Review Article

Kidney Diseases

Kidney Dis 2019;5:18–22
DOI: 10.1159/000492670

Received: August 3, 2018
Accepted: August 6, 2018
Published online: October 12, 2018

Deep Reinforcement Learning in Medicine

Anders Jonsson

Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain

Keywords
Artificial intelligence · Reinforcement learning · Deep learning

Abstract
Reinforcement learning has achieved tremendous success in recent years, notably in complex games such as Atari, Go, and chess. In large part, this success has been made possible by powerful function approximation methods in the form of deep neural networks. The objective of this paper is to introduce the basic concepts of reinforcement learning, explain how reinforcement learning can be effectively combined with deep learning, and explore how deep reinforcement learning could be useful in a medical context.

© 2018 S. Karger AG, Basel

Introduction

Reinforcement learning (RL) algorithms have experienced unprecedented success in the last few years, reaching human-level performance in several domains, including Atari video games [1] or the ancient games of Go [2] and chess [3]. This success has been largely enabled by the use of advanced function approximation techniques in combination with large-scale data generation from self-play games. The aim of this paper is to introduce basic RL algorithms and describe state-of-the-art extensions of these algorithms to deep learning. We also discuss the potential of RL in medicine and review the literature to study existing practical applications of RL. Even though RL offers several benefits in comparison to other artificial intelligence (AI) techniques, such as the ability to optimize long-term benefit to patients rather than immediate benefit, there are a number of obstacles that have to be overcome in order to apply RL on a large scale.

Reinforcement Learning

RL is an area of machine learning concerned with sequential decision problems [4]. Concretely, a learner or *agent* interacts with an *environment* by taking actions, and the aim of the agent is to maximize its expected cumulative reward. Each action affects the next, and the agent cannot simply maximize the immediate reward that it will get, but rather has to plan ahead and select actions that will maximize reward in the long run.

Notation: Given a finite set X, a probability distribution on X is a vector $\mu \in \mathbb{R}^X$ whose elements are nonnegative (i.e., $\mu(x) \geq 0$ for each $x \in X$) and whose sum equals 1 (i.e., $\Sigma_x \mu(x) = 1$). We use $\Delta(X) = \{\mu \in \mathbb{R}^X : \Sigma_x \mu(x)$

Contribution from the 2nd meeting of "Science for Dialysis," organized at the University Hospital of Bellvitge, L'Hospitalet de Llobregat, Barcelona, Spain, on September 28, 2018.

KARGER

© 2018 S. Karger AG, Basel

E-Mail karger@karger.com
www.karger.com/kdd

Anders Jonsson
Department of Information and Communication Technologies
Universitat Pompeu Fabra, Roc Boronat 138
ES–08018 Barcelona (Spain)
E Mail:anders.jonsson@upf.edu

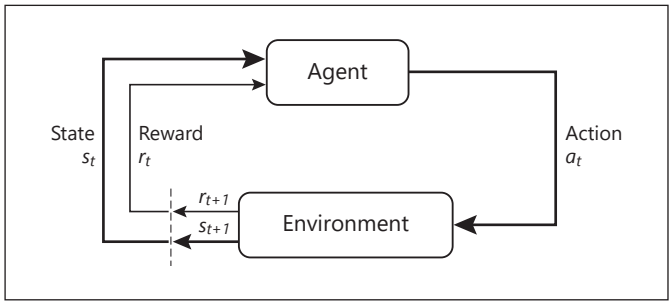

Fig. 1. Illustration of the agent-environment interface [4].

$= 1, \mu(x) \geq 0\ (\forall x)\}$ to denote the set of all such probability distributions.

Markov Decision Processes

Sequential decision problems are usually modelled mathematically as Markov decision processes, or MDPs. An MDP is a tuple $M = (S, A, P, r)$, where

- S is the finite state space,
- A is the finite action space,
- $P : S \times A \to \Delta(S)$ is the transition function, with $P(s' \mid s, a)$ denoting the probability of moving to state s' when taking action a in state s,
- $r : S \times A \to \mathbb{R}$ is the reward function, with $r(s, a)$ denoting the expected reward obtained when taking action a in state s.

