# Scaling All-Goals Updates in Reinforcement Learning Using Convolutional Neural Networks

**Fabio Pardo, Vitaly Levdik, Petar Kormushev**
Robot Intelligence Lab, Imperial College London, United Kingdom
{f.pardo, v.levdik, p.kormushev}@imperial.ac.uk

**Abstract**

Being able to reach any desired location in the environment can be a valuable asset for an agent. Learning a policy to navigate between all pairs of states individually is often not feasible. An *all-goals updating* algorithm uses each transition to learn Q-values towards all goals simultaneously and off-policy. However the expensive numerous updates in parallel limited the approach to small tabular cases so far. To tackle this problem we propose to use convolutional network architectures to generate Q-values and updates for a large number of goals at once. We demonstrate the accuracy and generalization qualities of the proposed method on randomly generated mazes and Sokoban puzzles. In the case of on-screen goal coordinates the resulting mapping from frames to *distance-maps* directly informs the agent about which places are reachable and in how many steps. As an example of application we show that replacing the random actions in $\varepsilon$-greedy exploration by several actions towards feasible goals generates better exploratory trajectories on Montezuma's Revenge and Super Mario All-Stars games.

## 1 Introduction

Reinforcement learning (RL) (Sutton and Barto 1998) environments can typically be separated into goal-reaching and reward-based types. In the former case, tasks like navigating a maze or moving an object to a target location directly justify the use of agents that are conditioned on both the observations of the world and the goals. In the latter case of RL environments, maximizing the discounted sum of future rewards provides a more subtle objective that can describe any given task. While learning can be more challenging in this case, it can sometimes be greatly simplified by learning a task-agnostic goal-directed policy in combination with a higher-level task-dependent one. (Bakker and Schmidhuber 2004; Kulkarni et al. 2016; Vezhnevets et al. 2017; Peng et al. 2017).

In both cases, learning to reach each goal independently would be very inefficient. However, because the goals conditioning the policy do not modify the dynamics of the environment, off-policy algorithms can greatly improve the sample efficiency by using *goal relabeling*. When replaying a past transition, this technique consists of substituting the given goal with another valid goal, in order to evaluate the action with respect to the new goal. Multiple strategies exist for selecting these goals. For example, within the list of future states visited in the episode from which the transition originated, as was proposed in Hindsight Experience Replay (HER) (Andrychowicz et al. 2017), randomly (Schaul et al. 2015; Veeriah, Oh, and Singh 2018) or from a generative model (Nair et al. 2018). However, in the case where all the possible goals can be enumerated, a single transition can be maximally used by updating the policy towards *all-goals*.

A traditional goal-conditioned network requires one forward pass per goal which can be intractable in the case of large goal sets. Instead, we propose to use a network producing Q-values for all combinations of actions and goals with a single observation in input. Furthermore, we propose to use convolutional neural networks (ConvNets) to scale to a large number of outputs simultaneously, exploiting the correlations between neighbouring goals.

We evaluate the accuracy and generalization properties of the proposed network on tasks consisting in finding the shortest path towards all points in random mazes and solving random Sokoban puzzles. Our results show that the proposed approach converges faster and achieves better final performance while being more stable than the the goal-in-input approach. We then demonstrate that the proposed network learns well in more visually complex and difficult to control environments. On Montezuma's Revenge we show that the capacity to query this large number of Q-values can allow an agent to explore well by selecting random but feasible goals. Finally, on Super Mario All-Stars, we show that when coupled with a task-learner DQN agent (Mnih et al. 2015) the exploration method significantly improves the performance.

The source code and videos are available on the website: https://sites.google.com/view/q-map-rl.

## 2 Related work

### 2.1 Goal relabeling strategies

Goal relabeling strategies rely on the fact that a single transition can be used to update the policy towards any goal. A

few different methods for selecting these goals have been proposed, some of which are explained bellow.

**Future states** In Hindsight Experience Replay (HER) (Andrychowicz et al. 2017), the goals are sampled from the list of subsequent states traversed in the remainder of the episode when selecting a transition. This allows to quickly propagate successful examples of goal reaching as all of the required transitions are present in the recorded episode. However, this approach limits the scope of goals considered.

**Random states** A simpler approach is to select goals uniformly over the set of goals which can be known or discovered through interaction (Schaul et al. 2015; Veeriah, Oh, and Singh 2018). This approach allows to learn to reach any known goal if trajectories can be discovered in the set of transitions or the model has enough capacity to generalize. One downside however, is that many goals can be unreachable from a given state or not useful and the value function can easily bootstrap from incorrect values.

