Decoupled Learning of Environment Characteristics for Safe Exploration

Pieter Van Molle [1] Tim Verbelen [1] Steven Bohez [1] Sam Leroux [1] Pieter Simoens [1] Bart Dhoedt [1]

Abstract

Reinforcement learning is a proven technique for an agent to learn a task. However, when learning a task using reinforcement learning, the agent cannot distinguish the characteristics of the environment from those of the task. This makes it harder to transfer skills between tasks in the same environment. Furthermore, this does not reduce risk when training for a new task. In this paper, we introduce an approach to decouple the environment characteristics from the task-specific ones, allowing an agent to develop a sense of survival. We evaluate our approach in an environment where an agent must learn a sequence of collection tasks, and show that decoupled learning allows for a safer utilization of prior knowledge.

1. Introduction

When using traditional reinforcement learning to train an agent for a specific task in an environment, the agent does not differentiate between the characteristics of the environment, and those of the task. This does not allow for an easy transfer of skills between tasks.

The above behavior can be problematic in real-world scenarios, where gathering experience is both costly and dangerous. Consider a warehouse for example, where autonomous drones are deployed. During training of the drones' policy, many have been lost due to crashes (Gandhi et al., 2017). Having to lose another series of drones when the objective changes would be far from optimal.

A better approach would be for the drones to have a notion of safety, or survival skills, regardless of their current task. In this paper, we introduce an approach to learn these survival skills independent of a task, by decoupling the environment characteristics from the task-specific ones when learning said task. We show that an agent that retains these skills between tasks exhibits a safer behavior than an agent that does not.

2. Reinforcement learning

In the reinforcement learning framework (Sutton & Barto, 1998), an agent interacts with its environment in a sequence of observations, actions and rewards. At each time-step t, the agent follows its policy π to take an action a_t, with respect to the observed state s_t of the environment. This results in a reward r_t and a new state s_{t+1} for the next time-step.

The objective of the reinforcement learning framework is to find a policy that maximizes the expected discounted return R_t

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \tag{1}$$

where γ is a discount factor between 0 and 1. Formally, this means finding a policy that follows the optimal action-value function $Q^*(s,a)$, defined as the maximum expected discounted return for taking an action a, given an observed state s, and following the optimal policy onwards.

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right] \tag{2}$$

Q-learning (Watkins & Dayan, 1992) is an off-policy algorithm that iteratively learns the optimal action-value function, by executing the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)). \tag{3}$$

Deep Q-learning (Mnih et al., 2013) approximates the optimal action-value function by using a deep Q-network (DQN). This is a deep neural network, parameterized by θ, that represents $Q(s,a;\theta)$.

3. Decoupled learning

When training for a task, an agent can gain a notion of survival by decoupling the environmental reward signals from

[1] Authors are with Ghent University - imec, IDLab, Department of Information Technology. Correspondence to: Pieter Van Molle <pieter.vanmolle@ugent.be>.

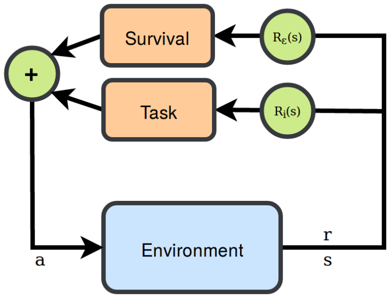

Figure 1. Architecture for decoupled learning.

the task-specific ones. This way, the agent can independently learn the environment characteristics. Formally, we define a reward function $R_\varepsilon(s)$ for the environment, and a reward function $R_i(s)$ for task i.

We can integrate decoupled learning in the Q-learning framework, by decomposing the action-value function

$$Q(s,a) = Q_\varepsilon(s,a) + Q_i(s,a) \tag{4}$$

where $Q_\varepsilon(s,a)$ is defined as the action-value function for survival and $Q_i(s,a)$ as the action-value function for task i. We can learn both functions iteratively by applying the appropriate reward function during updates, instead of using the global reward. This is illustrated by Figure 1.

When training for a new task, we can leave the learned survival function $Q_\varepsilon(s,a)$ unmodified, and must only learn the new task function. Furthermore, the survival function is used to safely navigate the environment while gathering experience for the new task.

As an example, consider the traditional cliff walking problem (Sutton & Barto, 1998), where an agent is separated from its goal by a cliff. The agent receives a positive reward reaching its goal, and a negative reward when it falls into the cliff. In this problem, we can interpret a negative reward as an environmental punishment and a positive reward as completing the task. In doing so, the cliff walking problem is transformed into a part inherent to the environment (don't fall into the cliff), and a part specific for the task (reach the goal). When training an agent for this problem, we decouple the environment characteristics by propagating a negative reward to the survival function, and a positive reward to the task function.

