Exact Learning Algorithms, Betting Games, and Circuit Lower Bounds[⋆]

Ryan C. Harkins and John M. Hitchcock

Department of Computer Science,
University of Wyoming

Abstract. This paper extends and improves work of Fortnow and Klivans [5], who showed that if a circuit class $\mathcal{C}$ has an efficient learning algorithm in Angluin's model of exact learning via equivalence and membership queries [2], then we have the lower bound $\mathrm{EXP}^{\mathrm{NP}} \not\subseteq \mathcal{C}$. We use entirely different techniques involving betting games [4] to remove the NP oracle and improve the lower bound to $\mathrm{EXP} \not\subseteq \mathcal{C}$. This shows that it is even more difficult to design a learning algorithm for $\mathcal{C}$ than the results of Fortnow and Klivans indicated.

1 Introduction

We continue the line of research basing hardness of learning results on computational complexity and cryptography (see for example [16,8,1,9]). Fortnow and Klivans [5] consider Angluin's model of exact learning from equivalence and membership queries [2]. In an equivalence query, the learner presents a hypothesis and asks if it is equivalent to the unknown target concept. If the hypothesis is equivalent, the learner has succeeded; otherwise, the learner is given an example on which the hypothesis is incorrect. In a membership query, the learner chooses an example and asks the value of the target concept on that example. To succeed, the learner must exactly identify the target concept. Fortnow and Klivans show that learning algorithms for a circuit class give lower bounds for that type of circuit in the class $\mathrm{EXP}^{\mathrm{NP}}$. Throughout this introduction, $\mathcal{C}$ denotes any nonuniform class of polynomial-size circuits. (Several results will be stated informally here but made precise in the body of the paper.)

Theorem 1.1. (Fortnow and Klivans [5]) *If there is an efficient exact learning algorithm for $\mathcal{C}$ using equivalence and membership queries, then* $\mathrm{EXP}^{\mathrm{NP}} \not\subseteq \mathcal{C}$.

While "efficient" typically means a polynomial-time algorithm, the result of Fortnow and Klivans as well as our results allow for exponential time and subexponentially many queries (i.e. $(2^{O(n)})$ time and $2^{n^{o(1)}}$ queries).

[⋆] This research was supported in part by NSF grants 0652601 and 0917417 and by an NWO travel grant. Part of this research was done while Hitchcock was on sabbatical at CWI.

For the case of exact learning algorithms that only make equivalence queries, a stronger lower bound follows immediately from results [10,6] connecting resource-bounded measure and dimension with Littlestone's model of online mistake-bound learning [11]. Combining these results with the fact that exact learning with equivalence queries is the same as online mistake-bound learning, we have the following, which was also noted in [5].

Theorem 1.2. (Hitchcock [6]) *If there is an efficient exact learning algorithm for $\mathcal{C}$ using equivalence queries, then* EXP $\not\subseteq \mathcal{C}$.

Given these two theorems, it is natural to ask whether we can prove a lower bound in EXP assuming the learning algorithm makes both equivalence and membership queries. Where does the NP oracle come from in Theorem 1.1? The proof of Theorem 1.1 separates into two parts, giving an indirect diagonalization. Assume that $\mathrm{EXP}^{\mathrm{NP}} \subseteq \mathcal{C}$.

(i) Because $\mathcal{C} \subseteq$ P/poly, $\mathrm{EXP}^{\mathrm{NP}}$ collapses and reduces to the Permanent [3,7,15].
(ii) Use the exact learning algorithm to learn the $\mathcal{C}$ circuits for the Permanent. An NP oracle is used to answer the equivalence queries. This yields a P^{NP} algorithm for the Permanent.

Combining (i) and (ii) gives $\mathrm{EXP}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}}$, a contradiction. Therefore we see that the NP oracle in Theorem 1.1 is for the equivalence queries. In contrast, no NP oracle is required in Theorem 1.2 where equivalence queries are allowed. The proof of Theorem 1.2 relies on martingales and a more direct measure-theoretic diagonalization, which is very different than the double-collapse argument and indirect diagonalization used for Theorem 1.1. This suggests hope that a more direct diagonalization approach may yield the desired improvement.

To improve Theorems 1.1 and 1.2, we must simulate the learning algorithm's queries while performing our diagonalization. Following [6], consider implementing this in the form of a martingale. The equivalence queries are no problem. We simply use the transformation in [11] that converts an equivalence query algorithm to a mistake-bound learning algorithm and apply the technique from [6]. We can therefore focus our effort on the membership queries. Unfortunately, we have been unable to discover a method for simulating membership queries in a martingale, due to the stringent requirement that a martingale must bet on all strings in lexicographic order. However, there is an extension of martingales called betting games that do help.

