Deep Planning – Deep Hierarchical Planning

Hankz

Abstract

Introduction

In AI planning community, planing systems aim to synthesize solutions to planning problems given domain models and initial and goal states as input. Despite the success of previous planning systems, most of them are based on the prior knowledge of domain models, represented by PDDL (Fox and Long 2003) for example, and symbolic description of states, represented by sets of propositions for example. Creating domain models and representing states with propositions (e.g., *on(A, B)*[1]), however, are both difficult and tedious, particularly in complex planning domains, e.g., real-time strategy games (Cowling et al. 2013; Buro 2003; Ontañón and Buro 2015), where although planning techniques have successfully been explored to solve specific tasks, they all require domain models provided as input and states (e.g. initial and goal states) are represented by logical symbols. There are also learning systems that aim to learn domain models from historical plan traces (), which likewise assume states are presented by logical symbols such as propositions.

In real-world applications, it is more practical for AI systems to acquire state information from raw data, such as images, instead of requiring human experts describe states with logical symbols. For example, in real-time strategy games, add pictures here ; in

The challenges are two folds. The first is how to effectively extract valuable information from raw data to represent real-world states. The second is how to bridge the connections between states and actions.

In this paper, we aim at learning Q-value function from historical data with deep representations of states, and exploit the Q-function to do planning. We call our approach `DeepPlanning`, which stands for learning Q-functions from **Deep** representations of states for **Planning**.

[1]In the *blocksworld* domain, the proposition *on(A, B)* indicates block A is on block B.

Related Work

Deep reinforcement learning
planning
action model learning
(Mikolov et al. 2013)

Problem Formulation

A planning problem can be defined by a triple $\langle s_0, g, \mathcal{A}\rangle$, where s_0 is an initial state composed of a set of propositions, g is a goal state. $\mathcal{A}$ is a set of action models, each of which is composed of a quadruple $\langle a, PRE, ADD, DEL\rangle$, where a is an action name with a list of parameters, PRE is a list of preconditions indicating the preconditions should be satisfied when action a is executed, ADD is a list of *adding* effects indicating the set of effects should be added after action a is executed, DEL is a list of *deleting* effects indicating the effects should be deleted after action a is executed. Note that we consider STRIPS action models in this paper for the sake of simplicity. Our approach presented below is extendable to general PDDL planning models. A solution to a planning problem is called a plan, denoted by $\langle a_1, a_2, \ldots, a_k\rangle$, which transits the initial state s_0 to goal g.

Since action models $\mathcal{A}$ are generally unknown in real world applications, in this paper we aim to consider the planning problem with only an initial state s_0 and goal g available. Specifically, given as input a set of pairs of initial states and goals $\langle s_0, g\rangle$, our approach learns a Q-value function (modeled by a deep network) by continuously executing actions to transit states s_0 to g, and leverage the learnt Q-value function to build action sequences (or plans) for solving new planning problems. Note that, different from planning problems in planning community, initial states and goals in this paper can be represented by images, i.e., an initial state (or a goal) is depicted by an image instead of a set of propositions.

Approach

An overview of our deep planning approach is shown in Algorithms 1 and 2.

Deep Reinforcement Learning

We consider the standard reinforcement learning setting where an agent interacts with an environment ϵ over a number of discrete time steps. At each time step t, the agent

receives a state s_t and selects an action a_t from some set of possible actions $\mathcal{A}$ according to its policy π, where π is a mapping from states s_t to actions a_t. In return, the agent receives the next state s_{t+1} and receives a scalar reward r_t. The process continues until the agent reaches a terminal state after which the process restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total accumulated return from time step t with discount factor $\gamma \in (0, 1]$. The goal of the agent is to maximize the expected return from each state s_t.

The action value $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a]$ is the expected return for selecting action a in state s and following policy π. The optimal value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ gives the maximum action value for state s and action a achievable by any policy. Similarly, the value of state s under policy π is defined as $V^\pi(s) = \mathbb{E}[R_t | s_t = s]$ and is simply the expected return for following policy π from state s.