**Imagined goals from a generative model** If the set of possible states is unknown and we wish to get more diverse goals than the ones discovered so far, it is possible to learn a generator from which the goals can be sampled. For example, a variational autoencoder (VAE) (Kingma and Welling 2014) can be trained to learn the distribution of states and used to generate new plausible goals (Nair et al. 2018). This approach can lead to more general-purpose goal-reaching policies but relies on the accuracy of the learned generator. Therefore, a mixture of imagined and random goals can be more advantageous (Nair et al. 2018).

**All goals** The previous approaches are sometimes referred to as *many-goals learning* (Veeriah, Oh, and Singh 2018) and are general enough to work with continuous and unknown goal spaces. However, many tasks of interest have a known finite number of goals. In which case, the best use of a single transition can be achieved by updating the policy towards all goals, hence the name all-goals updates. For example, an independent Q-value for each of the goals can be learned separately with tabular Q-learning (Kaelbling 1993). The main issue with this approach is that the number of updates scales linearly with the number of goals.

### 2.2 Using ConvNets to represent value functions

Training several function approximations in parallel quickly becomes impractical in the case of representing many value functions. Multiple works showed how a shared torso fully connected to multiple heads can be used to learn several value functions (Osband et al. 2016; Van Seijen et al. 2017; Cabi et al. 2017; Tavakoli, Pardo, and Kormushev 2018). A few approaches aim at representing value functions using convolutional neural networks, some of which are described bellow.

**RL with unsupervised auxiliary tasks** In UNREAL (Jaderberg et al. 2017), the main policy is trained along multiple unsupervised auxiliary policy heads to help generate useful features. The pixel-control task trains the agent to predict which actions generate the most change in the input frames. The head specific to this task uses transposed convolutions to generate a three dimensional array of Q-values. These values were however never used to control the agent and were not specialised in goal-reaching.

**Value Iteration Networks** A VIN module (Tamar et al. 2016) creates a reward frame which is fed into a convolutional layer generating Q-values which are then max-pooled along the action channel to produce a frame which is iteratively fed into the module until it represents the state-values of the optimal policy. An attention mechanism is then applied on the value frame to create features for a policy trained via reinforcement learning. A VIN module can in theory perform planning as demonstrated on goal-reaching tasks. While VIN is related to our model it has many differences. First, it does not directly use the generated values to act. Second, for goal-reaching tasks, those values are the expected returns when starting from each state location for a goal given in input while our approach represents the values of starting from a given state to reach all the possible goals. Finally, VIN performs planning by iteratively recomputing the values at every step while our approach is solely based on learning a direct mapping from inputs to values.

## 3 Background

**Q-values** We consider the standard reinforcement learning framework (Sutton and Barto 1998), in which an agent interacts sequentially with its environment, formalized as a Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $r(s, a, s')$ and state-transition function $p(s'|s, a)$. At each time step $t$, the agent generates an action $a_t$ sampled from its policy $\pi(a_t|s_t)$ conditioned on the state $s_t$. The environment responds by providing a new state $s_{t+1}$ and a reward $r_{t+1}$. 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 transitions to itself, providing no reward.

The action-value function of the policy is defined as:

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

It indicates the quality (Q-value) of each possible immediate action when following the policy afterwards.

**Q-learning** In the Q-learning algorithm (Watkins and Dayan 1992), the action-value function of the optimal policy $\pi^*$ is iteratively approximated by updating the estimated Q-values:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a')\right)$$

It uses previously experienced transitions $(s, a, s', r)$ and a learning rate $\alpha$. In $\varepsilon$-greedy exploration, this learned

action-value function can be used to take greedy actions $a = \arg\max_a Q(s, a)$ or random actions uniformly $a \sim \mathcal{U}(\mathcal{A})$ with probability $\varepsilon$. Finally, the fact that the target $r + \gamma \max_{a'} Q(s', a')$ 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 another mechanism.

**Generalized and Universal Value Functions** While an action-value function is usually specific to the rewards defining the task, the Generalized Value Functions (GVFs) (Sutton et al. 2011) $Q_g^\pi(s, a)$ are trained with pseudo-reward functions $r_g(s, a, s')$ that are specific to each goal $g$. 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. The Universal Value Function Approximators (UVFAs) (Schaul et al. 2015) 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.

# 4 Proposed model

## 4.1 Training and usage

While UVFA-like architectures take observations and goals in input and generate Q-values for every action in output, our model, named Q-map, only takes observations in input and generates Q-values for every goal and action in output. For example, in the case of a grid of 2D goal coordinates, when a stack of observation frames are provided in input, another stack of 2D *Q-frames* are generated where the rows and columns represent the goal locations and the number of frames represents the number of actions. Note that the height and width of the observations and generated frames can be different for example if the observations require more resolution as shown in some of the following experiments.

