INFLUENCE-AWARE MEMORY FOR DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Making the right decisions when some of the state variables are hidden, involves reasoning about all the possible states of the environment. An agent receiving only partial observations needs to infer the true values of these hidden variables based on the history of experiences. Recent deep reinforcement learning methods use recurrent models to keep track of past information. However, these models are sometimes expensive to train and have convergence difficulties, especially when dealing with high dimensional input spaces. Taking inspiration from *influence-based abstraction*, we show that effective policies can be learned in the presence of uncertainty by only memorizing a small subset of input variables. We also incorporate a mechanism in our network that learns to automatically choose the important pieces of information that need to be remembered. The results indicate that, by forcing the agent's internal memory to focus on the selected regions while treating the rest of the observable variables as Markovian, we can outperform ordinary recurrent architectures in situations where the amount of information that the agent needs to retain represents a small fraction of the entire observation input. The method also reduces training time and obtains better scores than methods that use a fixed window of experiences as input to remove partial observability in domains where long-term memory is required.

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. This type of problems can be modeled as *partially observable Markov decision processes (POMDP)* (Kaelbling et al., 1996). The model is an extension of the MDP framework (Puterman, 2014), 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. Assume for example that we want to control the traffic lights of a single intersection in a large city using only local information. Depending on the structure of the traffic network, some of the hidden variables (e.g. number of cars in other parts of the city) can have a strong impact on the agent's performance. Therefore, to be able to infer the influence of such variables on the intersection, the agent would benefit from memorizing the sequence of observations. However, not all the observed variables are equally important for estimating the traffic density outside the local region. In this case, it should be sufficient to remember how many cars left the intersection and when.

In this paper, we investigate whether we can learn policies that can control agents in the presence of uncertainty, by memorizing only a fraction of the input variables. Previous work on *influence-based abstraction (IBA)* (Oliehoek et al., 2012) demonstrates that the non-Markovian dependencies in the transiton and reward functions can be fully monitored by keeping track of a subset of variables in the history of actions and observations. Here we show how to adapt this approach to high dimensional reinforcement learning (RL) problems where no explicit model of the world is available. In particular, we use the insights from IBA to propose a different way of organizing the recurrent neural network (RNN) used for policy and value 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; thus imposing an inductive bias on the function. We also introduce a mechanism in our model that learns to direct the aim of the RNN towards the most relevant elements of the input, and thus allows the method to be used in general POMDP environments. We test our method on a simple traffic control task and four different Atari video games (Bellemare et al., 2013), obtaining equal performance to networks that try to overcome the partial observability issues by feeding a sliding window of 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 such cases, 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 agent's internal memory and forcing it to focus on the selected regions, we can speed up the learning process and find better policies than those obtained using full recurrent models.

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. The concepts of POMDP and IBA, which are the basis of our work, are outlined in Section 3. Section 4 explains how to adapt IBA to deep RL and describes the Influence Network structure. Finally, we discuss the results of our experiments (Section 5) and set the direction of future work (Section 7).

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 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 input space. These variables determine the traffic density at each road segment. The intuition is that a memoryless agent controlling the traffic lights in A could make instantaneous decisions based on the cars that are inside the red circle at the current timestep. 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.

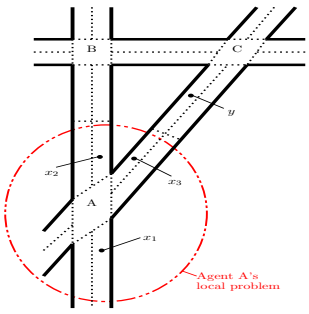


Figure 1: Traffic network.

3 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 (Kaelbling et al., 1996; Oliehoek et al., 2012).

3.1 Partially Observable Markov Decision Process

Definition 3.1 (POMDP) *A POMDP is a tuple* $\mathcal{M} = \langle S, A, T, R, \Omega, O \rangle$ *where* S *is the state space,* A *is the set of actions,* T *is the transition probability function, with* $T(s_t, a_t, s_{t+1}) = Pr(s_{t+1}|a_t, s_t)$*. That is, the probability of* s_{t+1} *being the next state given that the action* a_t *is taken in state* s_t*.* $R(s_t, a_t)$ *is defines the reward for taking action* a_t *in state* s_t*,* Ω *is the observation space,* O *is the observation probability function,* $O(a_t, s_{t+1}, o_{t+1}) = Pr(o_{t+1}|a_t, s_{t+1})$*, the probability of receiving observation* o_{t+1} *after taking the joint action* a_t *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 (Sutton & Barto, 1998). 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