Intuitively, the agent controls which action to select, while the environment controls the outcome of each action. In each round t, the agent observes a state $s_t \in S$ and selects an action $a_t \in A$. As a result, the environment returns a new state $s_{t+1} \sim P(\bullet \mid s_t, a_t)$ and reward $r_{t+1} \sim r(s_t, a_t)$. This process is illustrated in Figure 1. By repeating the procedure for $t = 0, 1, 2, \ldots$ the result is a *trajectory*

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots.$$

The aim of the agent is to select actions as to maximize some measure of expected cumulative reward. The most common reward criterion is the *discounted cumulative reward*

$$\mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_2 + \ldots\right] = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right],$$

where $\gamma \in (0, 1]$ is a *discount factor* which ensures that the sum remains bounded.

The decision strategy of the agent is represented by a *policy* $\pi : S \to \Delta(A)$, with $\pi(a \mid s)$ denoting the probability that the agent selects action a in state s. For each policy π, we can define an associated *value function* V^π, which measures how much expected reward the agent will accumulate from a given state when acting according to π. Specifically, the value in state s is defined as

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = s\right].$$

The values of consecutive states satisfy a recursive relationship called the *Bellman equations*:

$$V^\pi(s) = \sum_a \pi(a \mid s)\left(r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s')\right).$$

As an alternative to V^π, we can instead define an *action-value function* Q^π, which measures how much expected reward the agent will accumulate from a given state when taking a specific action and then acting according to π. The action-value for state s and action a is defined as

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = s, a_0 = a\right].$$

There is a straightforward relationship between the value function V^π and action-value function Q^π:

$$V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s, a),$$

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s').$$

Consequently, the Bellman equations can be stated either for V^π or for Q^π.

We can also define the *optimal value function* V^* as the maximum amount of expected reward that an agent can accumulate from a given state. The optimal value function in state s is given by $V^*(s) = \max_\pi V^\pi(s)$, i.e., the maximum value among the individual policies. The *optimal policy* π^* is the policy that attains the maximum value in each state s, i.e., $\pi^*(\bullet \mid s) = \arg\max_\pi V^\pi(s)$. The optimal values of consecutive states satisfy the *optimal Bellman equations*

$$V^*(s) = \max_a \left(r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s')\right).$$

Just as before, we can instead define the *optimal action-value function* Q^*, which is related to V^* as follows:

$$V^*(s) = \max_a Q^*(s, a),$$

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s').$$

RL Algorithms

Most RL algorithms work by maintaining an estimate $\hat{\pi}$ of the optimal policy, an estimate $\hat{V}$ of the optimal value

function, and/or an estimate $\hat{Q}$ of the optimal action-value function. If the transition function P and reward function r are known, $\hat{\pi}$ and $\hat{V}$ can be estimated directly. Specifically, from the Bellman equations we can derive a *Bellman operator* T^{π} which can be applied to a value function $\hat{V}$ to produce a new value function $T^{\pi}\hat{V}$ defined as

$$T^{\pi}\hat{V}(s) = \sum_{a} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \hat{V}(s') \right).$$

Likewise, from the optimal Bellman equations we can derive an *optimal Bellman operator* T^* that, when applied to a value function $\hat{V}$, is defined as

$$T^{*}\hat{V}(s) = \max_{a} \left(r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \hat{V}(s') \right).$$

Value iteration works by repeatedly applying the optimal Bellman operator T^* to an initial value function estimate $\hat{V}_0$:

$$\hat{V}_{k+1} = T^{*}\hat{V}_k, \quad k = 0, 1, 2, \ldots$$

If the value of each state is stored in a table, value iteration is guaranteed to eventually converge to the optimal value function V^*.

