Learning a Symbolic Planning Domain through the Interaction with Continuous Environments

Elena Umili [1]*, Emanuele Antonioni [1]*, Francesco Riccio [1], Roberto Capobianco [1], Daniele Nardi [1], Giuseppe De Giacomo[1] †

[1]Department of Computer Control and Management Engeneering Antonio Ruberti (DIAG) of Sapienza University of Rome, Italy.
E-mail $\{lastname\}$@diag.uniroma1.it

Abstract

One of the main challenges in AI is performing dynamic tasks by using approaches that efficiently predict the environment's future outcomes. State-of-the-art planners can reason effectively with symbolic representations of the environment. However, when the environment is continuous and unstructured, manually extracting an ad-hoc symbolic model to perform planning may be infeasible. Deep Reinforcement Learning is known to automatically learn compact representations of the state space through interaction with the environment. However, it is not suitable for planning, giving up the efficiency we would gain by predicting the consequences of actions. This work focuses on continuous state-space MDPs and proposes an approach that naturally combines interaction, symbolic representation learning, and symbolic online planning. Our system leverages experience-data gained from the environment to autonomously learn a symbolic planning model composed of: (1) a symbol grounding model to switch from continuous to symbolic space and vice versa; (2) a symbolic transition model; (3) a value function for symbolic states. This model is used at training time to lead the interaction with the world. At each interaction step, we perform fast symbolic online planning over a finite horizon to choose the action to execute in the environment. The success of this strategy in the environment implicitly validates our automatically extracted symbolic model, since the system is able to effectively plan actions in the original MDP by reasoning only in the finite and symbolic domain. The approach has been evaluated on several continual OpenAI gym environments, addressing successfully both control problems and games.

Introduction

The ability to make decisions autonomously is a key feature of artificial intelligence agents. When the agent has to perform a task in a dynamic and complex environment, its decision-making capabilities are strictly influenced by what the agent knows about its environment and how well it can predict its evolution. Automated planning is centered on finding a correct plan to solve a specific task, reasoning on the knowledge of the scenario, often expressed in a symbolic form. Just as humans manipulate ideas in their heads in order to plan their future actions, so do planners with *propositional symbols*. In this way, they can easily perform long-term projections in the future and plan actions accordingly. For example a human, or a symbolic planner, can plan the actions to turn on a car, knowing that: ″when I *turn the key* my car is *turned on*" - where 'turn the key' is an action modifying the truth value of the symbol 'turned on'- avoiding modeling irrelevant details such as the car color, the weather, and so on. Although this approach can be very performant on symbolic domains (Ghallab, Nau, and Traverso 2016), formalizing continuous and dynamic environments in such symbolic form is extremely hard and it is usually manually carried out by a human expert. On the other hand, there has recently been an increasing interest in Deep Reinforcement Learning (DRL) (Arulkumaran et al. 2017). DRL techniques automatically extract from data an effective policy to achieve the task in the environment, interacting with it through a trial-and-error process, and they do not require any models of the environment's states and transitions. However, by using DRL, it is not possible to plan over future actions. Moreover, the policy and the learned model are a black box, given the lack of explicability of the final result at the end of the training.

This paper proposes an algorithm inspired by both planning and Reinforcement Learning to *automatically* extract a symbolic planning domain for complex dynamic environments, this allows to exploit the advantages of planning techniques in this particular domains without the need of expert modeling. We develop an interactive learning algorithm that maps the environment state to an arbitrary size finite set of propositional symbols, tackling the symbol grounding problem (Steels 2008), and jointly learns the environment dynamics over this symbolic space. The system interacts with the environment formalized as a Markov Decision Process (MDP) over continuous variables. The experience data gained from the interaction are exploited to learn the following function approximators: a function for grounding the propositional symbols from the continuous observations; a value function approximator for evaluating symbolic states; a transition function approximator over the symbolic state space to predict the next symbolic states from the previous one and the corresponding action. This automatically learned planning domain is used to lead the interaction with

[*] Both authors equally contributed to this paper.

[†] Research partially supported by the ERC Advanced Grant WhiteMech (No. 834228) and by the EU ICT-48 2020 project TAILOR (No. 952215).

the environment. At each interaction step, we perform online planning (Ghallab, Nau, and Traverso 2016) over a finite horizon, and we select the first action of the computed plan as the action to take in the environment.

We evaluate the approach on several OpenAI gym environments with continuous state, including control problems and games. We also perform experiments setting different sizes for the finite symbols-set modelling the environment. Experiments show that, with a sufficiently large symbols-set, it is actually possible to plan the continuous environment's actions, reasoning only in the symbolic domain extracted from data. Furthermore, we show that we can use the learned planning domain to achieve different tasks in the same environment, through a reward shaping based on the learned symbolic representation. Experiments confirm that the symbolic representation-based reward and the transition model efficiently guides the agent towards different goals in the environment.