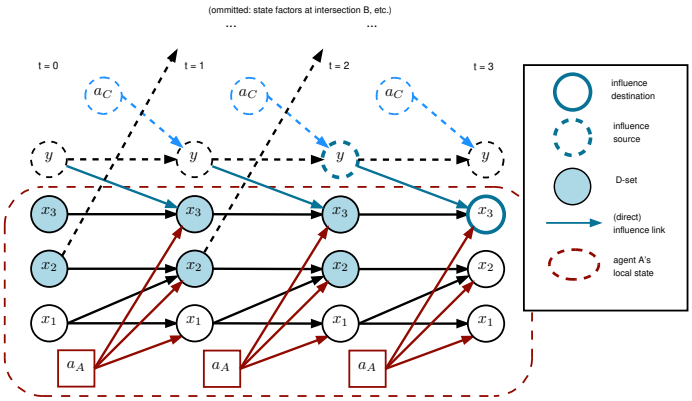


Figure 2: Simplified representation of the traffic example as a DBN unrolled over time. We use x for factors that belong to the agent's local observation and y to denote external state variables. The arrows illustrate the dependencies between the factors. For simplicity, some of the external state variables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

general, the agent is required to keep track of their past experiences to make the right action choices. Policies are therefore mappings from the history of past actions and observations h to actions.

3.2 Influence-based Abstraction

Conceptually the idea of *influence-based abstraction* (IBA) is: (1) choose a subset of observable variables that can fully describe the effect of the agent's actions on the hidden state variables, (2) build a model that can predict the influence of the hidden variables on the transition and reward functions based on the history of the selected observable variables, and finally, (3) use this influence predictor together with the current observation input to find the optimal policy. We illustrate this concept using the traffic example in Figure 1. The dynamics of the problem can be represented with a dynamic Bayesian network (DBN) (Pearl, 1988). The agent's local observation consists of the nodes labeled with x in the DBN depicted in Figure 2 where we unrolled three timesteps. According to the diagram, x_2 depends on the value of x_1 and x_2 at the previous timestep. The network also shows how the local state is affected by 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.

By inspecting both Figure 1 and the Bayesian network in Figure 2 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. In particular, we say that y is an *influence source* for x_3, the *influence destination*.

To find the optimal policy, we need to be able 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 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. Hence, x_3 does not follow the Markov property, since it does not only depend on the present local 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. The d-separating set d defines the smallest subset of variables in our local history that we need to condition on to estimate the effect of the influence sources. In other words, the d-separating set is the group of variables that d-separates (Bishop, 2006, Ch. 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.

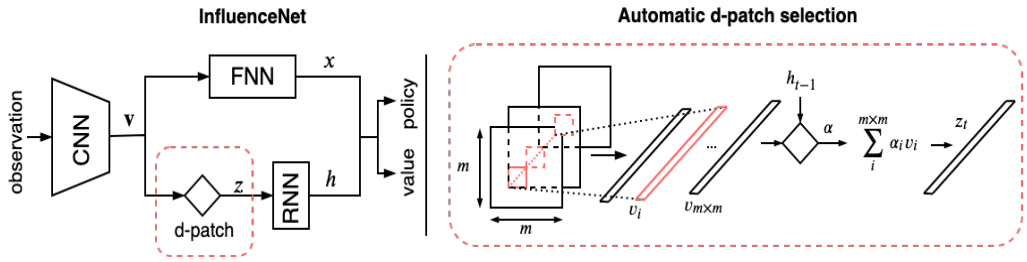


Figure 3: InfluenceNet architecture (left). Automatic d-patch selection (right)

4 INFLUENCE-AWARE MEMORY

The upshot of IBA is that one can replace the dependence on the full history of actions and observations by a dependence on the d-separating set and create local models without any loss in value. 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 to tackle complex problems where a model of the environment is not available. One of the advantages of using neural networks as function approximators is that we can directly work on high dimensional sensory input without having to extract more descriptive features. For instance, rather than using state variables in our traffic scenario, we can use images centered around intersection A to capture the traffic density.

As previously discussed, when receiving only partial information about the state space from the environment, some of the observed variables are no longer Markovian. This makes the usual deep RL methods inappropriate. An agent could still control the system by only using our most recent observation, but this might lead to sub-optimal decisions due to missing information on the hidden variables. Hence, it is essential that we treat the problem as a POMDP, and collect the information contained in the history of the past actions and observations. A naive approach to do so would be to store a finite window of past 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 dependence is too long. An alternative would be training an RNN to maintain and update a context of what has occurred up to the current timestep (Hausknecht & Stone, 2015; Wierstra et al., 2007). Unfortunately, this is particularly expensive for high-dimensional input spaces and even more so when 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 assumption is that in most complex problems we can capture the non-Markovian dependencies, by conditioning on the history of a small subset of local variables as long as that subset d-separates the influence sources from the rest of observable variables. However, since a DBN model characterizing the transition dynamics is not given, we can only rely on the reward signal to estimate the effect of the influence sources. This also means that explicitly identifying an exact d-separating is not possible. 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 propose to define ‘d-patches’, small regions in the input space, and feed these into recurrent layers. As such, we effectively distinguish two types of observable variables: those that we need to remember 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 with a fully connected feedforward neural network (FNN). However, to be able to account for 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 the RNN can maintain an internal memory of what happens there.

