in parallel to the recurrent network, and thus the attention mechanism can not just focus on what is important to remember but also needs to provide the RNN with enough information to predict current state values (see equation 5). This is probably the main reason for the poor results of their method on the Atari games. In Breakout, for example, knowing the layout of the bricks, and the position of the paddle is essential to estimate action values but this information does not need to be remembered and can just be processed by an FNN. Other works apply attention to facilitate the interpretation of the agent's behavior [20, 30] or to tackle multi-agent problems [13].

6 Conclusion

In this paper, we investigated if ideas from IBA can be encoded as inductive biases into a neural network. We tried to exploit the spatial structure of certain POMDP environments to reduce the dimensionality of the input to the recurrent layers and be able to learn the value function more effectively. The proposed architecture suits environments where the amount of information that the agent needs to retain represents a small fraction of the entire observation. Our results indicate that, by feeding into the recurrent cells only the d-patches while processing the rest of the image with an FNN, the InfluenceNet model facilitates the learning process, outperforms ordinary recurrent architectures and reduces training times. Moreover, the experiments reveal that while InfuenceNet can easily hold long-term memories, stacking multiple frames rapidly becomes impractical as the need for memorizing events in the distant past increases. Finally, the analysis of the model architecture in Section 4.3 gives strong evidence to confirm that the way our agents behave and represent their knowledge is aligned with the IBA framework.

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

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

[3] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[4] Finale Doshi-Velez. The infinite partially observable Markov decision process. In *Advances in Neural Information Processing Systems 22*, pages 477–485, 2009.

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

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

[7] Geoffrey Gordon. Stable function approximation in dynamic programming. In *Proc. of the Twelfth International Conference on Machine Learning*, pages 261–268, 1995.

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

[9] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On inductive biases in deep reinforcement learning. *arXiv preprint arXiv:1907.02908*, 2019.

[10] Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics*, pages 162–190. Springer, 1992.

[11] Marcus Hutter. *Universal Artificial Intellegence - Sequential Decisions Based on Algorithmic Probability*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005.

[12] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2117–2126, 2018.

[13] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2961–2970, 2019.

[14] Leslie Pack Kaelbling, Michael Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of AI Research*, 4:237–285, 1996.

[15] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *Proc. of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3*, pages 238–245, 1994.

[16] Yun-En Liu, Travis Mandel, Eric Butler, Erik Andersen, Eleanor O'Rourke, Emma Brunskill, and Zoran Popovic. Predicting player moves in an educational game: A hybrid approach. In *Proc. of the 6th International Conference on Educational Data Mining*, pages 106–113, 2013.

[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. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[18] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[19] Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

[20] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems*, pages 12329–12338, 2019.

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

[22] Frans A Oliehoek, Stefan Witwicki, and Leslie P Kaelbling. A sufficient statistic for influence in structured multiagent environments. *arXiv preprint arXiv:1907.09278*, 2019.

[23] Frans A. Oliehoek, Stefan J Witwicki, and Leslie Pack Kaelbling. Influence-based abstraction for multiagent systems. In *AAAI12*, 2012.

[24] Judea Pearl. *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[25] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley, 1994.

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

[27] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proc. of the International Conference on Machine Learning*, pages 284–292. Morgan Kaufmann, 1994.

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

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

[30] Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. *arXiv preprint arXiv:2003.08165*, 2020.

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

[32] Joel Veness, Kee Siong Ng, Marcus Hutter, William T. B. Uther, and David Silver. A Monte-Carlo AIXI approximation. *Journal of AI Research*, 40:95–142, 2011.

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

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

[35] David Wingate. Predictively defined representations of state. In *Reinforcement Learning*, pages 415–439. Springer, 2012.

[36] Britton D. Wolfe. *Modeling Dynamical Systems with Structured Predictive State Representations*. PhD thesis, University of Michigan, 2009.

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