In value-based model-free reinforcement learning methods, the action value function is represented using a function approximate, such as a neural network. Let $Q(s, a; \theta)$ be an approximate action-value function with parameters θ. The updates to θ can be derived from a variety of reinforcement learning algorithms. One example of such an algorithm is Q-learning, which aims to directly approximate the optimal action value function:

$$Q(s, a; \theta) \approx Q^*(s, a). \tag{1}$$

In one-step Q-learning, the parameters θ of the action value function $Q(s, a; \theta)$ are learned by iteratively minimizing a sequence of loss functions, where the ith loss function defined as

$$\mathcal{L}(\theta_i) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2] \tag{2}$$

where s' is the state encountered after state s.

Deep Planning

To solve our planning problem, we need to consider "goals" in the Q-learning framework. We extend the Q-value function to consider goals by replacing $Q(s, a)$ with $Q(s, a, g)$. We thus reformulate Equation (1) as shown below:

$$Q(s, a, g, \theta) \approx Q^\pi(s, a, g). \tag{3}$$

As a result, the objective function (similar to Equation (5)) by mean-squared error in Q-values can be defined by,

$$\mathcal{L}(\theta_i) = \mathbb{E}[\underbrace{(r + \gamma \max_{a'} Q(s', a', g, \theta_{i-1})}_{target} - Q(s, a, g, \theta_i))^2]. \tag{4}$$

Deep Hierarchical Planning

Given an initial state s_0 and a goal g, there are a set of tasks with partial orders to be accomplished and transit s_0 to g provided that the planning problem is solvable. Thus, the first phase is to select tasks sequentially. Our high-level idea is to iteratively select tasks based on current state s (which is initialized to be s_0) and update s to a new state after accomplishing the selected tasks. To do this, we define a Q-value function $Q^\pi(\{s, g\}, \tau) = \mathbb{E}[R_t | s_t = s, g, \tau]$, where π is a policy defined as a mapping from $\{s, g\}$ to τ, τ is a task and $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the accumulated return from time step t with discount factor $\gamma \in (0, 1]$, where r_{t+k} is a scalar reward. The Q-value function $Q^\pi(\{s, g\}, \tau)$ is the expected return for selecting τ in state-pair $\{s, g\}$ by following policy π. The optimal Q-value function can be defined by $Q^*(\{s, g\}, \tau) = \max_\pi Q^\pi(\{s, g\}, \tau)$, which gives the maximum value for state-pair $\{s, g\}$ and task τ achievable by any policy.

Due to the difficulty of optimizing the Q-value function $Q^*(\{s, g\}, \tau)$, we exploit a neural network with parameter θ $Q(s, g, \tau; \theta)$ to approximate the $Q^*(\{s, g\}, \tau)$, i.e.,

$$Q(s, g, \tau; \theta) \approx Q^*(\{s, g\}, \tau).$$

In one-step Q-learning, the parameters θ of the Q-value function $Q(s, g, \tau; \theta)$ are learned by iteratively minimizing a sequence of loss functions $\mathcal{L}(\theta)$, where the ith loss function $\mathcal{L}(\theta)$ defined by

$$\mathcal{L}(\theta_i) = \mathbb{E}[(r + \gamma \max_{\tau'} Q(s', g, \tau'; \theta_{i-1}) - Q(s, g, \tau; \theta_i))^2] \tag{5}$$

where s' is the state encountered after state s.