4.1 INFLUENCE NETWORK

The Influence Network (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 an FNN while the RNN receives only small preselected regions, the d-patches, combines them with its previous internal state and outputs the

influence prediction[1]. 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. This is only possible because the output of the CNN maintains the same spatial configuration as the input image.

4.2 AUTOMATIC D-PATCH SELECTION

Pinpointing the regions that need to be fed into the RNN requires some amount of prior domain knowledge. Besides, the d-patches might differ from one state to another in some situations. To overcome these potential limitations, we introduce a mechanism that allows the network to automatically select the most relevant regions at each particular state and thus, enables the use of InfluenceNet in more general POMDP domains where future rewards and transitions only depend on the current observation and a small subset of variables in the agent's history.

The method is based on the attention mechanism (Xu et al., 2015; Vaswani et al., 2017) and can be summarized as follows: first, the image is processed by the CNN which produces $m \times m$ vectors v of size D, where D is the number of filters in the last 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. A weighted linear combination of these vectors, which is called context vector z_t, is then fed into the RNN, $z_t = \sum_{i=0}^{m \times m} \alpha^i v_t^i$, where α^i is computed using a two-layer fully connected FNN that takes as input the i-th vector v_t^i and h_{t-1} followed by a softmax operator that calculates the weight for each of the regions; thus, $\sum_{i=0}^{m \times m} \alpha^i = 1$ (see Figure 3).

The purpose of the d-patch selection mechanism is to assist the RNN in focusing on the most relevant elements of the input image based on its current internal memory and decide which parts are important to remember. Intuitively, if we impose a constraint on the amount of information that is fed into the RNN and optimize the network via backpropagation through time, the algorithm should learn to put more weight on those regions of the memory input that influence the agent's future transitions and rewards the most, thus allowing the RNN to focus on the d-separating variables while discarding the rest.

5 EXPERIMENTS

The InfluenceNet[2] model was tested on a simple traffic control task and four atari video games against two other different network configurations: a model with no internal memory (FNN) and a full recurrent model with no d-patch selection (RNN). The goal of these experiments was to evaluate whether our approach improves over full recurrent models and see if it represents a better choice for handling partial observability than using a fixed-sized window of past frames. Moreover, we also analyzed whether the d-patch selection mechanism can capture the regions containing the d-separating variables and if this information can be used to estimate the future influence of the hidden state variables.

5.1 TRAFFIC CONTROL

In this environment, the agent must optimize the traffic flow at the intersection in Figure 4. 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

[1]The network is not being optimized to predict future states. However, we call this a prediction to be consistent with the IBA literature.

[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) (Schulman et al., 2017).

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 immediate reward consists of a penalty of -0.1 for each car that is not moving. We built the environment using SUMO (Simulator

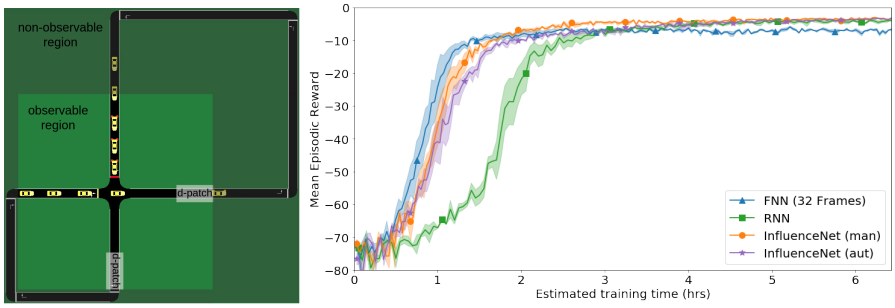


Figure 4: Traffic control task (left): The observable region corresponds to the non-shaded box centered at the intersection. The hand-specified d-patches are placed at the edges of the left and bottom road segments. Model performance comparison (right): Mean episodic reward as a function of training time . Shaded areas indicate the standard deviation of the mean.

of Urban Mobility) (Lopez et al., 2018). 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. Besides, the size of the observable region and the time delay between actions and traffic lights response were chosen so that an agent that merely reacts to cars entering the local region will be too late and cars will need 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. Agents need to anticipate cars appearing in the incoming lanes and switch the lights in time for the cars to continue without stopping [3].

The results depicted in Figure 4 show that both the manual and automatic versions of the InfluenceNet model outperform a FNN model that receives a fixed memory of 32 frames. This implies that memorizing the information contained 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 bigger 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 greatly reduced. Table 1 shows a runtime performance comparison of the three model architectures.

