Q-MAP: A CONVOLUTIONAL APPROACH FOR GOAL-ORIENTED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Goal-oriented learning has become a core concept in the reinforcement learning (RL) framework, extending the reward signal as a sole way to define tasks. Generalized value functions (GVFs) utilize an array of independent value functions, each trained for a specific goal, while universal value function approximators (UVFAs) enable generalization between goals by providing them in input. As parameterizing value functions with goals increases the learning complexity, efficiently reusing past experience to update estimates towards several goals at once becomes desirable, but requires independent updates per goal for both GVFs and UVFAs. Considering that a significant number of RL environments can support spatial coordinates as goals - such as on-screen location of the character in ATARI or SNES games, we propose a novel goal-oriented agent called Q-map that utilizes an autoencoder-like neural network to predict the minimum number of steps towards each coordinate in a single forward pass. This architecture is similar to Horde with parameter sharing and allows the agent to discover correlations between visual patterns and navigation. For example learning how to use a ladder in a game could be transferred to other ladders later. We show how this network can be efficiently trained with a 3D variant of Q-learning to update the estimates towards all goals at once. While the Q-map agent could be used for a wide range of applications, we propose a novel exploration mechanism in place of ε-greedy that relies on goal selection at a predicted target distance followed by several steps taken towards it, thus allowing the agent to take much longer and coherent exploratory steps in the environment. We demonstrate the accuracy and generalization qualities of the Q-map agent on a grid-world environment and then demonstrate how the proposed exploration mechanism allows the agent to explore much further than random walks on the notoriously difficult Montezuma's Revenge game and finally show how the combination of Q-map with a task-learner DQN agent improves the performance on the Super Mario All-Stars game.

1 INTRODUCTION

In Reinforcement Learning setting an agent attempts to discover and learn to solve a particular task in the environment. The task is generally specified in the form of a reward signal that the agent strives to maximize. While such reward signal is sufficient to describe any task in the environment, it can be advantageous to provide the agent with the task description in form of a specific goal to reach. These goals could be provided by a designed curriculum, or generated by a different agent and have shown to be advantageous for exploration in sparse-reward tasks.

Generalized value functions (GVFs) formalize the concept of goal-oriented Reinforcement Learning. This is achieved by creating a fixed array (also called horde) of independent off-policy value function agents (daemons) each simultaneously trained to achieve an agent-specific goal. UFV extends GVF approach by utilising a single value function that is provided with the goal specification in input, thus also enabling the agent to generalise to previously unseen goals.

One of the main challenges in such goal-parameterized policies is to make the best use of the data collected so far, to generalize to goals that were not necessarily selected when the data were collected. Training off-policy, using random goals and bootstrapping from the estimated value of next states allows in theory to learn to reach all goals simultaneously. In Horde, this requires to update

separately each of the daemons using a single transition while in UVFA, each goal needs to be passed in input to the same policy independently. Because the number of goals can be very large, possibly infinite, both of these approaches quickly become intractable if one wishes to update simultaneously towards all goals using a single transition.

While goal specification can be very complex, for many Reinforcement Learning environments goals specified in simple terms of spacial coordinates are sufficient to describe majority of the task. ATARI or NES games are a great example of such environments, where the agent's location is easily identifiable, and thus the goal can be defined in terms of on-screen coordinates. These goals are likely to be highly correlated both to the neighbouring goals, as well as the objects present in visual state.

Our first contribution is an agent, called Q-map, that uses a convolutional autoencoder-like architecture for goal-oriented reinforcement learning, that can be used to efficiently and simultaneously (in a single forward pass) produce value estimates for entire range of possible goals in compatible environments. We describe how to efficiently train such an agent and show that such approach is able to generalize to unseen environments to a good degree.

As a demonstration how Q-maps can be used, our second contribution is an exploration algorithm that exploits the capacity of the Q-map to quickly provide an estimated distance towards all goals and a policy to reach one selected randomly at a desired distance. We demonstrate how the ε-greedy exploration mechanism in DQN (Mnih et al., 2015) can be replaced with random goals reaching, expanding the exploration boundaries much further, especially in environments where actions can cancel each other or lead to terminal states.

