Multi-Agent Machine Learning
Multi-Agent Machine Learning
A Reinforcement Approach

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

WILEY
## Contents

### Preface
Preface ................................................................................................................. ix

### Chapter 1 A Brief Review of Supervised Learning

1.1 Least Squares Estimates ................................................................. 1
1.2 Recursive Least Squares ................................................................. 5
1.3 Least Mean Squares ....................................................................... 6
1.4 Stochastic Approximation .............................................................. 10

References ........................................................................................................ 11

### Chapter 2 Single-Agent Reinforcement Learning

2.1 Introduction ....................................................................................... 12
2.2 n-Armed Bandit Problem ................................................................. 13
2.3 The Learning Structure ................................................................. 15
2.4 The Value Function ....................................................................... 17
2.5 The Optimal Value Functions ......................................................... 18
   2.5.1 The Grid World Example ....................................................... 20
2.6 Markov Decision Processes ............................................................ 23
2.7 Learning Value Functions ............................................................... 25
2.8 Policy Iteration .............................................................................. 26
2.9 Temporal Difference Learning ....................................................... 28
2.10 TD Learning of the State-Action Function .................................... 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.10 Policy Hill Climbing</td>
<td>114</td>
</tr>
<tr>
<td>4.11 WoLF-PHC Algorithm</td>
<td>114</td>
</tr>
<tr>
<td>4.12 Guarding a Territory Problem in a Grid World</td>
<td>117</td>
</tr>
<tr>
<td>4.12.1 Simulation and Results</td>
<td>119</td>
</tr>
<tr>
<td>4.13 Extension of $\text{LR}_R\text{-I}$ Lagging Anchor Algorithm to Stochastic Games</td>
<td>125</td>
</tr>
<tr>
<td>4.14 The Exponential Moving-Average Q-Learning (EMA Q-Learning) Algorithm</td>
<td>128</td>
</tr>
<tr>
<td>4.15 Simulation and Results Comparing EMA Q-Learning to Other Methods</td>
<td>131</td>
</tr>
<tr>
<td>4.15.1 Matrix Games</td>
<td>131</td>
</tr>
<tr>
<td>4.15.2 Stochastic Games</td>
<td>134</td>
</tr>
<tr>
<td>References</td>
<td>141</td>
</tr>
</tbody>
</table>

Chapter 5 Differential Games ........................................ 144

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>144</td>
</tr>
<tr>
<td>5.2 A Brief Tutorial on Fuzzy Systems</td>
<td>146</td>
</tr>
<tr>
<td>5.2.1 Fuzzy Sets and Fuzzy Rules</td>
<td>146</td>
</tr>
<tr>
<td>5.2.2 Fuzzy Inference Engine</td>
<td>148</td>
</tr>
<tr>
<td>5.2.3 Fuzzifier and Defuzzifier</td>
<td>151</td>
</tr>
<tr>
<td>5.2.4 Fuzzy Systems and Examples</td>
<td>152</td>
</tr>
<tr>
<td>5.3 Fuzzy Q-Learning</td>
<td>155</td>
</tr>
<tr>
<td>5.4 Fuzzy Actor–Critic Learning</td>
<td>159</td>
</tr>
<tr>
<td>5.5 Homicidal Chauffeur Differential Game</td>
<td>162</td>
</tr>
<tr>
<td>5.6 Fuzzy Controller Structure</td>
<td>165</td>
</tr>
<tr>
<td>5.7 $Q(\lambda)$-Learning Fuzzy Inference System</td>
<td>166</td>
</tr>
<tr>
<td>5.8 Simulation Results for the Homicidal Chauffeur</td>
<td>171</td>
</tr>
<tr>
<td>5.9 Learning in the Evader–Pursuer Game with Two Cars</td>
<td>174</td>
</tr>
<tr>
<td>5.10 Simulation of the Game of Two Cars</td>
<td>177</td>
</tr>
<tr>
<td>5.11 Differential Game of Guarding a Territory</td>
<td>180</td>
</tr>
<tr>
<td>5.12 Reward Shaping in the Differential Game of Guarding a Territory</td>
<td>184</td>
</tr>
<tr>
<td>5.13 Simulation Results</td>
<td>185</td>
</tr>
<tr>
<td>5.13.1 One Defender Versus One Invader</td>
<td>185</td>
</tr>
<tr>
<td>5.13.2 Two Defenders Versus One Invader</td>
<td>191</td>
</tr>
<tr>
<td>References</td>
<td>197</td>
</tr>
</tbody>
</table>
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


