Average-Case Complexity of Learning Polynomials

Frank Stephan[*]
Mathematisches Institut
Universität Heidelberg
Im Neuenheimer Feld 294
69120 Heidelberg, Germany
fstephan@math.uni-heidelberg.de

Thomas Zeugmann[†]
Institut für Theoretische Informatik
Medizinische Universität Lübeck
Wallstraße 40
23560 Lübeck, Germany
thomas@tcs.mu-luebeck.de

Abstract

The present paper deals with the average-case complexity of various algorithms for learning univariate polynomials. For this purpose an appropriate framework is introduced. Based on it, the learnability of univariate polynomials evaluated over the natural numbers and of univariate polynomials defined over finite fields is analyzed.

Our results are manifold. In the first case, convergence is measured not relative to the degree of a polynomial but with respect to a measure that takes the *degree* and the *size* of the coefficients into account. Then standard interpolation is proved not to be the best possible algorithm with respect to the average number of examples needed.

In general, polynomials over finite fields are not uniquely specified by their input-output-behavior. Thus, as a new form of data representation the remainders modulo other polynomials is proposed and the expected example complexity is analyzed for a rather rich class of probability distributions.

1 Introduction

Learning concepts efficiently has attracted considerable attention during the last decade. However, within the field of inductive inference traditionally the main emphasis has been put on analyzing the update time, i.e., the effort to compute a *single* new hypothesis. On the other hand, starting with Valiant's [21] pioneering paper, the total amount of examples and/or time needed to solve a given learning problem has become quite popular. Nevertheless, the complexity bounds proved within the PAC model are usually worst-case bounds.

Since experimental studies have shown quite often a large gap between the worst-case bounds proved and the actual runtime observed, several authors advocated to analyze the average-case behavior of learning algorithms (cf., e.g., [7, 10, 12, 14, 15, 16, 17, 18]). We continue along this line of research.

Within this paper we deal with the problem to learn efficiently univariate integer valued polynomials as well as univariate polynomials over finite fields from two different sources of information. The underlying model is Gold [9]-style *learning in the limit*, i.e., the learner has to produce a sequence of hypotheses that stabilizes to a correct and finite description of the target polynomial. We always choose the target class of all relevant polynomials as hypothesis space.

Classically, the source of information are incrementally growing sequences of pairs argument–value. Angluin and Smith [2] describe two methods for learning integer valued polynomials in this setting. The first method is *identification by enumeration* (cf. Gold [9]). Here, a canonical enumeration of all target polynomials is assumed and the learner searches on every input for the first polynomial in the enumeration that matches the data. Clearly, it then converges to the first enumerated polynomial that equals the target.

The second method is learning by interpolation, i.e., the learner always computes the interpolation polynomial from the data given. Polynomial interpolation is a widely studied and well understood problem (cf., e.g., Bini and Pan [4]). So in the general case, there are algorithms which synthesize a formula for a desired polynomial over the rational or real numbers from $n+1$ pairs $(x, g(x))$ where g is the desired polynomial and n is its degree.

We aim to compare these methods with respect to their *average-case* example and time complexity. Part of our motivation is a result obtained by Gold [9] and generalized by Jantke and Beick [13] stating that *identification by enumeration* is an optimally data efficient method. Here, the data efficiency of a learner is measured by the quantity of data it needs to converge to a

[*]This author was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant no. Ste 967/1–1.

[†]Supported by the Grant-in-Aid for Scientific Research in Fundamental Areas from the Japanese Ministry of Education, Science, Sports, and Culture under grant no. 10558047.

correct hypothesis. If L_1 and L_2 are two learning algorithms, then L_1 is as *data efficient* as L_2 iff, for every admissible information presentation and every target, L_1 does not need more data than L_2 to converge. L_1 is *strictly more data efficient* than L_2 if L_1 is as data efficient as L_2 but there is a target and a data presentation for it such that L_1 needs strictly less data than L_2 until convergence. Finally, a learning method is called *optimally data efficient* if there is no other learner that is strictly more data efficient.

While these investigations have been undertaken in a setting where learning is required from *every* information presentation, our goal is to analyze the *average-case* complexity of these two methods. We define our average-case model by specifying a rather rich class of probability distributions over the natural numbers (cf. Section 2). Then, every datum x will be drawn independently at random with a certain probability. Thus, one obtains randomly generated sequences $((x, g(x)))$ and the learner is fed incrementally growing initial segments of the sequence generated.

Then we consider the problem of learning polynomials over a finite field. For seeing the main two differences, let us assume that we have to learn polynomials defined over the finite field $\mathbb{F}_2$.