The source code and videos can be found at sites.google.com/view/q-map-rl.

2 Q-MAP

In this section we briefly introduce the goal-oriented Reinforcement Learning followed by the description of the proposed Q-map agent capable to produce a simultaneous action-distance estimate towards a vast number of goals in the environment. We also show that the proposed agent has good generalization properties to environments never encountered during the training.

2.1 BACKGROUND

We consider the standard reinforcement learning framework in which an agent interacts sequentially with its environment, formalized as a Markov Decision Process (MDP) with state space S, action space A, reward function $r(s, a, s')$ and state-transition function $p(s'|s, a)$. At each time step t, the agent provides an action a_t based on the current state s_t and sampled from its policy $\pi(a_t|s_t)$. The environment responds by providing a new state s_{t+1} and a reward $r_{t+1}(s, a, s')$. Some states can be terminal, meaning that no more interaction is possible after reaching them, which can be simply considered as a deadlock state that only transition to itself, providing no reward. The action-value function of the policy

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi(a'|s')} \left[r(s, a, s') + \gamma Q^\pi(s', a') \right]$$

indicates the quality (Q-value) of each possible action when following the policy. In the Q-learning algorithm, the action-value function of the optimal policy π^* is iteratively approximated by updating the estimated Q-values

$$Q_{t+1}(s, a) \leftarrow (1 - \alpha) Q_t(s, a) + \alpha \left(r + \gamma \max_{a'} Q_t(s', a') \right)$$

using previously experienced transitions (s, a, s', r) and a learning rate α. At each time step, this learned action-value function can be used to take greedy actions $a = \arg\max_a Q(s, a)$ or random actions uniformly $a \sim \mathrm{U}(\mathrm{A})$ with probability ε. This basic exploration method is called ε-greedy and using the estimate of the value at the next step to improve the estimate at the current step is known as bootstrapping. Finally, the fact that the target $r + \gamma \max_{a'} Q_t(s', a')$ does not rely on the policy used to generate the data, allows Q-learning to learn off-policy, efficiently re-using previous transitions stored in a replay buffer or generated by a 3rd party.

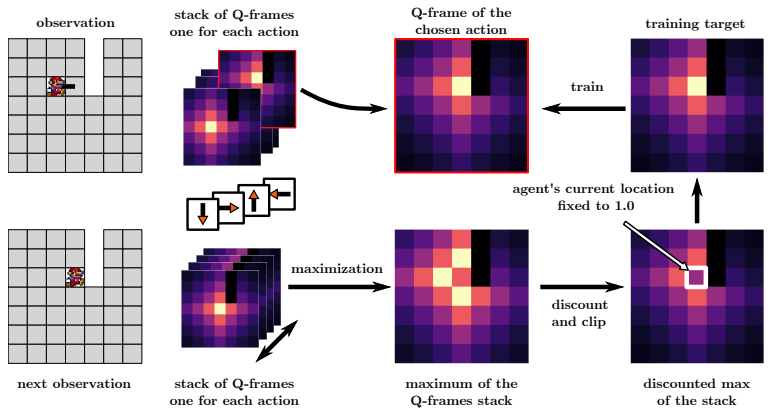

Figure 1: Illustration of the training process for a Q-map agent. The architecture allows to efficiently update the prediction towards all goals at once.

While an action-value function is usually specific to the rewards defining the task, general value functions (GVFs) $Q_g^\pi(s, a)$ are trained with pseudo-reward functions $r_g(s, a, s')$ that are specific to goals. The Horde architecture combines a large number of independent GVFs trained to predict the effect of actions on sensor measurements and can be simultaneously trained off-policy. Universal value function approximators (UVFs), extend the concept of GVFs by adopting a unique action-value function $Q^\pi(s, a, g)$ parameterized by goals and states together, enabling interpolation and extrapolation between goals.

