Appendix

Influence-aware Memory Architectures for Deep Reinforcement Learning

A. Implementation details

Warehouse and traffic control: The FNN and the LSTM baselines that we used in the warehouse and the traffic control environments, have two hidden layers. Only the second layer in the LSTM baseline is recurrent. This seems to work much better than the reverse option (i.e. first layer recurrent and second layer feedforward). Moreover, to ensure a fair comparison, the size of the first layer in IAM is set equal to the size of the first layer in the FNN and the LSTM baselines. That is, adding together the number neurons in the FNN and the RNN channels. For instance, if the first layer of the FNN and the LSTM baselines is 640 neurons, a valid configuration for IAM would be 512 feedfoward and 128 recurrent neurons. Such that the FNN and the RNN combined also add up to 640. When doing observation-stacking the FNN baseline is provided the last 8 observations in the warehouse and 32 observations in the traffic environment. On the other hand, the gradients in the recurrent architectures (LSTM and IAM) are backpropagated for 8 and 32 timesteps (sequence length parameter). We did extensive testing to try to find the best possible network configuration for each model. These are reported in Table 1 for the warehouse and Table 2 for the traffic environment. We tried different combinations of $\{128, 256, 512, 640\}$ and $\{32, 64, 128, 256\}$ for the size of the first and second feedfoward layers, respectively. As reported in section 5.3 (Figure 3), we also tried with $\{8, 16, 32, 64, 128, 256\}$ neurons for the recurrent layers in LSTM and IAM.

Table 1. Hyperparameters used for Warehouse Commissioning.

Warehouse						
Model	FNN		LSTM		IAM	
num frames	32		1		1	
time horizon	16		16		16	
seq length	-		8		8	
FNN	layer 1	layer 2	layer 1	-	layer 1	layer 2
neurons	640	256	640		512	256
RNN	None		layer 2		layer 1	
rec neurons	-		128		128	

Table 2. Hyperparameters used for Traffic Control.

Traffic Control						
Model	FNN		LSTM		IAM	
num frames	8		1		1	
seq length	-		32		32	
time horizon	128		128		128	
FNN	layer 1	layer 2	layer 1	-	layer 1	layer 2
neurons	256	64	256		128	64
RNN	None		layer 2		layer 1	
rec neurons	-		128		128	

Flickering Atari: As explained in Section 4.1, when the observations are images, instead of feeding them directly into the FNN or the RNN, we first preprocess them with a CNN. The FNN and LSTM baselines use the same architecture reported by Mnih et al. (2015) and Hausknecht & Stone (2015) respectively. The IAM maintains the same CNN configuration, and then connects 128 feedforward and 128 recurrent neurons in parallel so that the total number of neurons (256) remains the same as in the FNN and LSTM baselines. Additionally, the attention mechanism, which computes A_t for every $\langle o_t, \hat{d}_t \rangle$, consists of a single head with 256 neurons. The FNN baseline receives the last 8 frames as input, whereas LSTM and IAM only receive 1 frame. To update the network, gradients in the recurrent models are backpropagated for 8 timesteps. The network configurations used for Flickering Atari, are provided in Table 3.

Table 3. Hyperparameters used for Flickering Atari.

Flickering Atari									
Model	FNN			LSTM			IAM		
num frames	8			1			1		
seq length	-			8			8		
time horizon	128			128			128		
CNN	layer 1	layer 2	layer 3	layer 1	layer 2	layer 3	layer 1	layer 2	layer 3
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		
neurons	512			-			256		
Attention	None			None			layer 1		
neurons	-			-			256		
num heads	-			-			1		
RNN	None			layer 1			layer 1		
rec neurons	-			512			256		

As for the hyperparameters specific to PPO we used the same values reported by Schulman et al. (2017) for all three models and the three domains. These are shown in Table 4.