Related Work

The use of learning approaches for determining compressed and interpretable representations of the agent environment has been a very active research topic in the last years. Following this idea in (Garnelo, Arulkumaran, and Shanahan 2016), authors propose an end to end Reinforcement Learning architecture based on a neural network back-end and a symbolic front-end for addressing tabular game environments. In this work, a neural network is used to perform a symbolic representation of the agent state-space, then the symbolic state is used to obtain an effective action policy. The idea of determining a symbolic state to learn a propositional planning domain is introduced in (Asai and Fukunaga 2017; Asai and Kajino 2019; Asai 2019; Asai and Muise 2020), the learned planner is then used to solve a puzzle task. The system is based on the use of two different AutoEncoders: 1) a State AutoEncoder for the propositional representation of the image, 2) an Action AutoEncoder for obtaining the action transition definition on the domain. The works by Asai and collegues address the theme of the stability of symbols, that is, the system's ability to maintain the representation unchanged for small perturbations; this goal is obtained by using a State AutoEncoder. Authors have extended the planner learning algorithm introducing First-Order State AutoEncoder, unsupervised architecture for grounding the first-order logic predicates and facts. In this architecture, each predicate models a relationship between objects by taking the interpretable arguments and returning a propositional value. A similar task has been faced in (Suárez-Hernández et al. 2020) using an unsupervised learning approach. Differently from the previous works, in (Bonet and Geffner 2019), authors address the problem of determining the first-order representation from a non-symbolic input. The model is not learned with deep learning approaches, but it is extracted from the state-space structure. The required input is a labeled directed graph. All the works mentioned so far address environments with a discrete, even if sometimes complex, setting. Our system is meant to address also fully continuous and dynamic problems with multiple continual variables; this allows to address more problems closer to the real world environment. Different kinds of scenarios are addressed in (Dittadi, Drachmann, and Bolander 2020). In this case, a Variational AutoEncoder is used to learn relevant features in Atari games given images as training data. The planning is done with RolloutIW using the features learned by the VAE. The algorithm proves that VAEs can learn meaningful representations that can be effectively used for planning with RolloutIW. Similarly, in (Corneil, Gerstner, and Brea 2018), an external Variational AutoEncoder is used to obtain a propositional representation of the environment's state space. The propositional state is then given as input for a tabular Reinforcement Learning system and a neural network approach for learning state transitions. The combined architecture is exploited to learn with improved sample efficiency plans for solving the proposed tasks. Both these works involve the use of external VAE's for the states' propositional encoding. In our work, the VAE code is directly learned through the reconstruction error and the Q-network one, creating a symbolic representation encoded following the task specification. This allows to have state task-based state representation, useful for representing both the environment and the task associated to it. The idea of determining a model for a dynamic environment is addressed in (Kurutach et al. 2018). In this paper, authors present infoGAN, a system based on Generative Adversarial Networks (Creswell et al. 2018) to learn a plannable representation of dynamic problems. This approach identifies intermediate observation points in the agent task execution and the formalization of them in several kinds of symbolic representation. However, this approach does not reason about the action plan, but about intermediate state formalization for trajectory path planning problems. Finally, in (François-Lavet et al. 2019), authors present CRAR (Combined Reinforcement via Abstract Representations), a hybrid architecture composed of a model-based and a model-free component that jointly infer a sufficient abstract representation of the environment. This is achieved by explicitly training both the model-based and the model-free components, including the joint abstract representation. The CRAR agent creates a low-dimensional representation that captures task-specific dynamics, even without any reward. This last work introduces an architecture for obtaining a restricted meaningful abstract representation of the state-space but still relies on a continuous encoded space; Our work instead relies on a fully propositional symbolic representation of the environment.

Method

Overview

We consider an agent interacting with an unknown environment. The environment is modeled as an infinite horizon Markov Decision Process MDP = $\{S, A, t, r, \gamma\}$. Where $S \subseteq \mathbb{R}^n$ is a continuous state space, $A = \{a_0, a_1, .., a_m\}$ is a discrete action space, $t : S \times A \rightarrow S$ is the environment transition function, $r : S \times A \rightarrow \mathbb{R}$ is the reward function and γ is the discount factor. The agent knows the current environment state s_t, it can take an action $a_t \in A$ and observe the action-outcome, namely the new state $s_{t+1} \in S$ and a reward $r_{t+1} \in \mathbb{R}$ received by the environment.

