Hybrid Reward Architecture for Reinforcement Learning

Harm van Seijen[1]
harm.vanseijen@microsoft.com

Mehdi Fatemi[1]
mehdi.fatemi@microsoft.com

Joshua Romoff[12]
joshua.romoff@mail.mcgill.ca

Romain Laroche[1]
romain.laroche@microsoft.com

Tavian Barnes[1]
tavian.barnes@microsoft.com

Jeffrey Tsang[1]
tsang.jeffrey@microsoft.com

[1]Microsoft Maluuba, Montreal, Canada
[2]McGill University, Montreal, Canada

Abstract

One of the main challenges in reinforcement learning (RL) is generalisation. In typical deep RL methods this is achieved by approximating the optimal value function with a low-dimensional representation using a deep network. While this approach works well in many domains, in domains where the optimal value function cannot easily be reduced to a low-dimensional representation, learning can be very slow and unstable. This paper contributes towards tackling such challenging domains, by proposing a new method, called Hybrid Reward Architecture (HRA). HRA takes as input a decomposed reward function and learns a separate value function for each component reward function. Because each component typically only depends on a subset of all features, the overall value function is much smoother and can be easier approximated by a low-dimensional representation, enabling more effective learning. We demonstrate HRA on a toy-problem and the Atari game Ms. Pac-Man, where HRA achieves above-human performance.

1 Introduction

In reinforcement learning (RL) (Sutton & Barto, 1998; Szepesvári, 2009), the goal is to find a behaviour policy that maximises the return—the discounted sum of rewards received over time—in a data-driven way. One of the main challenges of RL is to scale methods such that they can be applied to large, real-world problems. Because the state-space of such problems is typically massive, strong generalisation is required to learn a good policy efficiently.

Mnih et al. (2015) achieved a big breakthrough in this area: by combining standard RL techniques with deep neural networks, they outperformed humans on a large number of Atari 2600 games, by learning a policy from pixels. The generalisation properties of their Deep Q-Networks (DQN) method is performed by approximating the optimal value function. A value function plays an important role in RL, because it predicts the expected return, conditioned on a state or state-action pair. Once the optimal value function is known, an optimal policy can easily be derived. By modelling the current estimate of the optimal value function with a deep neural network, DQN carries out a strong generalisation on the value function, and hence on the policy.

The generalisation behaviour of DQN is achieved by regularisation on the model for the optimal value function. However, if the optimal value function is very complex, then learning an accurate low-dimensional representation can be challenging or even impossible. Therefore, when the optimal value function cannot easily be reduced to a low-dimensional representation, we argue to apply a new, complementary form of regularisation on the target side. Specifically, we propose to replace the reward function with an alternative reward function that has a smoother optimal value function, but still yields a reasonable—but not necessarily optimal—policy, when acting greedily.

A key observation behind regularisation on the target function is the difference between the *performance objective*, which specifies what type of behaviour is desired, and the *training objective*, which provides the feedback signal that modifies an agent's behaviour. In RL, a single reward function often takes on both roles. However, the reward function that encodes the performance objective might be awful as a training objective, resulting in slow or unstable learning. At the same time, a training objective can differ from the performance objective, but still do well with respect to it.

Intrinsic motivation (Stout et al., 2005; Schmidhuber, 2010) uses the above observation to improve learning in sparse-reward domains. It achieves this by adding a domain-specific intrinsic reward signal to the reward coming from the environment. Typically, the intrinsic reward function is potential-based, which maintains optimality of the resulting policy. In our case, we define a training objective based on a different criterion: smoothness of the value function, such that it can easily be represented by a low-dimensional representation. Because of this different goal, adding a potential-based reward function to the original reward function is not a good strategy in our case, since this typically does not reduce the complexity of the optimal value function.

Our main strategy for constructing a training objective is to decompose the reward function of the environment into n different reward functions. Each of them is assigned to a separate reinforcement-learning agent. Similar to the Horde architecture (Sutton et al., 2011), all these agents can learn in parallel on the same sample sequence by using off-policy learning. For action selection, each agent gives its values for the actions of the current state to an aggregator, which combines them into a single action-value for each action (for example, by averaging over all agents). Based on these action-values the current action is selected (for example, by taking the greedy action).

We test our approach on two domains: a toy-problem, where an agent has to eat 5 randomly located fruits, and Ms. Pac-Man, a hard game from the ALE benchmark set (Bellemare et al., 2013).

2 Related Work

Horde architecture. Our HRA method builds upon the Horde architecture (Sutton et al., 2011). The Horde architecture consists of a large number of 'demons' that learn in parallel via off-policy learning. Each demon trains a separate general value function (GVF) based on its own policy and *pseudo-reward* function. A pseudo-reward can be any feature-based signal that encodes useful information. The Horde architecture is focused on building up general knowledge about the world, encoded via a large number of GVFs. HRA focusses on training separate components of the environment-reward function, in order to achieve a smoother value function to efficiently learn a control policy.

Learning with respect to multiple reward functions is also a topic of multi-objective learning (Roijers et al., 2013). So alternatively, HRA can also be viewed as applying multi-objective learning in order to smooth the value function of a single reward function.

Options / Hierarchical Learning. This work is also related to options (Sutton et al., 1999; Bacon et al., 2017), and more generally hierarchical learning (Barto & Mahadevan, 2003; Kulkarni et al., 2016). Options are temporally-extended actions that, like HRA's heads, can be trained in parallel based on their own (intrinsic) reward functions. However, once an option has been trained, the role of its intrinsic reward function is over. A higher-level agent that uses an option sees it as just another action and evaluates it using its own reward function. This can yield great speed-ups in learning and help substantially with better exploration, but they do not directly make the value function of the higher-level agent less complex. The heads of HRA represent values, trained with components of the environment reward. Even after training, these values stay relevant, because the aggregator uses the values of all heads to select its action.

