POLITECNICO DI MILANO
Master's Degree in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

FAST REINFORCEMENT LEARNING USING DEEP STATE-ACTION FEATURES

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Supervisor: Prof. Marcello Restelli
Co-supervisor: Matteo Pirotta, Ph.D.

Master's Thesis by:
Daniele Grattarola (student ID 853101)

Academic Year 2016-2017

Contents

Chapter 1

Background

Introduce the section

1.1 Deep Learning

Deep Learning (DL) is a branch of machine learning which exploits *Artificial Neural Networks* (ANN) with more than one hidden layer to learn an abstract representation of the input space [1].

Deep learning techniques can be applied to the three main classes of problems of machine learning (supervised, semi-supervised, and unsupervised), and have been used to achieve state-of-the-art results in a variety of learning tasks.

1.1.1 Artificial Neural Networks

Feed-forward Artificial Neural Networks (ANN) are function approximators inspired by the connected structure of neurons and synapses in the animal brain.

1.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of ANN inspired by the visual cortex in animal brains. CNNs exploit spatially-local correlations in the neurons of adjacent layers through the use of a *receptive field*, a set of weights which is used to transform a local subset of the input neurons of a layer.

1.1.3 Autoencoders

Autoencoders are a type of ANN which are used to learn a sparse and compressed representation of the input space, by sequentially compressing and reconstructing the inputs under some sparsity constraint (typically by minimizing the $L1$ norm of the activations in the innermost hidden layers).

1.2 Reinforcement Learning

1.2.1 Markov Decision Processes

Markov Decision Processes (MDP) are discrete-time, stochastic control processes, that can be used to describe the interaction of an *agent* with an *environment*.

Formally, MDPs are defined as 7-tuples $(S, S^t, A, P, R, \gamma, \mu)$, where:

- S is the set of observable states of the environment.
  When the set observable states coincides with the true set of states of the environment, the MDP is said to be *fully observable*. We will only deal with fully observable MDPs without considering the case of *partially observable* MDPs.
- $S^T \subseteq S$ is the set of *terminal states* of the environment, meaning those states in which the interaction between the agent and the environment ends. A sequence of states observed by an agent during an interaction with the environment and ending in a terminal state is usually called an *episode*.
- A is the set of actions that the agent can execute in the environment.
- $P : S \times A \times S \to [0, 1]$ is a *state transition function* which, given two states $s, s' \in S$ and an action $a \in A$, represents the probability of the agent going to state s' by executing a in s.
- $R : S \times A \to \mathbb{R}$ is a *reward function* which represents the reward that the agent collects by executing an action in a state.
- $\gamma \in (0, 1)$ is a *discount factor* with which the rewards collected by the agent are diminished at each step, and can be interpreted as the agent's interest for rewards further in time rather than immediate.
- $\mu : S \to [0, 1]$ is a probability distribution over S which models the probability of starting the exploration of the environment in a given state.

Episodes are usually represented as sequences of tuples

$$[(s_0, a_0, r_0, s_1), ..., (s_{n-1}, a_{n-1}, r_{n-1}, s_n)]$$

called *trajectories*, where (s_i, a_i, r_i, s_{i-1}) represent a transition of the agent to state s_{i+1} by taking action a_i in s_i and collecting a reward r_i, and $s_n \in S^T$.

In Markov Decision Processes, the modeled environment must satisfy the *Markov property*, meaning that the reward and transition functions of the environment must only depend on the current state and action, rather than the past state-action trajectory of the agent.

In other words, an environment is said to satisfy the Markov property when its one-step dynamics allow to predict the next state and reward given only the current state and action.

Policy

The behavior of the agent in an MDP can be defined as a probability distribution $\pi : S \times A \to [0, 1]$ called a *policy*, which given $s \in S, a \in A$, represents the probability of selecting a as next action from s.

An agent which uses this probability distribution to select its next action when in a given state is said to be *following* the policy.

Value Functions

Starting from the concept of policy, we can now introduce a function that evaluates how good it is for an agent following a policy π to be in a given state. This evaluation is expressed in terms of the expected return, i.e. the expected discounted sum of future rewards collected by an agent starting from a state while following π, and the function that computes it is called the *state-value function for policy* π (or, more commonly, simply *value function*).