Table 4. PPO hyperparameters.

learning rate	2.5e-4
discount γ	0.99
GAE λ	0.95
memory size	128
batch size	32
num. epoch	3
num. workers	8
entropy β	1.0e-2
clip ϵ	0.1
value coeff. c_1	1

B. Runtime performance

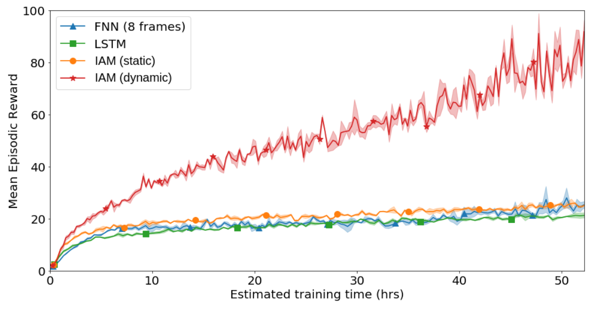
Table 5 is an empirical analysis of the runtime performance of the three architectures. The values are the average wall-clock time in milliseconds of evaluating (forward pass) and updating (backward pass) the models. The FNN baseline receives a stack of 8, 32 and 8 observations in the Warehouse, Traffic Control and Flickering Atari environments, respectively. On the other hand, gradients in LSTM and IAM are backpropagated for 8, 32 and 8 timesteps when updating the networks in each of the three environments. An evaluation corresponds to a forward pass through the network. One update involves 3 epochs over a batch of 1024 experiences. The test was run on an Intel Xeon CPU E5-1620 v2.

Table 5. Runtime performance in milliseconds

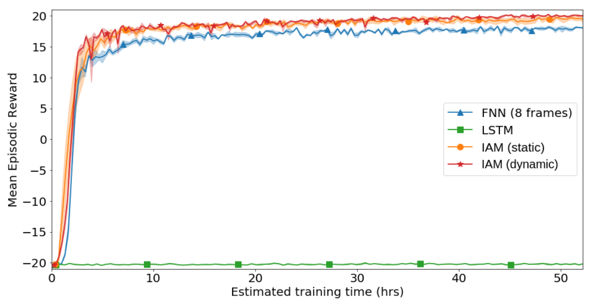
	Warehouse		Traffic		Flickering Atari	
	Evaluation	Update	Evaluation	Update	Evaluation	Update
FNN (obs-stacking)	1.03 ± 0.41	81.96 ± 1.29	0.90 ± 0.16	63.21 ± 0.71	3.82 ± 0.53	215.95 ± 7.54
LSTM	1.27 ± 0.27	151.10 ± 2.98	1.27 ± 0.12	167.36 ± 10.41	5.41 ± 1.13	285.19 ± 24.60
IAM	1.32 ± 0.50	82.60 ± 4.87	1.05 ± 0.13	144.96 ± 13.51	3.47 ± 1.84	163.30 ± 28.19

C. Learning curves

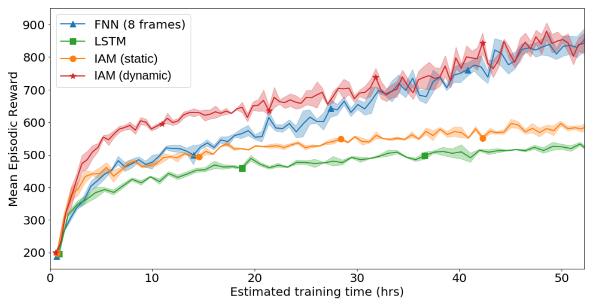
Figure 1 shows the mean episodic reward comparison of the two baselines FNN and LSTM and the two versions of IAM, static and dynamic, as a function of training time on the *flickering* Atari games. Since we could not run all our experiments on the same machine, training time is estimated using the values provided in Table 5.