3 Model

Consider a Markov decision process (MDP), which models an agent interacting with an environment at discrete time steps t. It has a state set $\mathcal{S}$, action set $\mathcal{A}$, environment reward function $R_{env} : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$, and transition probability function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. At time step t, the agent observes state $s_t \in \mathcal{S}$ and takes action $a_t \in \mathcal{A}$. The agent observes the next state s_{t+1}, drawn from the the transition probability function $P(s_t, a_t)$, and a reward $r_t = R_{env}(s_t, a_t)$. The behaviour is defined by a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which represents the selection probabilities over actions. The goal of an agent is to find a policy that maximises the expected return, which is the discounted sum of rewards: $G_t := \sum_{i=0}^{\infty} \gamma^i\, r_{t+i}$, where the discount factor $\gamma \in [0, 1]$ controls the importance of immediate rewards versus future rewards. Each policy π has a corresponding action-value function that gives the expected return, conditioned on the state and action, when acting according to that policy:

$$Q^\pi(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a, \pi] \tag{1}$$

Model-free methods improve their policy by iteratively improving an estimate of the optimal action-value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, using sample-based updates. By acting greedily with respect to Q^* (i.e., taking the action with the highest Q^*-value in every state), the optimal policy π^* is obtained.

3.1 Hybrid Reward Architecture

Because a Q-value function is high-dimensional, it is typically approximated with a deep network with parameters θ: $Q(s, a; \theta)$. DQN estimates the optimal Q-value function by minimising the sequence of loss functions:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i^{DQN} - Q(s, a; \theta_i))^2]\,, \tag{2}$$

$$\text{with} \qquad y_i^{DQN} = r + \gamma_{a'} Q(s', a'; \theta_i^-), \tag{3}$$

where θ_i^- are the parameters of a target network that gets frozen for a number of iterations, while the online network $Q(s, a; \theta_i)$ is updated.

Let the reward function of the environment be R_{env}. We propose to regularise the target function of the deep network, by splitting the reward function into n reward functions, weighted by w_i:

$$R_{env}(s, a) = \sum_{k=1}^{n} w_k R_k(s, a)\,, \qquad \text{for all } s, a, s', \tag{4}$$

and training a separate reinforcement-learning agent on each of these reward functions. There are infinitely many different decompositions of a reward function possible, but to achieve smooth optimal value functions the decomposition should be such that each reward function is mainly affected by only a small number of state variables.

Because each agent has its own reward function, each agent i also has its own Q-value function associated with it: $Q_i(s, a; \theta)$. To derive a policy from these multiple action-value functions, an aggregator receives the action-values for the current state from the different agents and combines them into a single set of action-values (i.e., a single value for each action), using the same linear combination as used in the reward decomposition (Equation 4).

$$Q_{\text{HRA}}(s, a; \theta) = \sum_{k=1}^{n} w_k Q_k(s, a; \theta)\,. \tag{5}$$

Because the different agents can share multiple lower-level layers of a deep Q-network, the collection of agents can be viewed alternatively as a single agent with multiple *heads*, with each head producing the action-values of the current state under a different reward function. A single vector θ can be used for the parameters of this network. Each head is associated with a different reward function. This is the view that we adopt in this paper. We refer to it as a Hybrid Reward Architecture (HRA). Figure 1 illustrates the architecture. The loss function for HRA is:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\sum_{k=1}^{n}(y_{k,i} - Q_k(s, a; \theta_i))^2\right]\,, \tag{6}$$

$$\text{with} \qquad y_{k,i} = R_k(s, a, s') + \gamma \max_{a'} Q_k(s', a'; \theta_i^-)\,. \tag{7}$$

By minimising this loss function, the different heads of HRA approximate the optimal action-value functions under the different reward functions: $Q^*_1, \dots, Q^*_n$. Furthermore, we define Q^*_{HRA} as follows:

$$Q^*_{\text{HRA}}(s,a) := \sum_{k=1}^{n} w_k Q^*_k(s,a) \qquad \text{for all } s, a\,.$$

Therefore, with the update target above, the aggregator's Q-values approximate Q^*_{HRA}. In general, Q^*_{HRA} is not equal to Q^*_{env}, the optimal value function corresponding to R_{env}. If HRA's policy performs badly with respect to R_{env}, a different aggregation scheme can be used, for example, instead of the mean over heads, an aggregator action-value could be defined as the max over heads, or a voting-based aggregation scheme could be used. Alternatively, an update target based on the expected Sarsa update rule (van Seijen et al., 2009) can be used:

$$y_{k,i} = R_k(s,a,s') + \gamma \sum_{a'} \pi(s',a') Q_k(s',a';\theta_i^-)\,. \tag{8}$$

In this case, minimisation of the loss function results in the heads approximating the action-values for π under the different reward functions: $Q^\pi_1, \dots, Q^\pi_n$. We define $Q^\pi_{\text{HRA}}(s,a) := \sum_{k=1}^n w_k Q^\pi_k(s,a)$. In contrast to Q^*_{HRA}, Q^π_{HRA} is equal to Q^π_{env}, as shown in the following theorem:

Theorem 1. *With an aggregator that implements Equation (5), for any reward decomposition and stationary policy π the following holds:*

$$Q^\pi_{\text{HRA}}(s,a) = Q^\pi_{env}(s,a) \qquad \textit{for all } s, a.$$

Proof. Substituting (4) in (1) gives:

$$\begin{aligned} Q^\pi_{env}(s,a) &= \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i \sum_{k=1}^{n} w_k R_k(s_{t+i}, a_{t+i}) | s_t = s, a_t = a, \pi\right], \\ &= \sum_{k=1}^{n} w_k \cdot \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_k(s_{t+i}, a_{t+i}) | s_t = s, a_t = a, \pi\right], \\ &= \sum_{k=1}^{n} w_k Q^\pi_k(s,a) = Q^\pi_{\text{HRA}}(s,a)\,. \end{aligned} \qquad \square$$

3.2 Improving Performance further by using high-level domain knowledge.

In its basic setting, the only domain knowledge applied to HRA is in the form of the decomposed reward function. However, one of the strengths of HRA is that it can easily exploit more domain knowledge, if available. Domain knowledge can be exploited in one of the following ways:

1. **Removing irrelevant features.** Features that do not affect the received reward in any way (directly or indirectly) only add noise to the learning process and can be removed.
2. **Identifying terminal states.** Terminal states are states from which no further reward can be received; they have by definition a value of 0. Using this knowledge, HRA can refrain from approximating this value by the value network, such that the weights can be fully used to represent the non-terminal states.