Formally, the state-value function associated to a policy π is a function $V^\pi : S \to \mathbb{R}$ defined as:

$$V^\pi(s) = E_\pi[R_t | s_t = s] \tag{1.1}$$

$$= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \tag{1.2}$$

where $E_\pi[\cdot]$ is the expected value given that the agent follows policy π, and t is any time step of an episode $[s_1, ..., s_t, ..., s_n]$ where $s_t \in S, \forall t = 1, ..., n$.

Similarly, we can also introduce a function that evaluates the goodness of taking a specific action in a given state, namely the expected reward obtained by taking an action $a \in A$ in a state $s \in S$ and then following

policy π. We call this function the *action-value function for policy* π denoted $Q^\pi : S \times A \to \mathbb{R}$, and defined as:

$$Q^\pi(s,a) = E_\pi[R_t | s_t = s, a_t = a] \tag{1.3}$$

$$= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \tag{1.4}$$

The majority of reinforcement learning algorithms is based on computing (or estimating) value functions to then control the behavior of the agent in order to maximize the expected reward collected during episodes.

We also note a fundamental property of value functions, which satisfy particular recursive relationships like the following *Bellman equation for* V^π:

$$\begin{aligned} V^\pi(s) &= E_\pi[R_t | s_t = s] \\ &= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \\ &= E_\pi[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s] \qquad (1.5) \\ &= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s,a,s')[R(s,a) + \\ &\quad + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s']] \qquad (1.6) \\ &= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s,a,s')[R(s,a) + \gamma V^\pi(s')] \qquad (1.7) \end{aligned}$$

Intuitively, relation (1.7) decomposes the state-value function as the sum of the immediate reward collected from a state s to a successor state s', and the value of s' itself; by considering the transition model of the MDP and the policy being followed, we see that the Bellman equation simply averages the expected return over all the possible (s, a, r, s') transitions, by taking into account the probability that these transitions occur.

1.2.2 Optimal Value Functions

In general terms, *to solve* a reinforcement learning task is to identify a policy that yields a sufficiently high expected return. In the case of MDPs with finite state and actions sets[1], it is possible to define the concept of *optimal policy* as the policy which maximizes the expected return collected by the agent in an episode.

[1]This specification is only required for formality, but is not expanded further in this work. Refer to SUTTON, BARTO for more details on the subjet of non-finite MDPs.

We start by noticing that state-value functions define a partial ordering over policies as follows:

$$\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$$

From this, the *optimal policy* π^* of an MDP is a policy which is better or equal than all other policies in the policy space.

The state-value function associated to π^* is called the *optimal state-value function*, denoted V^* and defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$$

As we did when introducing the value functions, given an optimal policy for the MDP it is also possible to define the *optimal action-value function* denoted Q^* (equivalence (1.9) in this definition highlights the relation between Q^* and V^*):

$$\begin{aligned} Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a) && (1.8)\\ &= E[r_{t+1} + \gamma V^*(s_{t+1}|s_t = s, a_t = a] && (1.9) \end{aligned}$$

.

Since V^* and Q^* are value functions of an MDP, they must satisfy the same type of recursive relations that we described in (1.7), in this case called the *Bellman optimality equations*.

The Bellman optimality equation for V^* expresses the fact that the value of a state associated to an optimal policy must be the expected return of the best action that the agent can take in that state:

$$\begin{aligned} V^*(s) &= \max_{a} Q^*(s,a) && (1.10)\\ &= \max_{a} E_{\pi^*}[R_t|s_t = s, a_t = a] && (1.11)\\ &= \max_{a} E_{\pi^*}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a] && (1.12)\\ &= \max_{a} E_{\pi^*}[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_t + k + 2|s_t = s, a_t = a] && (1.13)\\ &= \max_{a} E_{\pi^*}[r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a] && (1.14)\\ &= \max_{a} \sum_{s' \in S} P(s,a,s')[R(s,a) + \gamma V^*(s')] && (1.15) \end{aligned}$$

The Bellman optimality equation for Q^* is again obtained from the definition as:

$$\begin{aligned} Q^*(s,a) &= E[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')||s_t = s, a_t = a] && (1.16)\\ &= \sum_{s'} P(s,a,s')[R(s,a) + \gamma \max_{a'} Q^*(s', a')] && (1.17) \end{aligned}$$

It is possible to notice that both Bellman optimality equations have a unique solution independent of the policy. If the dynamics of the environment (R and P) are fully known, it is possible to solve the system of equations associated to the value functions (i.e. one equation for each state in S) and get an exact value for V^* and Q^* in each state.

1.2.3 Value-based optimization

One of main algorithm classes for solving reinforcement learning problems is based on searching an optimal policy for the MDP by trying to compute either of the optimal value functions, and then deriving a policy based on them.

