Hybrid Reward Architecture for Reinforcement Learning

Harm van Seijen, Joshua Romoff, Tavian Barnes, Mehdi Fatemi, Romain Laroche, Jeffrey Tsang

Travis LaCroix
CPSC 295
March 12, 2018

- It is challenging to *generalise* RL methods to large real-world problems (because state space is often massive, so learning a good policy efficiently is difficult).

- It is challenging to *generalise* RL methods to large real-world problems (because state space is often massive, so learning a good policy efficiently is difficult).
- Mnih et al. (2015) combined standard RL techniques with deep neural networks, they achieved above-human performance on a large number of Atari 2600 games, by learning a policy from pixels.

- It is challenging to *generalise* RL methods to large real-world problems (because state space is often massive, so learning a good policy efficiently is difficult).
- Mnih et al. (2015) combined standard RL techniques with deep neural networks, they achieved above-human performance on a large number of Atari 2600 games, by learning a policy from pixels.
- They achieve generalisation of their *Deep Q-Networks* (DQNs) by approximating the optimal-value function.

- It is challenging to *generalise* RL methods to large real-world problems (because state space is often massive, so learning a good policy efficiently is difficult).
- Mnih et al. (2015) combined standard RL techniques with deep neural networks, they achieved above-human performance on a large number of Atari 2600 games, by learning a policy from pixels.
- They achieve generalisation of their *Deep Q-Networks* (DQNs) by approximating 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.

- The authors propose to replace the optimal value function as target for training with an alternative value function that is easier to learn, but still yields a reasonable—but generally not optimal—policy, when acting greedily with respect to it.

- The authors propose to replace the optimal value function as target for training with an alternative value function that is easier to learn, but still yields a reasonable—but generally not optimal—policy, when acting greedily with respect to it.
- To do this, they decompose the reward function of the environment into n different reward functions.

- The authors propose to replace the optimal value function as target for training with an alternative value function that is easier to learn, but still yields a reasonable—but generally not optimal—policy, when acting greedily with respect to it.
- To do this, they decompose the reward function of the environment into n different reward functions.
- Each reward function is assigned a separate RL agent.

- The authors propose to replace the optimal value function as target for training with an alternative value function that is easier to learn, but still yields a reasonable—but generally not optimal—policy, when acting greedily with respect to it.
- To do this, they decompose the reward function of the environment into n different reward functions.
- Each reward function is assigned a separate RL agent.
- They point out that all these agents can learn in parallel on the same sample sequence by using off-policy learning (Similar to Horde architecture of Sutton, et al. (2011)).

Aside on Horde Architecture:

- A Horde consists of a large number of independent reinforcement learning sub-agents—these are referred to as "demons".

Aside on Horde Architecture:

- A Horde consists of a large number of independent reinforcement learning sub-agents—these are referred to as "demons".
- Each demon is responsible for answering a single predictive question about the world (in the form of a value function), thereby contributing to the system's overall knowledge.

Aside on Horde Architecture:

- A Horde consists of a large number of independent reinforcement learning sub-agents—these are referred to as "demons".
- Each demon is responsible for answering a single predictive question about the world (in the form of a value function), thereby contributing to the system's overall knowledge.
- Each demon has its own policy, reward function, termination function, and terminal-reward function unrelated to those of the base problem.

Aside on Horde Architecture:

- A Horde consists of a large number of independent reinforcement learning sub-agents—these are referred to as "demons".
- Each demon is responsible for answering a single predictive question about the world (in the form of a value function), thereby contributing to the system's overall knowledge.
- Each demon has its own policy, reward function, termination function, and terminal-reward function unrelated to those of the base problem.
- Learning proceeds in parallel by all demons simultaneously so as to extract the maximal training information from whatever actions are taken by the system as a whole.

Main Idea:
Each agent gives its action-values of the current state to an aggregator, which combines them into a single value for each action. The current action is selected based on these aggregated values.

Markov Decision Process given by

$$\langle \mathcal{S}, \mathcal{A}, P, R_{env}, \gamma \rangle$$

modeling an agent interacting with an environment at discrete times steps, t, where:

$\mathcal{S}$ is a set of states,

$\mathcal{A}$ is a set of actions,

$R_{env} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is an environment reward function,

$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition probability function.