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
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:

\[ y(n) = ax(n) + b \] (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

\[ y(n) = \phi^T \theta \] (1.2)

where \( \phi^T = [x(n) \ 1] \) and \( \theta^T = [a \ b] \). How one chooses \( \phi \) 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

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

The term \( \hat{\theta} \) is the estimate of the LS parameter \( \theta \). 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

\[
\frac{\partial V}{\partial \hat{\theta}} = \sum_{n=1}^{N} (y(n) - \phi^T(n)\hat{\theta})\phi(n) \\
= \sum_{n=1}^{N} \phi(n)y(n) - \sum_{n=1}^{N} \phi(n)\phi^T(n)\hat{\theta} \]

(1.4)

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

\[
\sum_{n=1}^{N} \phi(n)\phi^T(n)\hat{\theta} = \sum_{n=1}^{N} \phi(n)y(n) \]

(1.5)
Solving for $\hat{\theta}$, we get the LS solution

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

(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

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

(1.7)

and one can define the prediction errors as

$$
e(n) = (y(n) - \phi^T(n)\hat{\theta})
$$

(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 $e(n)$ 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
\[ y(n) = \phi^T(n)\theta + v(n) \]  
(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
\[ y(n) = -a_1 y(n-1) - a_2 y(n-2) + b_1 u(n-1) + b_2 u(n-2) + v(n) \]  
(1.10)
Then the model structure would be defined in \( \phi(n) \) as
\[ \phi^T(n) = [y(n-1) \ y(n-2) \ u(n-1) \ u(n-2)] \]  
(1.11)
In general, one can write an arbitrary \( k \)th-order autoregressive exogenous (ARX) model structure as
\[ y(n) = -a_1 y(n-1) - a_2 y(n-2) - \cdots - a_m y(n-k) + b_1 u(n-1) + b_2 u(n-2) + \cdots + b_{n-k} u(n-k) + v(n) \]  
(1.12)
and \( \phi(n) \) takes the form
\[ \phi^T(n) = [y(n-1) \ \cdots \ y(n-m) \ u(n-1) \ \cdots \ u(n-m)] \]  
(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 one's 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
\[ \Phi = [\phi(1) \ \phi(2) \ \cdots \ \phi(N)] \]  
(1.14)
and the output matrix as
\[ Y = [y(1) \ y(2) \ \cdots \ y(N)] \]  
(1.15)
Then one can write the LS estimate as

\[ \hat{\Theta} = (\Phi \Phi^T)^{-1} \Phi Y \]  

(1.16)

Furthermore, one can write the prediction errors as

\[ E = Y - \Phi^T \hat{\Theta} \]  

(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

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

(1.18)

where \( \lambda \leq 1 \). The term \( \lambda \) 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 to 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

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

(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:

\[
\hat{\theta}(n+1) = \hat{\theta}(n) + L(n+1)(y(n+1) - \phi^T(n+1)\hat{\theta}(n))
\]

\[
L(n+1) = \frac{P(n)\phi(n+1)}{\lambda + \phi^T(n+1)P(n)\phi(n+1)}
\]

\[
P(n+1) = \frac{1}{\lambda} \left( P(n) - \frac{P(n)\phi(n+1)\phi^T(n+1)P(n)}{\lambda + \phi^T(n+1)P(n)\phi(n+1)} \right)
\]

(1.20)

One implements Eq. (1.20) by initializing the parameter estimation vector \(\hat{\theta}\) to the user's 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

\[ y(n) = b_0u(n) + b_1u(n-1) + \cdots + b_ku(n-k) + v(n) \]  

(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

\[ e(n) = y(n) - \hat{y}(n) \]

(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

\[ J(n) = E[e^2(n)] \]

(1.23)

We can write the squared error term as

\[ e^2(n) = (y(n) - \phi^T(n)\hat{\theta})^2 \]

\[ = y^2(n) - 2y(n)\phi^T\hat{\theta} + \hat{\theta}^T\phi(n)\phi^T(n)\hat{\theta} \]

(1.24)

We take the expectation and get

\[ E[e^2(n)] = E[y^2(n)] - 2\hat{\theta}^T E[y(n)\phi(n)] + \hat{\theta}^T E[\phi(n)\phi^T(n)]\hat{\theta} \]

(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 \( \sigma_y, p, \) 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

\[ J(n) = \sigma_y^2 - 2\hat{\theta}^T p + \hat{\theta}^T R\hat{\theta} \]

(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

\[ \frac{\partial J(n)}{\partial \hat{\theta}} = \frac{\partial \sigma_y^2}{\partial \hat{\theta}} - 2\frac{\partial \hat{\theta}^T p}{\partial \hat{\theta}} + \frac{\partial \hat{\theta}^T R\hat{\theta}}{\partial \hat{\theta}} \]