Policy iteration instead starts with an initial policy estimate $\hat{\pi}_0$ and alternates between a policy estimation step and a policy improvement step. In the policy estimation step, we simply estimate the value function $\hat{V}^{\hat{\pi}_k}$ associated with the current policy $\hat{\pi}_k$. To do so, we can repeatedly apply the Bellman operator $T^{\hat{\pi}_k}$ to an initial value function estimate $\hat{V}_0$:

$$\hat{V}_{n+1} = T^{\hat{\pi}_k}\hat{V}_n, \quad n = 0, 1, 2, \ldots.$$

In the policy improvement step, we update the policy such that it is greedy with respect to the value function estimate $\hat{V}^{\hat{\pi}_k}$:

$$\hat{\pi}_{k+1} = \arg\max_{\pi} T^{\pi}\hat{V}^{\hat{\pi}_k}.$$

If the value of each state is stored in a table, policy iteration is also guaranteed to eventually converge to the optimal value function V^*.

If the transition function P and reward function r are unknown, we have to resort to different techniques. In this case, $\hat{\pi}$, $\hat{V}$, and $\hat{Q}$ have to be estimated from *transitions* on the form s_t, a_t, r_{t+1}, s_{t+1}. Unlike value iteration and policy iteration, which update the values of all states in each iteration, *temporal difference* (TD) methods update the value of a single state for a given transition. The most popular TD method is Q-learning [5], which maintains an estimate $\hat{Q}_t$ of the optimal action-value function, which is updated after each transition $(s_t, a_t, r_{t+1}, s_{t+1})$. The new estimate $\hat{Q}_{t+1}$ is identical to $\hat{Q}_t$ for each state-action pair different from (s_t, a_t), while $\hat{Q}_{t+1}(s_t, a_t)$ is given by

$$\hat{Q}_{t+1}(s_t, a_t) = (1 - \alpha_t)\hat{Q}_t(s_t, a_t) + \alpha_t \left(r_{t+1} + \gamma \max_{a} \hat{Q}_t(s_{t+1}, a) \right).$$

Here, α_t is a *learning rate*. If the value of each state-action pair is stored in a table and α_t is appropriately tuned, Q-learning is guaranteed to eventually converge to the optimal action-value function Q^*, even if the transition function P and reward function r are unknown.

Deep RL

In most realistic domains, the state space S is too large to store the estimated value function $\hat{V}$ in a table. In this case, it is common to *parameterize* $\hat{V}_\theta$ (or $\hat{\pi}_\theta$, $\hat{Q}_\theta$) on some parameter vector θ. The value in a state is completely determined by the current parameters in θ, and the update rules for RL algorithms are modified such that they no longer update the values of states directly, but rather the parameters in θ. In deep RL, $\hat{V}_\theta$ (or $\hat{\pi}_\theta$, $\hat{Q}_\theta$) is represented using a deep neural network, with θ being the parameters of the network. When the input is an image, a convolutional neural network is typically deployed, such as the one in Figure 2.

A deep Q network, or DQN [1], is a deep neural network that estimates the action-value function $\hat{Q}_\theta$. Given a transition s_t, a_t, r_{t+1}, s_{t+1}, the parameters θ of the neural network are updated as to minimize the Bellman error

$$r_{t+1} + \gamma \max_{a} \hat{Q}_\theta(s_{t+1}, a) - \hat{Q}_\theta(s_t, a_t).$$

In order to prevent overfitting, the algorithm performs a technique called *experience replay* in which many transitions are stored in a database. In each iteration, a number of transitions are sampled at random from the database in order to update the network parameters θ.