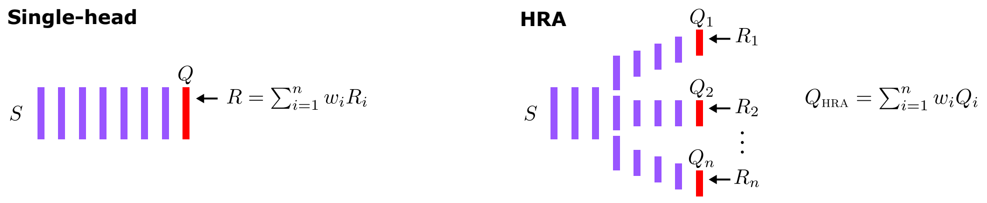

Single-head
Q
S
$R = \sum_{i=1}^n w_i R_i$
HRA
Q_1
R_1
Q_2
R_2
Q_n
R_n
S
$Q_{\text{HRA}} = \sum_{i=1}^n w_i Q_i$

Figure 1: Illustration of Hybrid Reward Architecture.

3. **Using pseudo-reward functions.** Instead of updating a head of HRA using a component of the environment reward, it can be updated using a pseudo-reward. In this scenario, a set of GVFs is trained in parallel using pseudo-rewards. Each head of HRA uses (an) appropriate GVF(s). This can often result in more efficient learning.

The first two types of domain knowledge can be used by any method, not just HRA. However, because HRA can apply this knowledge to each head individually, it can exploit domain knowledge to a much greater extent. We show this empirically in Section 4.1.

4 Experiments

4.1 Fruit Collection task

In our first domain, we consider an agent that has to collect fruits as quickly as possible in a 10×10 grid. There are 10 possible fruit locations, spread out across the grid. For each episode, a fruit is randomly placed on 5 of those 10 locations. The agent starts at a random position. The episode ends after 300 steps or when all 5 fruits have been eaten, whichever comes first.

We compare the performance of DQN with HRA using the same network. The training objective for DQN gives +1 reward for each fruit and uses $\gamma = 0.95$. For HRA, we decompose this reward function into 10 different reward functions, one per possible fruit location. The network consists of a binary input layer of length 110, encoding the agent's position and whether there is a fruit on each location. This is followed by a fully connected hidden layer of length 250. This layer is connected to 10 heads consisting of 4 linear nodes each, representing the action-values of the 4 actions under the different reward functions. Finally, the mean of all nodes across heads is computed using a final linear layer of length 4 that connects the output of corresponding nodes in each head. This layer has fixed weights with value 1/10 (i.e., it implements Equation 5). The difference between HRA and DQN is that DQN updates the network from the fourth layer using loss function (2), whereas HRA updates the network from the third layer using loss function (6).

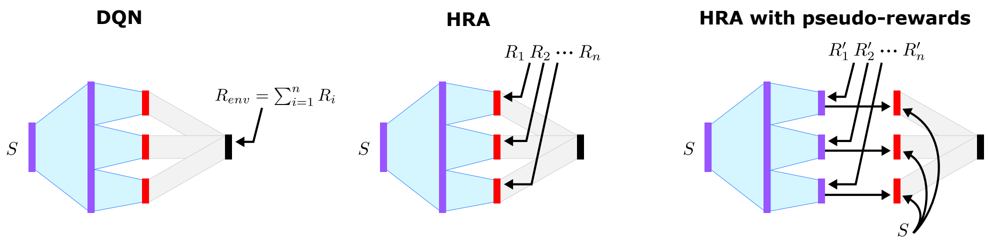


Figure 2: The different network architectures used.

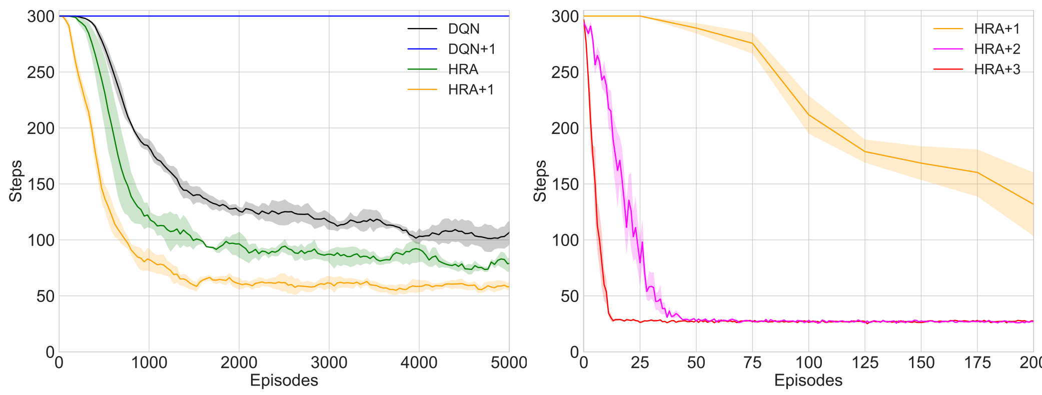


Figure 3: Results on the fruit collection domain, in which an agent has to eat 5 randomly placed fruits. An episode ends after all 5 fruits are eaten or after 300 steps, whichever comes first.

Besides the full network, we test using different levels of domain knowledge, as outlined in Section 3.2: 1) removing the irrelevant features for each head (providing only the position of the agent + the corresponding fruit feature); 2) the above plus identifying terminal states; 3) the above plus using pseudo rewards for learning GVFs to go to each of the 10 locations (instead of learning a value function associated to the fruit at each location). The advantage is that these GVFs can be trained even if there is no fruit (anymore) at a location. The head for a particular fruit copies the Q-values of the GVF corresponding to the fruit's location, or outputs 0s if there is currently no fruit at the location. We refer to these as *HRA+1*, *HRA+2* and *HRA+3*, respectively. For DQN, we also tested a version that was applied to the same network as *HRA+1*; we refer to this version as *DQN+1*.

We performed experiments with update targets that estimate an optimal policy (Equation 7) and update targets that evaluate a uniformly random policy (using Equation 8). Acting greedily with respect to the Q-values of a uniformly random policy evaluated under reward function R_{env}, can in some domains yields a very good performance with respect to R_{env}. We optimise the step-size for each method separately.

