Separating Models of Learning from Correlated and Uncorrelated Data

Ariel Elbaz, Homin K. Lee, Rocco A. Servedio*, and Andrew Wan

Department of Computer Science
Columbia University
{arielbaz,homin,rocco,atw12}@cs.columbia.edu

Abstract. We consider a natural framework of learning from correlated data, in which successive examples used for learning are generated according to a random walk over the space of possible examples. Previous research has suggested that the Random Walk model is more powerful than comparable standard models of learning from independent examples, by exhibiting learning algorithms in the Random Walk framework that have no known counterparts in the standard model. We give strong evidence that the Random Walk model is indeed more powerful than the standard model, by showing that if any cryptographic one-way function exists (a universally held belief in public key cryptography), then there is a class of functions that can be learned efficiently in the Random Walk setting but not in the standard setting where all examples are independent.

1 Introduction

It is a commonly held belief in machine learning that having access to correlated data – for example, having random data points that differ only slightly from each other – is advantageous for learning. However, we are not aware of research that rigorously validates this belief from the vantage point of the abilities and limitations of computationally efficient learning. Our work is motivated by this disparity.

We study a natural model of learning from correlated data, by considering a framework in which the learning algorithm has access to successive examples that are generated by a *random walk*. We give strong evidence that learning is indeed easier, at least for some problems, in this framework of correlated examples than in the standard framework in which no correlations exist between successive examples.

1.1 Background

In the well-known Probably Approximately Correct (PAC) learning model introduced by Valiant [16], a learning algorithm is given access to a source $EX_{\mathcal{D}}(c)$

* Supported in part by NSF CAREER award CCF-0347282 and a Sloan Foundation Fellowship.

of labelled examples each of which is drawn *independently* from a fixed probability distribution $\mathcal{D}$ over the space of possible instances. The goal of the learning algorithm is to construct (with high probability) a high-accuracy hypothesis for the target concept c with respect to $\mathcal{D}$.

Aldous and Vazirani [1] introduced and studied a variant of the PAC learning model in which successive examples are generated according to a Markov process, i.e. by taking a random walk on an (exponentially large) graph. Subsequent work by Gamarnik [8] extended this study to infinite Markov chains and gave bounds on the sample complexity required for learning in terms of the VC dimension and certain mixing properties of the underlying Markov chain. Neither [1] nor [8] considered computational issues for learning algorithms in the Random Walk framework.

In this paper we consider an elegant model of learning from Random Walk examples that is well suited for computational analyses. This model was introduced by Bartlett, Fischer and Höffgen [2] and subsequently studied by Bshouty *et al.* [6]. In this framework (described in detail in Section 2), successive examples for the learning algorithm are produced sequentially according to an unbiased random walk on the Boolean hypercube $\{0,1\}^n$. The PAC goal of constructing a high-accuracy hypothesis for the target concept with high probability (where accuracy is measured with respect to the stationary distribution of the random walk, i.e. the uniform distribution on $\{0,1\}^n$) is unchanged. This is a natural way of augmenting the model of uniform distribution PAC learning over the Boolean hypercube (which has been extensively studied, see e.g. [4, 5, 7, 12, 13, 14, 15, 17] and references therein) with the ability to exploit correlated data.

Bartlett *et al.* gave polynomial-time learning algorithms in this model for several concept classes including Boolean threshold functions in which each weight is either 0 or 1, parities of two monotone conjunctions over $x_1, \ldots, x_n$, and Disjunctive Normal Form (DNF) formulas with two terms. These learning algorithms are *proper*, meaning that in each case the learning algorithm constructs a hypothesis representation that belongs to the class being learned. Since proper learning algorithms were not known for these concept classes in the standard uniform distribution model, this gave the first circumstantial evidence that having access to random walk examples rather than uniform independent examples might bestow a computational advantage.

More recently, Bshouty *et al.* [6] gave a polynomial-time algorithm for learning the unrestricted class of all polynomial-size DNF formulas over $\{0,1\}^n$ in the Random Walk model. Since no comparable polynomial-time algorithms are known in the standard uniform distribution model (and their existence is a well-studied open question for which an affirmative answer would yield a $1000 prize, see [3]), this gives stronger evidence that the Random Walk model is strictly more powerful than the normal uniform distribution model. Thus, it is natural to now ask whether the perceived superiority of random walk learning over uniform

distribution learning can be rigorously established under some widely accepted hypothesis about efficient computation.[1]

1.2 Our Results

In this work we give such a separation, under a generic cryptographic hardness assumption, between the Random Walk model and the uniform distribution model. Our main result is a proof of the following theorem:

Theorem 1. *If any cryptographic one-way function exists, then there is a concept class over $\{0,1\}^n$ that is PAC learnable in poly(n) time in the Random Walk model but is not PAC learnable in poly(n) time in the standard uniform distribution model.*

We emphasize that the separation established by Theorem 1 is computational rather than information-theoretic. It will be evident from our construction that the concept class of Theorem 1 has poly(n) VC dimension, and thus the class can be learned using poly(n) many *examples* even in the distribution-independent PAC learning model; the difficulty is in obtaining a *polynomial-time* algorithm.

We remind the reader that while the existence of any one-way function is a stronger assumption than the assumption that P≠NP (since at this point it is conceivable that P≠NP but one-way functions do not exist), it is an almost universally accepted assumption in cryptography and complexity theory. (In particular, the existence of one-way functions is the weakest of the many assumptions on which the entire field of public-key cryptography is predicated.) We also remind the reader that all known representation-independent computational hardness results in learning theory (where any efficiently evaluatable hypothesis representation is allowed for the learning algorithm, as is the case in Theorem 1 above) rely on cryptographic hardness assumptions rather than complexity-theoretic assumptions such as P≠NP.

