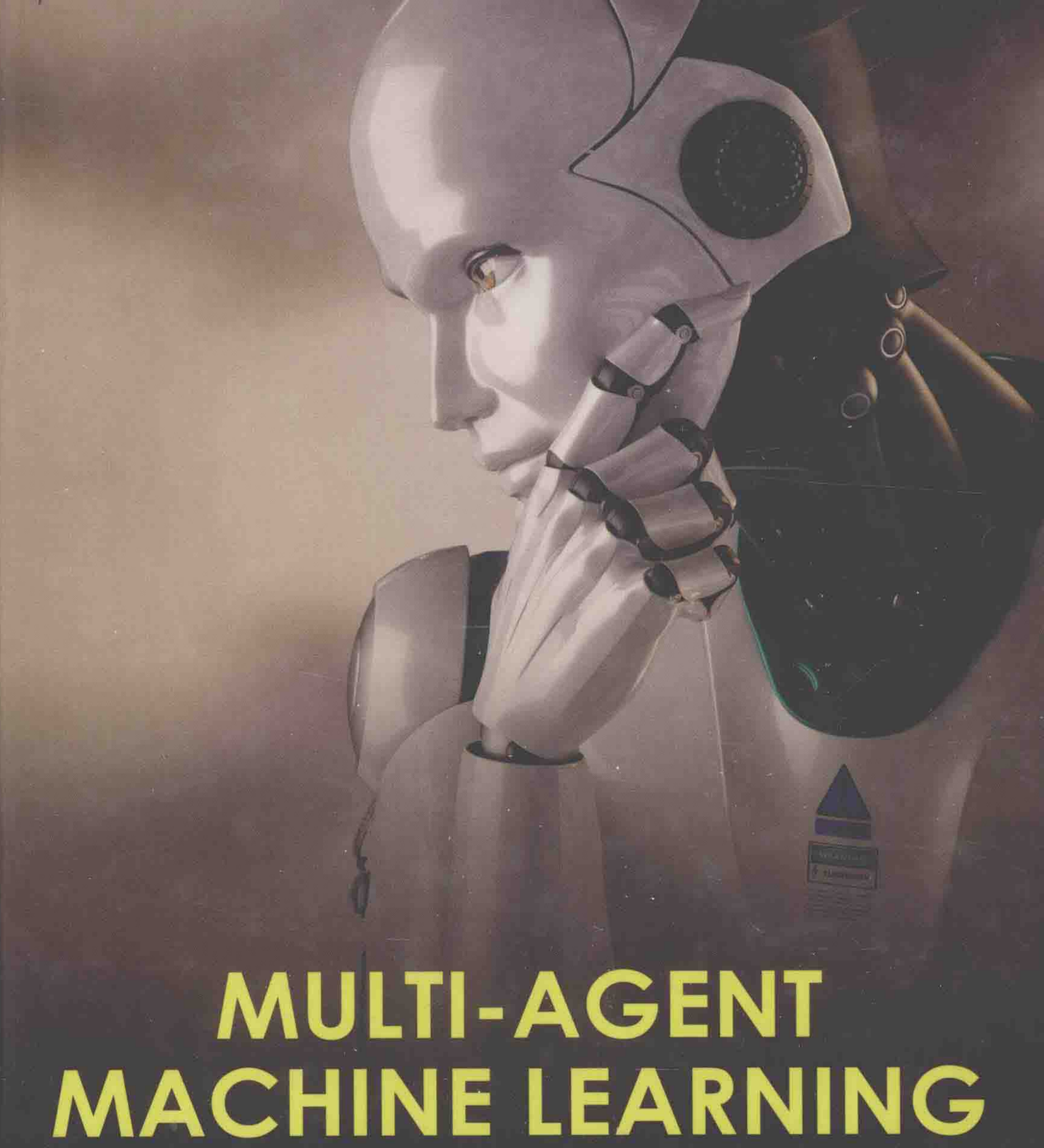

MULTI-AGENT MACHINE LEARNING

A Reinforcement Approach

Howard M. Schwartz

WILEY

Multi-Agent Machine Learning

A Reinforcement Approach

Howard M. Schwartz
Department of Systems and Computer Engineering
Carleton University

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data:

Schwartz, Howard M., editor.
Multi-agent machine learning : a reinforcement approach / Howard M. Schwartz.
pages cm
Includes bibliographical references and index.
ISBN 978-1-118-36208-2 (hardback)
1. Reinforcement learning. 2. Differential games. 3. Swarm intelligence. 4. Machine learning. I. Title.
Q325.6.S39 2014
519.3–dc23

