FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 14, No. 1, April 2001, 000-000

EXPLORATION-EXPLOITATION TRADE-OFF IN MACHINE LEARNING

Dragoljub Pokrajac, Aleksandar Lazarević and Zoran Obradović

Abstract: A survey of machine learning problems involving exploration-exploitation trade-off is presented. Theoretical and practical properties of existing algorithms for online learning tasks including K-armed bandit problem, apple-tasting and reinforcement learning are discussed. Several open problems in this area are described and their importance is emphasized.

Key words: Machine learning, K-armed bandit problem, reinforcement learning, on-line learning.

1. Introduction

Machine learning is concerned with the fundamental and experimental research on the development of intelligent computer systems capable of advanced forms for automatic improving from experience [25], [34]. One of the main tasks of machine learning is to provide techniques, methods and algorithms for learning, inference and explanation of natural, social and behavioral patterns and associations.

In general, all machine learning problems can be divided into off-line and on-line learning. In off-line learning, all the experience (examples) used for learning are available at the time when the learning starts. The primary goal is to construct a computer program (learning algorithm) that minimizes the error when testing is performed. Although a plethora of learning algorithms minimize the number of errors from experience (training error), it is important to emphasize that this number is not necessary related to the performance of the learner on the new unseen data [2].

Manuscript received March 8, 2001.

The authors are with Center for Information Science and Technology, Temple University, 303Wachman Hall (038-24), 1805 N. Broad St., Philadelphia, PA 19122, USA (e-mails: [pokie,aleks,zoran]@ist.temple.edu).

In an on-line learning model, learning occurs in sessions and the basic goal is to minimize the number of mistakes during a session [1], [27] and [31]. The number of trials per session (the length of the session) can be finite or infinite. In each trial, examples from experience and prior knowledge are provided to a "learner" (learning algorithm) that chooses one of given alternatives and performs decision. A "teacher" (an oracle from outer world) may then submit information about quality of the decision made. In on-line learning, learning and testing are not strongly separated. The process of learning is continuous: every time a new knowledge is presented, a learning algorithm refines hypotheses and thus adapts itself to the non-stationarity of the environment. The on-line model seems to be more realistic compared to the off-line model, since it is analog to the continuous process of human learning extending through the entire life.

During the process of on-line learning, the learner is confronted with a dilemma: whether to *exploit* the knowledge acquired during the learning process or to *explore* new actions through stochastic behavior instead of deterministic one. Exploration brings potential benefits, if a chosen action leads to a long-term performance improvement, but also carries latent risk, if a chosen action is a sub-optimal. Hence, there is a problem of achieving a well-balanced trade-off between exploration and exploitation.

Throughout this paper, the exploration-exploitation dilemma that naturally arises in various machine-learning tasks is discussed and some of the proposed solutions and heuristics to cope with this problem are examined. In Section 2 a mistake bound model of learning is introduced, while in Sections 3, 4 and 5 theoretical and practical aspects of the existing algorithms for on-line learning tasks are considered, as well as related open problems. In Section 3 the K-armed bandit problem is discussed followed by the "apple-tasting" problem presented in Section 4 and the reinforcement learning introduced in Section 5.

2. Mistake Bound Model

Many machine learning problems involve the task of classifying examples into one of possible categories (classes). These problems are referred to as *classification problems*. In the mistake bound model of learning [8], [34], a learning algorithm (learner) is evaluated by the total number of misclassification mistakes it makes before an optimal decision system (correct hypothesis) is learned or before all examples available for training are exhausted. In each trial, a training example is presented to the learner. After the learner classifies the given example, the teacher provides a correct answer regardless

of the learner's classification. Typically, we are interested in determining *optimal mistake bounds*. Therefore, the goal is to determine the maximal number of mistakes *opt(C)* made by the *best possible algorithm* in the case of the worst possible sequence of examples, when the most difficult concept from a class C is to be learned. In the case of binary classification problems, when all examples are assigned either a positive or a negative class, there are two possible errors during the classification. A false negative classification occurs when an example with a positive label is assigned a negative label. A false positive classification happens when a learner classifies a negative example as a positive one. The supremums of the numbers of false negative classification errors and the false positive classification errors are denoted as M_- and M_+ respectively, where the upper bound $opt(C) = M_- + M_+$ can be easily determined [27].

Consider the on line *halving algorithm* [27], [30], [37], where the learner maintains a collection of hypotheses consistent with the examples submitted so far and in each trial classifies according to the majority of "votes" in the current collection. After true classification is obtained from the teacher, non-consistent hypotheses are removed from the collection. If there are $|C|$ hypotheses in a concept class C, each mistake of a halving algorithm eliminates at least half of them, so if $|C|$ is finite, there is a finite $O(lg|C|)$ number of mistakes, before either all hypotheses are eliminated (if target hypothesis is not in class C), or a target hypothesis is learned.

A lower bound for $opt(C)$ can be obtained using the theory of Vapnik-Chernovenkis (VC) dimension [46], [47], [48]. According to its definition, VC dimension of a class C is $VC(C)$, if there exists $VC(C)$ examples such that each of their $2^{VC(C)}$ possible class assignments is consistent with some hypothesis from C. Therefore, given an algorithm prediction on a sequence of $VC(C)$ examples, the true class can be different than the prediction on each example. Hence, VC(C) is the lowest worst case bound for the number of mistakes and therefore the total number of classification errors is bounded from the both sides as

$$VC(C) \leq M_- + M_+ \leq lg|C|. \tag{1}$$

In a generalized mistake bound problem, Littlestone [28] investigated the existence of a learning algorithm A that reaches M_+ and M_-, where M_+ and M_- satisfy a constraint predicate P[1]. As an answer for this question,

[1]In general, constraint predicate is a conjunction of predicates containing M_+ and M_-. The simplest case of a predicate is $M_+ + M_- < B$, representing classical mistake bound problem, where B is the total number of mistakes.

the SCS algorithm is proposed [28] and is shown that given constraints P and a concept class C, the existence of the learning algorithm A implies that M_+ and M_- achieved by SCS algorithm satisfy constraints P on C. The main characteristic of the SCS algorithm is that it recursively reduces the problem of the existence of algorithm A to the existence of the same algorithm with a reduced input constraint P. According to [28], SCS may require a time-consuming examination of a target concept class.

The CCS algorithm, also proposed in [28], is guaranteed to satisfy constraints P whenever the size of class C is sufficiently small. Nevertheless, the problem of an efficient determination of the bound for the class size and the existence of an efficient algorithm for the classes of an arbitrary size is still open. Both proposed algorithms are difficult to implement, their computational complexity is not explicitly determined and they are of a rather theoretic importance[2].