It is possible to extract a notion of the distance γ^{k-1} between the current state and the desired goal in terms of the number of actions k by training GVF or UVF based agents with a pseudo-reward of 1 and termination in case the goal is reached, or otherwise rewarding 0. While the agent could in theory learn to reach any goal by attempting to reach them one at a time, the process can become very slow with increasing number of possible goals. However it is theoretically possible to use a single transition to update off-policy the Q-value estimates for the entire range of available goals. In Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), the goals used for the updates are chosen in the episode that gave a transition (s_t, a_t, s_{t+1}), after the step t while in Schaul et al. (2015a), the goals are simply chosen randomly, as updating for all becomes computationally difficult, requiring two forward passes through the network for every goal update.

2.2 The proposed agent

The core concept of the proposed agent called Q-map is to use a convolutional auto-encoder-like neural network to simultaneously represent all the GVFs of all possible on-screen locations for all possible actions from the raw screen pixels. This allows to efficiently share weights and to identify correlations between visual patterns such as walls, enemies or ladders and patterns in the distance towards on-screen locations. This means, for example, that learning how to navigate ladders can be transferred to other ladders later in a game. While stack of game frames are given in input, a stack of 2D Q-frames are produced in output where a single voxel at column x, row y and depth a represents the distance from the current state to the on-screen coordinates (x, y) given action a.

A great advantage of the Q-map architecture is that it allows one to query the estimated distance towards all goals in a single pass through the network and to very efficiently create a learning target from these estimates with just an additional pass using a 3D version of the Q-learning algorithm, where the rewards are now matrices comprising of zeros with only the value at the next coordinates set at 1 and Q-values are the 3D tensors consisting of a stack of 2D Q-frames with the maximization of Q-values performed on the depth axis and no bootstrapping at that location. Because the Q-values are bounded to $[0, 1]$, clipping of the estimated Q-frames at the next state can be performed to speedup the learning. As described in Figure 1, creating the target for an update thus consists of five steps: 1. perform a forward pass through the network with the next observation 2. clip and discount the values 3. maximize through the depth axis 4. replace the value at the next coordinate with 1 or in case of a terminal state, use a zero matrix with the 1 at the location 5. offset the frame