Asynchronous advantage actor-critic, or A3C [6], maintains both an estimate $\hat{\pi}_\theta$ of the policy (the actor) and an estimate $\hat{V}_\phi$ of the value function (the critic). Given a transition $(s_t, a_t, r_{t+1}, s_{t+1})$, the parameter vector θ of $\hat{\pi}_\theta$ is updated using the regularized policy gradient rule

$$\nabla_\theta \log \hat{\pi}_\theta(a_t \mid s_t) \hat{A}_\phi(s_t, a_t) + \beta \nabla_\theta H(\hat{\pi}_\theta(\bullet \mid s_t)),$$

where $\hat{A}_\phi(s_t, a_t)$ is an estimate of the *advantage function*

$$\hat{A}_\phi(s_t, a_t) = r_{t+1} + \gamma \hat{V}_\phi(s_{t+1}) - \hat{V}_\phi(s_t),$$

$H(\hat{\pi}_\theta(\bullet \mid s_t))$ is the *entropy* of the policy $\hat{\pi}_\theta$ in state s_t, and the parameter β controls the amount of regularization. The stability of the algorithm increases by using n-step returns, i.e., reward accumulated during n consecutive transitions. Moreover, the vectors θ and ϕ often share parameters, e.g., in a neural network setup all non-output

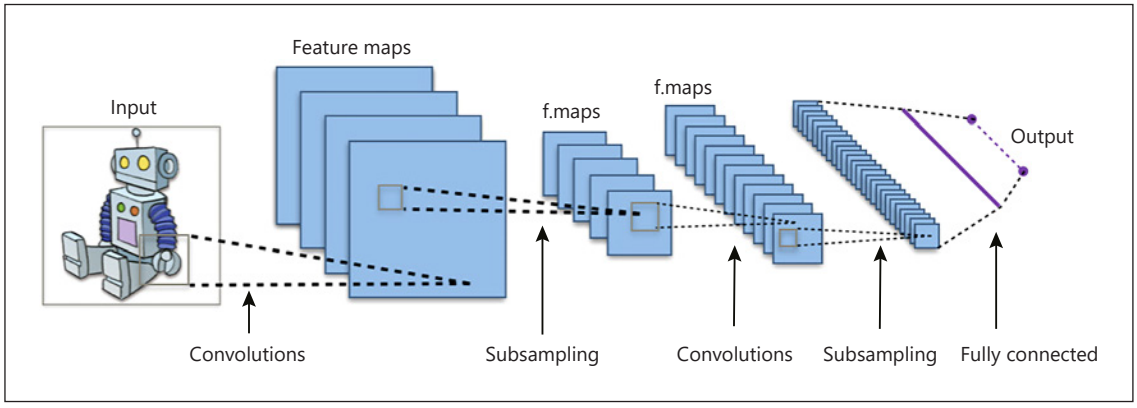

Fig. 2. Convolutional neural network. By Aphex 34 (CC BY-SA 4.0 [https://creativecommons.org/licences/by-sa/4.0]), from Wikimedia Commons.

layers are shared, and only the output layers differ for $\hat{\pi}_\theta$ and $\hat{V}_\phi$.

AlphaZero [3, 7] also maintains both a policy estimate $\hat{\pi}_\theta$ and a value estimate $\hat{V}_\phi$. The parameters are not updated using the policy gradient rule; rather, the algorithm performs Monte-Carlo tree search (MCTS) to estimate a target action distribution $p(\bullet)$ given by the empirical visitation count of each branch of the search tree (MCTS also determines which action a_t to perform next). The parameters are then updated using the loss function

$$l = (R_t - \hat{V}_\phi(s_t))^2 - p(\bullet)^T \log \hat{\pi}_\theta(\bullet \mid s_t) + \beta \| \theta \|^2,$$

where R_t is the observed return from state s_t, and β again controls the amount of regularization.

RL in Medicine

Many decision problems in medicine are by nature sequential. When a patient visits a doctor, the doctor has to decide which treatment to administer to the patient. Later, when the patient returns, the treatment previously administered affects their current state, and consequently the next decision regarding future treatment. This type of decision problem can be effectively modelled as an MDP and solved using RL algorithms.

In most AI systems implemented in medicine, the sequential nature of decisions is ignored, and the systems instead base their decisions exclusively on the current state of patients. RL offers an attractive alternative to such systems, taking into account not only the immediate effect of treatment, but also the long-term benefit to the patient.