One of applications for generalized mistake bound theory is the minimization of a *generalized loss*. With the "cost" of a false negative and false positive prediction equal to a and b, respectively, the generalized loss is equal to $aM_- + bM_+$. Hence, its minimization is equivalent to generalized mistake bound problem with the constraint P equal to $aM_- + bM_+ < B$.

Using the asymptotic notation [12], it can be shown [28] that in the case of a generalized loss, the smallest B is equal to $\Theta(lg|C|)$, such that the constraint is satisfied for any sequence of trials. This generalizes a mistake bound obtained by the *halving* algorithm. The existence of an algorithm, less complex than SCS (or CCS), that would guarantee $\Theta(lg|C|)$ performance bounds for the generalized loss is an open problem.

3. K-Armed Bandit Problem

The problem of the K-armed bandit [18], [22], [42], [6] can be formally defined as follows: Given K generators[3] of random rewards x_i, $i = 1, \ldots, K$ with unknown probability distributions $p_i(x_i)$, $i = 1, \ldots, K$, in each trial during a session of the finite length, choose one of the generators and pick up the corresponding current reward. The goal is to determine the strategy of choosing generators such that an expected cumulative reward is maxi-

[2] One of main application of these algorithms is to be subroutine of apple-tasting generic algorithm, described in [21], thus proving tighter bound for its worst-case behavior.

[3] Due to analogy with one-armed bandit slot machine, in literature is preferred to define K-armed bandit as a problem of proper choice of one of *arms* of K-armed slot machine, instead of random number generators. However, we prefer terminology of random number generators, since it avoids unnecessary abstraction.

mized. This problem is closely related to the theoretic explanation of genetic algorithms [18], [19], [22] that simultaneously solve K-armed bandit in the space of binary valued patterns-schemata. In addition, K-armed bandit problem is closely associated with the theory of dynamic allocation indices and stochastic scheduling tasks [17].

The performance of a learning algorithm in the K-armed bandit problem and its generalizations are measured by regret, defined as a difference between the best possible cumulative reward obtained by a sequence of optimal generator choices in given sessions and an actual cumulative reward achieved by the algorithm. The task is to minimize the regret, considering its worst-case bounds as functions of the number of trials in session T and the number of generators K.

A naive strategy to solve the K-armed bandit problem is a *greedy* strategy, where the rewards from the generator i that had the highest average reward $Q_t(i)$ in the past are always chosen. Since the expected reward estimation obtained by averaging actual rewards is only an approximation, and a *greedy* strategy selects a single generator (that appeared to be the best), the behavior of other generators cannot be properly explored and the estimates of their expected rewards may not be adequately adjusted. However, it is still beneficial to explore behavior of other generators, although they do not appear to be optimal based on the current information. Therefore, a trade-off between exploration and exploitation is very often necessary, and the methods to achieve it include [42]:

- $\varepsilon - greedy$ strategy: instead of always selecting an action having the maximal current average reward, choose among other actions with a small predefined probability ε. This strategy achieves the better exploration-exploitation trade-off if the variances of the distributions $p_i(x_i)$ are higher, since the action with the currently largest average is not necessarily the optimal one. Oppositely, if these distributions have zero variance, a *greedy* algorithm is the optimal, since the expected current reward estimation is always equal to its true value. A further improvement includes reducing ε over time and making a learning strategy gradually greedier as the estimation becomes more confident.
- $Softmax$ action selection using Boltzman distribution: the probability that generator i is chosen at the moment t is proportional to $exp(Q_t(i)/\tau(t))$, where $\tau(t)$ is a positive function decreasing with t and $Q_t(i)$ is the highest average reward. At the beginning of the session, when "temperature" $\tau(t)$ is high, transitions between actions mimic random walk because of approximately equal choice probabilities. On

the other side, with increased time t, it can be assumed that an estimation of expected rewards through averaging is approximately correct, so the difference between actions regarding their rewards can be emphasized by decreasing $\tau(t)$. It can be shown [22] that in the optimal (but unrealistic) case the numbers of trials devoted to the best and to the other generators should be exponentially related, and therefore a heuristic applied in $softmax$ selection seems to have a solid theoretic foundation.

- *Reinforcement comparison*: For each action a, a value $p(a)$ called $preference$ is maintained. When an action occurs, the preference is updated as $p(a) = p(a) + \alpha \cdot (x_i - \bar{r})$ where $\bar{r}$ is the average reward through *all* actions in the previous trials and α is a coefficient of proportionality. As a result, whenever an action results in the reward higher than the average, its $preference$ increases and vice versa. Similar to $softmax$ action comparison, the probability that an action is chosen is proportional to $exp(p(a))$. The variant of this method, called *pursuit method* [42], combines both $preferences$ and averaged rewards per each chosen generator.

It can be proven [3] that the upper bound for regret in K-bandit problem is $O(lgn)$, where n is the length of the session.

When the K-bandit problem is non-stationary, i.e. when parameters of the distributions $p_i(x_i)$ vary in time, it is convenient to introduce weighted average of rewards, where more recent rewards receive exponentially higher weights. Explicitly, the recurrent formula for current average reward after k-th execution of action a is: $Q_{k+1}(a) = Q_k(a) + \gamma \cdot (r_{k+1} - Q_k(a))$, where more recent awards are emphasized, such that the non-stationarity of distributions can be tracked.

3.1 Adversarial K-armed bandit problem

The classical K-armed bandit problem presumes the existence of probability distributions independent of the learner behavior and associated with particular random generators x_i, $i = 1, \ldots, K$. In [4] the adversarial model of a K-armed bandit is considered, where the teacher has a complete control on the assignment of rewards to each of the random generators. In addition, the teacher can be $non - oblivious$ by allowing the rewards to depend on the learner's actions. All the results obtained for non-oblivious teachers are also valid for a less challenging case of an oblivious teacher, where reward assignment may depend only on time (in the case of non-stationarity).

The K-armed bandit problem can be viewed as an example of a *partial-*

information game, where at time step t the learner obtains a reward only from the generator i. In the case of a *full information* game, after determining which generator to choose, the learner knows the rewards of all generators. Assuming an appropriate choice of a coefficient α and rewards confined in the interval [0,1], the *hedge* algorithm [4] guarantees $O(\sqrt{T \ln K})$ regret, where T is the length of a session. For each random generator i, the *hedge* algorithm maintains the score $s_i(t)$ representing the sum of all rewards associated with the generator i up to the time step t. At the $full\ information\ game$, all rewards are available regardless of the actual learner decisions. The hedge algorithm chooses a generator randomly according to the probability density that is proportional to $(1+\alpha)^{s_i(t)}$. Here, one can easily recognize the same ideas as in $softmax$ method for a standard K- armed bandit problem.