$\gamma \in [0, 1]$ is a discount factor.

Markov Decision Process given by

$$\langle \mathcal{S}, \mathcal{A}, P, R_{env}, \gamma \rangle$$

modeling an agent interacting with an environment at discrete times steps, t, where:

At time step t, the agent observes $s_t \in \mathcal{S}$.

Markov Decision Process given by

$$\langle \mathcal{S}, \mathcal{A}, P, R_{env}, \gamma \rangle$$

modeling an agent interacting with an environment at discrete times steps, t, where:

At time step t, the agent observes $s_t \in \mathcal{S}$.

(Also) at time step t, the agent takes action $a_t \in \mathcal{A}$.

Markov Decision Process given by

$$\langle \mathcal{S}, \mathcal{A}, P, R_{env}, \gamma \rangle$$

modeling an agent interacting with an environment at discrete times steps, t, where:

At time step t, the agent observes $s_t \in \mathcal{S}$.

(Also) at time step t, the agent takes action $a_t \in \mathcal{A}$.

The agent observes the next state, s_{t+1} drawn from $P(s_t, a_t, \cdot)$.

Markov Decision Process given by

$$\langle \mathcal{S}, \mathcal{A}, P, R_{env}, \gamma \rangle$$

modeling an agent interacting with an environment at discrete times steps, t, where:

At time step t, the agent observes $s_t \in \mathcal{S}$.

(Also) at time step t, the agent takes action $a_t \in \mathcal{A}$.

The agent observes the next state, s_{t+1} drawn from $P(s_t, a_t, \cdot)$.

The agent receives reward $r_t = R_{env}(s_t, a_t, s_{t+1})$.

Markov Decision Process given by

$$\langle \mathcal{S}, \mathcal{A}, P, R_{env}, \gamma \rangle$$

modeling an agent interacting with an environment at discrete times steps, t, where:

At time step t, the agent observes $s_t \in \mathcal{S}$.

(Also) at time step t, the agent takes action $a_t \in \mathcal{A}$.

The agent observes the next state, s_{t+1} drawn from $P(s_t, a_t, \cdot)$.

The agent receives reward $r_t = R_{env}(s_t, a_t, s_{t+1})$.

The agent's behaviour is defined by the policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, representing selection probabilities over actions.

Goal:

Find a policy π^* that maximises the expectation of the return, given by the discounted sum of the individual rewards:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Goal:

Find a policy π^* that maximises the expectation of the return, given by the discounted sum of the individual rewards:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Each policy, π, has an action-value function, giving the expected return conditioned on the state and the action:

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

Goal:

Find a policy π^* that maximises the expectation of the return, given by the discounted sum of the individual rewards:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Each policy, π, has an action-value function, giving the expected return conditioned on the state and the action:

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

The optimal policy, π^* can be found by iteratively improving an estimate of the optimal action-value function,

$$Q^*(s, a) := \max_\pi Q^\pi(s, a).$$

The Q-value function can be approximated using a function approximator with a weight vector θ.

The Q-value function can be approximated using a function approximator with a weight vector θ.
We estimate $Q(s, a)$ using the function $Q(s, a; \theta)$.

The Q-value function can be approximated using a function approximator with a weight vector θ.
We estimate $Q(s, a)$ using the function $Q(s, a; \theta)$.
DQN uses deep neural networks as a function approximator and improves the estimate of Q^* by minimising the sequence of loss functions given by

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s, a; \theta_i)\right)^2\right] \tag{1}$$

The Q-value function can be approximated using a function approximator with a weight vector θ.
We estimate $Q(s, a)$ using the function $Q(s, a; \theta)$.
DQN uses deep neural networks as a function approximator and improves the estimate of Q^* by minimising the sequence of loss functions given by

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s, a; \theta_i)\right)^2\right] \tag{1}$$

with

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \tag{2}$$

Definition:

The Q-value function that minimises the loss function is called the Training Target.

Definition:

The Q-value function that minimises the loss function is called the Training Target.

Definition

A training target is Consistent if it results in a policy that is optimal under the reward function of the environment when acting greedily with respect to the training target.

Definition:

The Q-value function that minimises the loss function is called the Training Target.

Definition

A training target is Consistent if it results in a policy that is optimal under the reward function of the environment when acting greedily with respect to the training target.