Buhrman et al. [4] introduced betting games to define a generalization of resource-bounded measure with applications to autoreducibility and the BPP vs. EXP problem. Betting games and martingales are similar; the difference is how the betting strategy is allowed to operate. A martingale is required to consider the strings at each length in lexicographic order. Betting games lift this requirement by allowing the betting strategy to pick the next string that it will bet on. We can easily simulate a membership query in a betting game – the strategy simply asks to make a prediction on the queried string and then it gets to see the answer for that string. This allows for a more powerful diagonalization and it suffices for our purposes:

Theorem 1.3. *If there is an exact learning algorithm for $\mathcal{C}$ using equivalence and membership queries, then there is a betting game which succeeds on $\mathcal{C}$ in the sense of [4].*

Buhrman et al. showed that it is also possible to diagonalize within EXP against betting games [4], just as is the case for martingales [12]. Formally, no betting game can succeed on all of EXP. Hence the desired improvement, our main result, follows from Theorem 1.3:

Theorem 1.4. *If there is an exact learning algorithm for $\mathcal{C}$ using equivalence and membership queries, then* $\mathrm{EXP} \not\subseteq \mathcal{C}$.

This results shows that designing an exact learning algorithm for many circuit classes $\mathcal{C}$ will be difficult, as for most classes it is an open problem whether $\mathrm{EXP} \subseteq \mathcal{C}$.

This paper is organized as follows. Precise technical definitions for betting games and exact learning are given in section 2. We construct betting games from exact learning algorithms in section 3.

2 Preliminaries

We use standard notation. The binary alphabet is $\Sigma = \{0,1\}$, the set of all binary strings is Σ^*, the set of all binary strings of length n is Σ^n, and the set of all infinite binary sequences is Σ^∞. The empty string is denoted by λ. We use the standard enumeration of strings, $\lambda, 0, 1, 00, 01, \ldots$, and a total ordering of strings corresponding to this enumeration. A language A can alternatively be seen as a subset of Σ^*, or as an element of Σ^∞ via identification with its characteristic sequence.

2.1 Betting Games

Betting games, which are also called nonmonotonic martingales, originated in the field of algorithmic information theory. In that setting they yield the notion of Kolmogorov-Loveland randomness (generalizing Kolmogorov-Loveland stochasticity) [14,13]. The concept was introduced to computational complexity by Buhrman et al. [4]. Our notation for betting games is taken predominantly from Merkle et al. [13]. First, for comparison, we recall the definition of a martingale:

Definition 2.1. *A* martingale *is a function $d : \Sigma^* \to [0, \infty)$ such that for all $w \in \Sigma^*$, we have the following averaging condition:*

$$d(w) = \frac{d(w0) + d(w1)}{2}.$$

Intuitively, a martingale is betting in order on the characteristic sequence of an unknown language. The martingale starts with finite initial capital $d(\lambda)$. The quantity $d(w)$ represents the current capital the martingale has after betting on the first $|w|$ bits of a sequence that begins with w. The quantities $\pi(w,0) = d(w0)/2d(w)$ and $\pi(w,1) = d(w1)/2d(w)$ represent the fraction of its current capital that the martingale is waging on 0 and 1, respectively, being the next bit of the sequence. This next bit is revealed and the martingale has $d(w0) = 2\pi(w,0)d(w)$ in the case of a 0 and $d(w1) = 2\pi(w,1)d(w)$ in the case of a 1.

Betting games are similar to martingales but have an additional capability of selecting which position in a sequence (or which string in a language) to bet upon next. Because of this added complexity, it is simpler to break the description of a betting game into pieces.

Definition 2.2. *A* betting game *is a system that bets nonmonotonically on an infinite sequence (or a language) and formally is a triple $G = (s, \pi, V)$ consisting of a scan rule s, a stake function π, and a capital function V.*