2014016950

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Preface

For a decade I have taught a course on adaptive control. The course focused on the classical methods of system identification, using such classic texts as Ljung [1, 2]. The course addressed traditional methods of model reference adaptive control and nonlinear adaptive control using Lyapunov techniques. However, the theory had become out of sync with current engineering practice. As such, my own research and the focus of the graduate course changed to include adaptive signal processing, and to incorporate adaptive channel equalization and echo cancellation using the least mean squares (LMS) algorithm. The course name likewise changed, from "Adaptive Control" to "Adaptive and Learning Systems." My research was still focused on system identification and nonlinear adaptive control with application to robotics. However, by the early 2000s, I had started work with teams of robots. It was now possible to use handy robot kits and low-cost microcontroller boards to build several robots that could work together. The graduate course in adaptive and learning systems changed again; the theoretical material on nonlinear adaptive control using Lyapunov techniques was reduced, replaced with ideas from reinforcement learning. A whole new range of applications developed. The teams of robots had to learn to work together and to compete.

Today, the graduate course focuses on system identification using recursive least squares techniques, some model reference adaptive control (still using Lyapunov techniques), adaptive signal processing using the LMS algorithm, and reinforcement learning using Q-learning. The first two chapters of this book present these ideas in an abridged form, but in sufficient detail to demonstrate the connections among the learning algorithms that are available; how they are the same; and how they are different. There are other texts that cover this material in detail [2–4].

The research then began to focus on teams of robots learning to work together. The work examined applications of robots working together for search and rescue applications, securing important infrastructure and border regions. It also began to focus on reinforcement learning and multiagent reinforcement learning. The robots are the learning agents. How do children learn how to play tag? How do we learn to play football, or how do police work together to capture a criminal? What strategies do we use, and how do we formulate these strategies? Why can I play touch football with a new group of people and quickly be able to assess everyone's capabilities and then take a particular strategy in the game?

As our research team began to delve further into the ideas associated with multiagent machine learning and game theory, we discovered that the published literature covered many ideas but was poorly coordinated or focused. Although there are a few survey articles [5], they do not give sufficient details to appreciate the different methods. The purpose of this book is to introduce the reader to a particular form of machine learning. The book focuses on multiagent machine learning, but it is tied together with the central theme of learning algorithms in general. Learning algorithms come in many different forms. However, they tend to have a similar approach. We will present the differences and similarities of these methods.

This book is based on my own work and the work of several doctoral and masters students who have worked under my supervision over the past 10 years. In particular, I would like to thank Prof. Sidney Givigi. Prof. Givigi was instrumental in developing the ideas and algorithms presented in Chapter 6. The doctoral research of Xiaosong (Eric) Lu has also found its way into this book. The work on guarding a territory is largely based on his doctoral dissertation. Other graduate students who helped me in this work include Badr Al Faiya, Mostafa Awheda, Pascal De Beck-Courcelle, and Sameh Desouky. Without the dedicated work of this group of students, this book would not have been possible.

H. M. Schwartz
Ottawa, Canada
September, 2013

References

[1] L. Ljung, *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice Hall, 2nd ed., 1999.

[2] L. Ljung and T. Soderstrom, *Theory and Practice of Recursive Identification*. Cambridge, Massachusetts: The MIT Press, 1983.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[4] Astrom, K. J. and Wittenmark, B., *Adaptive Control*. Boston, Massachusetts: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1994, ISBN = 0201558661.

[5] L. Buşoniu and R. Babuška, and B. D. Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst. Man Cybern. Part C*, Vol. 38, no. 2, pp. 156–172, 2008.

Contents

Chapter 1
A Brief Review of Supervised Learning

There are a number of algorithms that are typically used for system identification, adaptive control, adaptive signal processing, and machine learning. These algorithms all have particular similarities and differences. However, they all need to process some type of experimental data. How we collect the data and process it determines the most suitable algorithm to use. In adaptive control, there is a device referred to as the *self-tuning regulator*. In this case, the algorithm measures the states as outputs, estimates the model parameters, and outputs the control signals. In reinforcement learning, the algorithms process rewards, estimate value functions, and output actions. Although one may refer to the recursive least squares (RLS) algorithm in the self-tuning regulator as a supervised learning algorithm and reinforcement learning as an unsupervised learning algorithm, they are both very similar. In this chapter, we will present a number of well-known baseline supervised learning algorithms.

