Reinforcement Learning for Business Modeling

Fernando S. Oliveira
ESSEC Business School, Singapore

INTRODUCTION

Inter-temporal decision making is a complex problem relevant to, among others, product differentiation (Hsu & Wang, 2004), portfolio optimization (Bae, Kim, & Mulvey, 2013; Kraft & Steffensen, 2013), inventory management (Murphy & Oliveira, 2010, 2013) and valuation of futures contracts (Gulpinar & Oliveira, 2012).

Even though in some inter-temporal decision problems the distributions representing uncertainty are well known, there are many other problems in which information is incomplete. A methodology that is able to derive dynamic optimal policies in the context of limited information is reinforcement learning (Sutton, 1988; Bertsekas & Tsitsiklis, 1996). Applications of reinforcement learning include, among many others, scheduling problems (Zhang, Zheng, & Hou, 2011), Webpage ranking (Derhami, Khodadadian, Ghasemzadeh, & Bidoki, 2013) and the development of a self-evolutionary manufacturing system (Shin, Ryu, & Jung, 2012).

For this reason reinforcement learning has been receiving increasing attention from the business analytics community, being one of the most important methodologies used in the analysis of complex systems. This review presented in this article focus on three of the major algorithms used to model reinforcement learning: *n*-armed bandit, temporal-differences, and Q-learning. This chapter summarizes the reinforcement learning theory emphasizing its relationship with dynamic programming (reinforcement learning algorithms may replace dynamic programming when a full model of the environment is *not* available).

The Reinforcement Learning Problem

This section presents the terminology used in models of learning. The first concept presented is the *Markov property* (Howard, 1971). Let $P\left(s_{t+1} = s' \mid s_t, s_{t-1}, ..., s_0\right), \quad \forall s', s_t, s_{t-1}, ..., s_0$ represent the transition probability from state s_t to a state s', where s_t is the state of a dynamic system at time *t*. If the state has the Markov Property it contains all the information required to determine the transition probabilities, $P\left(s_{t+1} = s' \mid s_t\right)$. A Markovian decision process is characterized by a tuple (*policy*, *reward function*, *value function*, *model of the environment*).

A *policy* $\pi : A(s) \rightarrow a$ is a rule of behavior that transforms states into actions. A *reward function* $u_{ij}^{a_t}$ defines the utility that an agent receives from choosing an action a_t in a certain state *i*, at a given time t. The ultimate goal of an agent is to maximize the total reward received in the long-run, his *Return* (R_t). If the horizon is finite with *T* stages, then $R_t = u_{s_t, s_{t+1}}^{a_t} + u_{s_{t+1}, s_{t+2}}^{a_{t+1}} \ldots + u_{s_T, s_{T+1}}^{a_T}$. If the time horizon is infinite, the return is the discounted sum (where $0 < \gamma < 1$ is the *discount* parameter) of each one of the rewards $R_t = \sum_{k=0}^{+\infty} \gamma^k u_{t+k+1}$.

A *value function*, *V(s)*, specifies the value of the state *s*, i.e., the total value of rewards an agent expects to receive until the end of the horizon by entering that state immediately. When the problem has discrete state spaces and actions, and if the agent has a perfect knowledge of the transition probabilities and rewards, in order to compute the optimal policy there is no need to interact with the

DOI: 10.4018/978-1-4666-5202-6.ch181

environment. The method used to compute the optimal policy is dynamic programming (Bertsekas, 2000). However, when the state space and actions are be very large, and even if there is no explicit model of the environment, this value function can still be approximated using least squares or neural networks (Ueno, Maeda, Kawanabe, & Ishii, 2011; Chen, Chen, & Gu, 2013).

A *model* of the environment is the set of state transition probabilities and transition rewards associated with every action in each state. One of the advantages of reinforcement learning is to enable the learner to have forward-looking behaviour without having an explicit model of the environment.

An algorithm that works *on-line* learns the value function and controls a real environment at the same time. On-line algorithms present a trade-off between *exploration* (an attempt to improve the knowledge possessed at a certain time) and *exploitation* (an attempt to profit from the knowledge of the environment). In order to increase its knowledge of the environment an agent may have to choose sub-optimal actions. If an agent chooses only optimal actions he may end up exploiting a local optimum, not converging to the *global optimum*. *Off-line* algorithms use simulated experience with a model of the environment and do not face the exploitation vs. exploration trade-off. These algorithms learn the control policy before applying it to the real world.

Next, dynamic programming and three of the main approaches to reinforcement learning are discussed, namely the *n*-armed bandit, the temporal differences and the Q-learning algorithms. Reinforcement learning and dynamic programming can be used to solve the same problems as they both aim to computing the *optimal policies* given that the environment follows a Markov process.

Dynamic Programming

Dynamic Programming is a well established area of research which is used in the solution of inter-temporal decision problems in different areas of business analytics, including, among many others, portfolio optimization (Bae, Kim, & Mulvey, 2013; Kraft & Steffensen, 2013), operational planning of hydrothermal electricity generation (Dias, Tomim, Marcato, Ramos, Brandi, Junior, & Filho, 2013) and job-shop scheduling (Gromicho, van Hoorn, Saldanha-da-Gama, & Timmer, 2012).

Bellman (1957) defined dynamic programming as the mathematical theory of multi-stage decision processes, and defined *decision process* as a system (possibly stochastic) in which the decision-maker has a choice of transformations that may be applied to the system at any time. An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision (this is the *Principle of Optimality*). Therefore, the dynamic programming problem is the one of computing the *optimal policies*, given that the environment follows a Markovian process and a known model of the environment. Consequently, it implies the solution of two related problems: prediction (policy evaluation) and control.

The *prediction* problem consists in estimating the value of each state of the environment for a certain policy π. Let E_π represent the expected value of a policy π, and $\pi(s,a)$ represent the probability of executing action a in state s, following a policy π. Further, let $P^a_{ss'}$ stand for the probability of the system moving from state s to state s', conditional on the agent choosing action a. Then the *state-value function* $V^\pi(s)$ represents the expected return of state s, following a policy π such that $V^\pi(s) = E_\pi\{R_t \setminus s_t = s\}$, which is equivalent to

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

The *control problem* takes into account that a given policy influences the value of a given action. Thus, let the *action-value function* $Q^\pi(s,a)$ rep-