5.2 Atari

We also ran the InfluenceNet model on four different Atari video games: Pong, Breakout, Space Invaders and Flickering Pong, a variant of the game of Pong introduced by Hausknecht & Stone (2015), 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. Most of the games contain moving sprites whose speed and direction cannot be measured using only the most recent frame and thus constitute two hidden variables. Even though the spatial structure is less clear than in the traffic problem, these hidden variables can still be seen as influence sources that need to be estimated because they affect the reward and transition functions. In Breakout, for example, we can predict where the ball will be in the following frames based on its position in previous observations. To do so, it is not necessary to memorize the whole set of pixels in the game screen but only the ones that are around the ball. Intuitively, the history of these pixels corresponds to the d-separating set since together with the most recent frame they are all we need to predict future observations.

[3] Videos showing the results of the traffic control experiment can be found at `https://www.dropbox.com/sh/d90ukiusix9ciyv/AAC4R3pE17p3qGetTvrSKs14a?dl=0`

Table 1: Empirical analysis of model architecture runtime performance: Average wall-clock time and standard deviation of evaluating (forward pass) and updating (backward pass) the three models on traffic control and Atari. Unroll in the recurrent models indicates the length of the sequence of experiences that is used to update the model. One update involves processing an entire mini-batch of experiences. The test was run on an Intel Xeon CPU E5-2660 v4.

	Traffic			Atari		
		Evaluation (ms)	Update (ms)		Evaluation (ms)	Update (ms)
FNN	32 frames	**3.16** ±0.79	311.25 ±32.89	4 frames	**4.06** ±1.08	150.44 ±19.80
				8 frames	**4.81** ±0.82	224.89 ±11.91
RNN	Unroll 32	6.00 ±1.83	386.59 ±83.91	Unroll 4	5.31 ±2.67	159.28 ±43.66
				Unroll 8	5.53 ±2.10	186.71 ±20.31
InfluenceNet	Unroll 32	3.67 ±1.99	**253.44** ±81.01	Unroll 4	5.00 ±2.12	**136.18** ±35.16
				Unroll 8	**4.89** ±1.12	**153.19** ±16.29

Agents trained using InfluenceNet on Atari are able to outperform the full recurrent model on three of the four games (see Table 2). By feeding the RNN the context vector z, which contains only the most critical information, we reduce the number of parameters in the RNN and ease the task of the recurrent cells. The full RNN model, however, does significantly better on the game of Space Invaders. We hypothesize that the fact that the game displays many moving objects that are also scattered around the game screen poses a problem for the d-patch selection mechanism which cannot choose among all the possible regions and is only able to average over many of them, which leads to lossy compression. This is supported by Figure 10 in Appendix B. Moreover, the model achieves similar scores to the ordinary FNN architecture on the three standard Atari games and better scores on Flickering Pong. 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 general POMDP domains. See Appendix A for learning curves on the four games. Table 1 shows the runtime performance comparison on the Atari games. Again, we see a performance improvement with respect to the full RNN architecture and when compared to the FNN model that receives the last 8 frames as input used on Flickering Pong.

Table 2: Mean final scores and standard deviation over 50K steps on the Atari games after 10M timesteps of training averaged over three trials.

	Breakout	Pong	Space Invaders	Flickering Pong
FNN 4 frames (8 frames)	**317.2** ±11.93	**20.66** ±0.02	1035 ±107.08	16.08 ±0.34(18.16 ± 0.13)
RNN	306.51 ±31.76	-20.06 ±0.04	**1396** ±36.38	-20.18 ±0.18
InfluenceNet	**323.67** ±13.75	**20.80** ±0.07	866.11 ±38.00	**20.01** ±0.18

5.3 AUTOMATIC D-PATCH SELECTION MAPS

To assess whether the d-patch selection mechanism can capture the regions in the observation space that have a higher influence on the hidden variables, we created d-patch selection maps. We placed a greyscale heatmap on top of the game screen to highlight the regions that are given more weight at each timestep. The two images on the left of Figure 5 show that the network is able to track the cars entering and leaving the intersection and therefore select the right d-patches that need to be memorized. Please refer to Appendix B to see the d-patch selection maps in the Atari games.

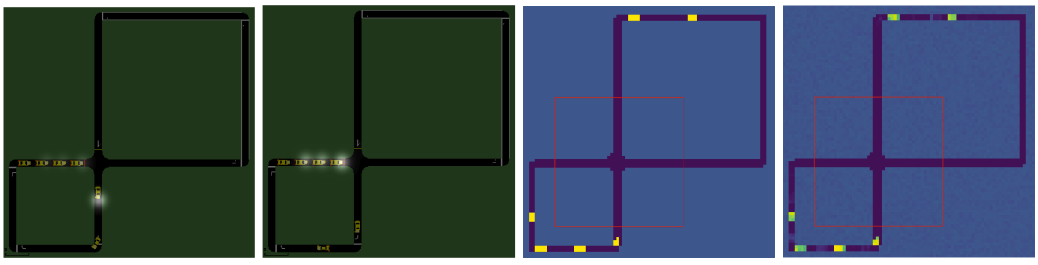


Figure 5: Two examples of the regions chosen by the d-patch selection mechanism (1st and 2nd images). True state (3rd image) and state predicted by the memory decoder (4th image).