These output values represent Q-values in the context of the goal-reaching reward function awarding 1 with episode-termination at the goal and 0 otherwise. The discount factor $\gamma$ creates exponentially decaying values $\gamma^{k-1}$ indicating the number of steps $k$ to the goal. A value of 0 indicates that the goal cannot be reached (for example in an obstacle) and 1 means that the goal will be reached at the next time step.

For a given transition, only the output frame corresponding to the taken action is updated using a mean-squared error with a target frame. This target is generated as follows: First, a forward pass in the model is performed with the next observation in input. Then, the generated frames are maximized over the action dimension to generate a single frame representing a measure of the minimum expected number of steps towards each of the goals. This frame is then clipped to the range $(0, 1)$ to reduce the under- and over-estimations. Finally, the frame is discounted by $\gamma$ and the location reached at the next step is set to 1. This procedure effectively uses one environmental transition to virtually create a set of independent one-step episodes, one per goal, with termination on success and partial-episode bootstrapping otherwise (Pardo et al. 2018).

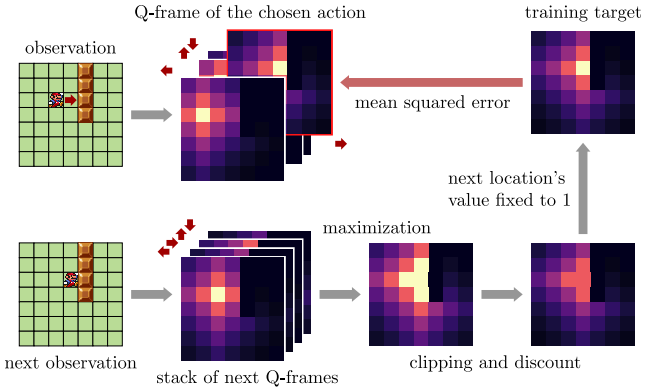


Figure 1: Training process for the proposed model, updating the prediction towards all goals at once.

To use the generated Q-frames to reach a specific goal, one only needs to take the vector of Q-values at the goal location and select the action of maximum value.

## 4.2 Neural network architectures

We considered multiple neural network architectures to support the proposed model. The first choice, which will be our baseline in the rest of the paper, is a UVFA-like network which takes a stack of observation and goal frames and generates the corresponding vector of Q-values, one for each action. To create a many-goals or all-goals update, the batch needs to consists of frames where the observation remains the same but the goal, represented by a one-hot frame, changes to cover all desired values. The batch output is then a set of vectors which can be reshaped to create the expected stack of Q-frames.

The second network uses an autoencoder-inspired architecture, with convolutions followed by fully-connected layers in turn followed by transposed convolutions. This network takes observation frames in input and generates the full Q-frames in output. The convolutional nature of this approach benefits from a potentially better capacity to share features and use correlations between the vision of the environment and the expected number of steps.

Finally, the last network architecture that we considered is composed solely of convolutions without compression and decompression. For example if the height and width of the Q-frames corresponds to the ones of the observations, strides 1 and padding "same" are used to keep the shape of the feature maps constant, not loosing any localization information.

## 4.3 Exploration with random goals

As a simple application case for a Q-map we propose an exploration method that replaces the noisy random actions frequently used to explore in reinforcement learning with a sequence of steps towards a goal. Using the Q-frames produced via Q-map, the goal is selected within a estimated close proximity to the current position and the actions are chosen greedily in order to reach it. While we do not expect this exploration method to outperform specialised

ones that use a variety of intrinsic signals, such as information gain (Kearns and Koller 1999; Brafman and Tennenholtz 2002), state visitation counts (Bellemare et al. 2016; Tang et al. 2017) or prediction error (Stadie, Levine, and Abbeel 2015; Pathak et al. 2017), it can be incorporated into most of those or used as a drop-in replacement for $\varepsilon$-greedy. It is worth noting that other works proposed to base the exploration on goal selection (Baranes and Oudeyer 2013; Florensa et al. 2017; Péré et al. 2018; Colas, Sigaud, and Oudeyer 2018) but to the best of our knowledge none of them relied on generating large and consistent steps in the environment using an auxiliary goal-reaching policy.

## 5 Experiments

In sections 5.1 and 5.2 we aim to evaluate the accuracy, training time and generalization properties of the proposed Q-map model on gridworld environments while in sections 5.3 and 5.4 we test Q-map in more visually complex environments and propose an application to exploration in reinforcement learning. In all of the experiments we use $\gamma = 0.9$ for the goal-reaching Q-functions and the neural networks are described using the notations: conv(filters, kernel sizes, strides) for convolutions (with padding "same" unless stated otherwise), deconv2d for transposed convolutions and dense(units) for dense layers. Elu activation functions are used for every layer except for the output ones. For more details, the full source code of the experiments is available at https://sites.google.com/view/q-map-rl.