The rest of the paper is structured as follows: Section 2 gives necessary definitions and background from cryptography and the basics of our random walk model. Section 3 gives a partial separation, and in Section 4 we show how the construction from Section 3 can be used to achieve a total separation and prove Theorem 1.

2 Preliminaries

2.1 Notation

We denote by $[n]$ the set $\{1,\ldots,n\}$. For an n-bit string $r \in \{0,1\}^n$ and an index $i \in [n]$, the i-th bit of r is denoted $r[i]$. We write $\mathcal{U}$ to denote the uniform distribution on $\{0,1\}^n$.

[1] Note that it is necessary to make some computational hardness assumption in order to separate these two learning models. It is easy to see that if P=NP, for instance, then the concept class of all polynomial-size Boolean circuits would be efficiently learnable in both these models (as well as far weaker models), and essentially all considerations about the computational complexity of learning would become trivial.

2.2 Learning Models

Recall that a concept class $\mathcal{C} = \cup_{n\in\mathbb{N}}\mathcal{C}_n$ is a collection of Boolean functions where each $f \in \mathcal{C}_n$ maps $\{0,1\}^n \to \{0,1\}$. A *uniform example oracle* for f is an oracle $EX_U(f)$ which takes no inputs and, when invoked, outputs a pair $\langle x, f(x)\rangle$ where x is drawn uniformly and independently from $\{0,1\}^n$ at each invocation.

Definition 1 (PAC learning). *A concept class $\mathcal{C}$ is* uniform distribution PAC-learnable *if there is an algorithm A with the following property: for any n, any target concept $f \in \mathcal{C}_n$, and any $\epsilon, \delta > 0$, if A is given access to oracle $EX_U(f)$ then A runs for poly$(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ time steps and with probability $1-\delta$ outputs a Boolean circuit h such that $\Pr_{x\in\mathcal{U}}[h(x) \neq c(x)] \leq \epsilon$.*

In the (uniform) Random Walk model studied in [2, 6], a *random walk* oracle is an oracle $EX_{RW}(f)$ which, at its first invocation, outputs an example $\langle x, f(x)\rangle$ where x is drawn uniformly at random from $\{0,1\}^n$. Subsequent calls to $EX_{RW}(f)$ yield examples generated according to a uniform random walk on the hypercube $\{0,1\}^n$. That is, if x is the i-th example, the $i+1$-st example is x', where x' is chosen by uniformly selecting one of the n bits of x and flipping it.

Definition 2 (PAC learning in the Random Walk model). *A concept class $\mathcal{C}$ is* PAC-learnable in the Random Walk model *if there is an algorithm A that satisfies Definition 1 above but with $EX_{RW}(f)$ in place of $EX_U(f)$.*

As in [6], it is convenient for us to work with a slight variant of the Random Walk oracle which is of equivalent power; we call this the *updating Random Walk* oracle and denote it by $EX_{URW}(f)$. If the last example generated by $EX_{URW}(f)$ was $x \in \{0,1\}^n$, the updating Random Walk oracle chooses a uniform index $i \in [n]$, but instead of flipping the bit $x[i]$ it replaces $x[i]$ with a uniform random bit from $\{0,1\}$ (i.e. it flips the bit with probability $1/2$ and leaves x unchanged with probability $1/2$) to obtain the new example x'. We say that such a step *updates* the i-th bit position.

An easy argument given in [6] shows that the Random Walk oracle can efficiently simulate the updating Random Walk oracle and vice versa, and thus any concept class that is efficiently learnable from one oracle is also efficiently learnable from the other. We introduce the updating Random Walk oracle because it is easy to see (and well known) that the updating random walk on the hypercube mixes rapidly. More precisely, we have the following fact which will be useful later:

Fact 1 *Let $\langle x, f(x)\rangle$ be a labeled example that is obtained from $EX_{URW}(f)$, and let $\langle y, f(y)\rangle$ be the labeled example that $EX_{URW}(f)$ outputs $n \ln \frac{n}{\delta}$ draws later. Then with probability at least $1-\delta$, the two strings x, y are uniformly and independently distributed over $\{0,1\}^n$.*

Proof. Since it is clear that x and y are each uniformly distributed, the only thing to check for Fact 1 is independence. This follows since y will be independent of

x if and only if all n bit positions are updated in the $n \ln \frac{n}{\delta}$ draws between x and y. For each draw, the probability that a particular bit is not updated is $(1 - \frac{1}{n})$. Thus after $n \ln \frac{n}{\delta}$ draws, the probability that any bit of r has not been updated is at most $n(1 - \frac{1}{n})^{n \ln \frac{n}{\delta}} \leq \delta$.. This yields the fact. ■

Note that Fact 1 implies that any concept class $\mathcal{C}$ that is uniform distribution PAC-learnable is also PAC-learnable in the Random Walk model, since we can obtain independent uniform random examples in the Random Walk model with essentially just a $\Theta(n \log n)$ slowdown.

2.3 Background from Cryptography

We write $\mathcal{R}_n$ to denote the set of all 2^{2^n} Boolean functions from $\{0,1\}^n$ to $\{0,1\}$. We refer to a function f chosen uniformly at random from $\mathcal{R}_n$ as a *truly random function.* We write D^f to denote a probabilistic polynomial-time (p.p.t.) algorithm D with black-box oracle access to the function f.