The results are shown in Figure 3 for the best settings. Interestingly, for DQN estimating the optimal policy performed better, while for HRA estimating the random policy performed better. Overall, HRA shows a clear performance boost over DQN, even though the network is identical. Furthermore, adding different forms of domain knowledge causes further large improvements. Whereas using a network structure enhanced by domain knowledge causes large improvements for HRA, using that same network for DQN results in not learning anything at all. The big boost in performance that occurs when the the terminal states are identified is due to the representation becoming a one-hot vector. Hence, we removed the hidden layer and directly fed this one-hot vector into the different heads. Because the heads are linear, this representation reduces to an exact, tabular representation. For the tabular representation, we used the same step-size as the optimal step-size for the deep network version.

4.2 ATARI game: Ms. Pac-Man

Our second domain is the Atari 2600 game Ms. Pac-Man (see Figure 4). Points are obtained by eating pellets, while avoiding ghosts (contact with one causes Ms. Pac-Man to lose a life). Eating one of the special power pellets turns the ghosts blue for a small duration, allowing them to be eaten for extra points. Bonus fruits can be eaten for further points, twice per level. When all pellets have been eaten, a new level is started. There are a total of 4 different maps and 7 different fruit types, each with a different point value. We provide full details on the domain in the supplementary material.

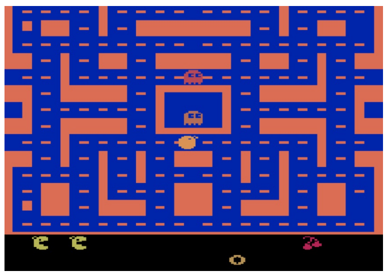


Figure 4: The game Ms. Pac-Man.

Baselines. While our version of Ms. Pac-Man is the same as used in literature, we use different preprocessing. Hence, to test the effect of our preprocessing, we implement the A3C method (Mnih et al., 2016) and run it with our preprocessing. We refer to the version with our preprocessing as 'A3C(channels)', the version with the standard preprocessing 'A3C(pixels)', and A3C's score reported in literature 'A3C(reported)'.

Preprocessing. Each frame from ALE is 210×160 pixels. We cut the bottom part and the top part of the screen to end up with 160×160 pixels. From this, we extract the position of the different objects and create for each object a separate input channel, encoding its location with an accuracy of 4 pixels. This results in 11 binary channels of size 40×40. Specifically, there is a channel for: Ms. Pac-Man, each of the four ghosts, each of the four blue ghosts (these are treated as different objects), the fruit plus one channel with all the pellets (including power pellets). For A3C, we combine the 4 channels of the ghosts into a single channel, to allow it to generalise better across ghosts. We do the same with the 4 channels of the blue ghosts. Instead of giving the history of the last 4 frames as done in literature, we give the orientation of Ms. Pac-Man as a 1-hot vector of length 4 (representing the 4 compass directions).

HRA architecture. We use a HRA architecture with one head for each pellet, one head for each ghost, one head for each blue ghost and one head for the fruit. Similar to the fruit collection task, HRA uses GVFs that learn the Q-values for getting to a particular location in the map (it learns separate GVFs for each of the four maps). The agent learns part of its representation during training: it starts of with 0 GVFs and 0 heads for the pellets. By wandering around the maze, it discovers new map locations it can reach, resulting in new GVFs being created. Whenever the agent finds a pellet at a new location it creates a new head corresponding to the pellet.

The Q-values of the head of an object (pellet/fruit/ghost/blue ghost) are simply the Q-values of the GVF that corresponds with the object's location (i.e., moving objects use a different GVF each time). If an object is not on the screen, all its Q-values are 0. Each head i is assigned a weight w_i, which can be positive or negative. For the head of a pellet/blue ghost/fruit the weight corresponds to the reward received when the object is eaten. For the regular ghosts (contact with one causes Ms. Pac-Man to lose a life), the weights are set to -1,000.

We test two aggregator types. The first one is a linear one that sums the Q-values of all heads multiplied with the weights (see Equation 5). For the second one, we take the weighted sum of all the heads that produce points, and normalise the resulting Q-values; then, we add the weighted Q-values of the heads of the regular ghosts.

For exploration, we test two complementary types of exploration. Each type adds an extra exploration head to the architecture. The first type, which we call *diversification*, produces random Q-values, drawn from a uniform distribution over [0, 20]. We find that it is only necessary during the first 50 steps, to ensure starting each episode randomly. The second type, which we call *count-based*, adds a bonus for state-action pairs that have not been explored a lot. It is inspired by upper confidence bounds (Auer et al., 2002). Full details can be found in the supplementary material.

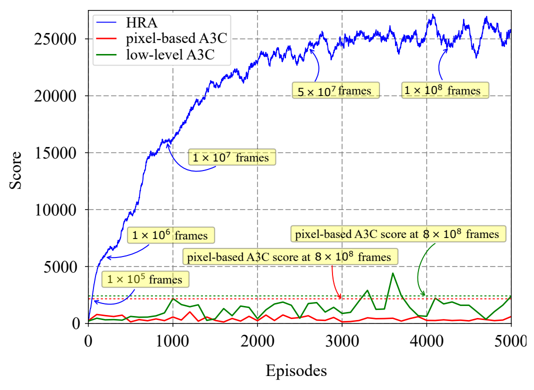
For our final experiment, we implement a special head inspired by *executive-memory* literature (Fuster, 2003; Gluck et al., 2013). When a human game player reaches the maximum of his cognitive and physical ability, he starts to look for favourable situations or even glitches and memorises them. This cognitive process is indeed memorising a sequence of actions (also called habit), and is not necessarily optimal. Our executive-memory head records every sequence of actions that led to pass a level without any kill. Then, when facing the same level, the head gives a very high value to the recorded action, in order to force the aggregator's selection. Note that our simplified version of executive memory does not generalise.

Evaluation metrics. There are two different evaluation methods used across literature which result in very different scores. Because ALE is ultimately a deterministic environment (it implements pseudo-randomness using a random number generator that always starts with the same seed), both evaluation metrics aim to create randomness in the evaluation in order to rate methods with more generalising behaviour higher. The first metric introduces a mild form of randomness by taking a random number of no-op actions before control is handed over to the learning algorithm. In the case of Ms. Pac-Man, however, the game starts with a certain inactive period that exceeds the maximum number of no-op steps, resulting in the game having a fixed start after all. The second metric selects random starting points along a human trajectory and results in much stronger randomness, and does result in the intended random start evaluation. We refer to these metrics as 'fixed start' and 'random start'.