### 5.1 Pathfinding in random mazes

The environment consists of a single pixel that can be moved in cardinal directions in a $16 \times 16$ area with traversable pathways surrounded by walls and generated such that any two points in the maze are only connected by one path. Actions towards walls result in the pixel remaining stationary. The observations consist of a stack of $16 \times 16$ RGB frames of the full view of the maze. The background is white, walls are black while the controlled pixel is in red. For the baseline model we use a green pixel to represent the goal.

We consider three duelling double DQN architectures for evaluation (Wang et al. 2016; Hasselt 2010; Mnih et al. 2015). The baseline architecture "goal-in-input" uses 2×conv(64, 4, 2)-dense(512) layers followed by a dense(256)-dense(4) advantage branch and a dense(256)-dense(1) state-value branch. The online network (not counting the target one) uses approximately 860K parameters. The second architecture Q-map "with compression" uses 2×conv(64, 4, 2)-dense(512)-dense(1024) layers followed by a deconv(64, 4, 2)-deconv(4, 4, 2) advantage branch and a deconv(64, 4, 2)-deconv(1, 4, 2) state-value branch. The online network uses approximately $1,260$K parameters. The third architecture Q-map "without compression" is composed of 4×conv(64, 4, 1) layers followed by a 3×deconv(64, 4, 1)-deconv(4, 4, 1) advantage branch and 3×deconv(64, 4, 1)-deconv(1, 4, 1) state-value branch. The online network uses approximately 800K parameters.

Instead of letting the agents interact with the environment we choose to generate a training and testing sets that are identical between the agents in order to remove any exploration bias when comparing the quality of the Q-frames generated. The training set is comprised of all possible transitions in $2,317$ different mazes for a total of $1,000,220$ transitions. The testing set is comprised of all possible starting points in 10 new mazes for a total of $1,075$ observations. The agents are trained with batches of 50 random transitions from the training set. The evaluation metric is the "success rate" which is a proportion of state-goal pairs where the agents correctly predict the greedy action towards all feasible goals, obtained by using the $\arg\max$ operator over the action dimension.

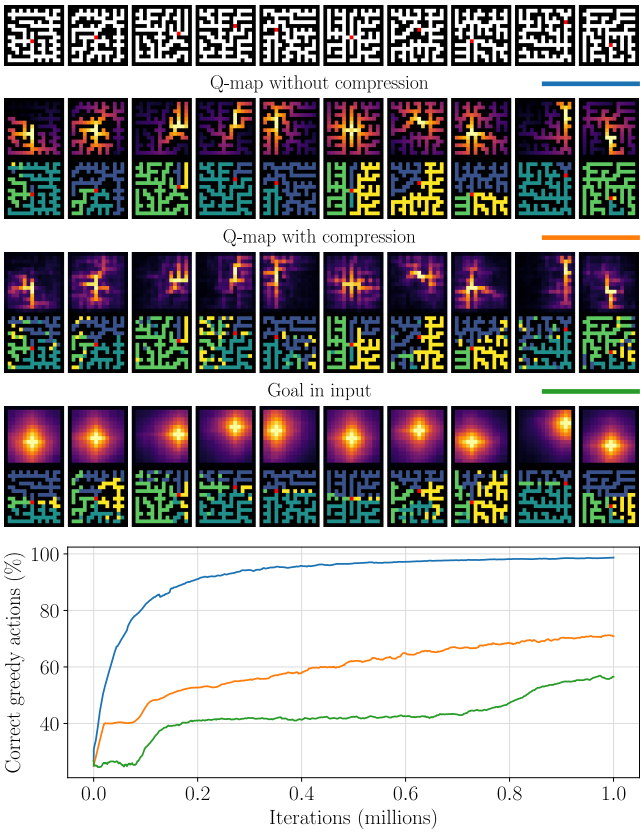


Figure 2: Learning to solve random mazes. Top: 10 unseen random mazes, then for each agent: the max Q-frames (maximized over the action dimension) and the first greedy action towards each goal. Bottom: percentage of correct greedy actions to take towards each possible goal. The Q-map architecture performs better than the baseline, in particular with the no-compression network.