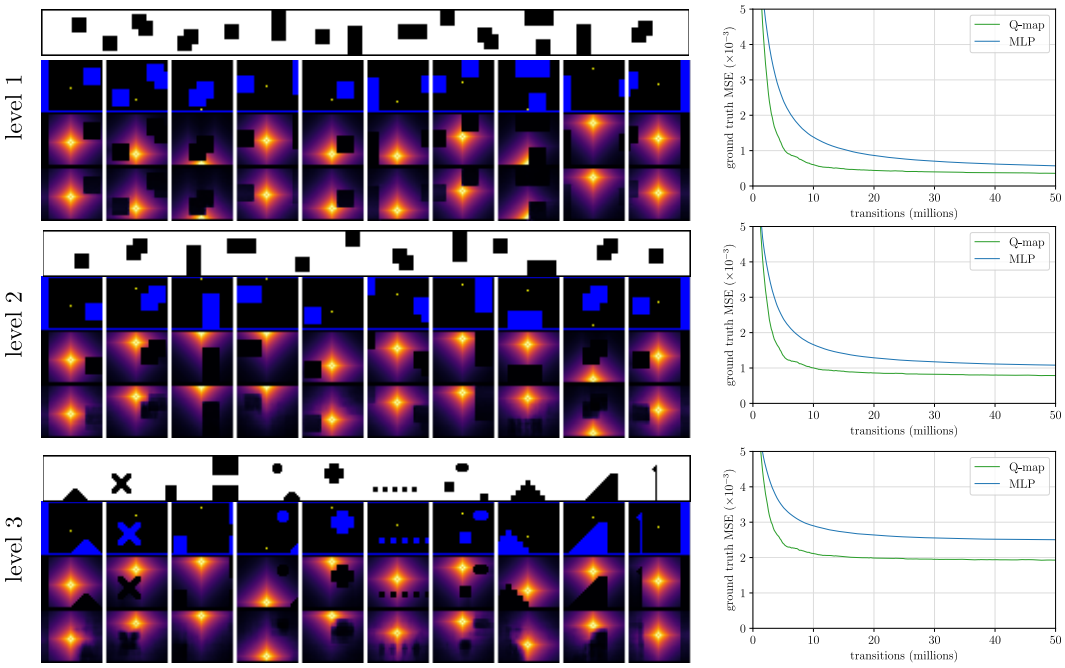

Figure 2: Left: For each level, the obstacles and walls are shown, with some observations provided to the agents, followed by the ground-truth Q-frames and predicted Q-frames (maximized over the action dimension). Right: Mean squared error (MSE) between the predictions and ground truth, for Q-maps using the proposed convolutional architecture and a simple multilayer perceptron (MLP). The proposed Q-map shows very accurate Q-frames on level 1 (used for training) and surprisingly good generalization on the two other levels even with radically different obstacles, especially when the convolutional architecture is used.

by the required number of pixels if the observations involve a sliding window that moved during the transition.

Q-maps are suited for environments in which it is possible to locate the agent's position in screen coordinates, which could either be provided by the environment (e.g. from the RAM in video games), or a separate classifier could be trained to localize the agent in the environment. While coordinates are used to create the target for the Q-learning, it does not however preclude one from only using raw frames as input for the Q-map agent. In some games, such as Super Mario Bros. (used later in the experiments), the screen scrolls while the player's avatar moves, and thus only a portion of an entire level is shown at once. In the proposed Q-map implementation we chose to use the coordinates available on the screen as possible goals and not coordinates over an entire level, thus the map is local to the area around the agent. While distance to goals could more directly be represented by the expected number of steps, the decay factor forces values to be bounded to $[0, 1]$, forcing the value of unreachable coordinates to naturally decay to 0 and the coordinates one step away to have a value of 1.

2.3 EXPERIMENTS

In order to measure the accuracy of the Q-map's representation of the minimum number of steps towards goals and to test its capacity to generalize to variety of states, we created a simple gridworld environment with three different arrangement of obstacles. The gridworld consist of a simple pixel that can be moved in cardinal directions in a 28×400 terrain with surrounding walls and non-traversable obstacles of various shapes. When an action would result in colliding with a wall or obstacle, the pixel is kept fixed. The observations consist of a a stack of 28×32 RGB frames corresponding to the view in a sliding window that moves horizontally with the pixel to keep it in the central column. Walls and obstacles are represented in blue while the pixel is represented in

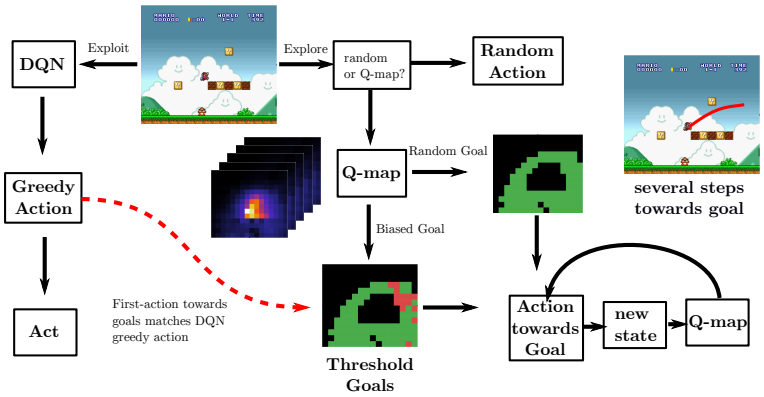

Figure 3: Illustration of the full action-decision pipeline of a DQN + Q-map agent.

yellow. The first and second levels share the same obstacle shapes with different locations while the third level consists of drastically different shapes of obstacles.

The Q-map is trained with transitions generated by randomly starting from any free locations in the environment and taking a random action. As a baseline we used a multi-layer perceptron (MLP) consisting of three fully-connected layers with layer normalization (Ba et al., 2016) and two output branches consisting of one fully-connected layer for each. The two outputs are aggregated in accordance to the dueling network architecture to represent the final Q-values (Wang et al., 2015), reshaped to create the Q-frames. The proposed convolutional autoencoder-like architecture consists of three layers of convolutions, followed by two fully-connected layers with layer normalization, followed by two output branches consisting of three transposed convolutions (also known as deconvolutions).