The hedge algorithm serves as a basis for $Exp3$ algorithm that works for the case of *partial information game* i.e. for adversarial K-armed bandit problem. The idea of $Exp3$ algorithm is to add a random component to the probability distribution obtained from the *hedge* algorithm. Having obtained a reward for a chosen generator i_t, a *simulated* reward vector is established, containing rewards for all generators (not only the reward for the chosen one!). Assuming that $p_j(t)$ is the probability obtained from the *hedge* algorithm that $j-th$ generator is to be chosen, the probability of drawing $j-th$ generator in $Exp3$ is

$$\hat{p}_j(t) = (1-\gamma)p_j(t) + \gamma\frac{1}{K}. \tag{2}$$

In the extreme case when $\gamma = 1$, the choice of a generator is completely random and independent of the rewards in the past. The weighted component γ/K is introduced to "soften" the decision of the *hedge*, since the rewards that are inputs of the *hedge* are not actual, but simulated.

As proposed in [4], components of a simulated reward vector are

$$\hat{x}_j(t) = \begin{cases} \dfrac{\gamma}{K}\dfrac{\hat{x}_i(t)}{\hat{p}_i(t)}, & \text{if} \quad j = i_t \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The normalization of $\hat{x}_j(t)$ by the probability $\hat{p}_{i_t}(t)$ is justified with the necessity to compensate infrequent occurrence of particular choices. Specifically, if the generator i_t is chosen with probability $\hat{p}_{i_t}(t)$, it will be chosen once per $(1/\hat{p}_{i_t}(t))$ trials on average. Therefore by introducing the normalization factor we account for the trials when a generator is not actually chosen

(and when a simulated reward corresponding to the generator is 0). The remaining part of normalization coefficient is chosen to confine simulated rewards in range [0,1], thus providing $O(\sqrt{T \ln K})$ regret. An interesting open problem arises regarding the existence of an alternative value assignment for simulated reward vector $\hat{x}_j(t)$ (particularly when $j \neq i_t$) that may lead to an improved upper bound for regret.

Setting appropriately parameters γ (for $Exp3$) and α (for embedded *hedge* algorithm), it is possible to achieve $O\left(T^{2/3}(K \ln K)^{1/3}\right)$ regret in a session of T steps. In general, when rewards are within a finite interval [a,b] instead of [0,1], the achieved regret is $\left((b-a)T^{2/3}(K \ln K)^{1/3}\right)$. The computational complexity of $Exp3$ and hedge algorithms is rather small. In each trial, there are $\Theta(K)$ computations of probabilities and score updates in the *hedge* and $\Theta(K)$ computations of probabilities and simulated rewards in the main routine of the $Exp3$ algorithm.

If the rewards in each trial are bounded, the maximal expected reward is $O(T)$ and it influences the values of the parameters γ and α in the $Exp3$ algorithm. When T is not known in advance, it is difficult to estimate the maximal expected reward, so a modification of $Exp3$ is necessary. An algorithm $Exp3.1$ [4] is such a modification, where sessions occur in *rounds* and in each round a *current* maximal expected reward is maintained. At the beginning of the $n-th$ round, the parameters γ and α are computed according to the *current* maximal expected reward that is set to 2^n. The score update is performed similar as in $Exp3$. Whenever a score for a generator is approaching the current maximal expected reward, n is increased by 1 and the next round is started. This algorithm guarantees $O\left(T^{2/3}(k \ln K)^{1/3} + K^2(\ln K)^{1/3}\right)$ regret [4]. For a sufficiently large number of trials, when $T = \Omega(K^{5/2})$, the regret for the $Exp3.1$ procedure achieves asymptotically the same bound as for the $Exp3$ algorithm.

The problem of a tight lower bound for the regret defined here[4] is still open. In [4], a lower bound is shown using the following reward distribution. Let the probability that a "good" random generator generates reward 1 is equal to $1/2 + aT^{1/2}K^{1/2}$, where a is a small positive constant. All other $K-1$ generators are "bad" having equiprobable rewards 1 or 0. It is claimed [4] that there is a constant probability that the "good" generator is sampled exactly $2T/K$ times during the session of the length T and

[4]The lower bound for regret, defined against the best expected strategy, is proven to be $\Omega(lgT)$ while there exists an algorithm that achieves $O(lg^2T)$ regret for oblivious and some restricted cases of a non-oblivious teacher[10]. As we can see, achieved boundaries in this case are relatively tight.

hence that the total cumulative reward is at most $T/2$. Therefore, the difference of cumulative rewards between "bad" and "good" random generators is not big enough to discriminate generators with a sufficient confidence. This conclusion leads to $\Omega(T^{1/2}K^{1/2})$ expected regret of any algorithm on this distribution for sufficiently large K and for $T \geq K$. Another upper bound for regret, $\Omega(\sqrt{T \lg K})$, follows from the fact that learning by mixture of experts is a generalization of adversarial K-bandit problem [11].

3.2 K-armed bandit problem and modeling of a mixture of experts

K-armed bandit problem can be further generalized to model learning by a mixture of experts [23]. The task of the learner is to perform a binary classification at each trial[5] generating the class prediction $\hat{y}$ based on the probability of a positive classification provided by each of K experts. After a classification is performed, the learner is supplied with the actual class label y. Unlike the adversarial K-armed bandit problem, where the learner is supposed to obtain reward from a chosen generator, here, we assume that the teacher submits decisions and rewards for each expert. On this way the "malicious" teacher can set expert decisions to further confuse the learner. Opposite to the K-armed bandit, where only one of generators is chosen on each trial, here a mixture of expert decisions is allowed. A regret per trial ε is defined as absolute difference of the true classification and the decision of the learner: $\varepsilon = |y - \hat{y}|$.

The worst regret in the case of a mixture of experts is $\Omega(\sqrt{T \lg K})$, as shown in [11]. This means that in the worst case any mixture of experts makes at least $\Omega(\sqrt{T \lg K})$ errors more than the best expert (if an expert provides a hard 0/1 decision instead of probability). This result is achieved through the game theory methods [36], where the decision process can be described as a game between the learner and the teacher. The learner has the intention to minimize regret, while the teacher wishes to increase it as much as possible.

In addition to this theoretical boundary, a practical algorithm to achieve $\Theta(\sqrt{T \lg K})$ worst-case regret is also proposed in [11]. At each step of the algorithm, the output prediction is based on the linear combination of prob-

[5] In [4] a more general case, when learner provides fuzzy decision (the probability of a positive class label) is considered; since lower bound is valid for an arbitrary learner, it is also valid for a learner that perform a hard decision, providing probabilities 1 or 0.

abilities ξ_i that represent outputs of particular experts

$$\hat{y} = F\Big(\sum w_i \xi_i\Big). \tag{4}$$

Here, the weights w_i are updated as

$$w_i(t+1) = w_i(t)U(\varepsilon), \tag{5}$$

and normalized to have the sum equal to 1 before the next trial. $U(\varepsilon)$ can be an arbitrary function that satisfy the following condition: $\beta^{\varepsilon} \leq 1-(1-\beta)\varepsilon$, for all $\varepsilon \in [0,1]$. If we choose $U(\varepsilon) = 1-(1-\beta)\varepsilon$, the rule for updating the weights reduces to $w_i(t+1) = w_i(t) - (1-\beta)\varepsilon w_i(t)$ which resembles the LMS rule [51], where the error is multiplied by an input vector, instead of a weight. If $U(\varepsilon) = \beta^{\varepsilon}$, the update rule resembles the weighted majority[6] algorithm, where $\varepsilon \in \{0,1\}$.