1.1 Least Squares Estimates

The least squares (LS) algorithm is a well-known and robust algorithm for fitting experimental data to a model. The first step is for the user to define a mathematical structure or model that he/she believes will fit the data. The second step is to design an experiment to collect data under suitable conditions. "Suitable conditions" usually means the operating conditions under which the system will typically operate. The next step is to run the estimation algorithm, which can take several forms, and, finally, validate the identified or "learned" model. The LS algorithm is often used to fit the data. Let us look at the case of the classical two-dimensional linear regression fit that we are all familiar with:

$$y(n) = ax(n) + b \tag{1.1}$$

In this a simple linear regression model, where the input is the sampled signal $x(n)$ and the output is $y(n)$. The model structure defined is a straight line. Therefore, we are assuming that the data collected will fit a straight line. This can be written in the form

$$y(n) = \phi^T \theta \tag{1.2}$$

where $\phi^T = \begin{bmatrix} x(n) & 1 \end{bmatrix}$ and $\theta^T = \begin{bmatrix} a & b \end{bmatrix}$. How one chooses ϕ determines the model structure, and this reflects how one believes the data should behave. This is the essence of machine learning, and virtually all university students will at some point learn the basic statistics of linear regression. Behind the computations of the linear regression algorithm is the scalar cost function, given by

$$V = \sum_{n=1}^{N} (y(n) - \phi^T(n)\hat{\theta})^2 \tag{1.3}$$

The term $\hat{\theta}$ is the estimate of the LS parameter θ. The goal is for the estimate $\hat{\theta}$ to minimize the cost function V. To find the "optimal" value of the parameter estimate $\hat{\theta}$, one takes the partial derivative of the cost function V with respect to $\hat{\theta}$ and sets this derivative to zero. Therefore, one gets

$$\begin{aligned} \frac{\partial V}{\partial \hat{\theta}} &= \sum_{n=1}^{N} (y(n) - \phi^T(n)\hat{\theta})\phi(n) \\ &= \sum_{n=1}^{N} \phi(n)y(n) - \sum_{n=1}^{N} \phi(n)\phi^T(n)\hat{\theta} \end{aligned} \tag{1.4}$$

Setting $\frac{\partial V}{\partial \hat{\theta}} = 0$, we get

$$\sum_{n=1}^{N} \phi(n)\phi^T(n)\hat{\theta} = \sum_{n=1}^{N} \phi(n)y(n) \tag{1.5}$$

Solving for $\hat{\theta}$, we get the LS solution

$$\hat{\theta} = \left[\sum_{n=1}^{N} \phi(n)\phi^T(n)\right]^{-1} \left[\sum_{n=1}^{N} \phi(n)y(n)\right] \tag{1.6}$$

where the inverse, $\left[\sum_{n=1}^{N} \phi(n)\phi^T(n)\right]^{-1}$, exists. If the inverse does not exist, then the system is not identifiable. For example, if in the straight line case one only had a single point, then the inverse would not span the two-dimensional space and it would not exist. One needs at least two independent points to draw a straight line. Or, for example, if one had exactly the same point over and over again, then the inverse would not exist. One needs at least two independent points to draw a straight line. The matrix $\left[\sum_{n=1}^{N} \phi(n)\phi^T(n)\right]$ is referred to as the *information matrix* and is related to how well one can estimate the parameters. The inverse of the information matrix is the covariance matrix, and it is proportional to the variance of the parameter estimates. Both these matrices are positive definite and symmetric. These are very important properties which are used extensively in analyzing the behavior of the algorithm. In the literature, one will often see the covariance matrix referred to as $P = \left[\sum_{n=1}^{N} \phi(n)\phi^T(n)\right]^{-1}$. We can write the second equation on the right of Eq. (1.4) in the form

$$\frac{\partial V}{\partial \hat{\theta}} = 0 = \sum_{n=1}^{N} (y(n) - \phi^T(n)\hat{\theta})\phi(n) \tag{1.7}$$

and one can define the prediction errors as

$$\epsilon(n) = (y(n) - \phi^T(n)\hat{\theta}) \tag{1.8}$$

The term within brackets in Eq. (1.7) is known as the *prediction error* or, as some people will refer to it, the *innovations*. The term $\epsilon(n)$ represents the error in predicting the output of the system. In this case, the output term $y(n)$ is the correct answer, which is what we want to estimate. Since we know the correct answer, this is referred to as *supervised learning*. Notice that the value of the prediction error times the data vector is equal to zero. We then say that the prediction errors are orthogonal to the data, or that the data sits in the null space of the prediction errors. In simplistic terms, this means that, if one has chosen a good model structure $\phi(n)$, then the prediction errors should appear as white noise. Always plot the prediction errors as a quick check to see how good your predictor is. If the errors appear to be correlated (i.e., not white noise), then you can improve your model and get a better prediction.