(1.27)
We then compute the partial derivative for each of the terms on the right-hand side of Eq. (1.27). Taking each term separately, we get

\[
\frac{\partial \sigma^2_y}{\partial \hat{\theta}} = 0
\]

\[
2 \frac{\hat{\theta}^T p}{\partial \hat{\theta}} = 2p
\]

\[
\frac{\partial \hat{\theta}^T R \hat{\theta}}{\partial \hat{\theta}} = 2R \hat{\theta}
\]

Substituting into Eq. (1.27), we get

\[
\frac{\partial J(n)}{\partial \hat{\theta}} = -2p + 2R \hat{\theta} = 0
\]

(1.29)

Solving for \( \hat{\theta} \), we get the solution for the optimal parameter estimate as

\[
\hat{\theta}^* = R^{-1} p
\]

(1.30)

Equation (1.30) is the well-known Wiener solution. However, the Wiener solution in Eq. (1.30) requires the computation of the inverse of a large matrix \( R \). Notice the similarity between the Wiener solution and the LS solution in Eq. (1.6). Let us say that we want to estimate the expectations in Eq. (1.25); then we would get the average by computing

\[
R_{avg} = \left[ \frac{1}{N} \sum_{n=1}^{N} \phi(n) \phi^T(n) \right]
\]

\[
p_{avg} = \left[ \frac{1}{N} \sum_{n=1}^{N} \phi(n) y(n) \right]
\]

(1.31)

Substituting the above values into Eq. (1.30), we get the LS solution given by Eq. (1.6). In essence, the LMS Wiener solution and the LS solution are essentially the same.

In the world of signal processing and in particular adaptive signal processing, the processing speed is very important. Furthermore, the model structure used in adaptive signal processing, especially for communication applications, can have many parameters. It would not be unusual to have 200 terms in the \( \phi(n) \) vector, which means the term \( k \) in Eq. (1.21) would be \( k = 200 \). In that case, the \( R \) matrix will be \( 200 \times 200 \), which would be prohibitively large to take the
inverse of in Eq. (1.30). As such, a gradient \textit{steepest descent} method is normally implemented. This is a very common technique throughout the fields of engineering and is very similar to the well-known Newton–Raphson method of finding the zeros and roots of various functions. The steepest descent method is an iterative method. The idea is to start with an initial guess of the parameter values: often one will simply choose zero for the parameter values. In the lexicon of signal processing, one would refer to the parameters as \textit{tap weights}. Then one iteratively adjusts the parameters such that one moves down the cost function along the gradient. Let us say that the current estimate of the parameter vector is $\hat{\theta}(\text{now})$; then we compute the next value of the parameter vector as

$$
\hat{\theta}(\text{next}) = \hat{\theta}(\text{now}) - \mu g
$$

(1.32)

where $g$ is the gradient and is given by the derivative of the cost function with respect to the parameter estimation vector, $\hat{\theta}$ as defined in Eq. (1.29). Then, substituting for $g$ in Eq. (1.32), we get

$$
\hat{\theta}(\text{next}) = \hat{\theta}(\text{now}) - \mu 2p - \mu 2R\hat{\theta}(\text{now})
$$

(1.33)

In recursive form, it is written as

$$
\hat{\theta}(n+1) = \hat{\theta}(n) - \mu 2p - \mu 2R\hat{\theta}(n)
$$

(1.34)

We can also write Eq. (1.34) in the form

$$
\hat{\theta}(n+1) = (I - \alpha R)\hat{\theta}(n) - \alpha p
$$

(1.35)

where $\alpha = 2\mu$. One may recognize, from systems theory, that if the eigenvalues of $(I - \alpha R)$ are less than 1, then the recursion in Eq. (1.35) will converge. This places a limit on the step size of the steepest descent method. We will come back to this point in the next section when we look at the stochastic approximation methods. The effect of step size is an important parameter in machine learning algorithms.