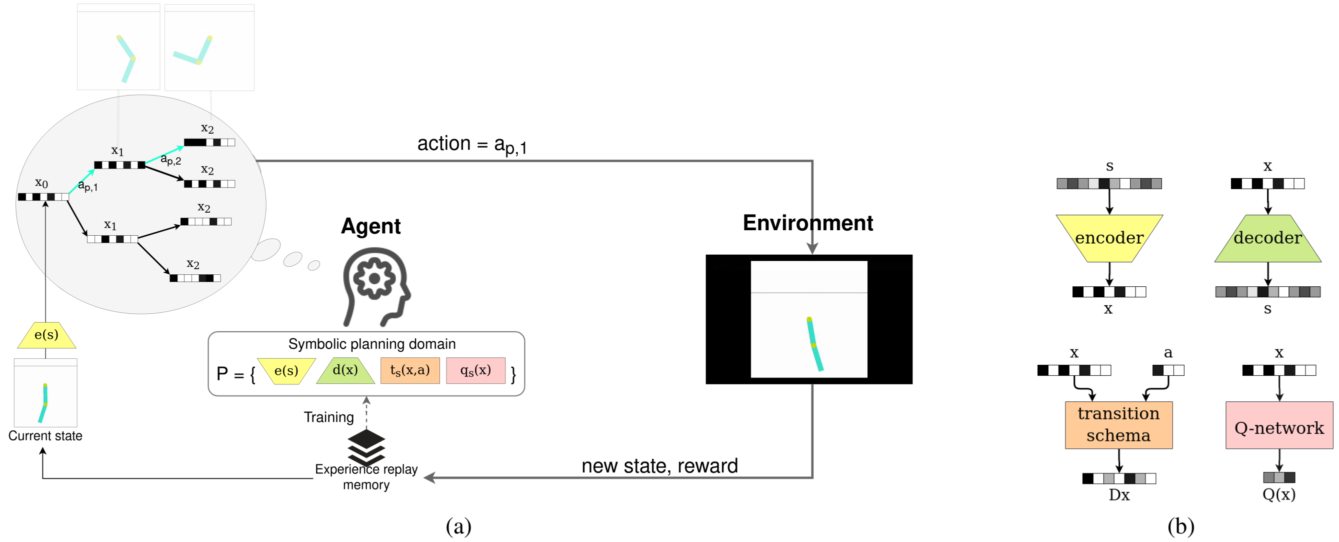


Figure 1: a) global view on the algorithm, b) neural networks trained by the algorithm

We want to learn a planning-based symbolic representation model of the reference environment interacting with the MDP. The idea of learning the environment model of MDPs, alongside the optimal action policy, is not new in Reinforcement Learning and is known also as model-based Reinforcement Learning (RL) (Moerland, Broekens, and Jonker 2020). Model-based RL algorithms usually learn the transition function and the reward function from the experience data gained through the interaction with the environment and use them to increase sample efficiency and temporal efficiency.

Our purpose in this work is not to increase the efficiency of the learning process, but to learn *another*, alternative, intrinsically different, representation of the same world, as similar as possible in its essence to a symbolic planning domain. Furthermore, we want use this model to effectively select actions to take in the MDP, so to achieve the desired task, as expressed by the MDP reward function. More formally, let's call $a_p(s)$ the action chosen by the planner when the environment is in state s, we want to maximize the expected cumulative discounted reward obtained by choosing actions according to the planner.

$$\sum_{t=1}^{\infty} \Big[\gamma^t r(s_t, a_p(s_t))\Big]_{s_t = t(s_{t-1}, a_p(s_{t-1}))} \tag{1}$$

Let's consider a symbolic state-space X composed of d_X propositional symbols.

$$X = \{0,1\}^{d_X} \tag{2}$$

At each time instant t, we can think of the observation s_t as the realization of a certain *situation* x_t identified by a specific truth value of the symbols in X. We define a situation as a boolean d_X-dimensional vector $x \in X$, and we can think of it as the latent symbolic representation of s. Since actions in A modify the observation that arises from the situation, we expect them to modify in some cases the underlying situation as well, and we want to model how this happens. In other words, we want to find a transition model t_s for the situation-space.

$$t_s : X \times A \rightarrow X \tag{3}$$

This model is conceptually similar to the effects of a PDDL-like (Ghallab et al. 1998) actions schema: given an action a and a situation x we want to learn which symbols to delete and add to x in order to have the next situation x'.

Furthermore, we want to know a measure of how much being in a particular situation is decisive for achieving the task in the environment. This is equivalent to know a numeric value function v_s over the symbolic space.

$$v_s : X \rightarrow \mathbb{R} \tag{4}$$

Last but not least, we want learn a *grounding* method for the symbols, namely a model e able to return the truth values of the symbols in every observed state $s \in S$, , and a function modeling the inverse mapping d.

$$e : S \rightarrow X \quad d : X \rightarrow S \tag{5}$$

While we want e to be accurate, since understanding the situation is essential for planning, we do not focus too much on the accuracy of d, since the more general and compact the symbolic representation of the situation is, the more varied are the possible manifestations of it in the environment.

The symbolic planner learned with the interaction can be seen therefore as a tuple of three functions.

$$P = \{e, d, t_s, v_s\} \tag{6}$$

Once we have learned P from experience we can use it to perform online planning during the agent-environment interaction in the abstract symbolic space. In other words, we may suggestively say that the agent is able to *abstractly* *imagine the future*, as it is shown in Figure 1(a). Abstract imagination is a fundamental part of our human intelligence: as humans we naturally make projections of our future reality by manipulating abstract ideas in our heads and we base our actions in the real world on those projections. In the same way the agent, starting from its current state s, can abstract the situation $x = e(s)$, imagine many possible finite-horizon-evolutions of the episode in its own idea-space, and choose the action leading to the best 'story in its head'.

Algorithm

Here we describe the interactive learning method from an high level point of view, as it is illustrated in Algorithm 1.

Algorithm 1: Interactive learning algorithm

```
Input: initial_state, desired_episode_reward
Output: P = $(e, d, t_s, v_s)$
P = initialize_models_randomly();
experience_buffer = empty_buffer();
$\epsilon$ = initialize_epsilon();
episode_reward = 0 ;
while episode_reward $\leq$ desired_episode_reward do
    s = initial_state;
    done = False ;            ▷ the episode is ongoing
    episode_reward = 0 ;
    while not done do
        mode = epsilon_greedy($\epsilon$);
        if mode == explore then
            a = random_action();
        else
            T = decide_finite_horizon();
            a = plan_action(P, s, T);
        done, s', r = execute(a);
        experience_buffer.append(s, a, s', r);
        episode_reward += r;
        s = s';
        batch = experience_buffer.extract_batch();
        P = train_models(P, batch);
    $\epsilon$ = decrease_epsilon();
```