One does not typically write the linear regression in the form of Eq. (1.2), but typically will add a white noise term, and then the linear regression takes the form

$$y(n) = \phi^T(n)\theta + v(n) \tag{1.9}$$

where $v(n)$ is a white noise term. Equation (1.9) can represent an infinite number of possible model structures. For example, let us assume that we want to learn the dynamics of a second-order linear system or the parameters of a second-order infinite impulse response (IIR) filter. Then we could choose the second-order model structure given by

$$y(n) = -a_1y(n-1) - a_2y(n-2) + b_1u(n-1) + b_2u(n-2) + v(n) \tag{1.10}$$

Then the model structure would be defined in $\phi(n)$ as

$$\phi^T(n) = \begin{bmatrix} y(n-1) & y(n-2) & u(n-1) & u(n-2) \end{bmatrix} \tag{1.11}$$

In general, one can write an arbitrary kth-order autoregressive exogenous (ARX) model structure as

$$\begin{aligned} y(n) = {} & -a_1y(n-1) - a_2y(n-2) - \cdots - a_my(n-k) \\ & + b_1u(n-1) + b_2u(n-2) + \cdots + b_{n-k}u(n-k) + v(n) \end{aligned} \tag{1.12}$$

and $\phi(n)$ takes the form

$$\phi^T(n) = \begin{bmatrix} y(n-1) & \cdots & y(n-m) & u(n-1) & \cdots & u(n-m) \end{bmatrix} \tag{1.13}$$

One then collects the data from a suitable experiment (easier said than done!), and then computes the parameters using Eq. (1.6). The vector $\phi(n)$ can take many different forms; in fact, it can contain nonlinear functions of the data, for example, logarithmic terms or square terms, and it can have different delay terms. To a large degree, one can use ones professional judgment as to what to put into $\phi(n)$. One will often write the data in the matrix form, in which case the matrix is defined as

$$\Phi = \begin{bmatrix} \phi(1) & \phi(2) & \cdots & \phi(N) \end{bmatrix} \tag{1.14}$$

and the output matrix as

$$Y = \begin{bmatrix} y(1) & y(2) & \cdots & y(N) \end{bmatrix} \tag{1.15}$$

Then one can write the LS estimate as

$$\hat{\Theta} = (\Phi\Phi^T)^{-1}\,\Phi Y \tag{1.16}$$

Furthermore, one can write the prediction errors as

$$E = Y - \Phi^T\hat{\Theta} \tag{1.17}$$

We can also write the orthogonality condition as $\Phi E = 0$.

The LS method of parameter identification or machine learning is very well developed and there are many properties associated with the technique. In fact, much of the work in statistical inference is derived from the few equations described in this section. This is the beginning of many scientific investigations including work in the social sciences.

1.2 Recursive Least Squares

The LS algorithm has been extended to the RLS algorithm. In this case, the parameter estimate is developed as the machine collects the data in real time. In the previous section, all the data was collected first, and then the parameter estimates were computed on the basis of Eq. (1.6). The RLS algorithm is derived by assuming a solution to the LS algorithm and then adding a single data point. The derivation is shown in Reference 1. In the RLS implementation, the cost function takes a slightly different form. The cost function in this case is

$$V = \sum_{n=1}^{N} \lambda^{(N-t)}(y(n) - \phi^T(n)\hat{\theta})^2 \tag{1.18}$$

where $\lambda \leq 1$. The term λ is known as the *forgetting factor*. This term will place less weight on older data points. As such, the resulting RLS algorithm will be able track changes to the parameters. Once again, taking the partial derivative of V with respect to $\hat{\theta}$ and setting the derivative to zero, we get

$$\hat{\theta} = \left[\sum_{n=1}^{N} \lambda^{(N-t)}\phi(n)\phi^T(n)\right]^{-1}\left[\sum_{n=1}^{N} \lambda^{(N-t)}\phi(n)y(n)\right] \tag{1.19}$$

The forgetting factor should be set as $0.95 \leq \lambda \leq 1.0$. If one sets the forgetting factor near 0.95, then old data is forgotten very quickly; the rule of thumb is that the estimate of the parameters $\hat{\theta}$ is approximately based on $1/(1-\lambda)$ data points.

The RLS algorithm is as follows:

$$\begin{aligned}
\hat{\theta}(n+1) &= \hat{\theta}(n) + L(n+1)(y(n+1) - \phi^T(n+1)\hat{\theta}(n)) \\
L(n+1) &= \frac{P(n)\phi(n+1)}{\lambda + \phi^T(n+1)P(n)\phi(n+1)} \\
P(n+1) &= \frac{1}{\lambda}\left(P(n) - \frac{P(n)\phi(n+1)\phi^T(n+1)P(n)}{\lambda + \phi^T(n+1)P(n)\phi(n+1)}\right)
\end{aligned} \tag{1.20}$$

One implements Eq. (1.20) by initializing the parameter estimation vector $\hat{\theta}$ to the users best initial estimate of the parameters, which is often simply zero. The covariance matrix P is typically initialized to a relatively large diagonal matrix, and represents the initial uncertainty in the parameter estimate.