Similar to $U(\varepsilon)$, shape of the function F for mapping a weighted sum of expert-predicted probabilities into the learner prediction in (4) depends on β. Since weights w_i are normalized, the weighted sum r of probabilities ξ_i in (4) is always in [0,1] range. The function F satisfies the following inequality

$$1 + \frac{\ln((1-r)\beta + r)}{2\ln(2/(1+\beta))} \leq F(r) \leq \frac{-\ln(1-r+r\beta)}{2\ln(2/(1+\beta))}, \text{ for all } r \in [0,1] \tag{6}$$

where F has the following form

$$F(r) = \begin{cases} 0, & r \leq 1/2 - c \\ 1/2 - (1-2r)/c, & 1/2 + c \leq r \leq 1/2 + c \\ 1, & \text{otherwise} \end{cases} \tag{7}$$

and parameter c is defined as

$$c = \frac{(1+\beta)\ln(2/(1+\beta))}{2(1-\beta)}. \tag{8}$$

Method for selecting the coefficient β, discussed in [11], is out of scope of this paper.

[6] In a *weighted−majority* algorithm [29] decision in each trial is performed according to the classification of hypotheses that have the maximal sum of weights. After obtaining a correct answer from teacher, weights of incorrect hypotheses decrease as multiplied by a constant learning rate $\alpha < 1$. By setting $\alpha = 0$, weightedmajority algorithm reduces on *halving*. Weighted majority algorithms with adaptive learning rates are discussed in [5].

It is important to observe that constraints (6)– (8) cannot be fulfilled by a step function, defined as $F(r) = 0$, $r \leq \Theta$ and $F(r) = 1$, $r > \Theta$. Therefore, the decision of this algorithm will always be *fuzzy*. An interesting open problem would be to determine the existence of an algorithm with 0/1 output having $\Theta(\sqrt{T \lg K})$ asymptotic behavior. Recall that the *weighted − majority* algorithm offers a mistake bound $O(M + lgK)$ where M is a mistake bound of the best algorithm in a pool of K experts.

4. Apple Tasting Model

In the mistake-bound model, a learner gets the true class label from the teacher regardless of its prediction. In contrast, in the apple tasting model [21] the learner gets the true class label from the teacher only if the example has been classified as *positive* by the learner. The name of this learning model originates from the following real world situation. When a person buys apples, she/he tries to judge about taste by *looking* at each apple: if an apple is not chosen, the person cannot know whether it is really tasty (and hence whether her/his selection was correct)[7]. The aim imposed to an apple tasting algorithm is the same as in the case of the mistake-bound model: to achieve a minimal possible loss, measured as the number of miss-classified trials in a session.

An obvious strategy for solving an apple-tasting problem is to decide a positive class in certain cases when "a common sense" governs toward negative decision and thus to examine whether "a common sense" was correct. This leads to the *generic apple − tasting algorithm*, Fig. 1.

```
Generic apple-tasting algorithm ( )
  • For each trial
    −Apply a decision algorithm A to predict a class label c.
    if c = 1,
      predict the positive class.
    else
      with probability p predict the positive class, and with
      complementary probability 1 − p predict the negative class.
```

[7]Further examples include: a bank should decide whether to issue a credit card to a person (if the card is not issued, bank cannot conclude whether the person would be a good customer); a company decides whether to send a potential customer a mail offer. If not sent, company cannot decide whether the ad would be successful and convince a customer to purchase goods.

Fig. 1. Generic apple-tasting algorithm

Using generic *apple − tasting algorithm* the learner can get information whether its decision strategy works correctly for negative classification by exploring cases on which teacher response does not provide a feedback. Here, the main issue of exploration-exploitation dilemma is to determine p. Assume that the algorithm A does not make negative mistakes, i.e. that its negative classification is always true. Then, it does not make sense to randomly predict positive when A decides negatively. On the other side, if negative mistakes are frequent, probability p should be larger. In [21], the probability p is set to $\sqrt{M_-/T}$, assuming that the number of trials T as well as performance of A is known. This leads to a $O(M_+ + 2\sqrt{M_- \cdot T})$ upper bound for expected number of errors of *generic apple − tasting algorithm*. This upper bound holds for even more sophisticated apple tasting algorithms, including an algorithm that can modify its decisions corresponding to information obtained from the teacher or an algorithm with the probability p decreasing in time as more information about the learner's ability is accumulated to correctly provide a negative classification.

An interesting problem is to estimate the optimal ratio M_+/M_- for an algorithm A embedded in *generic apple−tasting algorithm*. In other words, given a mistake bound $M_+ + M_-$ for an algorithm A, the optimal balance between negative and positive mistakes should be determined. Given M_- negative mistakes, the lower bound for positive mistakes M_+ is established as $M_+ \geq M_- \left(1/e \cdot |C|^{1/(M_- - 1)} - 1\right)$, where $|C|$ is the size of the hypothesis class[8]. Let us define algorithm $A(k)$ as on Fig.2.

```
A(k)
  If k = 0
    predict positive if and only if at least one hypothesis,
    from the set of hypotheses C_t consistent with the previous
    trials, predicts positive;
  else
    if A(k) did not make false-negative mistake
      predicts positive if and only if at least Ψ_k(Ψ_{k-1}^{-1}(| C |))
      hypotheses[1] in C_t predict positive;
    else call A(k − 1);
```

Fig. 2. A recursive decision algorithm embedded in generic apple-tasting algorithm.

[8] This theory is applicable only on finite classes of hypotheses.

It is easy to see that $A(0)$ does not make false-negative mistakes. If at least one of hypotheses from C_t votes is positive, $A(0)$ decides positive[9] so there is no risk that we might incorrectly perform a negative decision. By mathematical induction $A(k)$ can be concluded to make no more than $k = M_-$ false-negative mistakes. Also by induction can be shown that $A(k)$ makes at most $M_-\left(|C|^{1/(M_-1)} - 1\right)$ positive missclassifications[10] and also that an expected mistake bound of $O\left(\ln(1 + \ln|C|/\ln T)\sqrt{T\ln|C|/\ln T}\right)$ errors in T trials can be achieved. The main problem in a practical realization of A is that it must evaluate and count outcomes of all hypotheses in set C_t, which can have the exponential size.