1. *A* finite assignment *is a sequence $x \in (\Sigma^* \times \Sigma)^*$. In essence, it is a list of strings examined thus far, each string coupled with an assignment, saying whether or not it is included in the language being bet upon. The set of all finite assignments is denoted FA.*
2. *The* scan rule *is a (partial) computable function $s : FA \to \Sigma^*$ from finite assignments to strings that looks at a finite assignment and determines the next string (or bit of a sequence) to bet on. The scan rule is limited in that it cannot select a string that already appears in the current finite assignment.*
3. *The* stake function *is a partial function $\pi : FA \times \Sigma \to [0,1]$. Its function is to examine the current finite assignment and determine what fraction of the capital to bet on either side. It carries a condition that $\pi(w,0) + \pi(w,1) = 1$.*
4. *The* capital function *is a partial function $V : FA \to [0,\infty)$ from finite assignments to nonnegative reals, and utilizes the stake function π. Its initial capital $V(\lambda)$ is finite. When $V(w)$ is defined, the scan rule $s(w)$ determines the next string to bet on, and $\pi(w,b)$ is the stake amount, the capital is updated according to the rule*

$$V(w \cdot (s(w), b)) = 2\pi(w,b)V(w). \tag{2.1}$$

A betting game's capital function also satisfies an averaging condition, in analogy with the definition of a martingale:

$$V(w) = \frac{V(w \cdot (s(w),0)) + V(w \cdot (s(w),1))}{2}.$$

Note that a betting game is a martingale when the scan rule always selects the next string in the lexicographic order.

Definition 2.3. *If a betting game G earns unbounded capital on a language A (in the sense that for every constant c there is a point at which the capital function exceeds c when betting on A), we say that G* succeeds *on A. The* success set *of a betting game G, denoted $S^\infty[G]$, is the set of all languages on which G succeeds.*

The ability of the betting game to examine a sequence nonmonotonically makes determining its running time complicated, since each language can induce a unique computation of the betting game. In other words, the betting game may choose to examine strings in different orders depending upon the language it is wagering against. Buhrman et al. looked at a betting game as an infinite process on a language, rather than a finite process on a string. They used the following definition:

Definition 2.4. *A betting game G runs in time $t(n)$ if for all languages A, every query of length n made by G occurs in the first $t(n)$ steps of the computation.*

Specifically, once a $t(n)$-time-bounded betting game uses $t(n)$ computational steps, its scan rule cannot go back and select any string of length n. We remark that in the results of this paper all betting games have the special form that they bet on all strings of each length before moving on to the next length, so the technical issue of measuring the run time is not important for this paper. In any case, the crucial result is that no exponential-time betting game is successful against the class EXP $=$ DTIME$(2^{n^{O(1)}})$.

Theorem 2.5. (Buhrman et al. [4]) *No $2^{n^{O(1)}}$-time betting game succeeds on* EXP*.*

In our results, we often refer to $2^{O(n)}$-time bounds for sake of comparison to the results of Fortnow and Klivans [5]. We note that our results also hold for $2^{n^{O(1)}}$-time bounds, but for simplicity we prefer to not focus on such minor issues in this paper.

2.2 Exact Learning

In general, a learning algorithm seeks to identify an unknown concept from some known class of concepts. We now review the basic notation and definitions for the exact learning model.

A *concept* is a Boolean function $c_n : \Sigma^n \to \Sigma$. For any string $x \in \Sigma^n$, if $c_n(x) = 1$, then x is positively classified as belonging to the concept, while if $c_n(x) = 0$, then x is classified as not belonging to the concept. A string x paired with the classification $c_n(x)$ is called an *example.* A concept c_n is often identified with the set of positive examples $\{x \mid c_n(x) = 1\} \subseteq \Sigma^n$. A *concept class* $\mathcal{C}_n$ is a set of concepts over Σ^n. A *concept class family* is a sequence $\mathcal{C} = \{\mathcal{C}_n\}_{n \geq 0}$ of concept classes.

A learning algorithm tries to identify a *target concept* drawn from $\mathcal{C}_n$, and often does this by forming a *hypothesis*, which is typically some concept in $\mathcal{C}_n$ that is consistent with (i.e. classifies correctly) all the examples seen thus far. In the exact learning paradigm, a learner $\mathcal{A}$ may make various sorts of queries to a teacher, and then, depending on the answers, formulate a hypothesis. This process repeats until $\mathcal{A}$ has successfully discovered the target concept. We will focus on two types of queries: equivalence queries and membership queries.

Definition 2.6. *An* equivalence query *is a request to the teacher to know if the current hypothesis matches the target concept. If the answer is yes, the teacher responds accordingly. If the answer is no, then the teacher provides the learner with a counterexample (an example that is incorrectly classified by the current hypothesis).*

Definition 2.7. *A* membership query *is a request to the teacher to know the classification of a specific string x. The teacher responds with $c_n(x)$, where c_n is the target concept.*

3 Exact Learning and Betting Games

Theorem 3.1. *Let $\mathcal{C} = \{\mathcal{C}_n \mid n \in \mathbb{N}\}$ be a concept class family, and let*

$$X = \{A \mid (\exists^\infty n) A_{=n} \in \mathcal{C}_n\}\,.$$

If there is an exact learning algorithm for $\mathcal{C}$ that learns each $\mathcal{C}_n$ in time 2^{cn} and makes no more than 2^{n-2} equivalence and membership queries, then there exists a betting game G that runs in time $O(2^{(c+2)n})$, such that $X \subseteq S^\infty[G]$.