In spite of the potential of RL in medicine, there are a number of obstacles that have to be overcome in order to apply RL algorithms in the hospital. RL algorithms typically learn by trial-and-error, but submitting patients to exploratory treatment strategies is of course not an option in practice. Instead, RL algorithms would have to learn from existing data collected using fixed treatment strategies. This process is called off-policy learning and will play an important role in practical RL algorithms, especially in a medical setting.

Another important issue is to establish what the reward should be, which in turn determines the behavior of the optimal policy. Defining an appropriate reward function involves weighing different factors against each other, such as the monetary cost of a given treatment versus the life expectancy of a patient. This dilemma is not exclusive to RL, however, and is already being discussed on a large scale in different countries.

There exist several examples of medical RL applications in the literature. RL has been used to develop treatment strategies for epilepsy [8] and lung cancer [9]. An approach based on deep RL was recently proposed for developing treatment strategies based on medical registry data [10]. Deep RL has also been used to learn treatment policies for sepsis [11].

In nephrology, the problem of anemia treatment in hemodialysis patients is particularly well-suited to model as a sequential decision-making problem. A common treatment for patients with chronic kidney disease are erythropoiesis-stimulating agents (ESAs), but the effects of ESAs are unpredictable, making it necessary to closely monitor the patient's condition. At regular time intervals, the medical team has to decide what action to take, and

in turn, this action will affect the patient's condition in the future. Several authors have proposed using RL to control the administration of ESAs [12, 13].

Acknowledgements

This work is partially funded by the grant TIN2015-67959 of the Spanish Ministry of Science.

Statement of Ethics

The author has no ethical conflicts to disclose.

Disclosure Statement

The author has no conflicts of interest to declare.

References

1 Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015 Feb;518(7540):529–33.
2 Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016 Jan;529(7587):484–9.
3 Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv. 2017;1712.01815.
4 Sutton RS, Barto AG. Introduction to Reinforcement Learning. 1st ed. Cambridge (MA): MIT Press; 1998.
5 Watkins CJ, Dayan P. Q-learning. In: Machine Learning. 1992. p. 279–92.
6 Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, et al. Asynchronous Methods for Deep Reinforcement Learning. arXiv. 2016;48:1–28.
7 Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Mastering the game of Go without human knowledge. Nature. 2017 Oct;550(7676):354–9.
8 Pineau J, Guez A, Vincent R, Panuccio G, Avoli M. Treating epilepsy via adaptive neurostimulation: a reinforcement learning approach. Int J Neural Syst. 2009 Aug;19(4):227–40.
9 Zhao Y, Zeng D, Socinski MA, Kosorok MR. Reinforcement learning strategies for clinical trials in nonsmall cell lung cancer. Biometrics. 2011 Dec;67(4):1422–33.
10 Liu Y, Logan B, Liu N, Xu Z, Tang J, Wang Y. Deep reinforcement learning for dynamic treatment regimes on medical registry data. 2017 IEEE International Conference on Healthcare Informatics (ICHI); 2017 Aug. p. 380–5.
11 Raghu A, Komorowski M, Ahmed I, Celi LA, Szolovits P, Ghassemi M. Deep reinforcement learning for sepsis treatment. CoRR. 2017;abs/1711.09602.
12 Escandell-Montero P, Chermisi M, Martínez-Martínez JM, Gómez-Sanchis J, Barbieri C, Soria-Olivas E, et al. Optimization of anemia treatment in hemodialysis patients via reinforcement learning. Artif Intell Med. 2014 Sep;62(1):47–60.
13 Martín-Guerrero JD, Gomez F, Soria-Olivas E, Schmidhuber J, Climente-Martí M, Jiménez-Torres NV. A reinforcement learning approach for individualizing erythropoietin dosages in hemodialysis patients. Expert Syst Appl. 2009;36(6):9737–42.