Informally, a one-way function is a function $f : \{0,1\}^n \to \{0,1\}^n$ that is computable by a poly(n) time algorithm but is hard to invert in the sense that no poly(n)-time algorithm can successfully compute f^{-1} on a nonnegligible fraction of outputs of f. (See [9] for a detailed definition and discussion of one-way functions.) In a celebrated result, Håstad *et al.* [11] showed that if any one-way function exists, then *pseudorandom function families* must exist as well.

Definition 3. *A* pseudorandom function family *[10] is a collection of functions* $\{f_s : \{0,1\}^{|s|} \to \{0,1\}\}_{s \in \{0,1\}^*}$ *with the following two properties:*

1. (efficient evaluation) *there is a deterministic algorithm which, given an* n*-bit seed* s *and an* n*-bit input* x*, runs in time poly*(n) *and outputs* $f_s(x)$;
2. (pseudorandomness) *for all polynomials* Q*, all p.p.t. oracle algorithms* D*, and all sufficiently large* n*, we have that*

$$\left| \Pr_{f \in \mathcal{R}_n} [D^f(1^n) \texttt{ outputs } 1] - \Pr_{s \in \{0,1\}^n} [D^{f_s}(1^n) \texttt{ outputs } 1] \right| < \frac{1}{Q(n)}.$$

The argument 1^n indicates that the "distinguisher" algorithm D must run in poly(n) time steps since its input is of length n. Intuitively, condition (2) above states that a pseudorandom function cannot be distinguished from a truly random function by any polynomial-time algorithm that has black-box access to the pseudorandom function with an inverse polynomial advantage over random guessing.

3 A Partial Separation

3.1 A first attempt

It is clear that in the Random Walk model a learning algorithm will get many pairs of examples that are adjancent vertices of the Hamming cube $\{0,1\}^n$,

whereas this will not be the case for a learner in the standard uniform distribution model (with high probability, a set of poly(n) many independent uniform examples from $\{0,1\}^n$ will contain no pair of examples that have Hamming distance less than $n/2 - O(\sqrt{n \log n})$). Thus, in attempting to separate the random walk model from the standard uniform distribution model, it is natural to try to construct a concept class using pseudorandom functions f_s but altered in such a way that seeing the value of the function on adjacent inputs gives away information about the seed s.

One natural approach is the following: given a pseudorandom function family $\{f_s : \{0,1\}^k \to \{0,1\}\}_{s \in \{0,1\}^k}$, one could define a concept class of functions $\{f'_s : \{0,1\}^k \times \{0,1\}^{\log k} \times \{0,1\} \to \{0,1\}\}_{s \in \{0,1\}^k}$ as follows:

$$f'_s(x,i,b) = \begin{cases} f_s(x) & \text{if } b = 0 \\ f_s(x) \oplus s[i] & \text{if } b = 1 \end{cases}$$

where x is a k-bit string, i is a $(\log k)$-bit string encoding an integer between 1 and k, and b is a single bit. A learning algorithm in the Random Walk model will be able to obtain all bits $s[1], \ldots, s[k]$ of the seed s (by waiting for pairs of successive examples $(x,i,b), (x,i,1-b)$ in which the final bit b flips for all k possible values of i), and will thus be able to exactly identify the target concept. However, even though a standard uniform distribution learner will not obtain any pair of inputs that differ only in the final bit b, it is not clear how to show that no algorithm in the standard uniform distribution model can learn the concept class to high accuracy. Such a proof would require one to show that any polynomial-time uniform distribution learning algorithm could be used to "break" the pseudorandom function family $\{f_s\}$, and this seems difficult to do. (Intuitively, this difficulty arises because the $b = 1$ case of the definition of f'_s "mixes" bits of the seed with the output of the pseudorandom function.) Thus, we must consider alternate constructions.

3.2 A partial separation

In this section we describe a concept class and prove that it has the following two properties: (1) A randomly chosen concept from the class is indistinguishable from a truly random function to any polynomial-time algorithm which has an $EX_U(\cdot)$ oracle for the concept (and thus no such algorithm can learn to accuracy $\epsilon = \frac{1}{2} - \frac{1}{\text{poly}(n)}$); (2) However, a Random Walk algorithm with access to $EX_{RW}(\cdot)$ can learn any concept in the class to accuracy $\frac{3}{4}$. In the next section we will extend this construction to fully separate the Random Walk model from the standard uniform model and thus prove Theorem 1.

Our construction uses ideas from Section 3.1; as in the construction proposed there, the concepts in our class will reveal information about the seed of a pseudorandom function to learning algorithms that can obtain pairs of points with only the last bit flipped. However, each concept in the class will now be defined by *two* pseudorandom functions rather than one; this will enable us to prove that

the class is indeed hard to learn in the uniform distribution model (but will also prevent a Random Walk learning algorithm from learning to high accuracy).

Let $\mathcal{F}$ be a family of pseudorandom functions $\{f_r : \{0,1\}^k \to \{0,1\}\}_{r \in \{0,1\}^k}$. We construct a concept class $\mathcal{G} = \{g_{r,s} : r, s \in \{0,1\}^k\}$, where $g_{r,s}$ takes an n-bit input that we split into four parts for convenience. As before, the first k bits x give the "actual" input to the function, while the other parts determine the mode of function that will be applied.

$$g_{r,s}(x,i,b,y) = \begin{cases} f_s(x) & \text{if } y = 0, b = 0 \\ f_s(x) \oplus r[i] & \text{if } y = 0, b = 1 \\ f_r(x) & \text{if } y = 1 \end{cases} \tag{1}$$

Here b and y are one bit and i is $\log k$ bits to indicate which bit of the seed r is exposed. Thus half of the inputs to $g_{r,s}$ are labeled according to f_r, and the other half are labeled according to either f_s or $f_s \oplus r[i]$ depending on the value of b.