The second phase is to select subtasks τ^{sub} (or primitive tasks, i.e., actions) to accomplish a given task t based on current state s. Analogy to the first phase, we define an Q-value function $Q^*(\{s, \tau\}, \tau^{sub})$ and approximate the Q-value function with a neural network with parameters δ $Q(s, \tau, \tau^{sub}; \delta)$, i.e.,

$$Q(s, \tau, \tau^{sub}; \delta) \approx Q^*(\{s, \tau\}, \tau^{sub}).$$

the parameters δ of the Q-value function $Q(s, \tau, \tau^{sub}; \delta)$ are learned by iteratively minimizing a sequence of loss functions $\mathcal{L}(\delta)$, where the ith loss function $\mathcal{L}(\delta_i)$ defined by

$$\mathcal{L}(\delta_i) = \mathbb{E}[(r + \gamma \max_{\tau^{sub'}} Q(s', \tau, \tau^{sub'}; \delta_{i-1}) - Q(s, \tau, \tau^{sub}; \delta_i))^2] \tag{6}$$

where s' is the state encountered after state s.

The learning procedure

Algorithm 1 Our Deep Hierarchical Planning: Learning Phase

input: a set of state pairs for training: $\mathcal{D} = \{\langle s_0, g \rangle\}$
output: parameters θ for the Q-value function network

1: Initialize experience replay memories $\{D_1, D_2\}$ and parameters $\{\theta_1, \theta_2\}$ for selecting task and subtasks, respectively.
2: Initialize exploration probability ϵ
3: **for** i=1, episodes **do**
4: Initialize planning problem and get start state s
5:
6: **end for**
7: sample $(s, t, t', r) \sim \mathcal{D}$

Algorithm 2 Our Deep Planning: Planning Phase

input: a set of state pairs for testing: $\mathcal{T} = \{\langle s_0, g\rangle\}$, Q-value network

output: Solution plans: $\mathcal{P}$

1:

Testing

The theory of reinforcement learning provides a normative account, deeply rooted in psychological and neuroscientific perspectives on animal behaviors, of how angels may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and uses these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems, the former evidenced by a wealth of neural data revealing notable parallels between the phasic signals emitted by dopaminergic neurons and temporal difference reinforcement learning algorithms. While reinforcement learning agents have achieved some successes in a variety of domains, their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. Here we use recent advances in training deep neural networks to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. We tested this agent on the challenging domain of classic Atari 2600 games. We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters. This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.

We set out to create a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks – a central goal of general artificial intelligence that has eluded previous efforts. To achieve this, we developed a novel agent, a deep Q-network, which is able to combine reinforcement learning with a class of artificial neural network known as deep neural networks. Notably, recent advances in deep neural networks, in which several layers of nodes are use to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data. We use one particularly successful architecture, the deep convolutional network, which uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields – inspired by Hubel and Wiesel's seminal work on feedforward processing in early visual cotex–thereby exploiting the local spatial correlations present in images, and building in robustness to natural transformations such as changes of viewpoint or scale.

We consider tasks in which the agent interacts with an environment through a sequence of observations, actions and rewards. The goal of the agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{r+2} + \ldots | s_t = s, a_t = a, \pi],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behavior policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a).

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximate such as a neural network is used to represent the action-value (all known as Q) function. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed experience replay that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. Second, we used an iterative update that adjusts the action-values towards target values that are only periodically updated, thereby reducing correlations with the target.

While other stable methods exist for training neural networks in the reinforcement learning setting, such as neural fitted Q-iteration, these methods involve the repeated training of networks de novo on hundreds of iterations. Consequently, these methods, unlike our algorithm, are too inefficient to be used successfully with large neural networks. e parameterize an approximate value function $Q(s, a; \theta_i)$ using the deep convolutional neural network shown in Fig. 1, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i. To perform experience replay we store the agent's experiences.

Experiments

References

Buro, M. 2003. Real-time strategy games: A new AI research challenge. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 1534–1535.

Cowling, P. I.; Buro, M.; Bída, M.; Botea, A.; Bouzy, B.; Butz, M. V.; Hingston, P.; Muñoz-Avila, H.; Nau, D. S.; and Sipper, M. 2013. Search in real-time video games. In *Artificial and Computational Intelligence in Games*. 1–19.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.

Ontañón, S., and Buro, M. 2015. Adversarial hierarchical-task network planning for complex real-time games. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1652–1658.