5.4 DECODING THE AGENT'S INTERNAL MEMORY

We also evaluated if the information stored in the agent's internal memory after selecting the d-patches and discarding the rest of the observable variables was sufficient to estimate the influence of the hidden state variables. To that end, we trained a decoder neural network on predicting the full state of the traffic network given the encoded observation and the internal memory, using a dataset of images and hidden activations collected after training the policy. The second image from the right in Figure 5 shows the full game screen, from which the agent only receives the region delimited by the red box. The image further to the 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 history of previous d-patches stored in the agent's internal memory. This implies that the InfluenceNet architecture can correctly capture the necessary information and rememeber how many cars left the intersection and when, without being explicitly trained to do so. More results and further details about how the decoder neural network was trained are provided in Appendix C.

6 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 (Singh et al., 1994; Littman, 1994; Mnih et al., 2015). Issues like chattering and divergence, however, make the applicability of these algorithms unsuitable for situations in which the observation space contains insufficient information (Gordon, 1995; Fairbank & Alonso, 2012). On the other hand, we have methods that more explicitly deal with non-Markovianity: policy search (Kaelbling et al., 1996; Ng & Jordan, 2000), learning POMDP-like models (Ghahramani, 2001; Doshi-Velez, 2009), predictive state representations (Wingate, 2012; Wolfe, 2009), EM-based methods (Vlassis & Toussaint, 2009; Liu et al., 2013), AIXI (Hutter, 2005; Veness et al., 2011), recurrent policies with internal state. (Wierstra et al., 2007; Hausknecht & Stone, 2015). 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 a priori for predicting future states are fed back into the network to update the internal memory while the rest are simply processed by an FNN. The attention mechanism has successfully been used in the fields of computer vision (Xu et al., 2015) and natural language processing (Vaswani et al., 2017). The method was also applied to deep RL (Sorokin et al., 2015) in a similar fashion to the automatic d-patch selection mechanism that we propose. However, while they show that the network is able to select the most important area in the game screen, it also discards relevant information. This is probably the main reason for the poor performance obtained by their method on some of the Atari games. In the case of Breakout, for example, knowing the layout of the bricks, and specially the position of the paddle that the agent controls, is essential to estimate action values. We mitigate these issues by not only feeding the RNN with the selected regions but also processing the full encoded image with a FNN.

7 CONCLUSION

In this paper, we investigated if ideas from *influence-based abstraction* can be adapted to complex RL problems, where no model of the environment is available. We tried to estimate the influence that hidden state variables exert in the agent's transitions and rewards by remembering a small fraction of the information available and while only receiving feedback from the reward signal. Our results indicate that, by feeding into the recurrent cells only the variables that belong to the d-separating set, we are able to simplify the neural network architecture, considerably reducing training times. Moreover, the inductive bias imposed in the InfluenceNet model seems to facilitate the learning process and offers a better alternative than ordinary recurrent architectures in situations where the amount of information that the agent needs to retain represents a small fraction of the entire observation input. Unlike other recurrent methods, we show that, by restricting the memory input to the relevant areas of the observable region, we can obtain on par results with models that try to ensure Markovianity by feeding a sliding window of past frames. The experiments reveal that, while the InlfuenceNet model can hold long-term memories, stacking multiple frames rapidly becomes impractical as the need for memorizing events in the distant past increases.

REFERENCES

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.

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

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

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)*, pp. 1–8. IEEE, 2012. ISBN 978-1-4673-1488-6.

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

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

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.

Marcus Hutter. *Universal Artificial Intellegence - Sequential Decisions Based on Algorithmic Probability*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005. ISBN 978-3-540-22139-5. doi: 10.1007/b138233.

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

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*, pp. 238–245, 1994.

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*, pp. 106–113, 2013.

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. URL `https://elib.dlr.de/124092/`.

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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

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

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

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

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

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

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*, pp. 284–292. Morgan Kaufmann, 1994.

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

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

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*, pp. 5998–6008, 2017.

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. doi: 10.1613/jair.3125. URL http://dx.doi.org/10.1613/jair.3125.

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

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

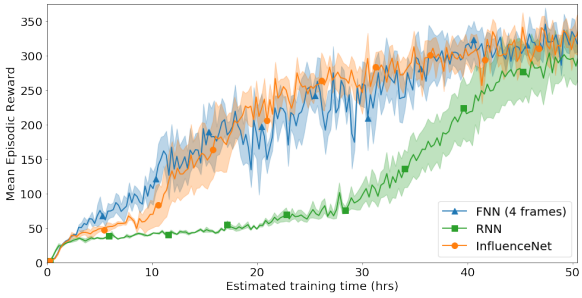
David Wingate. Predictively defined representations of state. In *Reinforcement Learning*, pp. 415–439. Springer, 2012.