As we have seen, an upper mistake bound can be established by construction of an apple- tasting algorithm based on a standard algorithm for on-line learning with mistake bounds. In contrast, by construction of a standard mistake bounds algorithm based on an apple- tasting algorithm, it is possible to compute a lower bound for the maximal expected number of mistakes in any apple-tasting algorithm. In [21], for any standard algorithm on a finite set of samples and a defined hypothesis class, if there are either at least M_- false-negative or M_+ false-positive errors, it is proven that the maximal expected number of errors in an apple tasting problem is $\Omega(\min(M_+, M_-\lfloor T/(M_+ + M_- - 1)\rfloor))$. Using generalized mistake bounds and this reduction technique, tight bounds $\Theta\left(\min(T, |C|, \sqrt{T\ln|C|/\ln(1 + T/\ln|C|)}\right)$ are obtained in [28][11].

Observe that upper and lower bounds mentioned above are rather pessimistic: they are valid even for a "malicious" teacher (gives always the most difficult examples). Nevertheless, if there is no adverse influence of learner's behavior on the sequence of examples submitted by a teacher, i.e. if a learner receives training samples randomly, it can be proven that the expected mistake bound of an apple-testing algorithm is $O\left(\sqrt{VC(C)\cdot T}\right)$ for finite-size hypothesis classes (here, $VC(C)$ denotes the VC dimension of a hypothesis class discussed in Section 2).

One of open problems in an apple tasting model of learning is the poten-

[9] Crucial assumption here is that at least one hypothesis from the class $|C|$ is consistent with all examples.

[10] Here, ratio M_+/M_- decreases with an increase in M_-, whereas in the case of Winnow algorithm [27] the ratio is practically constant as $M_+ \leq (1/\alpha - \alpha)/\alpha \cdot M_- + \Theta(1/\alpha)$ where $\alpha < 1$ is a learning parameter. This implies that Winnow algorithm is probably not appropriate for embedding in an apple-tasting algorithm.

[11] Observe similarity of these bounds and the regret bound for K-bandit problem, which opens the question of similarity of these two problems that is briefly discussed in [21].

tial computational complexity improvement for theoretic algorithms primarily used to prove upper error bounds. The challenge is to find an approximate algorithm with similar error bounds, but with a much smaller complexity. If grouping consistent hypotheses according to their classification of a particular training sample were possible without actual hypotheses evaluation on the sample, the proposed algorithm $A(k)$ would become feasible regardless of the hypothesis class size.

Intuitively, a learning algorithm that has a large M_+/M_- ratio will perform well as an apple tasting algorithm (since less attention should be paid to undetectable false-negative mistakes). Such an algorithm could be constructed using a generalized loss [28]. Another interesting possibility is to estimate on line the probability that a given sample is positive (or negative), and to apply cost-sensitive classifiers by setting a cost of false negative mistakes higher. A similar idea is used in [14] for an off-line classification, using bagging [9], where the predictors built on different samples randomly drawn from training set are combined in the hope that the accuracy of the ensemble of predictors is greater than the single predictors. The main problem here is to adapt bagging to on-line learning. One of possible ideas is to have for each learner in an ensemble a fixed size buffer and with some probability to enable a replacement of a random sample from the buffer with each new training sample. An additional problem in this realization is the computational complexity of ensemble algorithms.

It would be interesting to generalize apple-tasting problem for multiple choices. Another possible generalization is to introduce probabilities q_i that the teacher will respond with a correct class label when learner's decision is a certain class i (in an apple-tasting model considered here, $q_0 = 0$, $q_1 = 1$). This would lead to real-life situations where teacher's response always occurs with some uncertainty.

5. Reinforcement Learning

Reinforcement learning [42] addresses the problem how an autonomous agent that senses and acts in its environment can learn to behave appropriately and to choose an optimal action to achieve its goals. The agent is moving through a finite state space and in each state it can perform one of the pre-specified actions. Each time the agent performs an action in its environment, a transition to a new state occurs, and the environment provides a reward or penalty to indicate the desirability of the proposed actions. For example, when training the agent to play a game, the trainer might provide a positive reward when the game is won, negative reward when it is lost, and

zero rewards in all other states. The task of the agent in reinforcement learning is to create a strategy for learning from these indirect, delayed rewards and to choose sequences of actions that produce the maximal cumulative reward.

In order to prevent an unlimited growth of cumulative reward for infinite length sessions and to introduce a real-life fact that it is good to get a reward as soon as possible, we introduce a discounting. Considering discounting, the cumulative reward is defined as

$$R = \sum_k \gamma^k r_k, \quad 0 < \gamma < 1, \tag{9}$$

where r_k is a current reward at the moment k, and γ is a constant that determines the relative value of delayed versus immediate rewards. It can be observed that the idea of discounting is similar to the emphasizing of recent weights in the ordinary K-armed bandit problem.

In general, both transitions to new states due to actions and related rewards are stochastic. For each action a defined in a state s, the transition to a new state s' occurs with a pre-specified probability and results with a random *reward* that satisfies a defined probability distribution.

A general reinforcement learning problem is determined by:

- Set S of states s;
- Set A of actions a;
- Policy π, specified by probability $\pi(s,a)$ that the action a will be taken at state s. The policy is *deterministic* when for a fixed state s, $\pi(s,a)$ is a non-zero probability for exactly one action: the action in each state is uniquely determined by the chosen policy. Otherwise, the policy is *stochastic*. In addition, every policy π is specified by the probability P^a_{ss} that during the action a, the transition from state s to state s' will occur and by expected rewards $R^\pi_{ss'}$ in the transition from s to s' under the action a.

The value $V^\pi(s)$ is defined as the average cumulative reward R when starting from state s and following the transition strategy determined by policy π. The value functions $V^\pi(s)$ satisfy the following system of Bellman equations [42]:

$$V^\pi(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P^a_{ss'} \left((R^a_{ss'} + \gamma V^\pi(s') \right) \quad \text{for all} \quad s \in S \tag{10}$$

In addition, the action-value function $Q^\pi(s,a)$ is defined as the expected cumulative reward if the action a is taken in state s and followed by actions determined by policy π.

To properly model a problem using reinforcement learning paradigm, the it has to satisfy the following Markov property: the next state and the current reward depend only on the current state and the currently performed action. Therefore, each state should contain all information from the past that is relevant to the decision of feature behavior (i.e. what action needs to be undertaken). Since K-armed bandit and similar problems can have infinite sessions and since information (rewards) from the beginning of the session may be relevant to proper decisions in future, modeling K-armed bandit problem with reinforcement learning seems to be improper. It is an open problem whether reinforcement learning can be reduced to K-armed bandit problem, and whether is possible to impose error bounds using such reduction. An interesting relation between reinforcement problem and mistake-bound algorithms is explored in [33].