We then evaluate both the accuracy and generalisation of the Q-frames by limiting the training experience of the agent only to the first level of the grid world, while evaluating the output against a generated ground-truth Q-frames on all 3 levels separately.

As can be seen in Figure 2 the Q-frames generated by the convolutional network are almost identical to the ground truth. The agent generalises extremely well on the second level, which consists of familiar shapes in different arrangements. To our own surprise, the Q-map was able to handle unfamiliar shapes used in the thirst level to a good degree. Overall the convolutional network has demonstrated significantly faster learning and better generalisation properties when compared to the MLP baseline, validating the hypothesis that the proposed convolutional architecture is well suited for the studied scenario.

While the girdworld environment is useful for the evaluation of the Q-map performance versus the ground-truth, it does not offer a demonstration of the true capacity of the Q-map agent to handle much more complex visual input. We thus trained the Q-map on MontezumaRevenge and Mario All Stars environments, with examples of the learned Q-frames shown in Figure 4 and Figure 5.

3 Exploring the environment with Q-map

3.1 Background

The simplest, but yet very popular, method to explore in reinforcement learning, called ε-greedy, relies on a probability ε to take a completely random action at any time instead of the best-guess greedy action determined by the agent. The parameter ε is usually decayed linearly from 1 to a small value through the training to ensure a smooth transition from complete exploration to almost complete exploitation. A major issue is that it relies on random walks to discover new transitions which can generate many repetitive and unnecessary or even destructive transitions. This approach can potentially fail to efficiently push out the exploration boundary to discover the reward signal required for training.

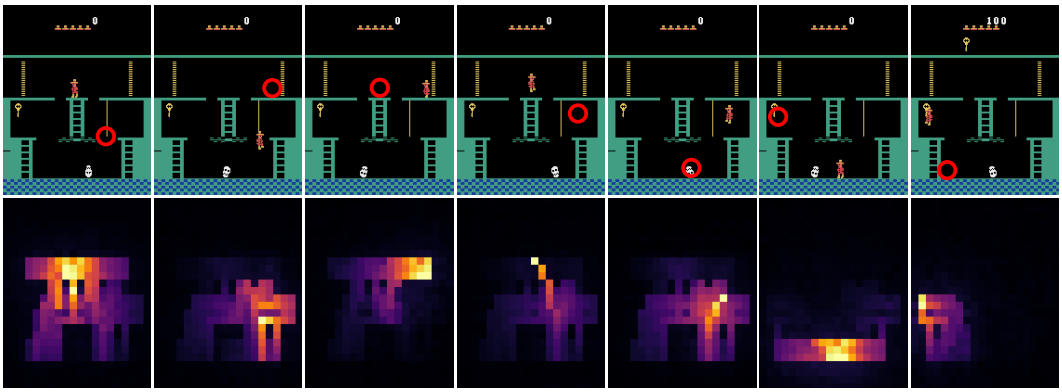

Figure 4: Example of learned Q-frames (maximized over the action dimension) on Montezuma's Revenge and how a simple random-goal walk allows to explore most of the first room. Goals are shown with circles.

Several methods have been described that bias the agent's actions towards novelty, primarily by using optimistic initialization or intrinsic rewards Oudeyer et al. (2007); Oudeyer & Kaplan (2009); Schmidhuber (1991; 2010). While the first approach is mainly compatible with tabular methods, the second relies on non-environmental rewards provided to the agent, for example, information gain Kearns & Koller (1999); Brafman & Tennenholtz (2002), state visitation counts Bellemare et al. (2016); Tang et al. (2017) or prediction error Stadie et al. (2015); Pathak et al. (2017). While these methods show success in sparse reward environments, they dynamically modify the agent's reward function and thus make the environment's MDP appear non-stationary to the agent.

If the agent is goal-oriented it is also possible to explore by providing a range of goals for the agent to reach. For example, Florensa et al. (2017) propose to label the ones that are at a desired level of difficulty and to use a GAN to generate similar goals, repeating the process enables a curriculum of increasingly difficult goals for the agent to progress through. A main issue with goal-oriented policies is that the goal representation needs to be sufficiently expressive to represent the required task in the environment, which can make the goal space excessively large and difficult to use.