4. Experiments

We evaluate our approach in an 11x11 gridworld environment, as seen on Figure 2. In this environment, an agent must perform a collection task, while avoiding obstacles.

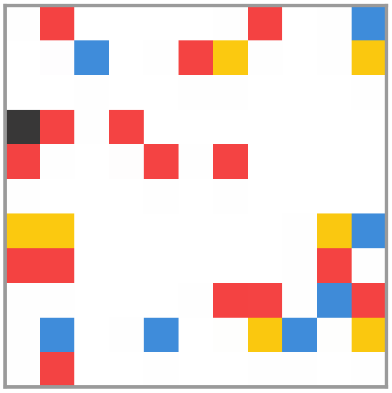

Figure 2. The environment. The agent (black) has to avoid the obstacles (red) and gather the collectibles (yellow or blue, depending on the task).

At the beginning of an episode, the agent spawns on a random space. Each other space can spawn either a collectible or an obstacle. These spawn with a probability of 0.05 for the collectibles and 0.15 for the obstacles. There are two types of collectibles, distinguished by color. A task consists of gathering as many collectibles of a single type. When an agent grabs a collectible, a new one of the same type appears on a random empty space. If the taken collectible was of the desired type, the agent receives a +1 reward. When crashing into an obstacle, the agent receives a -1 reward, and the episode ends instantly. Otherwise, an episode lasts for a maximum of 50 steps.

We train an agent on the first collection task, change the task, and apply different methods to train the agent on the second task. We compare three methods:

Naive learning To learn the first task, we use a single neural network to represent $Q(s,a;\theta)$. The weights θ of this network are randomly initialized. When learning the second task, we randomly re-initialize these weights.

Transfer learning Once again, a single neural network with weights θ is used to learn the first task. Next, the trained weights are used to bootstrap learning the second task.

Decoupled learning We use our decoupled learning approach, using two neural networks, to represent both the survival function $Q_\varepsilon(s,a;\theta)$ and the task-specific function $Q_i(s,i;\phi)$, during the first task. When training for the second task, we reuse the survival network, and only train a new network for the task.

The RGB state representation serves as input for each network. This input layer is followed by a convolutional layer

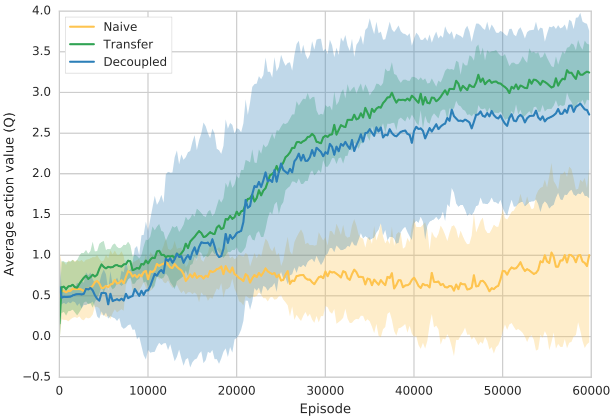

Figure 3. Learning curves for the second task, evaluated over a fixed set of states. All plots are an average over 9 random seeds.

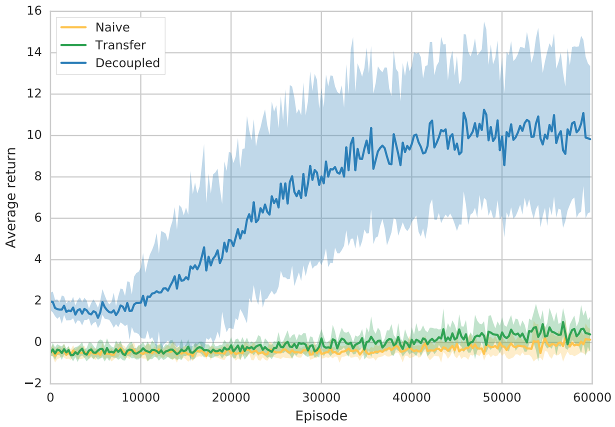

Figure 4. Comparison of the training performance of the naive, transfer and decoupled agent while learning the second task. All plots are an average over 9 random seeds.

with kernel size 3x3 and 64 filters, a max pooling layer, and two more convolutional layers with kernel size 3x3 and 32 filters. Another max pooling layer is followed by two fully connected layers, with 64 and 16 hidden units respectively. The output of each network consists of four values, representing the estimated Q-values for each of the four possible actions.

We use a replay memory of 10,000 experiences, which is continuously updated as the agent learns the first task. For the second task, a new replay memory is generated in different ways depending on the evaluated approach. In the naive case, the replay memory is initialized with random experiences. For transfer learning, the memory is filled with experiences of the agent performing the first task. When evaluating the decoupled approach, we use the survival network, trained during task one, to generate the replay memory.