From V^* or Q^*, it is easy to determine an optimal policy:

- Given V^*, for each state $s \in S$ there will be an action which maximizes the Bellman optimality equation (1.10). Any policy that assigns positive probability to only this action (or actions) is an optimal policy. This approach therefore consists in performing a one-step forward search on the state space to determine the best action from the current state.

- Given Q^*, the optimal policy is that which assigns positive probability to the action which maximizes $Q^*(s, a)$; this approach exploits the intrinsic property of the action-value function of representing the *goodness* of actions, without performing the one-step search on the successor states.

In this section we will describe some of the most important value-based approaches to reinforcement learning, which will be useful in the following sections of this work.

We will not deal with equally popular methods like *policy gradient* or *actor-critic* approaches, even though they have been successfully applied in conjunction with deep learning to solve complex environments (see 1.3 and **??**).

1.2.4 Dynamic Programming

The use of dynamic programming techniques to solve a reinforcement learning problem is based on recursively applying some form of the Bellman equation, starting from any initial policy π until convergence to π^*.

In this class of algorithms, we identify two main approaches: *policy iteration* and *value iteration*.

Policy iteration is based of the following theorem:

Theorem 1 (Policy improvement theorem) *Let π and π' be a pair of deterministic policies such that*

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s), \forall s \in S$$

Then, $\pi' \geq \pi$, i.e.

$$V^{\pi'}(s) \geq V^{\pi}(s), \forall s \in S$$

This approach works by iteratively computing the value functions associated to the current policy, and then improving that policy by making it act greedily with respect to the value functions, such that:

$$\pi'(s) = \arg\max_{a \in A} Q^{\pi}(s, a) \tag{1.18}$$

which, for Theorem 1, improves the expected return of the policy since:

$$Q^{\pi}(s, \pi'(s)) = \max_{a \in A} Q^{\pi}(s, a) \geq Q^{\pi}(s, \pi(s)) = V^{\pi}(s)$$

.

This continuous improvement is applied until the inequality in the previous equation becomes an equality, i.e. when the improved policy satisfies the Bellman optimality equation (1.10). Since the algorithm gives no assurances on the number of updates required for convergence, some stopping conditions are usually introduced to end the process when the new value function did not change substantially after the update (ϵ-*convergence*) or a certain threshold number of iterations has been reached.

Starting from a similar basis, the *value iteration* approach computes the value function associated to an initial policy, but then applies a contraction operator which iterates over sequentially better value functions without actually computing the associated greedy policy.

The contraction operator which ensures convergence is the *Bellman optimality backup*:

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} P(s, a, s')[R(s, a) + \gamma V(s')] \tag{1.19}$$

Similarly to policy iteration, convergence is ensured but without guarantees on the number of steps, and therefore it usual to terminate the iteration according to some stopping condition.

1.2.5 Monte Carlo Methods

Dynamic programming approaches exploit the exact solution of a value function which can be computed starting from a policy, but in general this requires to have a perfect knowledge of the environment's dynamics an may also not be tractable on sufficiently complex MDPs.

Monte Carlo methods are a way of solving reinforcement learning problems by only using *experience*, i.e. a collection of *sample trajectories* from an actual interaction of an agent in an environment.

This is often referred to as a *model-free* approach because, while the environment (or a simulation thereof) is still required to observe the sample trajectories, it is not necessary to have an exact knowledge of the transition model and reward function of the MDP.

Despite the differences with dynamic programming, this approach is still centered on the same two-step process of policy evaluation and improvement.

To estimate the value of a state $V^\pi(s)$ under a policy π with Monte Carlo methods, it is sufficient to consider a set of episodes collected under π.

The value of the state s will be computed as the average of the returns collected following a *visit* of the agent to s, for all occurrences of s in the collection [2].

This same approach can be also used to estimate the action-value function, simply by considering the occurrence of state-action pairs in the collected experience rather than states only.

Finally, the policy is improved by computing its greedy variation (1.18) with respect to the estimated value functions and the process is iteratively repeated until convergence, with a new set of trajectories collected under each new policy.

1.2.6 Temporal Difference Learning

SARSA

Q-learning

1.2.7 Fitted Q-Iteration

1.3 Deep Reinforcement Learning

1.3.1 Deep Q-Learning

[2] This is not always true: a variation of this algorithm exists, which only considers the average returns following the *first* visit to a state in each episode.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.