Traditional applications of reinforcement learning involve game playing [43], robot control [13, 32, 40] and distributed agents [38], while recent research includes new applications such as on-line clustering [26], network routing [45, 52], wireless [44] and financial market [35].

5.1 Properties of reinforcement learning algorithms

Standard methods for solving reinforcement learning problems consist of two steps [42]:

- Policy evaluation, where the value functions $V^{\pi}(s)$ (or action-value functions $Q^{\pi}(s,a)$) are determined for the current policy π;
- Policy improvement (exploration), where the policy π is updated towards the higher levels of the value function or the action-value function.

It is interesting to notice that this schema is similar to expectation-maximization algorithm [15] since the value computation in the policy evaluation phase is followed by the maximization in the improvement phase. The policy evaluation is typically performed through an iterative process, stopped when either the update in successive iterations is smaller than a pre-specified value or the pre-specified number of iterations is reached. The extreme case of the later stopping criterion, known as generalized policy iteration (GPI), occurs when only one iteration is performed in a policy evaluation phase and it is frequently exploited in reinforcement learning algorithms.

Reinforcement learning algorithms can be designed to perform either *on-policy* or *off-policy*. When in *on-policy* regime, the learning algorithms attempt to evaluate and/or improve the policy currently used for making decisions. In contrast, when perform *off-policy*, the learning methods can

estimate value functions of one policy while following another one in the state transitions. The off-policy techniques can explore larger classes of policies but typically perform worse in on-line learning, as compared to on-policy methods.

According to the policy evaluation method, three major classes of reinforcement learning algorithms can be distinguished: dynamic programming (DP) methods, Monte-Carlo (MC) methods and Temporal Difference (TD) learning.

Dynamic programming. In this method, policy evaluation is based on an iterative solution of Bellman equations (10), while policy improvement is accomplished by a greedy approach, where a policy is chosen to maximize the current action-value function. The time complexity of dynamic programming (DP) methods in a worst case is a polynomial in number of states and actions as opposite to the extensive search, which is exponential in the number of states. However, the method is still not practical for problems with a huge number of states. Fig.3 presents a GPI version of DP method with one iteration per policy evaluation.

DP-GPI(A,S)

- For all states s initialize $V(s)$ to arbitrary values;
- Repeat until convergence:

  For each state s

  1. Update the estimated value
     $$V(s) = \max_a \sum_{s'} P^a_{ss'} \left(R^a_{ss'} + \gamma V^\pi(s')\right)$$
  2. Update the policy in s such that the action that maximises the value function is chosen:
     $$\pi(s) = \max_a \sum_{s'} P^a_{ss'} \left(R^a_{ss'} + \gamma V^\pi(s')\right)$$

Fig. 3. Dynamic programming with generalized policy iteration (DP-GPI) reinforcement learning algorithm.

Monte-Carlo (MC) Methods. MC methods are attractive when only action value estimates for some states are necessary to be traced. The key idea is to estimate the action-value function $Q^\pi(s,a)$ as the average of cumulative rewards obtained by repetition of the same specified action a in the state s during on-line or simulated evaluation of the current policy. The policy π, which can be stochastic or deterministic, is then adjusted to reflect the changes in the estimated action-value function.

In generic MC reinforcement learning algorithms, the policy evaluation phase consists of an infinite number of iterations and each state-action

pair can be a starting point for the session with a non-zero probability (the assumption of exploring starts). The infinite number of iterations can be reduced to one step of evaluation performed per each session using GDI-style MC methods. Releasing the exploring starts assumption leads to the on- policy methods, where the effect of random exploration of states and actions is achieved through the application of stochastic policies (Fig.4). Hence, such methods cannot guarantee that the obtained stochastic policy is the best considering overall performance. As an alternative, an off-policy version of MC algorithm can be used. It can be shown [42] that off-policy algorithm for determining optimal deterministic policy π can be obtained by applying the variant of the algorithm, presented in Fig. 4, on a stochastic policy π'.

On-policy MC-GPI(A,S)

- Choose an arbitrary stochastic policy π;
- For all states s and actions a initialize $Q(s,a)$, $\pi(s,a)$ to arbitrary values;
- Initialize set Returns(s,a)<-empty set.
- Repeat until convergence;
  1. Generate a session using policy π;
  2. Policy evaluation:
     - For each pair state s-action a appearing in session:
       - Append cumulative reward R after the first occurrence of the pair state-action into set Returns(s,a);
       - Compute a new action-value function $Q(s,a) =$ average(Returns(s,a))
  3. Policy improvement:
     - For each state s in the session:
       - Find the action $a^\star$ that maximizes $Q(s,a)$;
       - Re-adjust stochastic policy π such that the action $a^\star$ has the highest probability to occur under policy: $\pi(s,a^\star) > \pi(s,a)$ for all $a \neq a^\star$.

Fig. 4. On-policy Monte Carlo algorithm with generalized policy iteration.

Temporal difference learning. The main idea of these methods is to update the action-value function $Q(s,a)$ (or the value function $V^\pi(s)$) during each state transition from s to s', using a current reward r and the functions $Q(s',a)$ or $V^\pi(s')$ in the next state s'. The update rule is similar to the rule used in the Least Mean Square (LMS) algorithm [20]. In "on-policy" version of TD learning, shown in Fig 5, the current action value function $Q(s,a)$ is updated toward its estimated value $\hat{Q}(s,a)$. In Q-learning [42, 49, 50], as "off-policy" variant of TD learning, in each state s, instead of the current action-value function $Q^\pi(s,a)$, the information of the currently

best cumulative reward for performed action a is kept independently of the current policy. In this algorithm, the update rule for policy improvement form Fig. 5 becomes: $Q(s,a) = Q(s,a) + \alpha\big(r + \max_{a'} Q(s',a') - Q(s,a)\big)$. Additional improvements of TD learning algorithms are possible by including domain information [42] and function approximations for the efficient storing of action-value function [7] and by integrating multi-step methods [49].

On-policy TD-GPI(A,S)

- Choose an arbitrary stochastic policy π;
- For all states s and actions a initialize $Q(s,a)$, $\pi(s,a)$ to arbitrary values;
- Repeat until convergence;

1. Generate a session using policy π. Initialize s as the first state in session and choose action according to the policy.
2. For each step in session while s is not the last state in a session

2a. Policy evaluation:
- Take action a and observe current reward r and the next state s';
- Choose next action a' in s' using the policy derived from Q;
- Compute the estimate $\hat{Q}(s,a)$ of the action-value function as: $\hat{Q}(s,a) = r + \gamma Q(s',a')$;

2b. Policy improvement:
- Update $Q(s,a)$ toward $\hat{Q}(s,a)$: $Q(s,a) + \alpha\big(\hat{Q}(s,a) - Q(s,a)\big)$, where α is a learning rate;
- Find action $a^\star$ that maximize $Q(s,a)$
- Re-adjust stochastic policy π such that action $a^\star$ has the highest probability to occur ($\pi(s,a^\star) > \pi(s,a)$ for all $a \neq a^\star$);

2c. Iterate: $s = s'$, $a = a'$.