The agent interacts with the environment by following a classic RL scheme. At each step it chooses an action $a \in A$ and executes it in the environment. The action outcome is the new state $s' \in S$ and the received reward $r \in \mathbb{R}$. The tuple (s, a, s', r) is the experience acquired through the step and it is stored in the experience buffer. Then, we train the models of P with a batch of data extracted randomly from the experience buffer. In particular, each model is implemented as a feed-forward neural network and is trained online - namely only with the data obtained through experience - no offline training is required. The action to execute is chosen with a classic epsilon-greedy exploration strategy: the agent performs random action choice with probability ϵ and abstract online planning with probability $1 - \epsilon$. ϵ is initialized to a large value and it is decreased over time. In this way, the agent relies more on the planner policy when the networks output becomes more consistent and robust. In order to select the action, the planner takes as input the current state s_t and the desired finite planning horizon T, that is increased as the mean reward grows. It plans future trajectories in the abstract space starting from the current situation $x_t = e(s_t)$, using the model t_s to expand abstract states for T steps. The symbolic-state trajectories are evaluated using the value function v_s, and finally the first action $a_{p,1}$ of the computed plan $(a_{p,1}, a_{p,2}, ..., a_{p,T})$ is selected for execution. The planning horizon is also adapted during training, starting from small length horizon when expansion and evaluation of plans are more prone to commit errors, and it is increased as the agent gains experience.

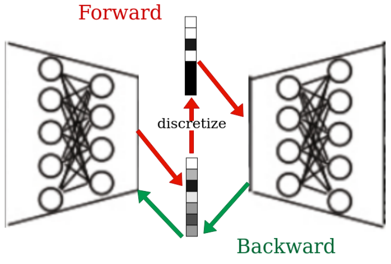


Figure 2: A neural network with an intermediary layer's output discretized only in forward propagation

The interaction stops when the agent achieves the task in the environment by choosing at each step the action returned by the symbolic planner, namely when the sum of the rewards obtained in the last episode reaches a certain desired cumulative reward value that depends on the task.

Learning the model

In this section we give details about each of the functions composing the model $P = \{e, d, t_s, v_s\}$ and how these are learned from data.

e**:** The mapping between the environment state-space to the symbolic space is implemented using a boolean-output encoder $e^{net} : S \to X = \{0, 1\}^{d_X}$, where d_X is the code length. Therefore, the encoder output space is composed of only 2^{d_X} possible different codes, and every single code is a possible interpretation over d_X propositional symbols. The boolean output is obtained by discretizing the sigmoid-activation function of the last layer with a discretization mechanism that will be described later.

$$e(s) = e^{net}(s) \tag{7}$$

d**:** A symbolic-input decoder, instead, models the inverse mapping $d^{net} : X \to S$. It reconstructs the environment observation in S from its symbolic representation x.

$$d(x) = d^{net}(x) \tag{8}$$

t_s**:** For the transition function in the symbolic space we trained a neural network model $t_\Delta^{net} : X \times A \rightarrow \{-1, 0, 1\}^d$, that learns the effects of actions on symbols in a PPDL-like formalism. t_Δ^{net} takes as input the symbolic state x and the action a, and it predicts which symbols the action modifies in the state. In particular, the output is discretized in three possible values: -1 for the symbols to delete from the state, 0 for the symbols that remain unchanged, 1 for the symbols to add to the state. Finally the next symbolic state x' is calculated as

$$t_s(x, a) = x + t_\Delta^{net}(x, a) \tag{9}$$

The discrete t_Δ^{net} output is achieved discretizing in $\{-1, 0, 1\}$ the value of a tanh activation function on the output layer.

v_s**:** To express the value of a symbolic state we analyzed two possible strategies:

1. learning the reward function r over the symbolic space,
2. learning the state-action pair quality function Q over the symbolic space.

The function $Q_\pi(s, a)$ is defined as the expected discounted cumulative reward of taking action a in the state s and then following the policy π. It is commonly used in model-free Reinforcement Learning algorithms like Q-learning (Jang et al. 2019). Although learning a Q function is much harder than learning a reward function, Q describes how much current actions influence future rewards, bringing much more information than the simple current reward function. For this reason we implemented the second choice, even if option one may still be a valid choice.

We define a Q-network taking as input the state in its symbolic form and returning an m-dimensional continuous vector $Q^{net} : X \rightarrow \mathbb{R}^m$, where m is the number of possible actions, and the i-th component of the output, $Q^{net}(x)_i$, is the expected cumulative discounted reward for taking action $a_i \in A$ from the symbolic state $x \in X$. To eliminate the dependency on a we define the value of a symbolic state as

$$v_s(x) = max_i Q^{net}(x)_i \tag{10}$$

Our system trains therefore three neural network models: e^{net}, t_Δ^{net} and Q^{net}, that are shown in Figure 1(b).

Training neural networks with symbolic or discrete layers Symbolic representations are compatible with symbolic planners, and they are often more interpretable and more computationally efficient than their continuous analogs. However, their discrete nature makes it difficult to be learned with neural networks.

Since discretization is a non-differentiable operation, in order to use backpropagation we cannot embed discretization as a layer in the network. On the other hand, since the discrete output is used as input by other models, we cannot discretize the layer outside the network. In fact, if we train the other models with *almost discrete* inputs, they will not predict the expected output when they are fed with *perfectly discrete* inputs.