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

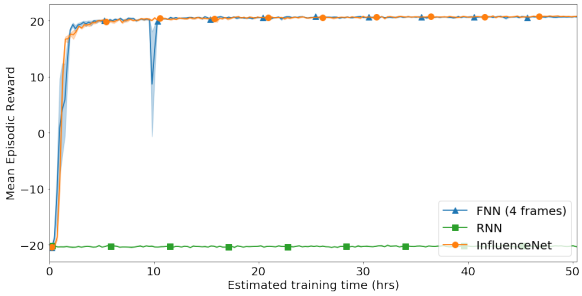
Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of the 32nd International Conference on Machine learning*, pp. 2048–2057, 2015.

A PERFORMANCE COMPARISON ON THE ATARI DOMAIN

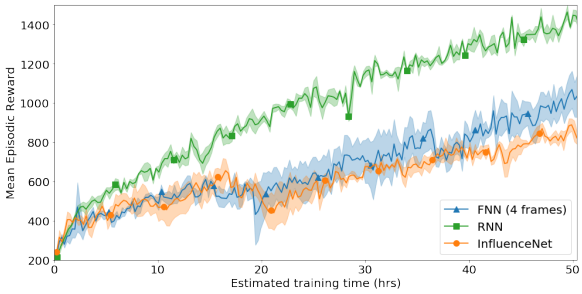
Figure 6 shows the mean episodic reward comparison of the three model architectures (FNN, RNN and InfluenceNet) as a function of training time on the Atari games. The FNN model receives 4 frames as input in all the four games except in Flickering Pong where we also tested a version with 8 frames. To update the network the recurrent models are unrolled 4 timesteps in the games of Breakout, Pong and Space Invaders and 8 timesteps in Flickering Pong. Since we could not run all our experiments on the same machine, training time is estimated using the values provided in Table 1.



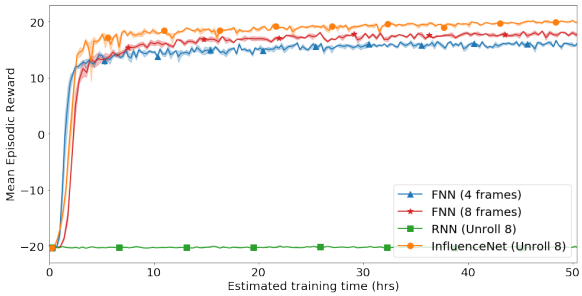
(a) Breakout



(b) Pong



(c) Space Invaders



(d) Flickering Pong

Figure 6: Model performance comparison (right): Mean episodic reward as a function of training time. Shaded areas indicate the standard deviation of the mean.

B AUTOMATIC D-PATCH SELECTION MAPS

Here we include a collection of images which show the weights that the automatic selection mechanism gives to the different regions of the game screen based on the agent's internal memory and the current observation. The network learns to focus on the regions that are important to memorize. This is not the case in Space Invaders where we believe that the fact that the game displays many moving objects that are also scattered around the game screen poses a problem for the d-patch selection mechanism which cannot choose among all the possible regions and is only able to average over many them (see Figure 10).