Fig. 5. On-policy temporal difference algorithm with generalized policy iteration (TD-GPI).

¿From Table 1, where the properties of presented reinforcement learning algorithms are summarized, it can be observed that dynamic programming methods can operate by estimating and updating the value functions $V^\pi(s)$ only when all model parameters (transition probabilities $P^a_{ss'}$ and expected rewards $R^\pi_{ss'}$) are known. In such scenarios, DP methods can also perform off-line. In contrast, only sample sequences of states, actions and current rewards are needed for Monte Carlo (MC) and Temporal Difference (TD) methods that operate by keeping track of the action-value functions $Q^\pi(s,a)$ instead of the value functions $V^\pi(s)$.

When reinforcement learning algorithms perform bootstrapping, the up-

Table 1: Comparison of major reinforcement learning algorithms.

Algorithm	Known model parameters	Off-line learning	Bootstraping	Step-wise Policy Update	Off-Policy learning	Convergence
Dynamic (DP) Programming	Yes	Yes	Yes	Yes	—	Guaranteed
Monte-Carlo (MC)	No	No	No	No	Yes	No proof for GPI methods
Temporal difference (TD)	No	No	Yes	Yes	Yes	Guaranteed

dates of the value functions or the action value functions at any state s depend on the estimates from other states. DP and TD methods belong to this category of algorithms, while in MC methods, the value functions or the action value functions are estimated and updated at each state independently. While policy evaluation and policy improvement for MC methods are intermixed on a session-by-session basis, in DP and TD methods the updated values and policies at one state are processed step-wise and can be immediately exploited for the next state transition. Therefore, TD methods are typically more suitable than MC algorithms when learning occurs in long sessions.

Convergence of major reinforcement learning algorithms has been extensively discussed in literature [39, 42, 50]. The convergence of dynamic programming methods guarantees achieving the optimal deterministic policy and follows from the policy improvement theorem defined in [42]. The theorem states that if action a, performed according to the new deterministic policy π'in state s $(a = \pi'(s))$ results in the higher action-value function than the current value function for all states s, then the new policy π' has higher value than the older one

$$\forall s, Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \Rightarrow \forall s, V^{\pi'}(s) \geq V^{\pi}(s) \tag{11}$$

Using equation (11), it can be shown that the convergence is achieved when DP values in the algorithm do not change in two successive sessions. Using the policy improvement theorem, the convergence of Monte Carlo (MC) methods was proven only for the case when the assumption of the infinite number of iterations in the policy evaluation phase is satisfied [42]. However, there is no proof of convergence for on-policy and off policy MC variants

based on generalized policy iteration (GPI). In these cases, the learning is usually completed when the updates of the action-value functions in two successive sessions are smaller than a pre-specified threshold. For "on-policy" variant of temporal-difference learning, there is a proof of convergence for a sufficiently small learning rate [39]. Finally, for Q-learning, representing the off-line version of temporal-difference method, the convergence is proven in [34, 50].

5.2 Open problems in reinforcement learning

One of the main open problems in the theory of reinforcement learning [41] is related to convergence of Monte Carlo GPI algorithms. Although there is a plethora of experimental evidence towards the convergence of this method, there is still no formal proof that the algorithm converges. In addition, there is no theoretical justification whether the methods that apply bootstrapping (such as TD learning) are faster than non- bootstrapping ones (e.g. Monte Carlo methods). Again, both experimental evidence and heuristic consideration suggest that the TD methods may require less time to achieve the optimal policy in comparison to Monte Carlo methods.

Another problem in reinforcement learning is the choice of stochastic policy. While $\varepsilon - greedy$ strategy (Section 3) is widely accepted, there is no proof whether it is computationally more efficient than using other strategies developed for K-armed bandit problem (e.g. softmax or reinforcement comparison).

Additional research is also necessary to determine the theoretical relationship between off-policy and on-policy methods. Off-policy methods typically can explore wider range of possible policies while at the same time performing sub-optimal (stochastic) policy. Similar to this problem is the choice of the optimal balance between policy evaluation and policy improvement phases of GPI algorithms. Therefore, both problems can be viewed as examples of typical exploration-exploitation dilemma. While DP and TD algorithms are proven to converge when only one iteration is performed per each policy evaluation, the number of iterations stays as a parameter of choice for performance optimization.

Finally, in spite of recent attempts [24] to apply the computation learning theory to reinforcement learning paradigm, there are still a lot of open questions, which include the problems how to determine VC dimension for particular policy classes of practical importance or how to apply the boosting algorithm [16] to the reinforcement learning.

6. Conclusions

In this paper, a brief survey of on-line machine learning problems, where the exploration-exploitation dilemma naturally arises, is presented. Our emphasis was on the worst-case performance boundaries as functions of model complexity and the number of trials in on- line learning. The study also includes practical heuristics for achieving these boundaries and a good exploration-exploitation balance. Also, some open problems arising in this interesting and relatively new area are emphasized.

REFERENCES

1. D. Angluin: *Queries and concept learning.*, Machine learning, Vol. 2, 1988,pp. 319-342.
2. M. Anthony, and P. Bartlett: *Neural Network Learning, Theoretical Foundations.*, Cambridge University Press, 1999.
3. P. Auer, N. Cesa-Bianchi and P. Fischer: *Finite-time analysis of the multiarmed bandit problem.* Neuro-COLT technical report NC-TR-00-084, 2000. (also on `http://www.neurocolt.org/abs/2000/abs00084.html`).
4. P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire: *Gambling in a rigged casino: The adversarial multi-armed bandit problem.*In: Proc. of the 36th Annual Symposium on the Foundations of Computer Science, 1995, pp. 322-331.
5. P. Auer, N. Cesa-Bianchi, and C. Gentile: *Adaptive and self-confident on-line learning algorithms.* Neuro-COLT technical report NC-TR-00-083, 2000. (also on `http://www.neurocolt.org/abs/2000/abs00083.html`).
6. D. A. Berry, and B. Fristedt: *Bandit Problems.* Chapman & Hall, 1985.
7. D. P. Bertsekas, and J. N. Tsitsiklis: *Neuro-Dynamic Programming.* Athena Scientific, 1996.
8. A. L. Blum: *Separating distribution-free and mistake-bound learning models over the boolean domain.* SIAM Journal on Computing, Vol. 23, No. 5, 1994, pp. 990-1000.
9. L. Breiman: *Bagging predictors.* Machine Learning, Vol. 24, No. 2, 1996, pp. 123-140.
10. N. Cesa-Bianchi, and P. Fischer: *Finite-time regret bounds for the multiarmed bandit problem.* In: Proc. of 15th International Conference on Machine Learning, Morgan Kaufmann, 1998, pp. 100-108.
11. N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, D. Haussler, R. Schapire, and M.K. Warmuth: *How to use expert advice,.* Journal of the ACM, Vol. 44, No.3, 1997, pp. 427-485.
12. T. Cormen, C. Leiserson, and R. Rivest: *Introduction To Algorithms.* MIT Press, 1990.
13. R. H. Crites, and A. G. Barto: *Improving elevator performance using reinforcement learning.* In: (D. Touretzky, M. Mozer, and M. Hasselmo, eds.) Proc. Advances in Neural Information Processing Systems 8, 1996.
14. P. Domingos: *MetaCost: A general method for making classifiers cost-sensitive.* In: Proc. Fifth international conference on knowledge discovery and data mining, San Diego, 1999, pp. 155-164.