The following lemma establishes that $\mathcal{G}$ is not efficiently PAC-learnable under the uniform distribution, by showing that a random function from $\mathcal{G}$ is indistinguishable from a truly random function to any algorithm which only has $EX_U(\cdot)$ access to the target concept. (A standard argument shows that an efficient PAC learning algorithm can be used to obtain an efficient distinguisher simply by running the learning algorithm and using its hypothesis to predict a fresh random example. Such an approach must succeed with high probability for any function from the concept class by virtue of the PAC criterion, but no algorithm that has seen only poly(n) many examples of a truly random function can predict its outputs on fresh examples with probability nonnegligibly greater than $\frac{1}{2}$.)

Lemma 1. *Let $g_{r,s} : \{0,1\}^n \to \{0,1\}$ be a function from $\mathcal{G}$ chosen by selecting r and s uniformly at random from $\{0,1\}^k$, where k satisfies $n = k + \log k + 2$. Let f be a truly random function. Then for any $\epsilon = \Omega(\frac{1}{\text{poly}(n)})$, no p.p.t. algorithm can distinguish between having oracle access to $EX_U(g_{r,s})$ versus oracle access to $EX_U(f)$ with success probability greater than $\frac{1}{2} + \epsilon$.*

Proof. The proof is by a hybrid argument. We will construct two intermediate functions, h_r and h'_r. We will show that $EX_U(g_{r,s})$ is indistinghable from $EX_U(h_r)$, $EX_U(h_r)$ from $EX_U(h'_r)$, and $EX_U(h'_r)$ from $EX_U(f)$. It will then follow that $EX_U(g_{r,s})$ is indistinguishable from $EX_U(f)$.

Consider the function

$$h_r(x,i,b,y) = \begin{cases} f(x) & \text{if } y = 0, b = 0 \\ f(x) \oplus r[i] & \text{if } y = 0, b = 1 \\ f_r(x) & \text{if } y = 1 \end{cases}. \tag{2}$$

Here we have simply replaced f_s with a truly random function. We claim that no p.p.t. algorithm can distinguish oracle access to $EX_U(g_{r,s})$ from oracle access

to $EX_U(h_r)$; for if such a distinguisher D existed, we could use it to obtain an algorithm D' to distinguish a randomly chosen $f_s \in \mathcal{F}$ from a truly random function in the following way. D' picks r at random from $\{0,1\}^k$ and runs D, answering D's queries to its oracle by choosing i, b and y at random, querying its own oracle to receive a bit q, and outputting q when both y and b are 0, $q \oplus r[i]$ when $y = 0$ and $b = 1$, and $f_r(x)$ when $y = 1$. It is easy to see that if D''s oracle is for a truly random function $f \in \mathcal{R}$ then this process perfectly simulates access to $EX_U(h_r)$, and if D''s oracle is for a randomly chosen $f_s \in \mathcal{F}$ then this process perfectly simulates access to $EX_U(g_{r,s})$ for r, s chosen uniformly at random.

We now consider the intermediate function

$$h'_r(x,i,b,y) = \begin{cases} f(x) & \text{if } y = 0 \\ f_r(x) & \text{if } y = 1 \end{cases}$$

and argue that no p.p.t. algorithm can distinguish oracle access to $EX_U(h_r)$ from access to $EX_U(h'_r)$. When $y = 1$ or both $y = 0$ and $b = 0$, both h_r and h'_r will have the same output. Otherwise, if $y = 0$ and $b = 1$ we have that $h_r(x,i,b,y) = f(x) \oplus r_i$ whereas $h'_r(x,i,b,y) = f(x)$. Now, it is easy to see that an algorithm with *black-box query access* to h_r can easily distinguish h_r from h'_r (simply because flipping the penultimate bit b will always cause the value of h_r to flip but will only cause the value of h'_r to flip half of the time). But for an algorithm that only has oracle access to $EX_U(\cdot)$, conditioned on never receiving the same string x twice (a condition that fails to hold only with negligible – in fact, inverse exponential – probability), it is easy to see that whether the oracle is for h_r or h'_r, each output value that the algorithm sees on inputs with $y = 0$ and $b = 1$ will be a fresh independent uniform random bit. (This is simply because a random function f can be viewed as tossing a coin to determine its output on each new input value, so no matter what $r[i]$ is, XORing it with $f(x)$ yields a fresh independent uniform random bit.)

Finally, it follows from the definition of pseudorandomness that no p.p.t. algorithm can distinguish oracle access to $EX_U(h'_r)$ from access to $EX_U(f)$. We have thus shown that $EX_U(g_{r,s})$ is indistinghable from $EX_U(h_r)$, $EX_U(h_r)$ from $EX_U(h'_r)$, and $EX_U(h'_r)$ from $EX_U(f)$. It follows that $EX_U(g_{r,s})$ is indistinguishable from $EX_U(f)$, and the proof is complete. ■

We now show that $g_{r,s}$ is learnable to accuracy $\frac{3}{4}$ in the Random Walk model:

Lemma 2. *There is an algorithm A with the following property: for any $\delta > 0$ and any concept $g_{r,s} \in \mathcal{G}$, if A is given access to a Random Walk oracle $EX_{RW}(g_{r,s})$ then A runs in time poly$(n, \log(1/\delta))$ and with probability at least $1 - \delta$, algorithm A outputs an efficiently computable hypothesis h such that $\Pr_{\mathcal{U}}[h(x) \neq g_{r,s}(x)] \leq \frac{1}{4}$.*

Proof. As described in Section 2, for convenience in this proof we will assume that we have an updating Random Walk oracle $EX_{URW}(g_{r,s})$.