Definition

When a training target results in a good, but suboptimal, policy when acting greedily with respect to the training target, it is called Semi-Consistent.

Goal:
Find an alternative training target for domains where the default training target, Q^*_{env}, is hard to learn.

Goal:
Find an alternative training target for domains where the default training target, Q^*_{env}, is hard to learn.

Proposal:
Decompose the reward function, R_{env}, into n reward functions as follows:

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

and then train a separate reinforcement learning agent on each of these reward functions.

Hybrid Reward Architecture:

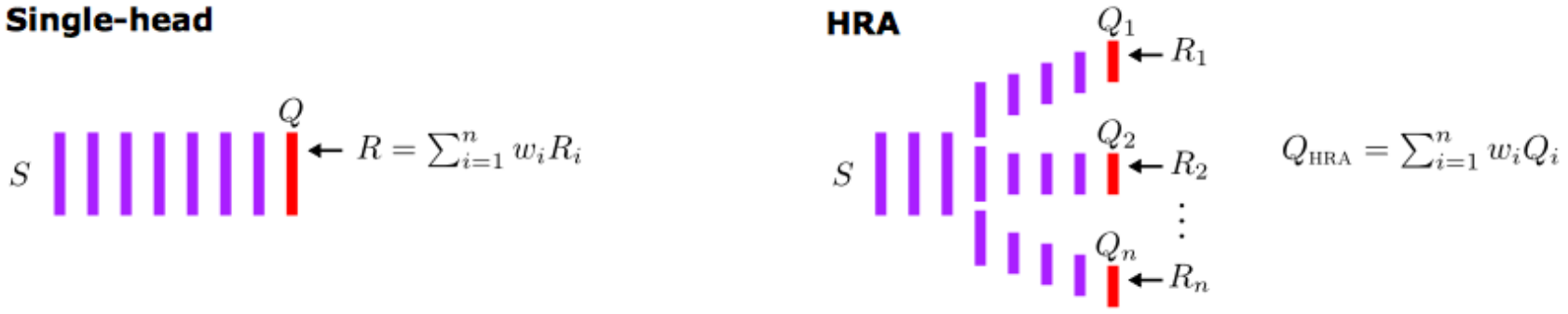


Figure 1: Illustration of Hybrid Reward Architecture.

Note:
Each agent, k, has its own reward function R_k, and therefore has its own Q-value function, Q_k.

Action selection for HRA is based on the sum of the agents' Q-value functions, denoted by Q_{HRA}:

$$Q_{\text{HRA}}(s, a; \theta) := \sum_{k=1}^{n} Q_k(s, a; \theta), \qquad \text{For all } s, a.$$

The sequence of loss functions associated with HRA is given by

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\sum_{k=1}^{n}\left(y_{k,i} - Q_k(s,a;\theta_i)\right)^2\right] \tag{3}$$

with

$$y_{k,i} = R_k(s,a,s') + \gamma \max_{a'} Q_k(s',a';\theta_{i-1}) \tag{4}$$

By minimising these loss functions, the individual agents of HRA approximate the action-value functions under the reward functions: $Q_1^*, \ldots, Q_n^*$.

By minimising these loss functions, the individual agents of HRA approximate the action-value functions under the reward functions: $Q_1^*, \ldots, Q_n^*$.

Furthermore, Q_{HRA} approximates Q_{HRA}^*:

$$Q_{\text{HRA}}^*(s,a) := \sum_{k=1}^{n} Q_k^*(s,a) \qquad \text{For all } s, a.$$

Alternative Training Target:
Evaluate the uniformly random policy, v, under each component reward function:

$$Q^v_{\text{HRA}}(s,a) := \sum_{k=1}^{n} Q^v_k(s,a).$$

Alternative Training Target:
Evaluate the uniformly random policy, v, under each component reward function:

$$Q^v_{\text{HRA}}(s,a) := \sum_{k=1}^{n} Q^v_k(s,a).$$

This can be learned using the SARSA update rule.