15. R. Duda, P. Hart, and D. Stork: *Pattern Classification.* 2nd edn, John Wiley and Sons, 2001.

16. Y. Freund, and R. Schapire: *A decision-theoretic generalization of on-line learning and an application to boosting.* Journal of computer and system science, Vol. 55, No. 1, 1997, pp. 119-139.

17. J. C. Gittins: *Multi-Armed Bandit Allocation Indices.* John Wiley, 1989.

18. D. Goldberg: *Genetic Algorithms in Search.* Optimization and Machine Learning, Addison-Wesley, 1989.

19. R. L. Haupt, and S. E. Haupt: *Practical Genetic Algorithms.* John Wiley & Sons, 1998.

20. S. Haykin: *Neural Networks: A Comprehensive Foundation.* 2nd ed, PrenticeHall, 1999.

21. D. Helmbold, N. Littlestone, and O. Long: *Apple tasting and nearly one-sided learning.* In: Proc. 33rd Annual Symposium on the Foundations of Computer Science, 1992, pp. 493-502.

22. J. Holland: *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1992.

23. R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton: *Adaptive mixtures of local experts.* Neural Computation, Vol. 3, 1991,pp. 79-87.

24. M. Kearns, Y. Mansour, and A. Ng: *A sparse algorithm for near-optimal planning in large markov decision processes.* In: Proc. IJCAI 1999, pp. 1324-1231.

25. M. Kearns, and U. Vazirani: *An Introduction to Computational Learning Theory.* MIT press, 1994.

26. A. Likas: *A reinforcement learning approach to on-line clustering.* Neural Computation 2001, to appear(also on `http://www.cs.uoi.gr/~arly/papers/rcluster.ps.gz`).

27. N. Littlestone: *Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm.* Machine learning, Vol. 2, 1987, pp. 285-318.

28. N. Littlestone, and P.Long: *On-line learning with linear loss constraints.* In: Proc. of the Sixth Annual Conference on Computational Learning Theory, 1993, pp. 412-421.

29. N. Littlestone, and M.Warmuth: *The weighted majority algorithm.* Information and Computation, Vol. 108, No. 2, 1994, pp. 212-261.

30. W. Maass: *On-line learning with an oblivious environment and the power of randomization.* In: Proc. 4th Annual ACM Workshop on Computational Learning Theory, Morgan Kaufmann, 1991, pp. 167-175.

31. W. Maass, and G. Turan: *Lower bound methods and separation results for on-line learning models.* Machine learning, Vol. 9, 1992, pp. 107-145.

32. S. Mahadevan, and J. Connell: *Automatic programming of behavior-based robots using reinforcement learning.* Artificial Intelligence, Vol. 55, 1992, pp. 311-365.

33. Y. Mansour: *Reinforcement learning and mistake bounded algorithms.* In: Proc. COLT 1999, pp.183-192.

34. T. Mitchell: *Machine Learning.* McGraw Hill 1997.

35. J. Moody, and M. Saffell: *Reinforcement learning for trading systems.* In: (M. S. Kearns, S. A. Solla, D. A. Cohn, eds.) Proc. Advances in Neural Information Processing Systems 11, 1999.

36. R. B. Myerson: *Game Theory: Analysis of Conflict.* Harvard University Press, 1997.

37. R. Rivest: *Lecture notes for machine learning course.* MIT 1994, `http://www.toc.lcs.mit.edu/ mona/lectures.html`.

38. C. Shelton, C. Isbell, M. Kearns, S. Singh, and P. Stone: *A social reinforcement learning agent.* In: Proc. Fifth. Int. conf. on Autonomous agents (Agents-2001) 2001, to appear.(also on `http://www.research.att.com/ mkearns/`).

39. S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári: *Convergence results for single-step on-policy reinforcement-learning algorithms.* Machine Learning, Vol. 38, No. 3, 2000, pp. 287-308.

40. P. Stone, T. Balch, and G. Kreatzschmarr (eds.): *RoboCup-2000: Robot soccer World Cup IV.* Springer Verlag, Berlin, 2001. (to appear).

41. R. Sutton: *Open theoretical questions in reinforcement learning.* In: Proc. Euro COLT'99, to appear in Springer Lecture Notes in Artificial Intelligence. (also on `ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/sutton-99.ps`).

42. R. Sutton, and A. Barto: *Reinforcement Learning, an Introduction.* A Bradford Book, MIT Press, 1998.

43. G. Tesauro: *Temporal difference learning and TD-Gammon.* Communications of the ACM, Vol. 38, No. 3, 1995, pp. 58-68. (also on `http://www.research.ibm.com/massive/tdl.html`).

44. X. Timothy: *Low Power Communications via Reinforcement Learning.* In: (S. A. Solla, T. K. Leen, K.-R. Müller, eds.) Proc. Advances in Neural Information Processing Systems 12, 2000.

45. K. Tumer and D. Wolpert: *Collective intelligence and Braess' paradox.* In the Proc. Seventeenth National Conference on Artificial Intelligence, Austin, 2000, pp. 104-109.

46. V. N. Vapnik: *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

47. V. N. Vapnik: *Statistical Learning Theory.* Wiley, 1998.

48. V. N. Vapnik and A. Y. Chervonenkis: *On the uniform convergence of relative frequencies of events to their probabilities.* Theory of Probab. and its Applications, Vol. 16, No. 2, 1971, pp. 264-280.

49. C. Watkins: *Learning with delayed rewards.* PhD thesis, Cambridge University, Department of Computer Science, Cambridge, England, 1989.

50. C. Watkins and P. Dayan: *Q-learning.* Machine Learning, Vol.8, 1992, pp. 279-292.

51. B. Widrow, and S. Stearns: *Adaptive Signal Processing.* Prentice-Hall 1995.

52. D. Wolpert, S. Kirshner, C. Merz, and K. Tumer: *Adaptivity in agent-based routing for data networks.* In: Proc. Fourth International Conference on Autonomous Agents Barcelona, Spain, 2000, pp. 396-403.