We give an algorithm that, with probablity $1-\delta$, learns all the bits of r. Once the learner has obtained r she outputs the following (randomized) hypothesis h:

$$h(x,i,b,y) = \begin{cases} \$ & \text{if } y = 0 \\ f_r(x) & \text{if } y = 1 \end{cases}$$

where \$ denotes a random coin toss at each invocation. Note that h incurs zero error relative to $g_{r,s}$ on inputs that have $y = 1$, and has error rate exactly $\frac{1}{2}$ on inputs that have $y = 0$. Thus the overall error rate of h is exactly $\frac{1}{4}$.

We now show that with probability $1-\delta$ (over the random examples received from $EX_{URW}(g_{r,s})$) the learner can obtain all of r after receiving $T = O(n^2k \cdot \log^2(n/\delta))$ many examples from $EX_{URW}(g_{r,s})$. The learner does this by looking at pairs of successive examples; we show (Fact 4 below) that after seeing $t = O(nk \cdot \log(k/\delta))$ pairs, each of which is independent from all other pairs, we obtain all of r with probability at least $1 - \frac{\delta}{2}$. To get t independent pairs of successive examples, we look at blocks of $t' = O(n \log(tn/\delta))$ many consecutive examples, and use only the first two examples from each such block. By Fact 1 we have that for a given pair of consecutive blocks, with probability at least $1 - \frac{\delta}{2t}$ the first example from the second block is random even given the pair of examples from the first block. A union bound over the t blocks gives total failure probability at most $\frac{\delta}{2}$ for independence, and thus an overall failure probability of at most δ.

We have the following simple facts:

Fact 2 *If the learner receives two consecutive examples $w = (x,i,0,0), w' = (x,i,1,0)$ and the corresponding labels $g_{r,s}(w), g_{r,s}(w')$, then the learner can obtain the bit $r[i]$.*

Fact 3 *For any $j \in [k]$, given a pair of consecutive examples from $EX_{URW}(g_{r,s})$, a learning algorithm can obtain the value of $r[j]$ from this pair with probability at least $\frac{1}{4kn}$.*

Proof. By Fact 2, if the first example is $w = (x,i,b,y)$ with $i = j$, $y = 0$ and the following example differs in the value of b, then the learner obtains $r[j]$. The first example (like every example from $EX_{URW}(g_{r,s})$) is uniformly distributed and thus has $i = j$, $y = 0$ with probability $\frac{1}{2k}$. The probability that the next example from $EX_{URW}(g_{r,s})$ flips the value of b is $\frac{1}{2n}$. ■

Fact 4 *After receiving $t = 4kn \cdot \log(k/\delta')$ independent pairs of consecutive examples as described above, the learner can obtain all k bits of r with probability at least $1 - \delta'$.*

Proof. For any $j \in [k]$, the probability that $r[j]$ is not obtained from a given pair of consecutive examples is at most $(1 - \frac{1}{4kn})$. Thus after seeing t independent pairs of consecutive examples, the probability that any bit of r is not obtained is at most $k(1 - \frac{1}{4kn})^t$. This yields the fact. ■

Thus the total number of calls to $EX_{URW}(g_{r,s})$ that are required is

$$T = t \cdot t' = O(nk \log(k/\delta)) \cdot O(n \log(tn/\delta)) = O(n^2 k \log^2(n/\delta)).$$

Since $k = O(n)$, Lemma 2 is proved. ■

4 A Full Separation

We would like to have a concept class for which a Random Walk learner can output an ϵ-accurate hypothesis for any $\epsilon > 0$. The drawback of our construction in Section 3.2 is that a Random Walk learning algorithm can only achieve a particular fixed error rate $\epsilon = \frac{1}{4}$. Intuitively, a Random Walk learner cannot achieve accuracy better than $\frac{3}{4}$ because on half of the inputs the concept's value is essentially determined by a pseudorandom function whose seed the Random Walk learner cannot discover. It is not difficult to see that for any given $\epsilon = \frac{1}{\text{poly}(n)}$, by altering the parameters of the construction we could obtain a concept class that a Random Walk algorithm can learn to accuracy $1-\epsilon$ (and which would still be unlearnable for a standard uniform distribution algorithm). However, this would give us a different concept class for each ϵ, whereas what we require is a single concept class that can be learned to accuracy ϵ for each $\epsilon > 0$.

In this section we present a new concept class $\mathcal{G}'$ and show that it achieves this goal. The idea is to string together many copies of our function from Section 3.2 in a particular way. Instead of depending on two seeds r, s, a concept in $\mathcal{G}'$ is defined using k seeds $r_1, \ldots, r_k$ and $k-1$ subfunctions $g_{r_1,r_2}, g_{r_2,r_3}, \ldots, g_{r_{k-1},r_k}$. These subfunctions are combined in a way that lets the learner learn more and more of the seeds $r_1, r_2, \ldots$, and thus learn to higher and higher accuracy, as she receives more and more examples.