$$
\begin{aligned}
Q^v_{env}(s,a) &= \mathbb{E}\left[\sum_{i=0}^{\infty}\gamma^i R_{env}(s_{t+i}, a_{t+i}, s_{t+1+i}) \middle| s_t = s, a_t = a, v\right] \\
&= \mathbb{E}\left[\sum_{i=0}^{\infty}\gamma^i \sum_{k=1}^{n} R_k(s_{t+i}, a_{t+i}, s_{t+1+i}) \middle| s_t = s, a_t = a, v\right] \\
&= \sum_{k=1}^{n}\mathbb{E}\left[\sum_{i=0}^{\infty}\gamma^i R_k(s_{t+i}, a_{t+i}, s_{t+1+i}) \middle| s_t = s, a_t = a, v\right] \\
&= \sum_{k=1}^{n} Q^v_k(s,a) := Q^v_{\text{HRA}}(s,a)
\end{aligned}
$$

HRA can exploit more domain knowledge, if available, in one of the following ways:

HRA can exploit more domain knowledge, if available, 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.

HRA can exploit more domain knowledge, if available, 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.

HRA can exploit more domain knowledge, if available, 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.
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 *General Value Functions* is trained in parallel using pseudo-rewards.

Fruit Collection Task

Consider an agent that has to collect fruits as quickly as possible in a 10×10 grid.

Fruit Collection Task

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.

Fruit Collection Task

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.

Fruit Collection Task

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 reward is $+1$ if a fruit gets eaten and 0 otherwise.

Fruit Collection Task

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 reward is $+1$ if a fruit gets eaten and 0 otherwise.
- An episode ends after all 5 fruits have been eaten or after 300 steps, whichever comes first.

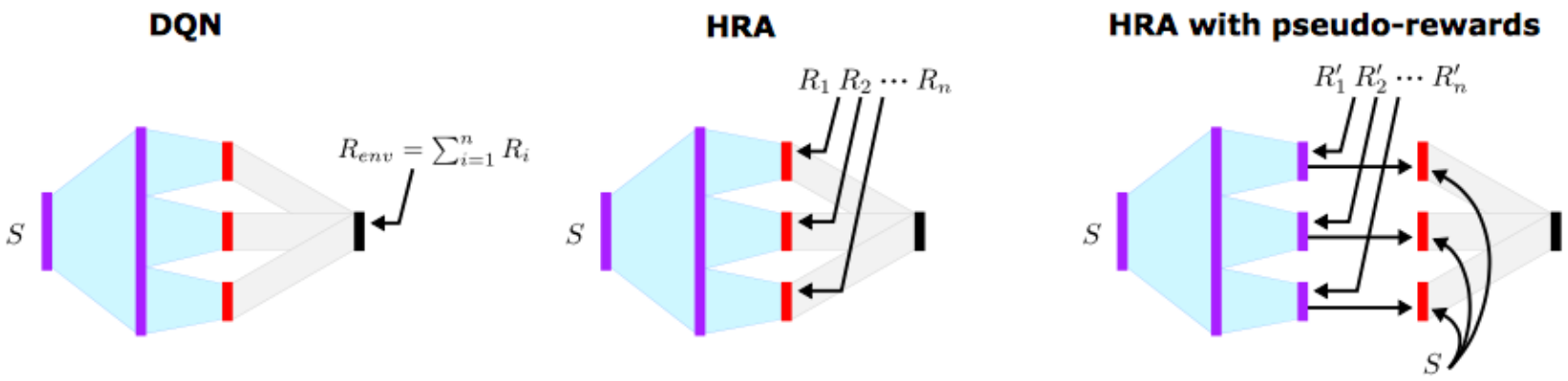


Figure 2: The different network architectures used.