Proof. Let $\mathcal{A}$ be the learning algorithm that learns concepts in $\mathcal{C}$. In other words, for each $n \in \mathbb{N}$, and for any target concept $c_n \in \mathcal{C}_n$, $\mathcal{A}$ can learn c_n using no more than 2^{cn} time, and making at most 2^{n-2} equivalence and membership queries.

Let $G(s, \pi, V)$ be described as follows. G effectively runs in stages, examining strings by length, betting on all strings of length n before betting on any string of size $n+1$. This is proper, since $\mathcal{A}$ learns concepts taken from $\mathcal{C}_n$, whose concepts only classify strings of length n. Therefore, we will apply two subscripts to the stages of calculation, the first to indicate string length, and the second to indicate how many stages have been executed at the string length.

G starts with capital $V_{0,0} = 2$, but it treats its capital as divided up into an infinite number of amounts, 2^{-n} for each n. Thus at each stage $(n, 0)$, the capital effectively is $V_{n,0} = 2^{-n}$ (with all previous winnings "banked" and untouchable). To reflect this, we will divide π in a class of functions $\{\pi_n\}_{n \geq 0}$, so that π_n only touches the capital $V_{n,i}$.

At each stage (n, i), the scan rule s runs the learning algorithm $\mathcal{A}$, which updates its current hypothesis $h_{n,i}$. In the process of formulating $h_{n,i+1}$, one of two things will happen:

- $\mathcal{A}$ will make a membership query to some string $x \in \Sigma^n$
- $\mathcal{A}$ will make an equivalence query

If $\mathcal{A}$ makes a membership query to x, s then checks to see if x occurs in w, the current finite assignment. If so, then s answers the query according to the label. If not, then we set $s(w) = x$ and $\pi_n(w, 0) = \pi_n(w, 1) = 1/2$. In other words, the betting game selects x and makes no bet on the outcome. Once the label for x is revealed, the finite assignment is updated ($w = w \cdot (x, b)$, where b is the label for x). The scan rule then provides the correct classification of x to $\mathcal{A}$ as the answer to the membership query. The betting game's capital is unchanged and the computation then proceeds onto stage $(n, i+1)$.

If $\mathcal{A}$ makes an equivalence query, then s proceeds in an online fashion. First, the scan rule selects $s(w) = x$, where x is the lexicographically least string in Σ^n that does not appear in w. The stake function computes the prediction $b = h_{n,i}(x)$ using the current hypothesis $h_{n,i}$ and sets $\pi_n(w, b) = 3/4$ and $\pi_n(w, 1-b) = 1/4$. The true classification of x is revealed and $V_{n,i}$ is updated according to (2.1). If $c_n(x) \neq h_{n,i}(x)$, then $(x, 1-b)$ is presented to $\mathcal{A}$ as a counterexample, and computation proceeds to stage $(n, i+1)$. If $c_n(x) = h_{n,i}(x)$, then computation proceeds to stage $(n, i+1)$, and $\mathcal{A}$ is treated as still making an equivalence query. This process repeats until either a counterexample is discovered or the strings of size n are exhausted.

Without loss of generality, we will assume that $\mathcal{A}$ always finishes with an equivalence query to ensure correctness of its hypothesis. Thus if $\mathcal{A}$ has formed the correct hypothesis, G will continue searching for a counterexample until all strings of length n are exhausted, and G moves onto $(n+1, 0)$, where it utilizes $\mathcal{A}$ to learn a new concept.

To examine the running time of G, we note first that $\mathcal{A}$ updates its hypothesis in 2^{cn} time. The remaining time is allotted to the scan rule, which takes only time linear in the size of the current finite assignment (which has size at most 2^{n+1}) to determine a string to select, and to updating the capital function, which can be done in time $O(2^n)$. Hence the aggregate time for G to make all queries of size n is $O(2^n \cdot 2^{cn} \cdot 2^n)$. Therefore G is an $O(2^{(c+2)n})$-time betting game.