4.1 The Concept Class $\mathcal{G}'$

We now describe $\mathcal{G}'$ in detail. Each concept in $\mathcal{G}'$ is defined by k seeds $r_1, \ldots, r_k$, each of length k. The concept $g'_{r_1,\ldots,r_k}$ is defined by

$$g'_{r_1,\ldots,r_k}(x,i,b,y,z) = \begin{cases} g_{r_{\alpha(z)},r_{\alpha(z)+1}}(x,i,b,y) & \text{if } \alpha(z) \in \{1,\ldots,k-1\} \\ f_{r_k}(x) & \text{if } \alpha(z) = k \end{cases}$$

As in the previous section x is a k-bit string, i is a $\log k$-bit string, and b and y are single bits. The new input z is a $(k-1)$-bit string, and the value $\alpha(z) \in [k]$ is defined as the index of the leftmost bit in z that is 1 (for example if $z = 0010010111$ then $\alpha(z) = 3$); if $z = 0^{k-1}$ then $\alpha(z)$ is defined to be k. By this design, the subfunction $g_{r_j,r_{j+1}}$ will be used on a $1/2^j$ fraction of the inputs to g'. Note that g' maps $\{0,1\}^n$ to $\{0,1\}$ where $n = 2k + \log k + 1$.

4.2 Uniform Distribution Algorithms Cannot Learn $\mathcal{G}'$

We first show that $\mathcal{G}'$ is not efficiently PAC-learnable under the uniform distribution. This is implied by the following lemma:

Lemma 3. *Let $g'_{r_1,\ldots,r_k} : \{0,1\}^n \to \{0,1\}$ be a function from $\mathcal{G}'$ chosen by selecting $r_1, \ldots, r_k$ uniformly at random from $\{0,1\}^k$, where k satisfies $n = 2k + \log k + 1$. Let f be a truly random function. Then for any $\epsilon = \Omega(\frac{1}{\text{poly}(n)})$, no p.p.t. algorithm can distinguish between having access to $EX_U(g'_{r_1,\ldots,r_k})$ versus access to $EX_U(f)$ with success probability greater than $\frac{1}{2} + \epsilon$.*

Proof. Again we use a hybrid argument. We define the concept classes $\mathcal{H}(\ell) = \{h_{r_1,\ldots,r_\ell;f} : r_1,\ldots,r_\ell \in \{0,1\}^k, f \in \mathcal{R}_k\}$ for $2 \leq \ell \leq k$. Each function $h_{r_1,\ldots,r_\ell;f}$ takes the same n-bit input (x,i,b,y,z) as $g'_{r_1,\ldots,r_k}$. The function $h_{r_1,\ldots,r_\ell;f}$ is defined as follows:

$$h_{r_1,\ldots,r_\ell;f}(x,i,b,y,z) = \begin{cases} g_{r_{\alpha(z)},r_{\alpha(z)+1}}(x,i,b,y) & \text{if } \alpha(z) < \ell \\ f(x) & \text{otherwise.} \end{cases}$$

Here as before, the value $\alpha(z) \in [k]$ denotes the index of the leftmost bit of z that is one (and we have $\alpha(z) = k$ if $z = 0^{k-1}$).

We will consider functions that are chosen uniformly at random from $\mathcal{H}(\ell)$, i.e. $r_1,\ldots,r_\ell$ are chosen randomly from $\{0,1\}^k$ and f is a truly random function from $\mathcal{R}_k$. Using Lemma 1, it is easy to see that for a distinguisher that is given only oracle access to $EX_U(\cdot)$, a random function from $\mathcal{H}(2)$ is indistinguishable from a truly random function from $\mathcal{R}_n$. We will now show that, for $2 \leq \ell < k$, if a random function from $\mathcal{H}(\ell)$ is indistinguishable from a truly random function then the same is true for $\mathcal{H}(\ell+1)$. This will then imply that a random function from $\mathcal{H}(k)$ is indistinguishable from a truly random function.

Let $h_{r_1,\ldots,r_{\ell+1}}$ be taken randomly from $\mathcal{H}(\ell+1)$ and f be a truly random function from $\mathcal{R}_n$. Suppose we had a distinguisher D that distinguishes between a random function from $\mathcal{H}(\ell+1)$ and a truly random function from $\mathcal{R}_n$ with success probability $\frac{1}{2} + \epsilon$, where $\epsilon = \Omega(\frac{1}{\text{poly}(n)})$. Then we can use D to obtain an algorithm D' for distinguishing a randomly chosen $f_s \in \mathcal{F}$ from a randomly chosen function $f \in \mathcal{R}_k$ in the following way. D' first picks strings $r_1,\ldots,r_\ell$ at random from $\{0,1\}^k$. D' then runs D, simulating its oracle in the following way. At each invocation, D' draws a random (x,i,b,y,z) and behaves as follows:

- If $\alpha(z) < \ell$, then D' outputs $\langle (x,i,b,y,z), g_{r_{\alpha(z)},r_{\alpha(z)+1}}(x,i,b,y)\rangle$.
- If $\alpha(z) = \ell$, then D' calls its oracle to obtain $\langle x', \beta\rangle$. If $y = b = 0$ then D' outputs $\langle (x',i,b,y,z), \beta\rangle$. If $y = 0$ but $b = 1$ then D' outputs $\langle (x',i,b,y,z), \beta \oplus r_\ell[i]\rangle$. If $y = 1$ then D' outputs $\langle (x',i,b,y,z), f_{r_\ell}(x)\rangle$.
- If $\alpha(z) > \ell$, D' outputs the labelled example $\langle (x,i,b,y,z), r(x)\rangle$ where $r(x)$ is a fresh random bit for each x. (The pairs $(x, r(x))$ are stored, and if any k-bit string x is drawn twice – which is exponentially unlikely in a sequence of poly(n) many draws – D' uses the same bit $r(x)$ as before.)