Table 1: Final scores.

method	fixed start	random start
best reported	6,673	2,251
human	15,693	15,375
A3C (reported)	—	654
A3C (pixels)	2,168	626
A3C (channels)	2,423	589
HRA	**25,304**	**23,770**

Results. Figure 5 shows the training curves; Table 1 the final score after training. The best reported fixed start score comes from STRAW (Vezhnevets et al., 2016); the best reported random start score comes from the Dueling network architecture (Wang et al., 2016). The human fixed start score comes from Mnih et al. (2015); the human random start score comes from Nair et al. (2015). We train A3C for 800 million frames. Because HRA learns fast, we train it only for 5,000 episodes, corresponding with about 150 million frames (note that better policies result in more frames per episode). The score shown for HRA uses normalisation and both exploration types. We try different combinations (with/without normalisation and with/without each type of exploration) for HRA. The score shown for HRA uses the best combination: with normalisation and



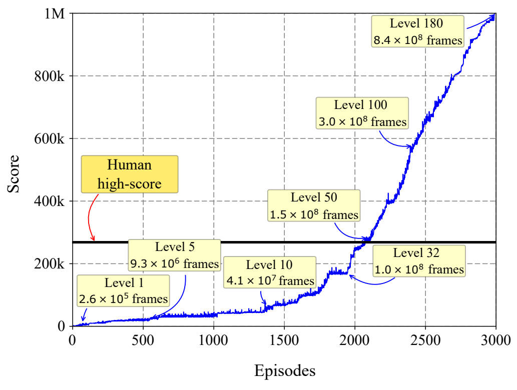


Figure 5: Training smoothed over 100 episodes. Figure 6: Training with trajectory memorisation.

with both exploration types. All of the combinations achieved over 10,000 points in training, except the combination with no exploration at all, which performed very poorly. With the best combination, HRA not only outperforms the state-of-the-art on both metrics, it also significantly outperforms the human score, convincingly demonstrating the strength of HRA.

Comparing A3C(pixels) and A3C(channels) in Table 1 reveals a surprising result: while we use advanced preprocessing by separating the screen image into relevant object channels, this did not significantly change the performance of A3C.

In our final experiment, we test how well HRA does if it exploits the weakness of the fixed-start evaluation metric by using a simplified version of executive memory. Using this version, we not only surpass the human high-score,[1], we achieve the maximum possible score of 999,990 points in less than 3,000 episodes. The curve is slow in the first stages because the model has to be trained, but even though the further levels get more and more difficult, the level passing speeds up by taking advantage of already knowing the maps. Obtaining more points is impossible, not because the game ends, but because the score overflows to 0 when reaching a million points. [2]

5 Discussion

One of the strengths of HRA is that it can exploit domain knowledge to a much greater extent than single-head methods. This is clearly shown by the fruit collection task: while removing irrelevant features causes a large improvement in performance for HRA, for DQN no effective learning occurred when provided with the same network architecture. Furthermore, separating the pixel image into multiple binary channels only makes a small improvement in the performance of A3C over learning directly from pixels. This demonstrates that the reason that modern deep RL struggle with Ms. Pac-Man is not related to learning from pixels; the underlying issue is that the optimal value function for Ms. Pac-Man cannot easily be mapped to a low-dimensional representation.

HRA solves Ms. Pac-Man by learning close to 1,800 general value functions. This results in an exponential breakdown of the problem size: whereas the input state-space corresponding with the binary channels is in the order of 10^{77}, each GVF has a state-space in the order of 10^3 states, small enough to be represented without any function approximation. While we could have used a deep network for representing each GVF, using a deep network for such small problems hurts more than it helps, as evidenced by the experiments on the fruit collection domain.

We argue that many real-world tasks allow for reward decomposition. Even if the reward function can only be decomposed in two or three components, this can already help a lot, due to the exponential decrease of the problem size that decomposition might cause.

[1]highscore.com reports *oyamafamily* as the world record holder with 266,330 points.

[2]For a video of HRA's final trajectory reaching this point, see: https://youtu.be/VeXNw0Owf0Y

References

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Bacon, P., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the Thirthy-first AAAI Conference On Artificial Intelligence (AAAI)*, 2017.

Barto, A. G. and Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Fuster, J. M. *Cortex and mind: Unifying cognition.* Oxford university press, 2003.

Gluck, M. A., Mercado, E., and Myers, C. E. *Learning and memory: From brain to behavior*. Palgrave Macmillan, 2013.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., and Tenenbaum, J. B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems 29*, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., Kumaran, H. King D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1928–1937, 2016.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, A. De, Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., and Silver, D. Massively parallel methods for deep reinforcement learning. In *In Deep Learning Workshop, ICML*, 2015.

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 2013.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Schmidhuber, J. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). In *IEEE Transactions on Autonomous Mental Development 2.3*, pp. 230–247, 2010.

Stout, A., Konidaris, G., and Barto, A. G. Intrinsically motivated reinforcement learning: A promising framework for developmental robotics. In *The AAAI Spring Symposium on Developmental Robotics*, 2005.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, Doina. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

Sutton, R.S., Precup, D., and Singh, S.P. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

Szepesvári, C. *Algorithms for reinforcement learning*. Morgan and Claypool, 2009.

van Hasselt, H., Guez, A., Hessel, M., Mnih, V., and Silver, D. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems 29*, 2016a.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016b.

van Seijen, H., van Hasselt, H., Whiteson, S., and Wiering, M. A theoretical and empirical analysis of expected sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 177–184, 2009.

Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., and Kavukcuoglu, K. Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems 29*, 2016.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

A Ms. Pac-Man - experimental details

A.1 General information about Atari 2600 Ms. Pac-Man

The second domain is the Atari 2600 game Ms. Pac-Man. Points are obtained by eating pellets, while avoiding ghosts (contact with one causes Ms. Pac-Man to lose a life). Eating one of the special power pellets turns the ghost blue for a small duration, allowing them to be eaten for extra points. Bonus fruits can be eaten for further increasing points, twice per level. When all pellets have been eaten, a new level is started. There are a total of 4 different maps (see Figure 7 and Table 2) and 7 different fruit types, each with a different point value (see Table 3).