Regarding our approach, we need both discrete and continuous outputs to be predicted by our system. For these reasons, discrete-output layers are trained using a particular mechanism: both the discretized and the continuous output tensors are maintained in the computational graph; the former is used during forward propagation, while the latter is used for backpropagation. This artifice, illustrated in Figure 2(b), solves both the issues mentioned above, because we use the continuous tensor for calculating the gradients necessary for updating the weights and we use the discrete tensor to feed the next layers of the model.

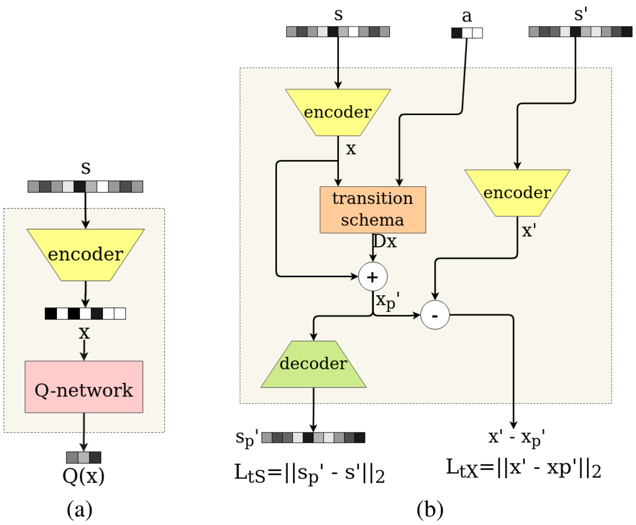


Figure 3: (a) the symbolic value model (b) computation of the transition model losses

Another possibility is to use the Gumbel-Softmax (Jang, Gu, and Poole 2017) as the activation function of the discrete layers. We tried to use it and we observed that the agent is able to increase the cumulative episode reward for a while, but finally fails to reach the desired episode reward value, even with very large size codings.

Value model training here we describe how we use the tuple (s, a_i, s', r) taken by the experience replay to update the value model, shown in figure 1(a)(b). For sake of clarity, we use the notation $Q^{net}(x, a_i)$ to indicate $Q^{net}(x)_i$.

We jointly train the encoder e^{net} and the symbolic q-network model Q^{net} minimizing the loss function

$$L_q(s, a_i, s', r) = ||Q^{net}(e^{net}(s), a_i) - V_t(s', r)||_2 \tag{11}$$

Where $V_t(s', r)$ is the expected cumulative reward according to the old Q-network model and it is calculated with the Bellman equation.

$$V_t(s', r) = r + \gamma \max_{a_i \in A} Q^{net}(e^{net}(s'), a_i)) \tag{12}$$

Transition model training Everything in the tuple except the reward is used to update the transition model also. So given (s, a, s'), we jointly train e^{net}, d^{net} and t_Δ^{net} minimizing two loss functions: (1) the transition prediction squared error in the symbolic space X, L_{tX} ; (2) the transition prediction squared error in the continuous original space S,

L_{tS}.Figure 3(b) shows how these two errors are calculated. Let's call the next symbolic state, as predicted by the transition model, x'_p.

$$x'_p(s,a) = t_s(e(s),a) = e^{net}(s) + t^{net}_{\Delta}(e^{net}(s),a) \quad (13)$$

We want to minimize the distance in the abstract space X between x'_p and the symbolic interpretation of s': $x' = e^{net}(s')$.

$$L_{tX}(s,a,s') = ||x'_p(s,a) - e^{net}(s')||_2 \quad (14)$$

L_{tS} is instead the distance in the continuous space S between the next state s' and its reconstruction from x'_p through the decoder: $d^{net}(x'_p)$.

$$L_{tS}(s,a,s') = ||d^{net}(x'_p(s,a)) - s'||_2 \quad (15)$$

L_{tS} regularizes the learning process of the symbolic representation. Notice the output of d^{net} is used only by this loss. In principle, we could do without decoding the symbolic states, since the planning component functions are designed to work only with the symbolic representation and never with the continuous one. In practice, if we do not consider loss L_{tS}, the training finds a local minimum where too much information is lost in encoding the states with e^{net}. In fact, according to loss L_{tX} only, the optimal e^{net} and t^{net}_{Δ} are the constant functions $e^{net} = \bar{x}$ and $t^{net}_{\Delta} = 0$, since this configuration of the models always leads to 0 error in predicting the next symbolic state. We prevent this degenerate solution by using the decoder, so to consider also the transition prediction error in the continuous space. In fact, L_{tS} tends to explode when the representation tries to collapse, since the decoder utterly fails in reconstructing different continuous states from the same symbolic representation.

To recap, L_{tX} tends to approach the representation of two states that were consecutive in the execution, while L_{tS} tends to keep in the symbolic representation some features useful for the reconstruction. The first property will be exploited for transfer learning, as it will be explained in a next section. Encoding in some way the 'temporal' distance between two states in the representation is very promising for planning, since the representation encodes 'how far' we are from reaching a particular state with the agent's actions. This means that distance between the current state and a desired one, in the symbolic space, can be considered itself as a quality function for planning toward that desired state.