It is straightforward to check that if D''s oracle is $EX_U(f_s)$ for a random $f_s \in \mathcal{F}$, then D' simulates an oracle $EX_U(h_{r_1,\ldots,r_{\ell+1}})$ for D, where $h_{r_1,\ldots,r_{\ell+1}}$ is drawn uniformly from $\mathcal{H}(\ell+1)$. On the other hand, we claim that if D''s oracle is $EX_U(f)$ for a random $f \in \mathcal{R}_k$, then D' simulates an oracle that is indistinguishable from $EX_U(h_{r_1,\ldots,r_\ell})$ for D, where $h_{r_1,\ldots,r_\ell}$ is drawn uniformly from $\mathcal{H}(\ell)$. Clearly the oracle D' simulates is identical to $EX_U(h_{r_1,\ldots,r_\ell})$ for $\alpha(z) \neq \ell$. For $\alpha(z) = \ell$, D' simulates the function h_{r_l} as in Equation 2 in the proof of Lemma 1, which is indistinguishable from a truly random function as proved in the lemma.

Thus the success probability of the distinguisher D' is the same as the probability that D succeeds in distinguishing $\mathcal{H}(\ell+1)$ from $\mathcal{H}(\ell)$. Recall that $\mathcal{H}(\ell)$ is

indistinguishable from a truly random function, and that D succeeds in distinguishing $\mathcal{H}(\ell+1)$ from a truly random function with probability at least $\frac{1}{2}+\epsilon$ by assumption. This implies that D' succeeds in distinguishing a randomly chosen $f_s \in \mathcal{F}$ from a randomly chosen function $f \in \mathcal{R}_k$ with probability at least $\frac{1}{2}+\epsilon-\frac{1}{\omega(\mathrm{poly}(n))}$, but this contradicts the pseudorandomness of $\mathcal{F}$.

Finally, we claim that for any p.p.t. algorithm, having oracle access to a random function from $\mathcal{H}(k)$ is indistinguishable from having oracle access to a random function from $\mathcal{G}'$. To see this, note that the functions $h_{r_1,\ldots,r_\ell;f}$ and $g'_{r_1,\ldots,r_\ell}$ differ only on inputs (x,i,b,y,z) that have $\alpha(z)=k$, i.e. $z=0^{k-1}$ (on such inputs the function $g_{r_1,\ldots,r_\ell}$ will output $f_{r_k}(x)$ whereas $h_{r_1,\ldots,r_\ell;f}$ will output $f(x)$). But such inputs are only a $\frac{1}{2^{\Omega(n)}}$ fraction of all possible inputs, so with overwhelmingly high probability a p.p.t. algorithm will never receive such an example. ■

4.3 Random Walk Algorithms Can Learn $\mathcal{G}'$

The following lemma completes the proof of our main result, Theorem 1.

Lemma 4. *There is an algorithm B with the following property: for any $\epsilon,\delta>0$, and any concept $g_{r_1,\ldots,r_k} \in \mathcal{G}'$, if B is given access to a Random Walk oracle $EX_{RW}(g_{r_1,\ldots,r_k})$, then B runs in time poly$(n,\log(1/\delta),1/\epsilon)$ and can with probability at least $1-\delta$ output a hypothesis h such that $\Pr_U[h(x)\neq g_{r_1,\ldots,r_k}(x)]\leq\epsilon$.*

Proof. The proof is similar to that of Lemma 2. Again, for convenience we will assume that we have an updating Random Walk oracle $EX_{URW}(g_{r_1,\ldots,r_k})$. Recall from Lemma 2 that there is an algorithm A that can obtain the string r_j with probability at least $1-\delta'$ given $t'=O(nk\cdot\log(n/\delta'))$ independent pairs of successive random walk examples

$$\Big(\langle w, g_{r_j,r_{j+1}}(w)\rangle, \langle w', g_{r_j,r_{j+1}}(w')\rangle\Big).$$

Algorithm B works in a sequence of v stages. In stage j, the algorithm simply tries to obtain t' independent example pairs for $g_{r_j,r_{j+1}}$ and then uses Algorithm A. Assuming the algorithm succeeds in each stage, after stage v algorithm B has obtained $r_1,\ldots,r_v$. It follows directly from the definition of $\mathcal{G}'$ that given $r_1,\ldots,r_v$, Algorithm B can construct a hypothesis that has error at most $\frac{3}{2^{v+2}}$ (see Figure 1) so we may take $v=\log\frac{1}{\epsilon}+1$ to obtain error at most ϵ. (Note that this implicitly assumes that $\log\frac{1}{\epsilon}+1$ is at most k; we deal with the case $\log\frac{1}{\epsilon}+1>k$ at the end of the proof.)

If the learner fails to obtain $r_1,\ldots,r_v$, then either:

1. Independence was not achieved between every pair of examples;
2. Algorithm B fails to acquire t' pairs of examples for $g_{r_j,r_{j+1}}$ in some stage j; or
3. Algorithm B acquires t' pairs of examples for $g_{r_j,r_{j+1}}$ but Algorithm A fails to obtain r_j in some stage j.