The difficulty in computing the recursion in Eq. (1.34) is the computation of the statistical terms $R$ and $p$, where $R$ is the information matrix or the autocorrelation matrix, and $p$ is the cross-correlation matrix. Their statistics are often unknown and have to be estimated as we did in Eq. (1.31). However, these estimates are computationally intensive and one has to wait until $N$ data points are collected. Instead, the LMS algorithm proposes that one estimate
these matrices based on a single data point at each sampling time, as
\[
\hat{R}(n) = \phi(n)\phi^T(n)
\]
\[
\hat{p}(n) = \phi(n)y(n)
\]
(1.36)

This is sometimes referred to as the dirty gradient method or the stochastic gradient method. The idea is that one has to descent in the general direction of the gradient and not exactly along the gradient. Think of yourself walking down a hill; you can either go straight down or, if it is really steep, you may choose to go back and forth traversing the hill, much the same way as a skier. Either way, you end up at the bottom of the hill. Now we substitute the estimates for \( \hat{R} \) and \( \hat{p} \) given by Eq. (1.36) into the recursive equation given by Eq. (1.34) and we get
\[
\hat{\theta}(n + 1) = \hat{\theta}(n) + 2\mu\phi(n)y(n) - 2\mu\phi(n)\phi^T(n)\hat{\theta}(n)
\]
(1.37)

Now we factor out the term \( 2\mu\phi(n) \) and get the standard LMS recursive algorithm as
\[
\hat{\theta}(n + 1) = \hat{\theta}(n) + 2\mu\phi(n)(y(n) - \phi^T(n)\hat{\theta}(n))
\]
(1.38)

Recall that the term in brackets on the right-hand side, given by \( (y(n) - \phi^T(n)\hat{\theta}(n)) \), is the prediction error or the innovation. The term \( \phi(n)\hat{\theta}(n) \) is the current prediction of the output \( y(n) \). If we compare the RLS algorithm in Eq. (1.20) to Eq. (1.38), we see that the update has a similar form. The update to the parameters is the previous estimate plus a matrix vector times the prediction error. In fact, it can be shown that at the stationary point, or the value at which the covariance matrix update takes the value \( P(n + 1) = P(n) \) in Eq. (1.20), the LMS algorithm is equivalent to the RLS algorithm for a particular set of parameters.

There is a vast literature on various implementations and convergence results associated with the LMS algorithm, but the key element for this book is that the machine learns the parameters of a preconceived model of the system based on the available experimental data and knowledge of the correct answer given by \( y(n) \). The new parameter is the old parameter plus a vector based on the data times the known error in the predicted output.

### 1.4 Stochastic Approximation

The method of stochastic approximation is an older method of system identification. In fact, it is a method of finding the zeros of a function and is very similar to both the RLS and LMS methods, and it is the fundamental structure for the Q-learning algorithms associated with reinforcement learning and
much of the machine learning literature. The early work in stochastic approximation comes from the work by Robbins and Monro [3] and Wolfowitz [4]. A good textbook on the topic is by Kushner and Yin [5].

Monro formulated the problem as finding the level at which a continuous function \( M(\theta) = \alpha \). Writing the problem in the form \( M(\theta) - \alpha = 0 \) converts it into the problem of finding the zeros of a function. If one knows the gradient of the function, then one can use the well-known Newton–Raphson method to find the zeros, but in this case one takes the noise-corrupted measurements of the function at different values of \( \theta \). One then makes small corrections to \( \theta \) in the estimated direction of zero.

The method of stochastic approximation and the theoretical proofs of stability are used in the proofs of convergence for several fundamental algorithms in reinforcement learning. Formulating the problem in a similar form to the previous sections, we get the function \( M(\theta) = (y(\theta) - \phi^T \theta) = 0 \) and we can write the prediction error and the error in getting to zero as \( \epsilon = (y - \phi^T \theta) \).

The stochastic approximation algorithm is

\[
\theta(n + 1) = \theta(n) - a_n (y - \phi^T \theta) \tag{1.39}
\]

where \( a_n \) is a variable step size that goes to zero such that

\[
0 < \sum_{n=1}^{\infty} a_n^2 = A < \infty \tag{1.40}
\]

References


Chapter 2
Single-Agent Reinforcement Learning

The objective of this chapter is to introduce the reader to reinforcement learning. A good introductory book on the topic is Reference 1 and we will follow their notation. The goal of reinforcement learning is to maximize a reward. The interesting aspect of reinforcement learning, as well as unsupervised learning methods, is the choice of rewards. In this chapter, we will discuss some of the fundamental ideas in reinforcement learning which we will refer to in the rest of the book. We will start with the simple $n$-armed bandit problem and then present ideas on the meaning of the “value” function.

2.1 Introduction

Reinforcement learning is learning to map situations to actions so as to maximize a numerical reward [1]. Without knowing which actions to take, the learner must discover which actions yield the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and all subsequent rewards [1]. Different from supervised learning, which is learning from examples provided by a knowledgeable external supervisor, reinforcement learning is used for learning from interaction [1]. Since it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations, the learner must be able to learn from its