Ms. Pac-Man is considered one of the hard games from the ALE benchmark set. When comparing performance, it is important to realise that there are two different evaluation methods for ALE games used across literature which result in hugely different scores (see Table 4). Because ALE is ultimately a deterministic environment (it implements pseudo-randomness using a random number generator that always starts with the same seed), both evaluation metrics aim to create randomness in the evaluation in order to discourage methods from exploiting this deterministic property and rate methods with more generalising behaviour higher. The first metric introduces a mild form of randomness by taking a random number of no-op actions before control is handed over to the learning algorithm. In the case of Ms. Pac-Man, however, the game starts with a certain inactive period that exceeds the maximum number of random no-op steps, resulting in the game having a fixed start after all. The second metric selects random starting points along a human trajectory and results in much stronger randomness, and does result in the intended random start evaluation.

The best method with fixed start evaluation is STRAW with 6,673 points (Vezhnevets et al., 2016); the best with random start evaluation is the dueling network architecture with 2,251 points (Wang et al., 2016). The human baseline, as reported by Mnih et al. (2015), is 15,693 points. The highest reported score by a human is 266,330.[3] All reported scores are shown in Table 4.

[3] See highscore.com: 'Ms. Pac-Man (Atari 2600 Emulated)'.

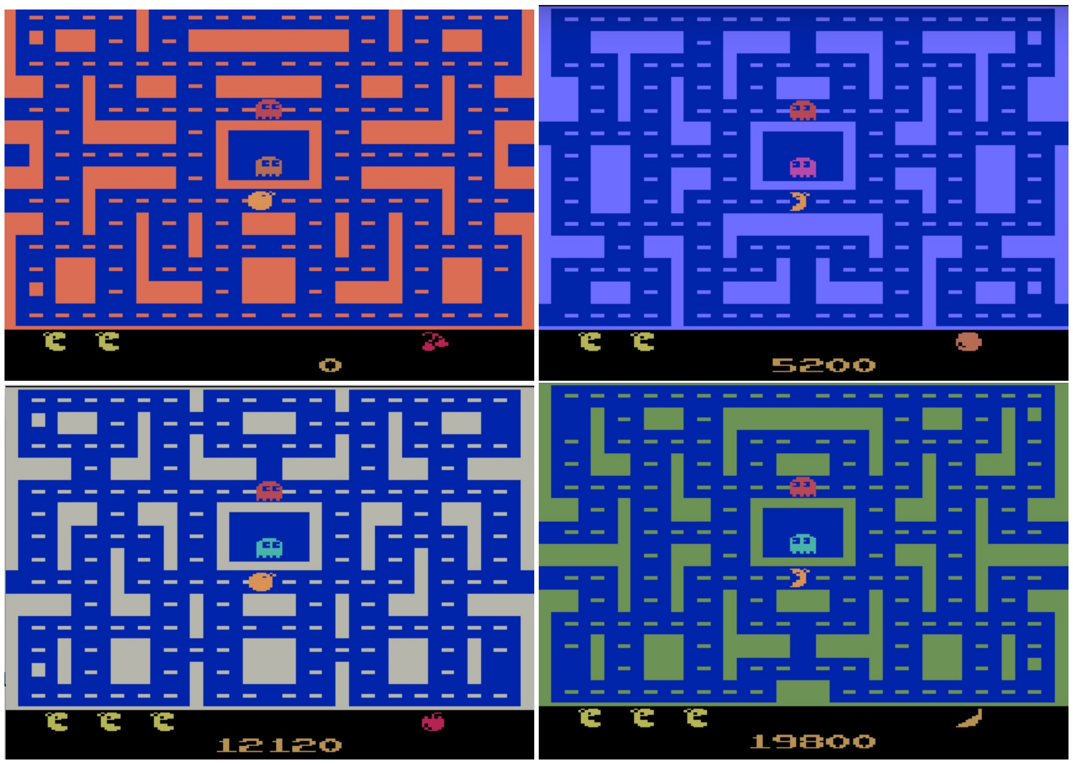


Figure 7: The four different maps of Ms. Pac-Man.

Table 2: Map type and fruit type per level.

level	map	fruit
1	red	cherry
2	red	strawberry
3	blue	orange
4	blue	pretzel
5	white	apple
6	white	pear
7	green	banana
8	green	$\langle random \rangle$
9	white	$\langle random \rangle$
10	green	$\langle random \rangle$
11	white	$\langle random \rangle$
12	green	$\langle random \rangle$
⋮	⋮	⋮

Table 3: Points breakdown of edible objects.

object	points
pellet	10
power pellet	50
1^{st} blue ghost	200
2^{nd} blue ghost	400
3^{th} blue ghost	800
4^{th} blue ghost	1,600
cherry	100
strawberry	200
orange	500
pretzel	700
apple	1,000
pear	2,000
banana	5,000

Table 4: Reported scores on Ms. Pac-Man for fixed start evaluation (called 'random no-ops' in literature) and random start evaluation ('human starts' in literature).

algorithm	fixed start	source	rand start	source
Human	15,693	Mnih et al. (2015)	15,375	Nair et al. (2015)
Random	308	Mnih et al. (2015)	198	Nair et al. (2015)
DQN	2,311	Mnih et al. (2015)	764	Nair et al. (2015)
DDQN	3,210	van Hasselt et al. (2016b)	1,241	van Hasselt et al. (2016b)
Prio. Exp. Rep	6,519	Schaul et al. (2016)	1,825	Schaul et al. (2016)
Dueling	6,284	Wang et al. (2016)	2,251	Wang et al. (2016)
A3C	—	—	654	Mnih et al. (2016)
Gorila	3,234	Nair et al. (2015)	1,263	Nair et al. (2015)
Pop-Art	4,964	van Hasselt et al. (2016a)	—	—
STRAW	6,673	Vezhnevets et al. (2016)	—	—

A.2 HRA architecture

GVF heads. Ms. Pac-Man state is defined as its position on the map and its direction (heading North, East, South or West). Depending on the map, there are about 400 positions and 950 states (not all directions are possible for each position). A GVF is created online for each visited Ms. Pac-Man position. Each GVF is then in charge of determining the value of the random policy of Ms. Pac-Man state for getting the pseudo-reward placed on the GVF's associated position. The GVFs are trained online with off-policy 1-step bootstrapping with $\alpha = 1$ and $\gamma = 0.99$. Thus, the full tabular representation of the GVF grid contains $nb_{maps} \times nb_{positions} \times nb_{states} \times nb_{actions} \approx 14M$ entries.