The Figure 2 shows the training success rate curves for the architectures as well as examples of the learned Q-frames for the test mazes. In the solid-color filled maze images the color of a pixel depicts the action-choice made by the corresponding agent towards that pixel from the given location. Logically all pathways towards a single direction from the agent's location should have the same color. The Q-map "without compression" architecture clearly outperforms the alternative, achieving $99\%$ success rate at $1M$ training it-

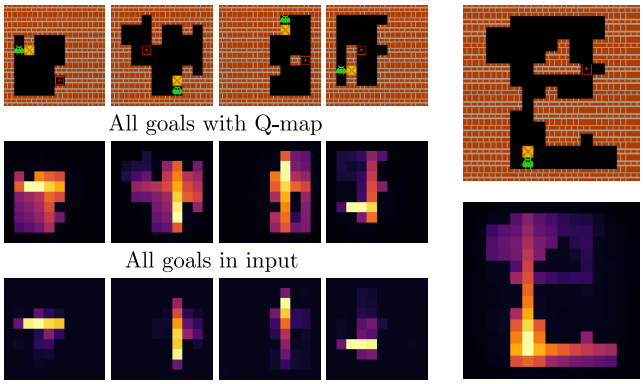


Figure 3: Left: Comparison between the max Q-frames learned by Q-map and the all-goals in input approaches on unseen $10 \times 10$ puzzles. Right: A separate example from a Q-map trained on $15 \times 15$ levels.

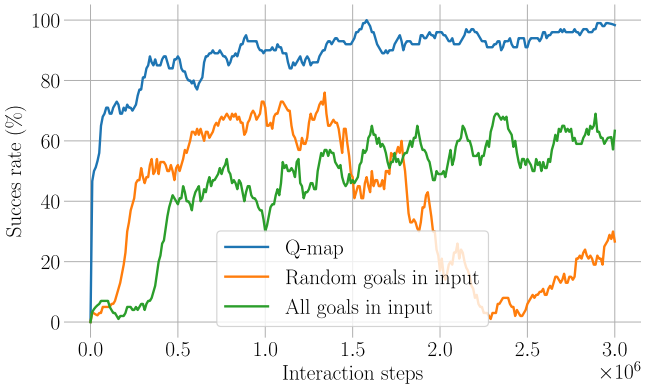


Figure 4: Success rate on unseen Sokoban puzzles. Q-map performs better than the goal-in-input baselines. The random-goals baseline is particularly unstable.

erations. The "Goal in input" architecture has learned what one could consider a naive first-pass solution to the task - that is a fading gradient centered on the agent's location in the maze, but failed to recognize walls of the maze. This is a common phase in training of all architectures, and indicates that the approach could require many more training iterations to improve further. The Q-map "with compression" based architecture performed significantly better but still struggled to produce sufficiently sharp Q-frames, unable to determine correct action choices outside of close proximity to the agent. It is worth noting that the Q-values for the maze walls are not specifically masked and the Q-map successfully learns to decay them to 0. Videos showing the Q-frames and actions through the training of all three architectures are available on the website.

### 5.2 Solving Sokoban puzzles

In this section we consider a notoriously difficult environment suited for planning (Racanière et al. 2017): Sokoban, from Gym-Sokoban (Schrader 2018). A significant difference to the previously considered Maze environment is that the goal is specified for a box which has to be carefully pushed by an agent-controlled avatar.

Similarly to the previous experiments, we compare the Q-map "without compression" model with two goal-in-input baselines. The two baselines differ by their goal-relabeling strategies: one uses random goals while the other one uses all goals. All the models are trained on the same batch size of 100. For the all-goals updates model, because there are 100 possible goals, a batch contains a single observation replicated 100 times with each of the possible goals.

Both of the baseline models uses a 5×conv(64, 5, 1)-2×conv(64, 5, 1, valid)-dense(256)-dense(4) network, while the Q-map "with compression" uses a 7×conv(64, 5, 1)-conv(4, 5, 1) network. We keep the total number of parameters roughly the same between the models with approximately 688K for the baselines and 626K for the Q-map.

We generate a unique $10 \times 10$ Sokoban level per episode including for evaluation. The models are provided with $10 \times 10$ RGB frames representing the walls, box and the agent's locations in red, green and blue respectively. In contrast to the Maze environment, we generate data by performing exclusively random actions and storing the state transitions in an unlimited buffer. This results in a less uniform training set that is more representative of an agent interacting with the environment, while using the same seeds generates identical transitions between agents.

Multiple generated Q-frames examples are given in Figure 3. The Q-map model was able to learn better representations showing how the box should be moved even in complex scenarios requiring a sequence of actions with long-term dependencies. Figure 4 shows the success rate of each of the considered models measured as the proportion of goals reached with greedy actions. Each point on the graph represents an average of 10 individual evaluation runs. The Q-map model straight away outperforms both of the baselines and keeps improving until almost perfect performance in a very stable way. Of the two baselines, the one trained with random-goals initially seems to learn faster but then deteriorates, suggesting that learning with all goals simultaneously is more stable.