Finally, RL algorithms that are able to learn off-policy can exploit data collected by a different exploratory agent or provided by demonstration. For example the GEP-PG agent (Colas et al., 2018) uses a policy parameter space search to parametrize an exploratory policy and generate diverse transitions for an off-policy DDPG (Lillicrap et al., 2015) agent to learn from.

3.2 The proposed exploration algorithm

The main idea of the proposed exploratory algorithm is to replace the random actions in ε-greedy by random goals. The proposed agent uses Q-map to explore and DQN to exploit. At each time step, the action selection follows a decision tree: With probability ε_r a completely random action can be selected to ensure pure exploration for Q-map and DQN. If no random action has been selected and no goal is currently chosen, a new one is chosen with probability ε_g by first querying Q-map for the predicted distance towards all the goals, filtering the ones too close or too far away (based on two hyper parameters specifying the minimum and maximum number of steps), choosing one at random and setting a time limit to reach it based on the predicted distance. If a goal is currently chosen, the Q-map is queried to take a greedy action in the direction of the goal and the time allotted to reach it is reduced. Finally, if no random action or goal has been chosen, DQN is queried to take a greedy action with respect to the task.

Because the number of steps spent following goals is not accurately predictable, ensuring that the average proportion of exploratory steps (random and goal-oriented actions) follows a scheduled proportion ε is not straightforward. To achieve a good approximation, we dynamically adjust the probability of selecting new goals ε_g by using a running average of the proportion of exploratory steps $\tilde{\varepsilon}$ and increasing or decreasing ε_g to ensure that $\tilde{\varepsilon}$ approximately matches ε. This allows us later

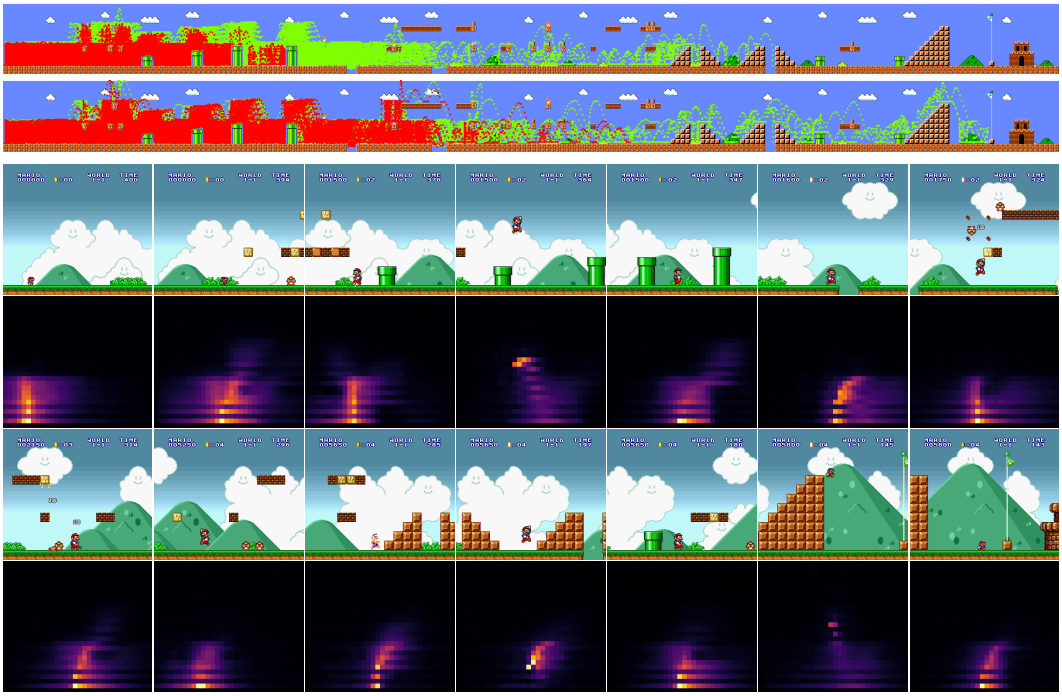

Figure 5: Top: Example of coordinates visited by the different agents after 2 million steps on the first level of Super Mario All-Stars. First image: Random walk (red) compared against the proposed Q-map random-goal walk (green). Second image: DQN using ε-greedy (red) compared against DQN with the proposed exploration (green). In both cases, Q-map allows to explore much further. Bottom: Example of learned Q-frames (maximized over the action dimension).

to compare the performance of our proposed agent and a baseline DQN with a similar proportion of exploratory actions.