Aggregator. For each object of the game: pellets, ghosts and fruits, the GVF corresponding to its position is activated with a multiplier depending on the object type. Edible objects' multipliers are consistent with the number of points they grant: pellets' multiplier is 10, power pellets' 50, fruits' 200, and blue and blue (edible) ghosts' 1,000. Initial tests showed that a ghosts' multiplier of -1,000 is a fair balance between gaining points and not being killed. Finally, the aggregator sums up all the activated and multiplied GVFs to compute a global score for each of the nine actions and chooses the action that maximises it.

Diversification head. The blue curve on Figure 8 reveals that this naïve setting performs badly because it tends to deterministically repeat a bad trajectory like a robot hitting a wall continuously. In order to avoid this pitfall, we need to add an exploratory mechanism. An ϵ-greedy exploration is not suitable for this problem since it might unnecessarily put Ms. Pac-Man in danger. A Boltzmann distributed exploration is more suitable because it favours exploring the safe actions. It would be

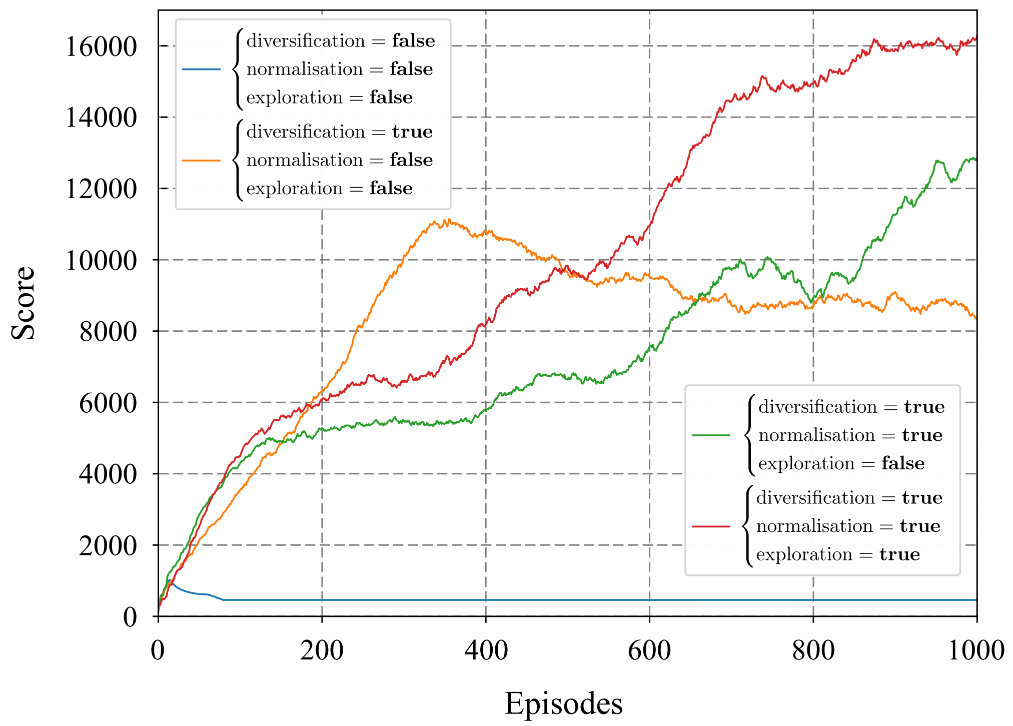


Figure 8: Training curves for incrementally head additions to the HRA architecture.

possible to apply this on top of the aggregator, but we chose here to instead add a diversification head that generates for each action a random value. This random value is drawn according to a uniform distribution in [0,20]. We found that it is only necessary during the 50 first steps, to ensure each episode starts randomly.

Score heads normalisation. The orange curve on Figure 8 shows that the diversification head solves the determinism issue. The so-built architecture progresses fast, up to 10,000 points, but then starts regressing. The analysis of the generated trajectories reveals that the system has difficulty in finishing levels: indeed, when only a few pellets remain on the screen, the aggregator gets overwhelmed by the ghost avoider values. The regression in score is explained by the fact that the more the system learns the more it gets easily scared by the ghosts, and therefore the more difficult it is for it to finish the levels. We solve this issue by modifying the additive aggregator with a normalisation over the score heads between 0 and 1. To fit this new value scale, the ghost multiplier is modified to -10.

Targeted exploration head. The green curve on Figure 8 grows over time as expected. It might be surprising at first look that the orange curve grows faster, but it is because the episodes without normalisation tend to last much longer, which allows more GVF updates per episode. In order to speed up the learning, we decide to use a targeted exploration head (teh), that is motivated by trying out the less explored state-action couples. The value of this agent is computed as follows:

$$value_{teh}(s,a) = \kappa \sqrt{\frac{\sqrt[4]{N}}{n(s,a)}}, \tag{9}$$

where N is the number of actions taken until now and $n(s,a)$ the number of times action a has been performed in state s. This formula is inspired from upper confidence bounds (Auer et al., 2002), but replacing the stochastically motivated logarithmic function by a less drastic one is more compliant with our need for bootstrapping propagation. Note that this targeted exploration head is not a replacement for the diversification head. They are complementary: diversification for making each trajectory unique, and targeted exploration for prioritised exploration. The red curve on Figure 8 reveals that the new targeted exploration head helps exploration and makes the learning faster. This setting constitutes the HRA architecture that is used in every experiment.

Executive memory head. When human game players reach the maximum of their cognitive and physical ability, they start to look for favourable situations or even glitches and memorise them. This cognitive process is referred as executive memory in cognitive science literature (Fuster, 2003; Gluck et al., 2013). The executive memory head records every sequence of actions that led to passing a level without losing any life. Then, when facing the same level, the head gives a high value to the recorded action, in order to force the aggregator's selection. *Nota bene:* since it does not allow generalisation, this head is only employed for the level-passing experiment.

A.3 A3C baselines

Since we use low-level features for the HRA architecture, we implement A3C and evaluate it both on the pixel-based environment and on the low-level features. The implementation is performed in a way to reproduce the results of Mnih et al. (2015).

They are both trained similarly as in Mnih et al. (2016) on 8×10^8 frames, with $\gamma = 0.99$, entropy regularisation of 0.01, n-step return of 5, 16 threads, gradient clipping of 40, and α is set to take the maximum performance over the following values: $[0.0001, 0.00025, 0.0005, 0.00075, 0.001]$. The pixel-based environment is a reproduction of the preprocessing and the network, except we only use a history of 2, because our steps are twice as long.