Finally, we note that the Q-map model could in theory support multiple boxes by increasing the dimensionality of the outputs to account for the combinatorial nature of the problem. We however leave this experiment for future work.

### 5.3 Montezuma's Revenge

So far we have demonstrated the efficiency of the proposed Q-map model in a selection of gridworld problems which, while challenging to solve, have very simple state spaces and action dynamics. In this section we use the Montezuma's Revenge (Atari 2600) game to evaluate the learning capability of Q-map and roughly compare the exploration boundaries between a random-action policy and a random-goal policy that utilizes Q-map for action selection. Environmental rewards are ignored and no task-learner agent is present in this experiment.

The actions are limited to "no-op, left, right, jump, jump-left, jump-right, down" and are repeated four times. The

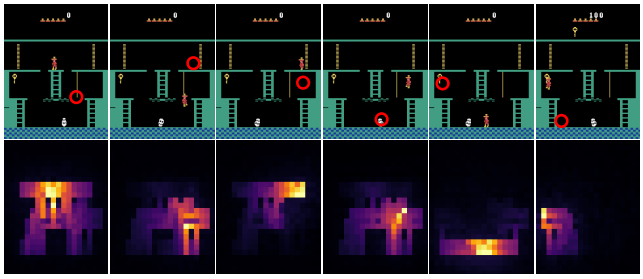

Figure 5: A random sequence of goal-reaching steps successfully exploring most of the first room in the Montezuma's Revenge game. The goals are shown with circles and the corresponding learned max Q-frames accurately represent reachable places.

Figure 6: Example of learned max Q-frames on Super Mario All-Stars. The Q-map successfully take into account the obstacles. The horizontal patterns are due to the action repeat.

game frames are zero-padded to reach a resolution of $160 \times 224$ before being scaled with a factor of $1/4$. Three greyscale $40 \times 56$ frames, spaced by two steps are stacked to produce the observations while another scaling factor of $1/2$ is used for the coordinates, limiting the Q-frames to $20 \times 28$.

We use a Q-map "with compression" to accommodate the resolution discrepancy between the observations and the output. The network uses conv(32,8,2)-conv(32,6,2)-conv(64,4,2)-dense(1024)-dense(1024) layers followed by a deconv(32, 4, 2)-deconv(32, 6, 2)-deconv(4, 8, 1) advantage branch and a deconv(32, 4, 2)-deconv(32, 6, 2)-deconv(1, 8, 1) state-value branch. A random goal is chosen within 15 to 30 predicted steps from the agent's current position. An individual goal-directed trajectory terminates upon either reaching the goal or exceeding $150\%$ of the original predicted number of steps. Furthermore, there is a chance to take a random action decayed linearly from $0.1$ to $0.05$.

Qualitatively, the learned Q-frames are sufficiently accurate for the random-goal policy to be able to navigate much further through the environment and even actively avoid the contact with the skull on the way towards a goal. Example Q-frames are shown in Figure 5. We also tracked the number of keys picked up by both of the policies during the training, with the random policy only reaching the key once in the $5$ million steps, while the random-goal policy first reached the key at the $1.2$ million steps and overall $398$ times.

### 5.4 Super Mario All-Stars

Finally, the proposed exploration method is used as a drop-in replacement for the $\varepsilon$-greedy exploration in a DQN agent on the Super Mario All-Stars game (SNES) (OpenAI 2018). The actions are limited to "no-op, left, right, up-left, up, up-right" and are repeated four times. The rewards from the game are divided by $100$ with $0$ bonus for moving to the right or penalty for game overs. Terminations by touching enemies or falling into pits and the coordinates of Mario and of the scrolling window are extracted from the RAM. Episodes are naturally limited by the timer of $400$ seconds present in the game, which corresponds to $2,402$ steps. Observations consist in three $56 \times 64$ grayscale frames spaced by two steps and the goal space is scaled down to $32 \times 28$.

We used the network described in Section 5.3 for Q-map. The network used for DQN has conv(32,8,2)-conv(32,6,2)-conv(64,4,2) layers followed by a dense(1024)-dense(6) advantage branch and a dense(1024)-dense(1) state-value branch. Furthermore, to account for the movement of the screen in the game, the target Q-frames are shifted accordingly before performing the updates.

Similarly to Section 5.3, we initially compare a random-goal Q-map based policy with a random-action policy. The states visited during 2 million steps of interaction are displayed in Figure 7). As can be seen, the proposed exploration method covers significantly larger area, allowing the agent to almost reach the end of the level without using any environmental or intrinsic reward.