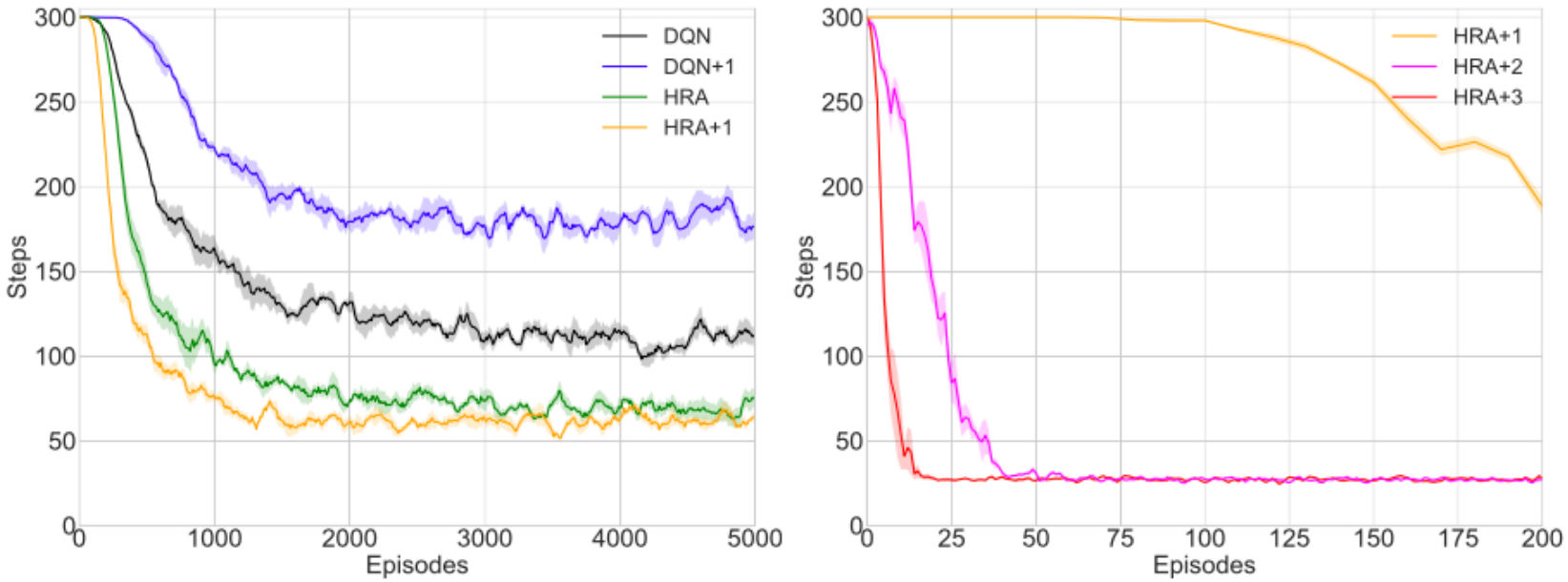
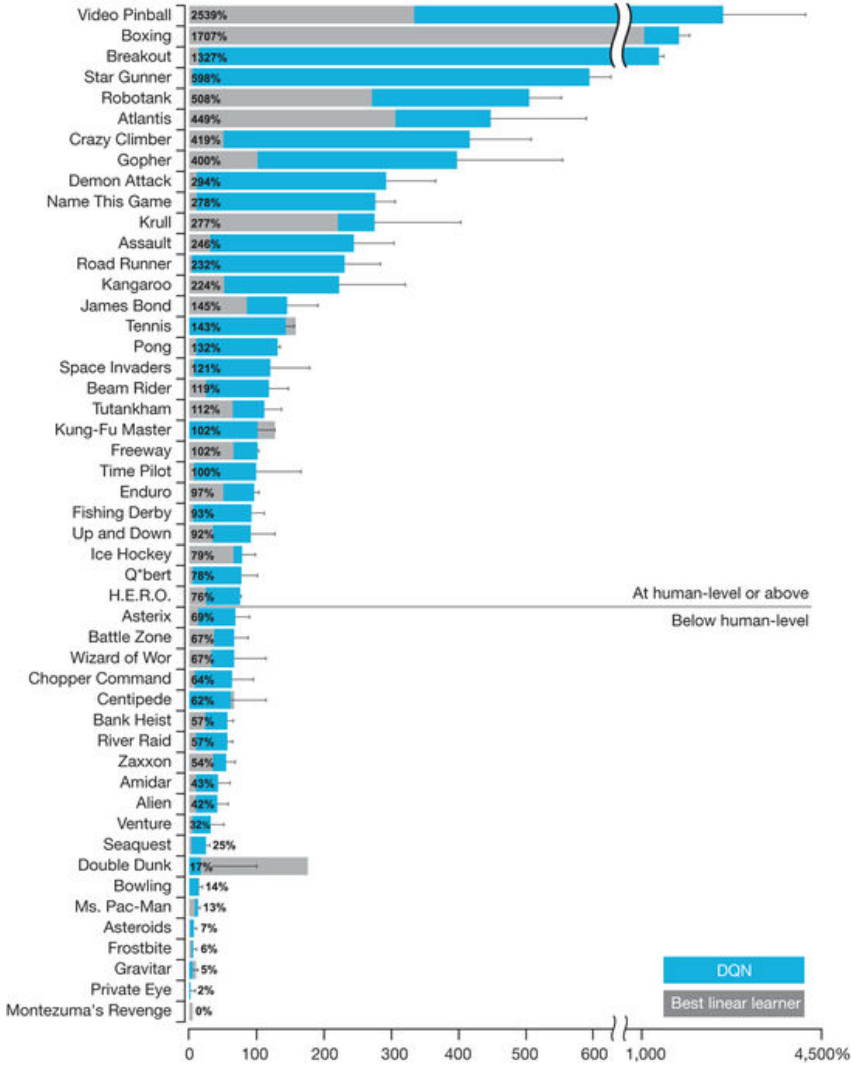