To see that G succeeds on X, it suffices to show that for infinitely many n, $V_{n,2^n} \geq 1$. We know that for any $A \in X$, there exist infinitely many n such that $A_{=n} \in \mathcal{C}_n$, and hence for infinitely many n, $\mathcal{A}$ will either correctly learn $A_{=n}$, or at least will classify correctly a sufficiently large number of strings in $A_{=n}$. Thus it suffices to show that for any sufficiently large n, $\mathcal{A}$ learns sufficiently quickly enough to bring in large earnings.

The worst case occurs if all 2^{n-2} queries are equivalence queries, to which a counterexample is ultimately found for each query. (If we allow for membership queries, the bet for each query is 0, and so the capital does not change regardless of the true label. The counterexample to the equivalence query, though, will decrease capital.) Since, by definition, each string of length n will be queried, we have:

$$V_{n,2^n} = V_{n,0} \cdot \left(\frac{1}{2}\right)^{2^{n-2}} \cdot \left(\frac{3}{2}\right)^{3 \cdot 2^{n-2}} = 2^{-n} \left(\frac{27}{16}\right)^{2^{n-2}} \geq 1$$

Hence for infinitely many n, G will "bank" one dollar, and therefore its earnings will increase unbounded. Therefore, $X \subseteq S^\infty[G]$.

Our improvement to the result of Fortnow and Klivans now follows as an immediate corollary to Theorems 2.5 and 3.1:

Corollary 3.2. *Under the assumptions of Theorem 3.1,* EXP $\not\subseteq X$.

Corollary 3.3. *Let C be any subset of* P/poly *that is exactly learnable in time* 2^{cn}*, making at most* 2^{n-2} *equivalence and membership queries. Then* $\mathrm{EXP} \not\subseteq C$*. Specifically, if* P/poly *is learnable under these assumptions, then* $\mathrm{EXP} \not\subseteq \mathrm{P/poly}$.

We remark that throughout this section, the stronger results where either EXP is replaced by $\mathrm{E} = \mathrm{DTIME}(2^{O(n)})$ or the learning algorithms are allowed to run in $2^{n^{O(1)}}$ time also hold via the same proofs.

References

1. Aizenstein, H., Hegedüs, T., Hellerstein, L., Pitt, L.: Complexity theoretic hardness results for query learning. Computational Complexity 7, 19–53 (1998)
2. Angluin, D.: Queries and concept learning. Machine Learning 2(4), 319–342 (1988)
3. Buhrman, H., Homer, S.: Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In: Shyamasundar, R.K. (ed.) FSTTCS 1992. LNCS, vol. 652, pp. 116–127. Springer, Heidelberg (1992)
4. Buhrman, H., van Melkebeek, D., Regan, K.W., Sivakumar, D., Strauss, M.: A generalization of resource-bounded measure, with application to the BPP vs. EXP problem. SIAM Journal on Computing 30(2), 576–601 (2001)
5. Fortnow, L., Klivans, A.R.: Efficient learning algorithms yield circuit lower bounds. Journal of Computer and System Sciences 75(1), 27–36 (2009)
6. Hitchcock, J.M.: Online learning and resource-bounded dimension: Winnow yields new lower bounds for hard sets. SIAM Journal on Computing 36(6), 1696–1708 (2007)
7. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proceedings of the 12th Annual ACM Symposium on Theory of Computing, pp. 302–309 (1980)
8. Kearns, M., Valiant, L.: Cryptographic limitations on learning Boolean formulae and finite automata. J. ACM 41(1), 67–95 (1994)
9. Kharitonov, M.: Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. J. of Comput. Syst. Sci. 50(3), 600–610 (1995)
10. Lindner, W., Schuler, R., Watanabe, O.: Resource-bounded measure and learnability. Theory of Computing Systems 33(2), 151–170 (2000)
11. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine Learning 2(4), 285–318 (1988)
12. Lutz, J.H.: Almost everywhere high nonuniform complexity. Journal of Computer and System Sciences 44(2), 220–258 (1992)
13. Merkle, W., Miller, J., Nies, A., Reimann, J., Stephan, F.: Kolmogorov-Loveland Randomness and Stochasticity. Annals of Pure and Applied Logic 138(1-3), 183–210 (2006)
14. Muchnik, A.A., Semenov, A.L., Uspensky, V.A.: Mathematical metaphysics of randomness. Theoretical Computer Science 207(2), 263–317 (1998)
15. Toda, S.: On the computational power of PP and $\oplus$P. SIAM Journal on Computing 20(5), 865–877 (1991)
16. Valiant, L.G.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)