We then evaluate an augmented agent that is comprised of a Q-map-based exploration and a DQN (Mnih et al. 2015) task-learner agent. For both this agent and the baseline we use an exploratory schedule that linearly decreases from $100\%$ to $5\%$. For the proposed agent, the random action probability is decreased from $10\%$ to $5\%$ over the course of the training. In order for the total proportion of exploration steps to match the planned schedule, the chance to start a new goal-reaching trajectory is then dynamically adjusted. Furthermore, to focus the exploration towards the task-learner policy a $50\%$ chance to select the goals with a first greedy action identical to the one from the task-learner is introduced.

To measure the performance of the proposed combined agent in terms of the sum of rewards collected per episode and the number of flags reached, we ran both agents for $5$ million steps with the same four seeds (0, 1, 2, 3) and reported the results in Figure 8. The initial performance of the augmented agent is worse than the baseline, likely due to the fact that early rewards can easily be collected with random movements. In turn the longer exploration horizon of the augmented agent enables it to learn to progress through the level faster and learn to reach the final flag consistently. In total, the baseline reached the flag $9$ times while the proposed agent reached it $33$ times with a final performance $30\%$ higher. Finally, we also tested the capacity of the Q-map model to adapt to a different level and found that the learning was significantly faster when transferring a pre-trained Q-map model from one level to another than when training from scratch as visible in the videos.

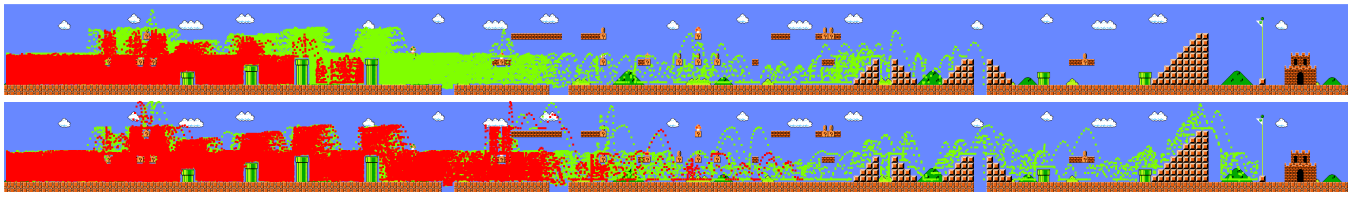

Figure 7: Coordinates visited after 2 million steps on the first level of Super Mario All-Stars. First image: Random walk (red) compared against the proposed Q-map random-goal walk (green). Second image: DQN using $\varepsilon$-greedy (red) compared against DQN with the proposed exploration (green). In both cases, Q-map allows to explore significantly further

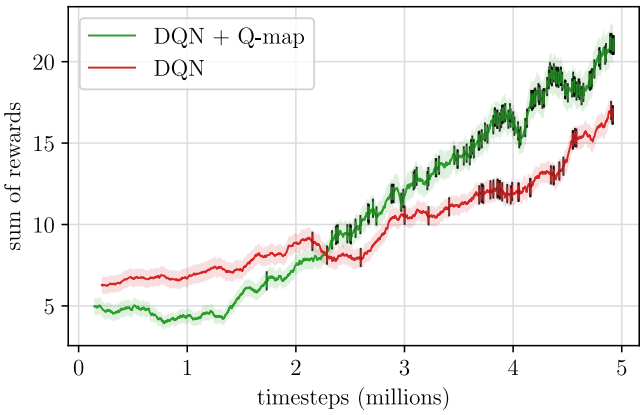


Figure 8: Performance comparison between $\varepsilon$-greedy exploration (red), and the proposed exploration (green) with confidence intervals of 99%. The vertical bars indicate flags reached. The proposed agent significantly outperforms the baseline, reaching the flag earlier and more frequently.

## 6 Discussion

In the experiments using Montezuma's Revenge and Super Mario All-Stars we assumed that the location of the avatar was available to create the updates. While this can appear as a strong assumption, it is not unreasonable to imagine that an agent could learn to localize its avatar or other controllable objects (Moniz, Patra, and Garg 2019; Sawada 2018) and use this knowledge when training a Q-map model.

Furthermore, it is worth noting that the proposed approach could be extended beyond images in observations, such as angles and velocities of objects or point clouds and the Q-frames could be replaced by Q-tensors to represent larger coordinate spaces architecturally achieved by using multi-dimensional convolutions. This could, for example, enable an agent to control robotic arms or flying drones.

Beyond the examples demonstrated in Section 5, learning to reach coordinates with Q-map can be useful in many scenarios. For example in hierarchical reinforcement learning, a high-level agent could be rewarded to provide useful goals to a low-level Q-map model. Also, a count-based exploration method could be combined with the estimated distance from a Q-map to select close and less visited coordinates. Furthermore, learning Q-map in itself could be a signal for a curiosity-like exploration algorithm.