One can implement the RLS algorithm as in Eq. (1.20), but the user should be careful that the covariance matrix P is always positive definite and symmetric. If the P matrix, because of numerical error by repeatedly computing the RLS, ceases to be positive definite and symmetric, then the algorithm will diverge. There are a number of well-developed algorithms to ensure that the P matrix remains positive definite. One can use a square-roots approach whereby the P matrix is factored into its Cholesky factorization or the UDU factorization. Such methods are described in Reference 1.

Let us examine Eq. (1.20) and notice that the update to the parameter estimate is the previous estimate plus a matrix $L(n)$ times the current prediction error. We will see this structure in almost every algorithm that will be described in machine learning. In this case, we have an actual correct answer, which is the measurement $y(n)$, and we call such algorithms *supervised learning*.

1.3 Least Mean Squares

In the field of signal processing, there are a few commonly used techniques to model or characterize the dynamics of a communications channel and then compensate for the effects of the channel on the signal. These techniques are referred to as *channel equalization* and *echo cancellation*. There are numerous books on adaptive signal processing and adaptive filtering [2]. Most of these techniques use the least mean squares (LMS) approach to identify the coefficients of a model of the channel. Once again, as in the LS and RLS algorithms, we must choose an appropriate model structure to define the communication channel dynamics. In the field of signal processing, one would typically use what is known as a *finite impulse response* (FIR) filter as the underlying model structure that describes the system. To maintain consistency with the previous section, one can write the channel dynamics as

$$y(n) = b_0u(n) + b_1u(n-1) + \cdots + b_ku(n-k) + v(n) \tag{1.21}$$

where $y(n)$ is the output of the filter, or the communications channel, at time step n, b_i are the filter coefficients that we want to estimate or *learn*, and $u(n)$ is the input signal. Typically, the signal $u(n)$ is the communication signal that we want to recover from the output signal $y(n)$. We define an error signal

$$\epsilon(n) = y(n) - \hat{y}(n) \tag{1.22}$$

where $\hat{y}(n) = \phi^T(n)\hat{\theta}$. This is the same signal as the prediction error in Eq. (1.8). The LMS algorithm defines the cost function as the expected value of the prediction errors as

$$J(n) = E[\epsilon^2(n)] \tag{1.23}$$

We can write the squared error term as

$$\begin{aligned}\epsilon^2(n) &= (y(n) - \phi^T(n)\hat{\theta})^2 \\ &= y^2(n) - 2y(n)\phi^T\hat{\theta} + \hat{\theta}^T\phi(n)\phi^T(n)\hat{\theta}\end{aligned} \tag{1.24}$$

We take the expectation and get

$$E[\epsilon^2(n)] = E[y^2(n)] - 2\hat{\theta}^TE[y(n)\phi(n)] + \hat{\theta}^TE[\phi(n)\phi^T(n)]\hat{\theta} \tag{1.25}$$

We then define the variance $\sigma_y = E[y^2]$ as the mean squared power, and the cross-correlation vector is defined as $p = E[y(n)\phi(n)]$. Then we define the information matrix, which is almost the same matrix as in Section 1.1, as $R = E[\phi(n)\phi^T(n)]$. If the system statistics are stationary, that is, the statistics do not change, the terms σ_y, p, and R, are constants and the cost function, as a function of changing $\hat{\theta}$, will have the shape of a bowl. The cost function $J(n)$ can be written as

$$J(n) = \sigma_y^2 - 2\hat{\theta}^Tp + \hat{\theta}^TR\hat{\theta} \tag{1.26}$$

Once again, as in Eq. (1.4), to find the *optimal* parameter estimate $\hat{\theta}$ to minimize the cost function, we take the partial derivative of the cost function $J(n)$ with respect to $\hat{\theta}$, and determine the value of $\hat{\theta}$ that sets the partial derivative to zero. We can take the partial derivative of $J(n)$ as

$$\frac{\partial J(n)}{\partial\hat{\theta}} = \frac{\partial\sigma_y^2}{\partial\hat{\theta}} - 2\frac{\hat{\theta}^Tp}{\partial\hat{\theta}} + \frac{\partial\hat{\theta}^TR\hat{\theta}}{\partial\hat{\theta}} \tag{1.27}$$