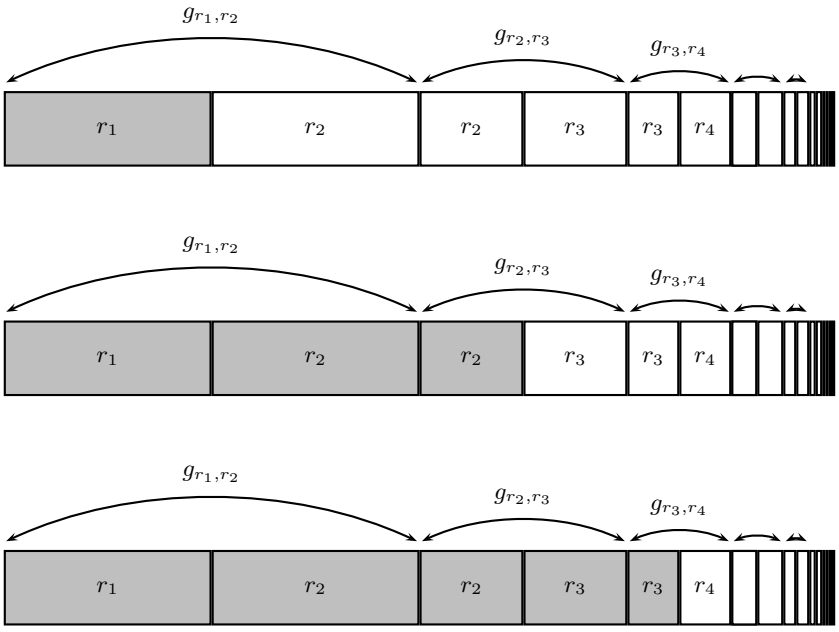

Fig. 1. Stages 1, 2 and 3 of Algorithm B. Each row represents the output values of $g'_{r_1,\ldots,r_k}$. After stage j the algorithm "knows" $r_1, \ldots, r_j$ and can achieve perfect accuracy on the shaded region.

We choose the total number of examples so that each of these probabilities is bounded by $\delta/3$ to achieve an overall failure probability of at most δ.

As will be clear from the analysis of cases (2) and (3) below, in total Algorithm B will use $4 \cdot 2^{v+1}t'$ pairs of examples in stages 1 through v, where t' will be bounded later. Each pair of examples is obtained by using the first two examples from a block of $s = O(n \log(v \cdot 2^{v+1}t'n/\delta))$ many consecutive examples from the updating Random Walk oracle. With this choice of s, the same argument as in the proof of Lemma 2 shows that the total failure probability for independence is at most $\frac{\delta}{3}$.

We bound (2) assuming full independence between all pairs of examples. In stage j, Algorithm B uses $4 \cdot 2^j t'$ pairs of examples. Observe that each pair of examples has both examples from $g_{r_j,r_{j+1}}$ with probability at least $2^{-(j+1)}$. By a Chernoff bound, the probability that less than t' of the example pairs in stage j are from $g_{r_j,r_{j+1}}$ is at most $e^{-\frac{t'}{8}}$. Thus the overall probability of failure from condition (2) is at most $ve^{-\frac{t'}{8}}$ which is at most $\delta/3$ for $t' \geq \ln(3v/\delta)$.

We bound (3) assuming full independence between all pairs of examples as well. In stage j, we know by Fact 4 that after seeing $t' = O(nk \log(3vk/\delta))$ pairs

of examples for $g_{r_j,r_{j+1}}$, the probability of failing to obtain r_j is at most $\delta/3v$. Hence the overall failure probability from condition (3) is at most $\frac{\delta}{3}$.

We thus may take $t' = O(nk\log(3vk/\delta))$ and achieve an overall failure probability of δ for obtaining $r_1, \ldots, r_v$. It follows that the overall number of examples required from the updating Random Walk oracle is $\mathrm{poly}(2^v, n, \log\frac{1}{\delta}) = \mathrm{poly}(n, \frac{1}{\epsilon}, \log\frac{1}{\delta})$, which is what we required.

Finally, we observe that if $\log\frac{1}{\epsilon} + 1 > k$, since $k = \frac{n}{2} - O(\log n)$ a $\mathrm{poly}(\frac{1}{\epsilon})$-time algorithm may run for, say, 2^{2n} time steps and thus build an explicit truth table for the function. Such a table can be used to exactly identify each seed $r_1, \ldots, r_k$ and output an exact representation of the target concept. ■

5 Acknowledgements

We warmly thank Tal Malkin for helpful discussions.

References

[1] D. Aldous and U. Vazirani. A Markovian extension of Valiant's learning model. In *Proceedings of the Thirty-First Symposium on Foundations of Computer Science*, pages 392–396, 1990.

[2] P. Bartlett, P. Fischer, and K.U. Höffgen. Exploiting random walks for learning. *Information and Computation*, 176(2):121–135, 2002.

[3] A. Blum. Learning a function of r relevant variables (open problem). In *Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop*, pages 731–733, 2003.

[4] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the Twenty-Sixth Annual Symposium on Theory of Computing*, pages 253–262, 1994.

[5] N. Bshouty, J. Jackson, and C. Tamon. More efficient PAC learning of DNF with membership queries under the uniform distribution. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 286–295, 1999.

[6] N. Bshouty, E. Mossel, R. O'Donnell, and R. Servedio. Learning DNF from Random Walks. In *Proceedings of the 44th IEEE Symposium on Foundations on Computer Science*, pages 189–198, 2003.

[7] N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 1996.

[8] D. Gamarnik. Extension of the PAC framework to finite and countable Markov chains. In *Proceedings of the 12th Annual Conference on Computational Learning Theory*, pages 308–317, 1999.

[9] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, New York, 2001.

[10] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807, 1986.

[11] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[12] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55:414–440, 1997.

[13] J. Jackson, A. Klivans, and R. Servedio. Learnability beyond AC^0. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, 2002.

[14] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing*, pages 372–381, 1993.

[15] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[16] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[17] K. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314–326, 1990.