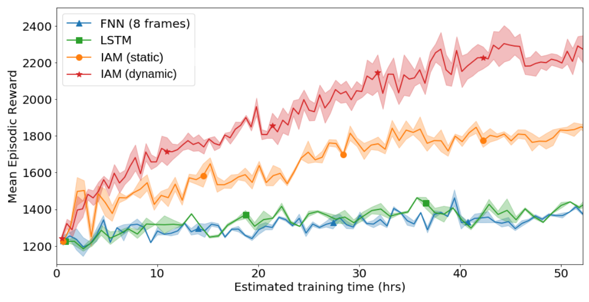
(a) Breakout



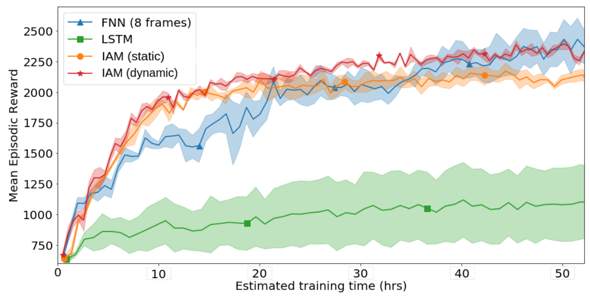
(b) Pong



(c) Space Invaders



(d) Asteroids



(e) MsPacman

Figure 1. Average score per episode as a function of training time. Shaded areas show the standard deviation of the mean.

D. Results on the original Atari games

We also tested IAM on the original Atari games. Although the full game screen is visible at all times, some of the games contain moving elements whose speed and direction can not be measured from just a single frame. This means that the original games are also POMDPs (Hausknecht & Stone, 2015). In DQN (Mnih et al., 2015) this issue is easily solved by stacking the last 4 frames and feeding them into the network. Our experiments show (Table 6), that even when frame-stacking seems to be the optimal solution, IAM can reach the same or even better performance of the FNN model while clearly outperforming the LSTM baseline.

E. 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 $\hat{d}_{t-1}$ 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

Table 6. Average score per episode and standard deviation over the last 200K timesteps after 50 hours of training on the original (non-flickering) Atari games. 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 (4 frames)	LSTM	IAM (static)	IAM (dynamic)
Breakout	$\mathbf{326.54 \pm 22.21}$	127.37 ± 26.13	295.73 ± 31.40	$\mathbf{339.42 \pm 21.96}$
Pong	$\mathbf{20.68 \pm 0.07}$	-20.26 ± 0.04	$\mathbf{20.73 \pm 0.08}$	$\mathbf{20.74 \pm 0.04}$
Space Invaders	975.39 ± 70.06	1177.47 ± 24.81	$\mathbf{1326.63 \pm 53.59}$	868.23 ± 37.34
Asteroids	2184.61 ± 62.83	1618.89 ± 41.93	2057.21 ± 109.65	$\mathbf{2792.45 \pm 60.40}$
MsPacman	$\mathbf{3859.91 \pm 751.30}$	667.58 ± 32.39	$\mathbf{4016.78 \pm 273.11}$	3294.29 ± 222.64

trained on a dataset containing screenshots and their corresponding network activations ($\hat{d}_{t-1}$ 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 2. The goal of this experiment was to confirm that although the RNN is only fed the last two elements of the binary vectors representing each lane, it can still uncover hidden state. 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. A video of this experiment can be found at `https://tinyurl.com/y9cvuz7l`.