We initialize the weights of each network using Xavier initialization (Glorot & Bengio, 2010). Training is done for 60.000 episodes, by means of the Adam algorithm (Kingma & Ba, 2014), with a minibatch size of 32 and a learning rate of 0.000025. For the second task, the naive agent is trained using ε-greedy learning, with ε linearly annealed from 1.0 to 0.1. The transfer agent always follows its policy during training, and the decoupled agent uses an ε-greedy strategy, but instead of sampling over the entire action space, the agent picks an action from a subset of safe actions, provided by the survival network.

To ascertain progress in re-training the agents, we apply each agent's action-value function as metric, as shown in Figure 3. It shows how the decoupled agent equals the transfer agent in convergence rate, albeit with a higher variance. The naive agent fails to converge within the given time frame.

Although the decoupled agent has the same convergence rate as the transfer agent, the first one learns a safer policy, resulting in a higher survivability. Because of this, the decoupled agent can take more steps each episode, which results in a higher episode return, as seen in Figure 4.

5. Related Work

In transfer learning, knowledge from a source task is used to learn a target task better than if transfer learning were not used, according to some metric such as training time or total accumulated reward (Taylor & Stone, 2009). Different approaches exist to transfer knowledge between tasks. Autonomous shaping (Konidaris & Barto, 2006) tackles a sequence of goal-directed reinforcement learning tasks by separating each task in a problem-space representation, which can be different for each task, and an agent-space representation, which is the same across tasks. Using the latter representation, a shaping function is learned that provides value predictions for novel states across tasks as to speed up learning. The separation of problem-space and agent-space can also be extended to the level of options (Sutton et al., 1999; Konidaris & Barto, 2007). Transfer learning via inter-task mapping (Taylor et al., 2007) uses hand coded task relationships to transform the action-value function from a source task to fit a target task with different state and/or action spaces. The MASTER method (Taylor et al., 2008) improves on the inter-task mapping by autonomously learning a mapping between a target task and one or more source tasks, by using experience the agent has gathered in the different task environments. When placing an agent in multiple environments, the agent itself is a common feature of each environment. By leveraging this stronger notion of an agent, the shared features framework (Konidaris et al., 2012) allows for both transfer of knowledge between a source and a target task, and a way to learn portable skills through a sequence of tasks. The Actor-Mimic method (Parisotto et al., 2015) involves a single policy network learning to act in a set of distinct tasks through the guidance of an expert teacher for each task. Furthermore, the learned representation of the policy network enables generalizing to new tasks without expert guidance. The use of successor features (Barreto et al., 2016), an extension of the successor representation (Dayan, 1993), combined with a generalized framework for policy improvement, allows an agent to perform well on a novel task if it has seen a similar task before.

6. Conclusion

In this paper, we present an approach for an agent to explicitly learn survival skills, by decoupling the environment characteristics from those of the task during training. This way, a learned representation of these characteristics can be transferred when training for a new task in the same environment. We compare our approach to both the naive method and the method of transfer learning. We evaluate each method by sequentially training an agent to gather different types of collectibles in a hostile environment. Our approach equals the method of transfer learning in terms of convergence, and, in addition, allows for a much safer utilization of prior knowledge, resulting in a higher episode return on average.

Following this paper, we plan to evaluate our approach in a real-world scenario, where a robot has to complete a series of tasks while crashing as little as possible.

Acknowledgements

Steven Bohez is funded by a PhD grant of the Agency of Innovation by Science and Technology in Flanders (IWT). We thank NVIDIA Corporation for their generous donation of a Titan X GPU.

References

Barreto, André, Munos, Rémi, Schaul, Tom, and Silver, David. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016.

Dayan, Peter. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Gandhi, Dhiraj, Pinto, Lerrel, and Gupta, Abhinav. Learning to fly by crashing. *arXiv preprint arXiv:1704.05588*, 2017.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pp. 249–256, 2010.

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

Konidaris, George and Barto, Andrew. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 489–496. ACM, 2006.

Konidaris, George and Barto, Andrew G. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pp. 895–900, 2007.

Konidaris, George, Scheidwasser, Ilya, and Barto, Andrew. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May):1333–1371, 2012.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Parisotto, Emilio, Ba, Jimmy Lei, and Salakhutdinov, Ruslan. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Taylor, Matthew E and Stone, Peter. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Taylor, Matthew E, Stone, Peter, and Liu, Yaxin. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8 (Sep):2125–2167, 2007.

Taylor, Matthew E, Kuhlmann, Gregory, and Stone, Peter. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pp. 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Watkins, Christopher JCH and Dayan, Peter. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.