Action selection

Here we describe how the models are used for online planning. We assume to know the planner functions e, t_s and v_s, since they are approximated by the neural networks trained with data from the experience replay. The planner takes the current continuous state s_0, and a desired planning horizon T and it has to select the action ensuring to achieve the best results over the next T future steps.

First the current symbolic state is calculated starting from s_0

$$x_0 = e(s_0) \quad (16)$$

then x_0 is expanded using the transition schema t_s for every possible action $a \in A$. By simulating the actions with the learned transition-schema, the agent reaches m new different symbolic states $x_1 = t_s(x_0, a_i)$. From each of them the expansion continues for all the possible actions, until the expanded tree reaches depth T. When the expansion stops the best trajectory in the tree is calculated as the path with maximum total value. Hence, the best plan is the sequence of actions corresponding to the best path of symbolic states.

$$(a_{p,1}, a_{p,2}, ..., a_{p,T}) = \underset{(a_1,a_2,...,a_T)}{\arg\max} \left[\sum_{i=1}^{i=T} v_s(x_i) \right]_{x_i = t_s(x_{i-1},a_i)} \quad (17)$$

Finally the first action of the computed plan is selected as the optimal action to take in the environment and it's executed by the agent.

Transfer learning

A benefit of using planning systems is that they can be reused infinite times, in order to calculate a plan for every possible couple (initial state, final state) in the domain. In RL instead, since the goal is expressed as a reward function, changing the goal is not so straightforward, and it consists of solving another MDP with a different reward function. Some techniques exist in transfer learning for RL (Zhu, Lin, and Zhou 2020), such as reward-shaping (Ng, Harada, and Russell 1999), that allow re-using part of the knowledge acquired solving an $MDP^1 = (S, A, t, r_{G_1}, \gamma)$ to solve an $MDP^2 = (S, A, t, r_{G_2}, \gamma)$, where r_{G_1} is the reward shaped to guide the agent to the goal G_1, and r_{G_2} is another reward function defined to lead the agent to the goal G_2. Regarding our approach, we would like to investigate if, once we have learned our symbolic planner $P^1 = (e^1, d^1, t^1_s, v^1_s)$ solving MDP^1, we can use it to guide the agent's actions towards a different goal G_2; or, in other words, if we can use P^1 to solve MDP^2.

In principle, we should be able to re-use the grounding method, e^1 and d^1, and the transition function t^1_s, but we cannot re-use for sure the quality function v^1_s, since it depends on r_{G_1}.

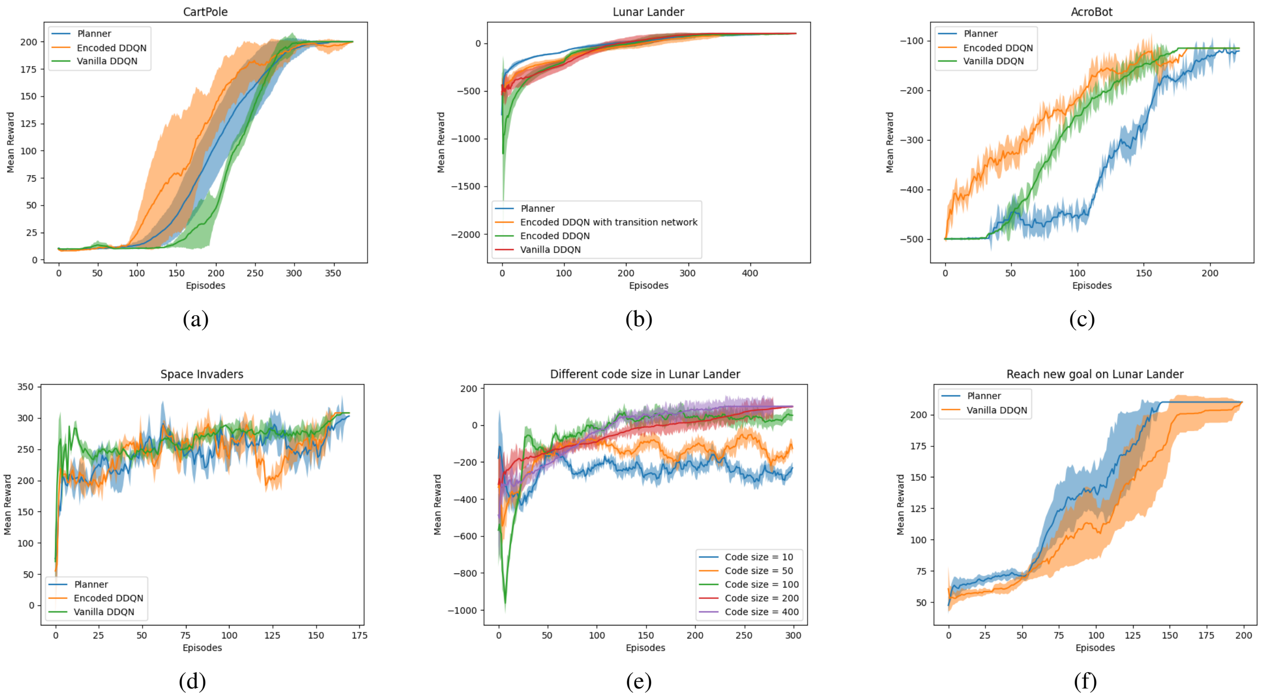
Since the symbolic representation tends to map closer states that were subsequent during execution, we examined the use of Jaccard distance between symbolic states, according the symbolic representation learned for goal G_1, to guide the agent toward goal G_2. We call this distance function Jd_{e^1,G_2}.

$$Jd_{e^1,G_2}(s) = 1 - 1/d_X \sum_{i=1}^{i=d_X} |e^1_i(s) - e^1_i(G_2)| \quad (18)$$

It is the distance between a generic state s, in its symbolic form according to e^1, and the goal $G2$, also converted in the same representation through e^1.