## 7 Conclusion

We proposed an all-goals Q-learning model that is computationally efficient and able to generalise to previously unseen goals. A single environmental transition is used to generate a value estimate for the full set of possible goals in a single forward pass in the network. This is achieved by utilising a convolutional architecture with the intuition that such a model would take advantage of the correlations between the input features and the similarities of the neighbouring goals in the environment. We have demonstrated that this approach significantly outperforms goal-in-input baselines and achieves nearly perfect success rate in gridworld maze pathfinding and Sokoban tasks. In addition we have shown that the Q-map model is capable to learn to navigate in visually complex environments such as Montezuma's Revenge and Super Mario All-Stars games. Finally we provide an example of an application where by replacing random $\varepsilon$-greedy exploration actions with random goal-directed trajectories we improve the performance of a DQN agent.

## Acknowledgments

The research presented in this paper has been supported by Dyson Technology Ltd. and computation resources were provided by Microsoft Azure.

## References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in Neural Information Processing Systems*.

Bakker, B., and Schmidhuber, J. 2004. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. *Conference on Intelligent Autonomous Systems*.

Baranes, A., and Oudeyer, P.-Y. 2013. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*.

Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*.

Cabi, S.; Gmez Colmenarejo, S.; Hoffman, M. W.; Denil, M.; Wang, Z.; and De Freitas, N. 2017. The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. *Conference on Robot Learning*.

Colas, C.; Sigaud, O.; and Oudeyer, P.-Y. 2018. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *International Conference on Machine Learning*.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. *Conference on Robot Learning*.

Hasselt, H. V. 2010. Double Q-learning. *Advances in Neural Information Processing Systems*.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2017. Reinforcement learning with unsupervised auxiliary tasks. *International Conference on Learning Representations*.

Kaelbling, L. P. 1993. Learning to achieve goals. *International Joint Conferences on Artificial Intelligence*.

Kearns, M., and Koller, D. 1999. Efficient reinforcement learning in factored mdps. *International Joint Conferences on Artificial Intelligence*.

Kingma, D. P., and Welling, M. 2014. Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A. n.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Neural Information Processing Systems*.

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

Moniz, J. R. A.; Patra, B.; and Garg, S. 2019. Compression and localization in reinforcement learning for atari games. *arXiv preprint arXiv:1904.09489*.

Nair, A. V.; Pong, V.; Dalal, M.; Bahl, S.; Lin, S.; and Levine, S. 2018. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*.

OpenAI. 2018. OpenAI Retro. https://github.com/openai/retro.

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep exploration via bootstrapped dqn. *Advances in Neural Information Processing Systems*.

Pardo, F.; Tavakoli, A.; Levdik, V.; and Kormushev, P. 2018. Time limits in reinforcement learning. *International Conference on Machine Learning*.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. *International Conference on Machine Learning*.

Peng, X. B.; Berseth, G.; Yin, K.; and Van De Panne, M. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics*.

Péré, A.; Forestier, S.; Sigaud, O.; and Oudeyer, P.-Y. 2018. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. *International Conference on Learning Representations*.

Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Jimenez Rezende, D.; Puigdomènech Badia, A.; Vinyals, O.; Heess, N.; Li, Y.; Pascanu, R.; Battaglia, P.; Hassabis, D.; Silver, D.; and Wierstra, D. 2017. Imagination-augmented agents for deep reinforcement learning. *Advances in Neural Information Processing Systems*.

Sawada, Y. 2018. Disentangling controllable and uncontrollable factors of variation by interacting with the world. *arXiv preprint arXiv:1804.06955*.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. *International Conference on Machine Learning*.

Schrader, M.-P. B. 2018. Gym-Sokoban. https://github.com/mpSchrader/gym-sokoban.

Stadie, B. C.; Levine, S.; and Abbeel, P. 2015. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

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

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. *International Conference on Autonomous Agents and Multiagent Systems*.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. *Advances in Neural Information Processing Systems*.

Tang, H.; Houthooft, R.; Foote, D.; Stooke, A.; Chen, X.; Duan, Y.; Schulman, J.; De Turck, F.; and Abbeel, P. 2017. #Exploration: A study of count-based exploration for deep reinforcement learning. *Advances in Neural Information Processing Systems*.

Tavakoli, A.; Pardo, F.; and Kormushev, P. 2018. Action branching architectures for deep reinforcement learning. *AAAI Conference on Artificial Intelligence*.

Van Seijen, H.; Fatemi, M.; Romoff, J.; Laroche, R.; Barnes, T.; and Tsang, J. 2017. Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems*.

Veeriah, V.; Oh, J.; and Singh, S. 2018. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. Feudal networks for hierarchical reinforcement learning. *International Conference on Machine Learning*.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning*.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning*.