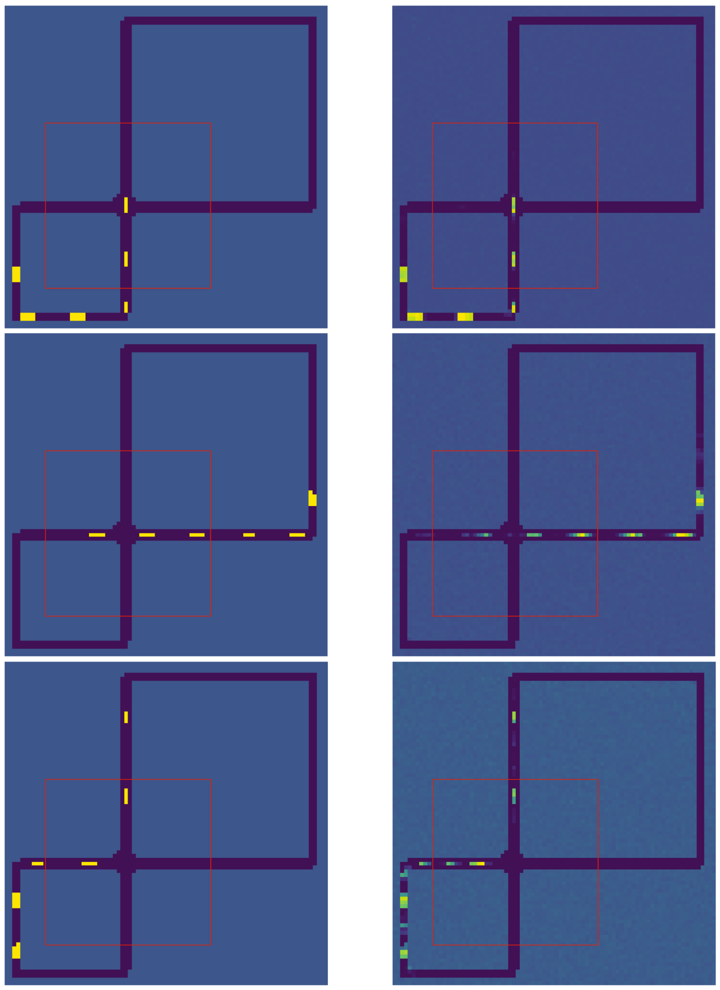


Figure 2. True state (left) and state predicted by the memory decoder (right)

F. Analysis of the hidden activations

We further analyze the information contained in $\hat{d}$ when playing Breakout (non-flickering) by looking at the activation patterns at different timesteps and their correlation with the ball velocity. The velocity vector v is computed by comparing the location of the ball at two subsequent frames. We use Canonical Correlation Analysis (CCA) (Hotelling, 1992) to obtain a lower dimensional representation of $\hat{d}$. In broad terms, CCA is a linear transformation that projects two sets of variables, in this case $\hat{d}$ and v, onto two subspaces that are highly correlated. This is done by iteratively finding linear combinations of $\hat{d}$ and v, known as canonical components, that maximize the correlation between them while being uncorrelated with the previous canonical components. We found that the first two canonical components of $\hat{d}$ are highly correlated with the two coordinates of the velocity vector ($\mathbf{0.98}$ and $\mathbf{0.90}$).

We also show that the FNN's output x, on the other hand, contains information that does not need to be memorized but is relevant for predicting action values. We applied CCA to compare x with the number of bricks destroyed at each frame and obtained a correlation coefficient of $\mathbf{0.96}$. The projections of the $\hat{d}$ and x onto the space spanned by their corresponding cannonical components are shown in the two scatter plots on the left of Figure 7 in the paper.

G. Attention maps

Here we include a collection of images which show the weights that the attention mechanism gives to the different regions of the game screen based on the agent's internal memory $\hat{d}$ and the current observation $\hat{o}_t$. The network seems to learn to focus on the regions that are important to memorize.