We tested the use of f_{e^1,G_2} as reward function to lead the agent toward G_2, with a generic model-free RL algorithm and with our learning algorithm.

In the second case, this is equivalent to define a new planning domain $P^2 = \{e^2, t^2_s, v^2_s\}$, initialize the models of P^2



(a)

(b)

(c)

(d)

(e)

(f)

Figure 4: Figure (a,b,c,d) show results on different continual gym environments as: Cartpole, Lunar Lander, Acrobot, Space Invaders. Figure (e) shows the results obtained using different code size on Lunar Lander. Figure (f) shows the results obtained retraining the system on a new goal in the Lunar Lander environment. The plots report on the y-axis the average mean reward (solid line) and standard deviation (shaded area) of the different training sessions.

as

$$e^2 = e^1, \quad t_s^2 = t_s^1, \quad v_s^2 = \text{RandomInitialization()} \quad (19)$$

and re-train the models running Algorithm 1 (by relying on everything as before, but the model initialization) in the $MDP^2 = (S, A, t, f_{e^1,G_2}, \gamma)$. This second training adjusts the models e^2, d^2 and t_s^2 and learns a new value function v_s^2 from zero, that is specific for the new goal.

Experiments show that model-free RL algorithms are able to converge to the new goal using the reward f_{e^1,G_2} that is based on the encoder learned by our algorithm for the old goal G_1. Furthermore, if we use our algorithm instead, the convergence is faster, since the planner can re-use also the transition model learned for the old task.

Experiments

We evaluated our approach on different continuous-state problems of the OpenAI Gym suite. The problems faced range from under-actuated robot control problems such as CartPole and Acrobot, continuous problems subject to scattered rewards such as Lunar Lander and Atari games such as Space Invaders. In the selected environments we compare our model-based algorithm with model-free algorithms: (1) a Vanilla Double Deep Q-Network (DDQN), (2) a DDQN algorithm that uses our symbolic Q-model based on propositional symbolic representation as Q-network, called in plots as ‘Encoded DDQN’ (3) a DDQN algorithm that uses our symbolic Q-model and learn also the transition-model on the symbolic space *without using it for planning*, that is labeled in plots as ‘Encoded DDQN with transition network’.

We remark that the symbolic representation learned by the encoded DDQN is different by the one learned by the encoded DDQN with the transition network, since also the learning of the transition network influences the representation learning.

Setup

All the experiments have been conducted on an Intel i7 processor and an Nvidia 1060 GPU. The chosen setting for the DDQN is $lr = 0.001$, $\gamma = 0.99$, the number of hidden layers is 4, and the number of units for each hidden layer is 256. When not differently indicated, the number of symbols used to represent the state is 200. The experiments have been conducted ten times for each environment and approach. Figure 4 show mean values (solid line) and standard deviation (shaded area) of the results obtained.

Training performance

In this section we compare the planning system’s performance with those of proven Deep Reinforcement Learning approaches, to demonstrate the persistence of sample efficiency nevertheless the use of the symbolic code. The results obtained by the system on the test environments demonstrate its ability to solve different types of continuous and dynamic environments as: Control problems of underactuated systems (CartPole and AcroBot); continuous and dynamic control problems with sparse reward (Lunar Lander); and problems with adversaries (Space Invaders).

CartPole and Acrobot We address the problem of controlling under-actuated systems, like the Cartpole and Acrobot Gym environments, using our online symbolic planner. The CartPole environment requires the agent to balance a pole connected to a motorized car. The goal is to keep the pole at an angle between +15° and -15° using the cart's movement. The state definition of the environment is: [p, $\dot{p}$, θ, $\dot{\theta}$], where p is the cart position and θ is the pole angle with respect to the cart. In the Acrobot environment, the agent can act only on the intermediate joint of a two revolute joints planar arms. The two links are initially hanging downwards, the goal is to reach a certain height with the end of the second link. The state definition of the Acrobot is so composed: [$\cos(\theta_1)$, $\sin(\theta_1)$, $\cos(\theta_2)$, $\sin(\theta_2)$, $\dot{\theta_1}$, $\dot{\theta_2}$], where θ_1 is the first joint angle and θ_2 is the second one. In both environments a code consisting of 200 propositional symbols have been used. Figures 4(a) and 4(c) show the results of our algorithm applied to the CartPole and Acrobot environments respectively. Our model-based algorithm reaches the desired average reward in a number of episodes comparable to the model-free methods DDQN and encoded DDQN, despite it has to learn much more information in order to plan appropriately, since the planner is composed by four neural networks, while the other model-free algorithms need to learn only the Q-network. In the Acrobot experiment we can notice a quick improvement of the mean reward around the 100-th episode, that is probably when the planner has reached enough information to start to properly plan the agent actions.