Figure 3: Results on the fruit collection domain, in which an agent has to eat 5 randomly placed fruits. An episode ends after all 5 fruits are eaten or after 300 steps, whichever comes first.

Test Bed: Ms. Pac-Man

Figure 4: The game Ms. Pac-Man.


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Ms. Pac-Man

- Uses 163 "sub-agents"

Ms. Pac-Man

- Uses 163 "sub-agents"
  - 154 Pellets,
  - 4 Ghosts,
  - 4 Blue Ghosts,
  - 1 Fruit.

Ms. Pac-Man

- Uses 163 "sub-agents"
  - 154 Pellets,
  - 4 Ghosts,
  - 4 Blue Ghosts,
  - 1 Fruit.
- For each object, there is a separate input channel which encodes its location with an accuracy of 4 pixels.

Ms. Pac-Man

- Uses 163 "sub-agents"
  - 154 Pellets,
  - 4 Ghosts,
  - 4 Blue Ghosts,
  - 1 Fruit.
- For each object, there is a separate input channel which encodes its location with an accuracy of 4 pixels.
- Each agent has its own reward function.

Ms. Pac-Man

- Uses 163 "sub-agents"
  - 154 Pellets,
  - 4 Ghosts,
  - 4 Blue Ghosts,
  - 1 Fruit.
- For each object, there is a separate input channel which encodes its location with an accuracy of 4 pixels.
- Each agent has its own reward function.
- The aggregate reward functions tell the "master" agent which action is best.

HRA architecture:

- R_{env} corresponds with the points of the game.

HRA architecture:

- R_{env} corresponds with the points of the game.
- Reward shaping occurs prior to decomposition.

HRA architecture:

- R_{env} corresponds with the points of the game.
- Reward shaping occurs prior to decomposition.
- HRA uses General Value Functions (GVFs) that learn pseudo Q-values.

HRA architecture:

- R_{env} corresponds with the points of the game.
- Reward shaping occurs prior to decomposition.
- HRA uses General Value Functions (GVFs) that learn pseudo Q-values.
- HRA starts with 0 GVFs and 0 heads for the pellets.

HRA architecture:

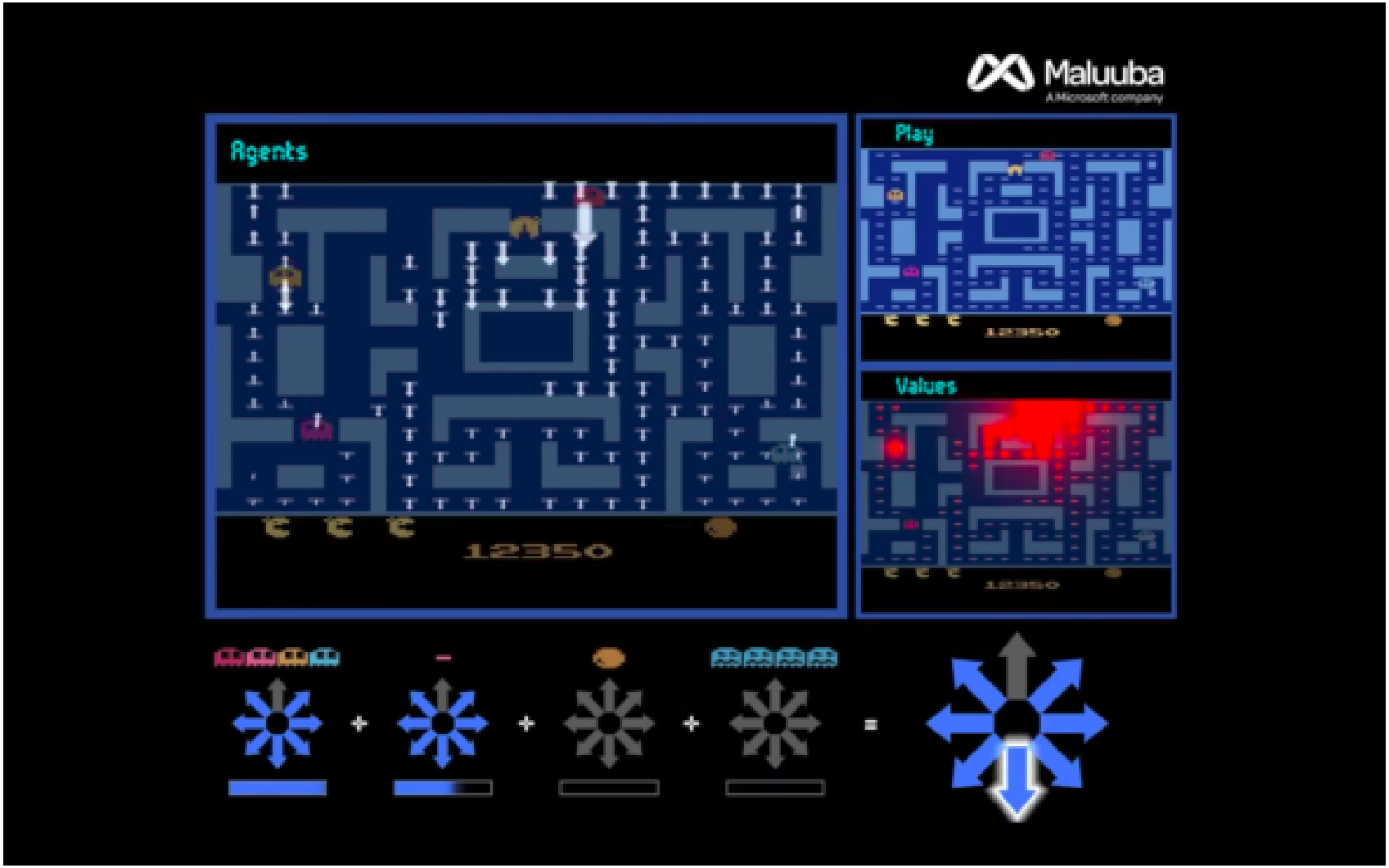
- R_{env} corresponds with the points of the game.
- Reward shaping occurs prior to decomposition.
- HRA uses General Value Functions (GVFs) that learn pseudo Q-values.
- HRA starts with 0 GVFs and 0 heads for the pellets.
- Discovery of map locations results in new GVFs being created.

HRA architecture:

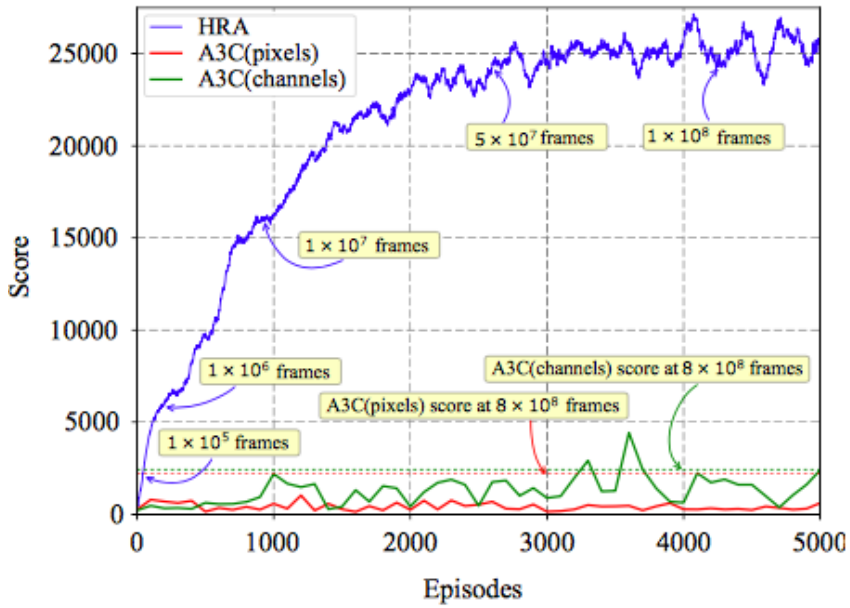
- R_{env} corresponds with the points of the game.
- Reward shaping occurs prior to decomposition.
- HRA uses General Value Functions (GVFs) that learn pseudo Q-values.
- HRA starts with 0 GVFs and 0 heads for the pellets.
- Discovery of map locations results in new GVFs being created.
- When an agent finds a pellet at a new location, it creates a new "head" corresponding to that location.