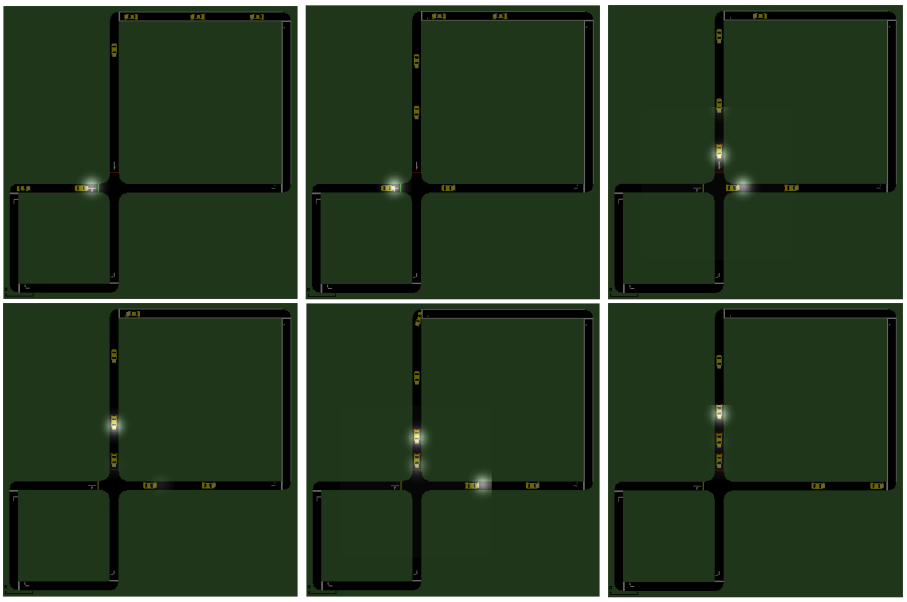


Figure 7: Traffic control

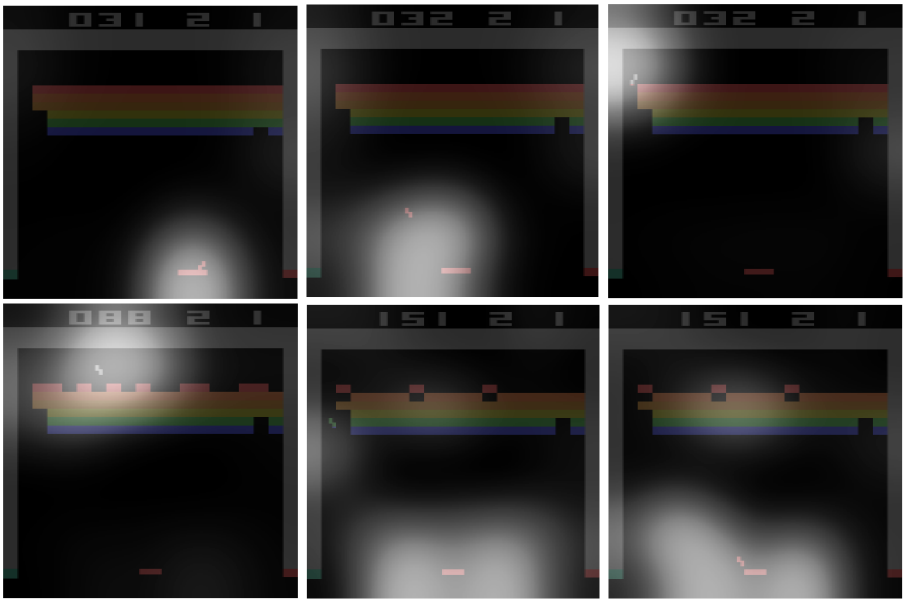


Figure 8: Breakout

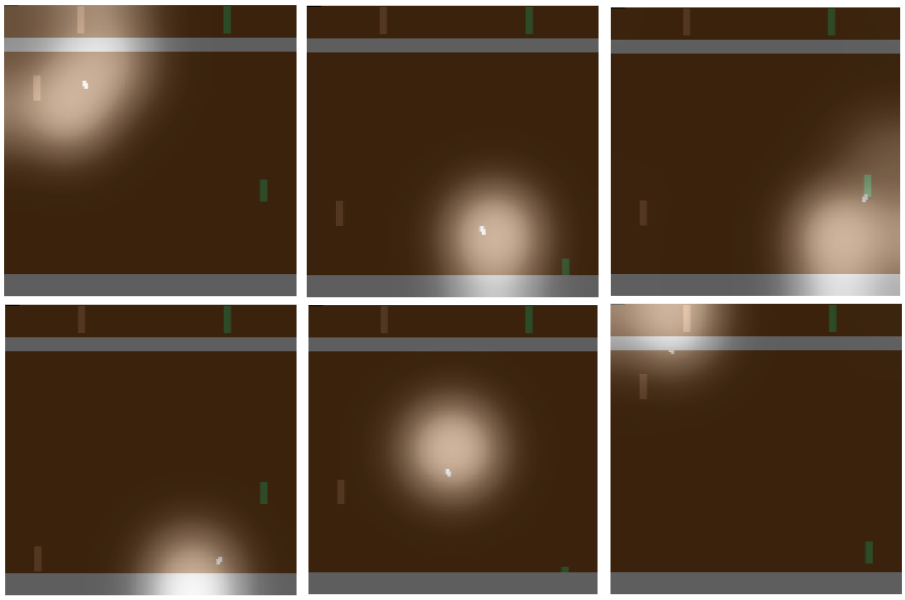


Figure 9: Pong

Figure 10: Space Invaders

C DECODING THE AGENT'S INTERNAL MEMORY

The memory decoder consists of a two-layer fully connected FNN which takes as input the RNN's internal memory h_{t-1} and the output x_t of the FNN (see Figure 3) and outputs a prediction for each of the pixels in the screen. The network is trained on a dataset containing screenshots and their corresponding network activations (h_{t-1} and x_t), using the pixel-wise cross entropy loss. The dataset is collected after training the agent's policy. The images are first transformed to grey-scale and then normalized to simplify the task. The results are shown in Figure 11. The goal of this experiment was to confirm that although the agent can only see the region delimited by the red boxes, the information regarding the number and location of cars outside the agent's observation is somehow stored in the internal memory. It is important to point out that the agent is only trained to maximize the

reward and has no explicit knowledge of the environment dynamics. We believe d-patch selection mechanism plays an important role since it ensures that the RNN focuses on those variables that can influence future transitions and rewards, namely the d-separating set, while ignoring the rest.