Unlike ε-greedy approach, where the action is either entirely random or greedy, it is possible to bias the exploratory actions towards greedy actions by selecting a goal such that the first action towards the goal is the same as the greedy action proposed by the task-learner agent. Such bias aims to reduce the "cancelling-out" actions usually experienced with random exploration, while still widening the exploration horizon towards the task-learner's intended direction.

Finally, it is worth noting that such an exploratory algorithm is entirely compatible with intrinsic reward driven exploration, as such reward can still be provided to the task-learner agent, which would in turn benefit from Q-map trajectories to discover both the intrinsic and environmental rewards.

3.3 Experiments

First we compare exploration horizons between an agent performing entirely random actions and a Q-map agent that selects random goals in 15-30 step expected length. Neither of the agents are aware of the environmental rewards or motivated to explore any areas of the environment specifically. We first test both of the agents on the MontezumaRevenge environment, as it is a popular and challenging RL exploration task. In our experiments the random action agent was unable to reach the key in 5M steps. In the same amount of steps the Q-map agent has inadvertently picked up the key 398 times, while exploring the environment (Figure4 shows examples of Q-map exploring the Montezuma environment). We chose Mario for the second environment, as can be seen in top-most section of Figure5, random agent (red) does not explore beyond 1/3 of the level, while Q-map agent (green) is able to traverse more than 2/3 of the level just by attempting to reach random unbiased goals.

We then evaluate a combined agent that is comprised of Q-map exploration agent and a DQN Mnih et al. (2015) task-learner agent. The implementation is based on OpenAI Baselines' DQN imple-

mentation Dhariwal et al. (2017), using TensorFlow Abadi et al. (2015). Inputs are a stack of three 64×45 grayscale frames normalized (-1 for white and 1 for black). Each sub-agent uses an independent neural network with identical encoder architectures. We used double Q-learning Hasselt (2010) with target network updates every 1000 steps, prioritized experience replay Schaul et al. (2015b) (with default parameters from Baselines) using a shared buffer of $500,000$ steps but separate priorities. The training starts after 1000 steps and the networks are used after 2000 steps. Both networks are trained every 4 steps with independent batches of size 32. Two Adam optimizers Kingma & Ba (2014) are used with learning rate $10^{-}4$ and default other hyperparameters from TensorFlow's implemetation. The probability of greedy actions from DQN increases from 0 to 0.95 linearly throughout the experiment.

The DQN sub-agent outputs 6 Q-values, one for each action, uses dueling Wang et al. (2015), relu activations and a discount factor of 0.99. The Q-map sub-agent outputs 6 frames of size 32×28 (2 times lower resolution than the input), totalling 5376 Q-values. The deconvolutions (convolutions transpose) use the same kernel shapes as in the encoder but smaller strides, and the elu activation function. Goals are selected within an expected range of 15 to 30 timesteps while the time given to reach them contains a 50% supplement to account for possible random movements interfering with the trajectories. The goal selection is biased towards the DQN sub-agent's greedy action with a probability of 0.5. The discount factor used to decay the value towards impossible states and force the agent to minimize time is 0.9. Finally, the probability of taking completely random action at any time is decayed from 0.1 to 0.05 throughout the length of the experiment.