With the low features, five channels of a 40×40 map are used embedding the positions of Ms. Pac-Man, the pellets, the ghosts, the blue ghosts, and the special fruit. The input space is therefore $5 \times 40 \times 40$ plus the direction appended after convolutions: 2 of them with 16 (respectively 32) filters of size 6×6 (respectively 4×4) and subsampling of 2×2 and ReLU activation (for both). Then, the network uses a hidden layer of 256 fully connected units with ReLU activation. Finally, the policy

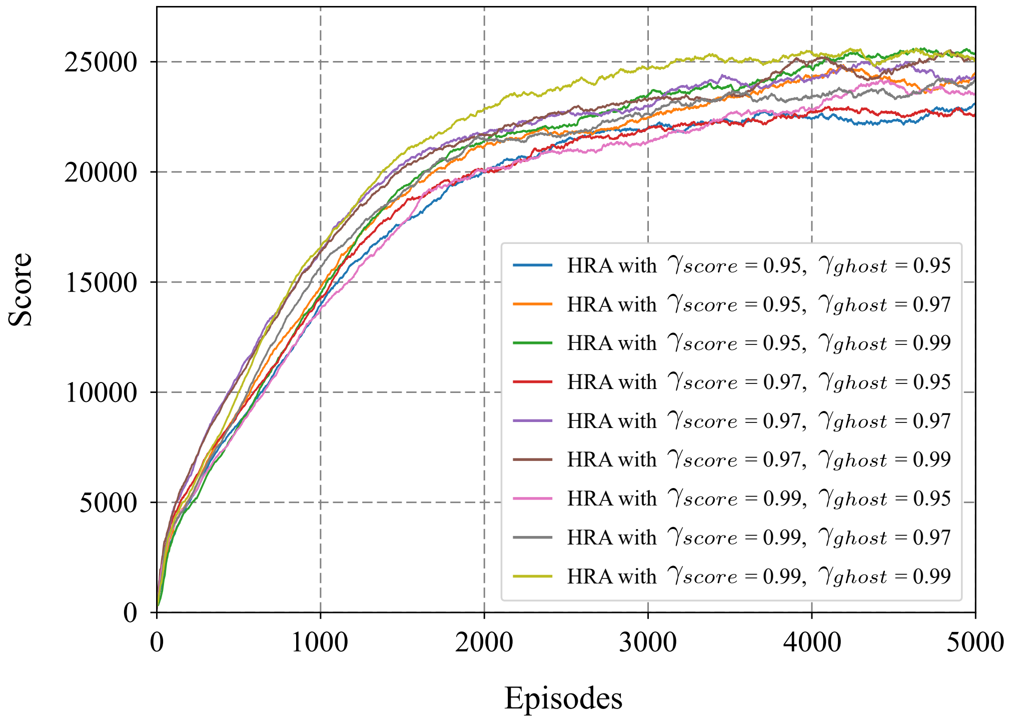


Figure 9: Gridsearch on γ values without executive memory smoothed over 500 episodes.

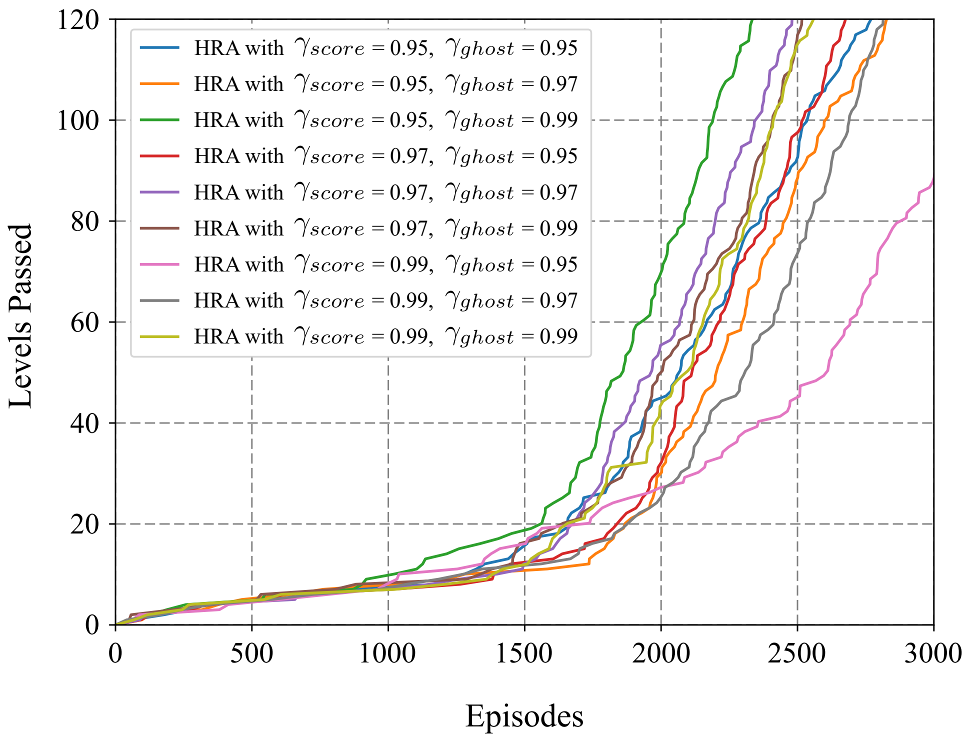


Figure 10: Gridsearch on γ values with executive memory.

head has $nb_{actions} = 9$ fully connected units with softmax activation, and the value head has 1 unit with a linear activation. All weights are uniformly initialised He et al. (2015).

A.4 Results

Training curves. Most of the results are already presented in the main document. For more completeness, we present here the results of the gridsearch over γ values for both with and without the executive memory. Values $[0.95, 0.97, 0.99]$ have been tried independently for γ_{score} and γ_{ghosts}.

Figure 9 compares the training curves without executive memory. We can notice the following:

- all γ values turn out to yield very good results,
- those good results generalise over random human starts,
- high γ values for the ghosts tend to be better,
- and the γ value for the score is less impactful.

Figure 10 compares the training curves with executive memory. We can notice the following:

- the comments on Figure 9 are still holding,
- and it looks like that there is a bit more randomness in the level passing efficiency.