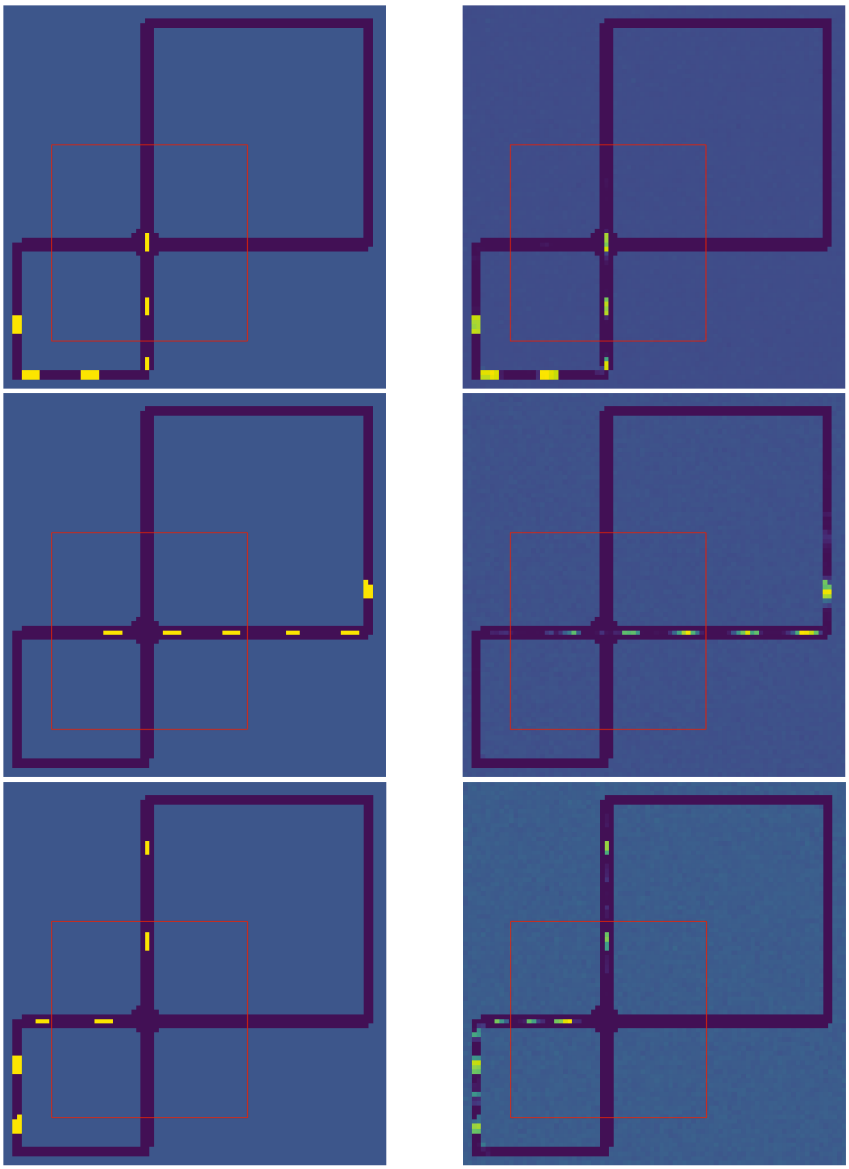


Figure 11: True state (left) and state predicted by the memory decoder (right)

D HYPERPARAMETERS

The hyperparameters used in the two experiments are provided in tables 3 and 4. We used the same hyperparameter configuration reported by Schulman et al. (2017) 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 for 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. In Atari, we used a sequence length

of 4 timesteps for the original Atari video games and 8 frames for Flickering Pong so as to provide the agent with the same amount of information as the FNN model. 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. Additionally, Table 4 includes the automatic d-patch selection network configuration. 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 3: Observation boxes and hyperparameters used 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	-		84		-	-
frame_width	-		84		-	-
Algorithm	PPO		PPO+RNN		PPO+InfluenceNet	
num_frames	32		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
FNN	layer 1		None		layer 1	
units	256		-		128	
RNN	None		layer 1		layer 1	
input	-		local		man. and aut. d-patches	
rec_units	-		256		128	
seq_length	-		32		32	

Table 4: Hyperparameters used for Atari.

Atari									
Algorithm	PPO			PPO+RNN			PPO+InfluenceNet		
num_frames	4 (8 Flickering Pong)			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 3	layer 1	layer 2	layer 3	layer 1	layer 2	layer 3
input	4 frames	-	-	1 frame	-	-	1 frame	-	-
filters	32	64	64	32	64	64	32	64	64
kernel_size	8	4	3	8	4	3	8	4	3
strides	4	2	1	4	2	1	4	2	1
FNN	layer 1			None			layer 1		
units	512			-			256		
Aut. d-patch	None			None			layer 1		layer 2
units	-			-			256		1
RNN	None			layer 1			layer 1		
input	-			full encoded image			d-patch (automatic selection)		
rec_units	-			512			256		
seq_length	-			4 (8 Flickering Pong)			4 (8 Flickering Pong)		