We compare the combined agent versus a DQN baseline on the first level of the Super Mario All-Stars game from the Super Nintendo Entertainment System (SNES) console, using the OpenAI Gym Retro Brockman et al. (2016). Transitions are deterministic, and the action set is limited to the 6 most basic ones: no action, move left and right, jump up and diagonally to the left or to the right. Terminations by touching enemies or falling into pits are detected from the RAM and only original SNES rewards are used and divided by 100. No bonus was provided when moving to the right and no penalty for dying. Typical rewards are 0.5 for breaking brick blocks, 1 for killing enemies, 2 for collecting coins, 4 for throwing turtle shells, 10 for eating consumables such as mushrooms, and 50 for reaching the final flag. The coordinates of Mario and of the scrolling windows were extracted from the RAM. Episodes are naturally limited by the timer of 400 seconds present in the game, which corresponds to 2394 steps.

Finally, to measure the performance of the proposed agent in terms of sum of the rewards collected per episode and number of flags reached we trained it and the baseline DQN for 5 million time steps with 4 different seeds and reported the results in Figure 6. Initially, the combined agent has worse performance than the DQN baseline - this is explained by the design of the game level, where frequent random actions, usually corresponding to jumps, would result in the agent gathering some early coins. However due to the shorter exploration horizon it much takes longer for the baseline to learn how to progress in the level and at 2M steps the combined agent overtakes the DQN in performance. The final performance of the combined agent is 30% better than the baseline with on average per seed 33 flags reached, versus 9 flags reached for the baseline.

4 Discussion

A major limitation of the proposed Q-map agent is the requirement to use coordinates as goals and to be able to track the agent's location. However, for compatible environments, Q-map offers a fast and efficient way to learn goal-oriented policies. It is worth noting that the approach could be extended beyond 2D to multidimensional observations, such as angles and velocities of objects, where Q-frames could be replaced by Q-tensors to represent larger coordinate spaces, architecturally achieved by using multi-dimensional transposed convolutions. This could, for example, enable the agent to handle scenarios involving robotic arms or a flying drones.

In this article we illustrated how Q-maps can be used to improve ε-greedy exploration, but the range of applications is much larger. Any scenario where learning to reach coordinates is useful can benefit from a Q-map approach. For example hierarchical reinforcement learning could combine a high-level agent rewarded with environmental rewards and outputting coordinates to reach for a low-level Q-map agent while a count-based exploration method could be combined with the estimated distance from a Q-map agent to select close and less visited coordinates.

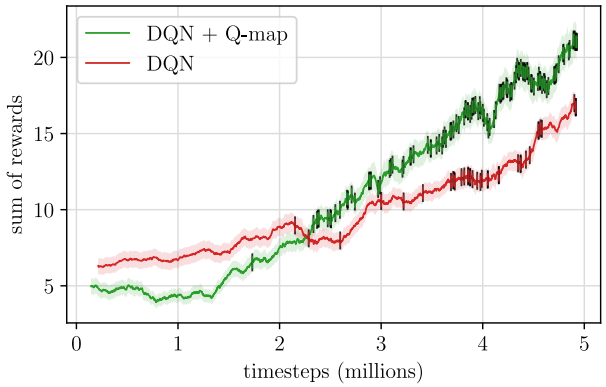

Figure 6: Performance comparison between DQN with ε-greedy exploration (red), and the proposed exploration (green) with confidence interval of 99%, averaged over 4 seeds. Vertical bars indicate flags reached (end of the level). While ε-greedy provides more rewards at the beginning (as completely random actions allow to easily get the first coins and jump on the first enemy), the proposed agent significantly outperforms the baseline, reaching earlier and more frequently the flag.

5 Conclusion

We proposed a novel reinforcement learning agent that efficiently learns to reach all possible screen coordinates in games. Each past transition can be used to update simultaneously every predictions at once using Q-learning while the convolutional autoencoder-like architecture allows the agent to learn correlations between visual patterns and ways to navigate the environment. We showed that the generated Q-frames match the ground truth and that the agent generalises well on a grid world environment. Furthermore, we proposed a novel exploration method to replace ε-greedy, based on the proposed Q-map agent, and demonstrated that it successfully manages to expand the exploration horizon on Montezuma's Revenge and Super Mario All-Stars allowing to significantly increase the performance of DQN.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.

Michael Kearns and Daphne Koller. Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, pp. 740–747, 1999.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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.

Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015a.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227, 1991.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2750–2759, 2017.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.