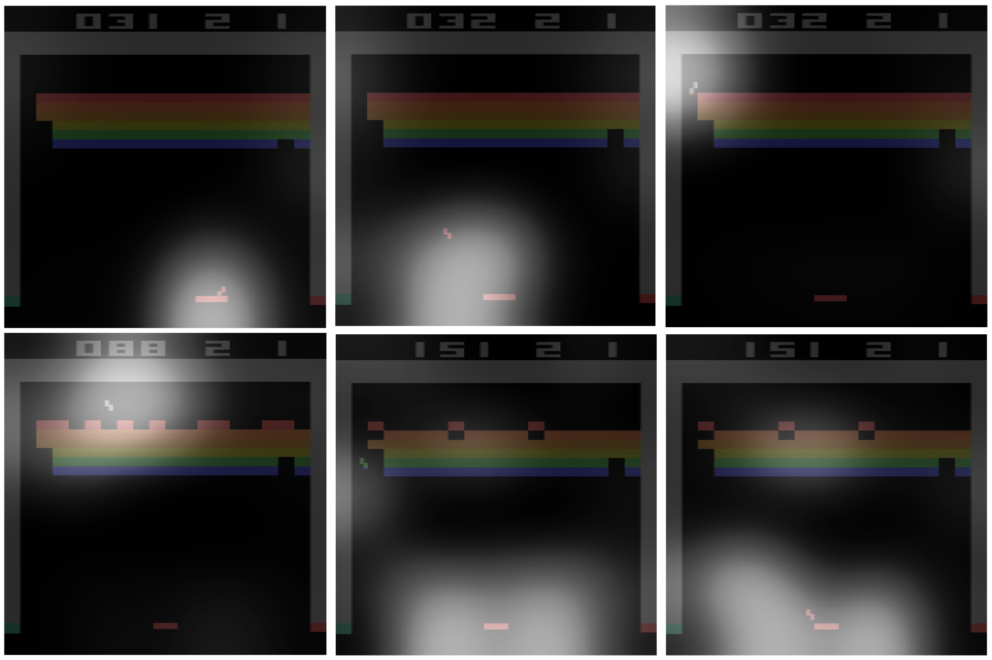


Figure 3. Breakout

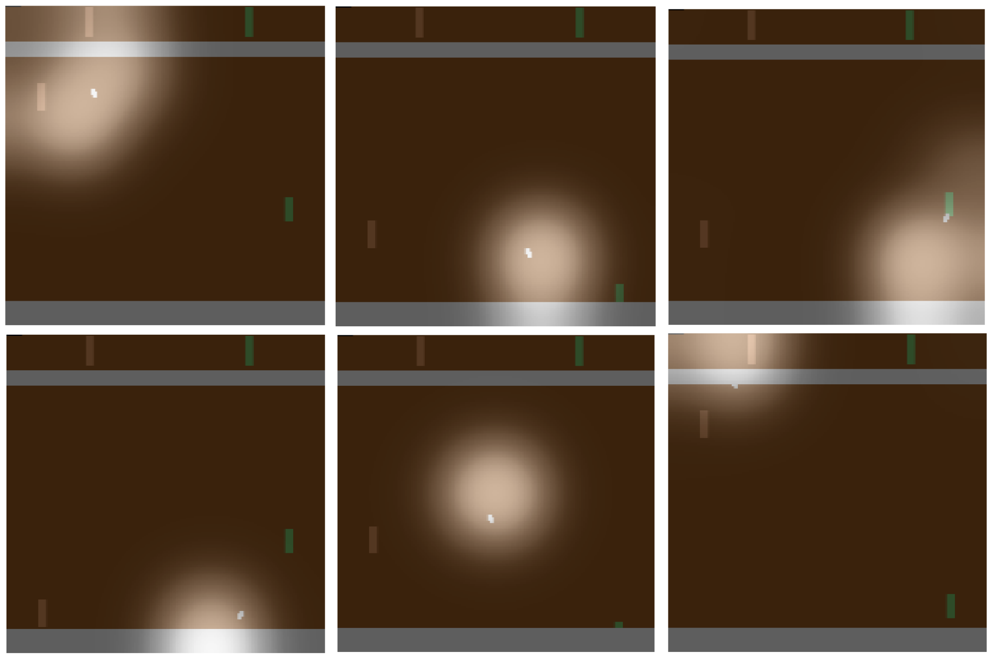


Figure 4. Pong

References

Hausknecht, M. and Stone, P. Deep recurrent Q-learning for partially observable MDPs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Hotelling, H. Relations between two sets of variates. In *Breakthroughs in statistics*, pp. 162–190. Springer, 1992.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.