Space Invaders The last environment proposed is an Atari Game: Space Invaders. The game RAM representation has been taken as input state. The state is therefore composed of 128 variables representing not intelligible task features. Figure 4(d) presents the comparison between our approach, Vanilla DDQN, and DDQN with encoded states. In this case, systems showed similar behavior, both in the number of episodes necessary to reach the average reward and the overall reward function's overall performance.

Code abstraction and generalization

We have evaluated how the system performance vary setting different sizes of the symbolic code in the Lunar Lander environment.

Figure 4(e) shows the obtained results. Results show clearly that properly setting the symbol-set size is critical, since insufficiently large symbol-sets prevent the system to converge. In the case of Lunar Lander, convergence to the desired final reward of 100 is achieved only with code sizes grater or equal to 100. Furthermore, there is a relation between the number of used symbols and the training sample efficiency, as shown in the increasing performance between the use of 100, 200, and 400 symbols.

We also have conducted experiments in using the planning domain acquired in one MDP to perform a different task in the same environment. We take as reference environment Lunar Lander. First we have trained a planning domain over 200 symbols reaching the original goal expressed by the reward, namely safely landing the spacecraft in the center of the ground. Then we have re-trained the planning models with the reward based on the encoder previously trained, expressed by Equation 17, whit the new goal equal to (0,1,0,0,0,0,0,0); hence, we want the spacecraft to stay straight in the center of the screen with zero angular and linear velocity. Figure 4(f) shows the comparative results of the planner and a Vanilla DDQN trained on the new reward. The use of the already trained encoding speeds up the learning of our system compared to a Vanilla DDQN. This demonstrates that the system creates encodings sufficiently generic to allow the execution of multiple tasks in the same environment.

Conclusions

In conclusion, this paper proposes an interactive learning algorithm inspired by both planning and Reinforcement Learning, for automatically learning a symbolic planning domain from a continuous-state MDP. The data gained by experience in the MDP are used to train online four neural network models, e^{net}, d^{net}, t_{Δ}^{net} and Q^{net}. These models allow for fast symbolic online planning over a finite horizon at each interaction step. The planner can reason only with the automatically grounded symbolic representation, without the need for complex trajectory propagation and evaluation in the original continuous state space. We have shown that reasoning in the symbolic space is enough to effectively guide the agent's action in the original continuous environment to achieve the task expressed by the MDP reward function. We started analyzing the use of the learned symbolic domain as a *complete* domain, namely a domain that can be used to plan a sequence of actions for any possible couple initial state-final state in the original state space. We have shown that the planning domain can be *retrained* so to achieve different goals in the same environment through a reward shaping based on the found symbolic representation. As a future direction, we aim at focusing on the domain re-usability trying to speed-up or even eliminate the second training necessary for achieving a new goal. In this regard, we remark that learning an uncertainty model for the transitions would be high beneficial, especially if we want to use the symbolic model offline and to plan towards goal states that have been little explored during training.

References

Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34(6): 26–38.

Asai, M. 2019. Unsupervised grounding of plannable first-order logic representation from images. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 583–591.

Asai, M.; and Fukunaga, A. 2017. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154* .

Asai, M.; and Kajino, H. 2019. Towards Stable Symbol Grounding with Zero-Suppressed State AutoEncoder. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 592–600.

Asai, M.; and Muise, C. 2020. Learning Neural-Symbolic Descriptive Planning Models via Cube-Space Priors: The Voyage Home (to STRIPS). *arXiv preprint arXiv:2004.12850* .

Bonet, B.; and Geffner, H. 2019. Learning first-order symbolic representations for planning from the structure of the state space. *arXiv* arXiv–1909.

Corneil, D.; Gerstner, W.; and Brea, J. 2018. Efficient model-based deep reinforcement learning with variational state tabulation. *arXiv preprint arXiv:1802.04325* .

Creswell, A.; White, T.; Dumoulin, V.; Arulkumaran, K.; Sengupta, B.; and Bharath, A. A. 2018. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine* 35(1): 53–65.

Dittadi, A.; Drachmann, F. K.; and Bolander, T. 2020. Planning From Pixels in Atari With Learned Symbolic Representations .

François-Lavet, V.; Bengio, Y.; Precup, D.; and Pineau, J. 2019. Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3582–3589.

Garnelo, M.; Arulkumaran, K.; and Shanahan, M. 2016. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518* .

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language .

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Jang, B.; Kim, M.; Harerimana, G.; and Kim, J. 2019. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access* PP: 1–1. doi:10.1109/ACCESS.2019.2941229.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. URL https://arxiv.org/abs/1611.01144.

Kurutach, T.; Tamar, A.; Yang, G.; Russell, S. J.; and Abbeel, P. 2018. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, 8733–8744.

Moerland, T. M.; Broekens, J.; and Jonker, C. M. 2020. Model-based Reinforcement Learning: A Survey.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287. Morgan Kaufmann.

Steels, L. 2008. The Symbol Grounding Problem has been solved. So what's next? *Symbols, embodiment and meaning. Academic Press, New Haven* URL http://www.csl.sony.fr/downloads/papers/2007/steels-07a.pdf.

Suárez-Hernández, A.; Segovia-Aguas, J.; Torras, C.; and Alenyà, G. 2020. STRIPS Action Discovery. *arXiv preprint arXiv:2001.11457* .

Zhu, Z.; Lin, K.; and Zhou, J. 2020. Transfer Learning in Deep Reinforcement Learning: A Survey.