HRA architecture:

- R_{env} corresponds with the points of the game.
- Reward shaping occurs prior to decomposition.
- HRA uses General Value Functions (GVFs) that learn pseudo Q-values.
- HRA starts with 0 GVFs and 0 heads for the pellets.
- Discovery of map locations results in new GVFs being created.
- When an agent finds a pellet at a new location, it creates a new "head" corresponding to that location.
- If an object is not on-screen, all its Q-values are 0.


Maluuba
A Microsoft company
Agents
12350
Play
Values

Video



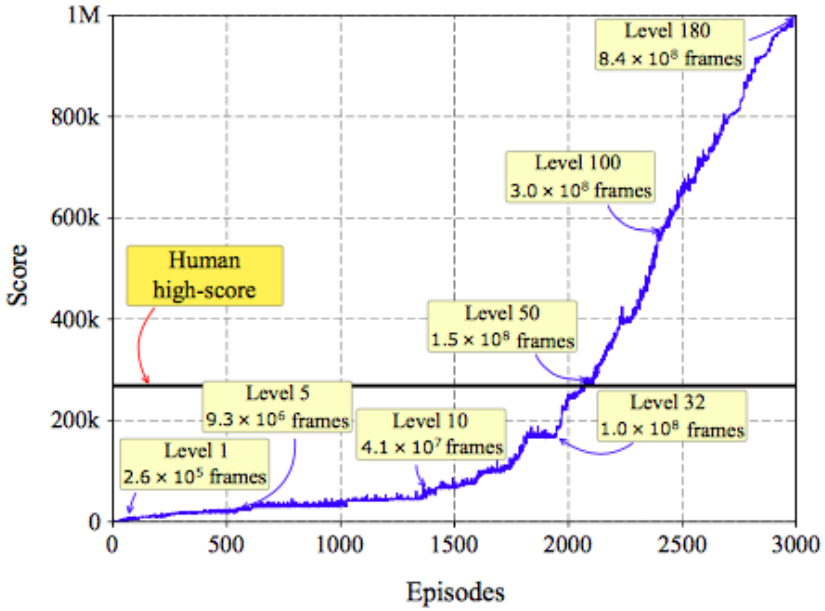


Figure 5: Training smoothed over 100 episodes. Figure 6: Training with trajectory memorisation.

- HRA solves Ms. Pac-Man by learning approximately 1,800 general value functions.

- HRA solves Ms. Pac-Man by learning approximately 1,800 general value functions.
- The input state-space is in the order of 10^{77} states.

- HRA solves Ms. Pac-Man by learning approximately 1,800 general value functions.
- The input state-space is in the order of 10^{77} states.
- However, each GVF has a state space in the order of 10^3 states.

- HRA solves Ms. Pac-Man by learning approximately 1,800 general value functions.
- The input state-space is in the order of 10^{77} states.
- However, each GVF has a state space in the order of 10^3 states.
- This constitutes a significant reduction in dimensionality of representation, which allows the AI to learn more quickly.

Summary:

- Different parts working together to achieve the same goal (well grounded in extant theories of how the brain works).

Summary:

- Different parts working together to achieve the same goal (well grounded in extant theories of how the brain works).
- Separating tasks into discrete targets decreases the problem size to make the problem more tractable.

Summary:

- Different parts working together to achieve the same goal (well grounded in extant theories of how the brain works).
- Separating tasks into discrete targets decreases the problem size to make the problem more tractable.
- Many real-world tasks may allow for reward decomposition.