(1) Now, the polynomials considered are defined only for two inputs, i.e., input 0 and input 1. Therefore they are not fully described by the mapping $x \to g(x)$ for $x \in \{0, 1\}$ and one has to look for other ways to describe them. The way chosen in the present paper is to supply the data as a random sequence of pairs (a, b) where b is the remainder polynomial obtained when the target polynomial g is divided by polynomial a. This generalization reintroduces a mode to describe the whole polynomial. We refer to this model as to *learning from remainder sequences.*

(2) In the standard case, all data fed to the learner are of the form $(x, g(x))$ and have the same information content. Thus, $n+1$ different data items — whatever they are — describe exactly the target polynomial g while n do not do it. This beautiful property is lost in the model considered here, i.e., when learning from remainder sequences. Taking for example a product $g_1 g_2$; then all pairs (a, b) where a is dividing g_1 do not contribute any knowledge about g_2, but there might be 2^n divisors for some polynomial g_1 of degree n^2 — for example obtained by multiplying n coprime polynomials of length n. So the degree is not an upper bound on the number of data-items needed to learn a polynomial. On the other hand, a single data-item can give a full description of the polynomial: if $\deg(a) > n$ then the b in the pair (a, b) is already the correct polynomial wanted.

For defining our average-case model, we have to introduce a class of probability distributions over the set $\mathbb{F}_q[x]$ of all relevant polynomials (where $\mathbb{F}_q$ is a finite field of order q). Clearly, this class should be chosen independently of the target polynomial. That is, we first fix the class of admissible probability distributions $\mathcal{D}$ and then analyze the expected complexity of learning from remainder sequences drawn with respect to some distribution from the class $\mathcal{D}$.

For defining $\mathcal{D}$, all polynomials from $\mathbb{F}_q[x]$ should have a non-zero probability except the zero polynomial. Since $\mathbb{F}_q[x]$ is infinite, the limit superior (as n tends to infinite) of the probability to show up for a polynomial of degree n has to be zero. That is, high degree polynomials have low probability and the higher the degree the smaller is the relevant probability. On the other hand, there is no reasonable cause to assume different probabilities for polynomials of the same degree. Thus, the distributions in $\mathcal{D}$ considered in the present work are all of a quasi uniform type.

The complexity of our learning algorithms is analyzed with respect to two average measures: The *example complexity* is the average number of examples used by the learning algorithm until it comes up with the correct hypothesis. The *time complexity* is the average number of computation steps until the correct polynomial g is found. Naturally, the example complexity is also a lower bound for the time complexity.

Next, we introduce some notions and notations used within the present paper.

By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the set of all natural numbers, and we set $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. The symbol $\mathbb{Z}$ is used for the set of all integers. Any finite field is denoted by $\mathbb{F}$. If $\mathbb{F}$ is a finite field, we write p for its characteristic and q for its order. Note that p is always a prime and $q = p^\ell$ for some $\ell \in \mathbb{N}^+$. Therefore, we usually write $\mathbb{F}_q$ for the, up to isomorphism, unique field of order q. For more information concerning finite fields the reader is referred to Berlekamp [3].

If g is any polynomial; then we use $\deg(g)$ to denote its degree.

Finally, we recall the following important proposition from probability theory that will be used frequently.

PROPOSITION 1 (Feller [8]). *Assume that there is a source of examples such that every example has with probability $r > 0$ a certain property u. Then the average number of examples to be drawn until some example satisfying u comes up is just $\frac{1}{r}$. The average number of examples necessary to draw n such examples is $\frac{n}{r}$.*

This paper is organized as follows. In Section 2 we present a new algorithm for learning integer valued polynomials from sequences argument-value and analyze its average-case complexity with respect to a rather large class of probability distributions. The average-case complexity of learning polynomials over finite fields from remainder sequences is studied in Section 3. Finally, we outline conclusions.

2 Learning Polynomials on Natural Numbers

Throughout this section, the target class is the set of all integer-valued polynomials. For learning a target polynomial g, the source of information given to the learner are pairs $(x, g(x))$, where $x \in \mathbb{N}$. Next, we have to specify the class of admissible probability distributions $\mathcal{D}$ over $\mathbb{N}$. That is, a datum x will then be drawn with probability $p(\{x\})$, and the learner is fed $(x, g(x))$. For defining a rather rich class of probability distributions, we make only the following two assumptions about p. First, $p(\{x\}) > 0$ for all $x \in \mathbb{N}$, since we have no reason to distinguish any $x \in \mathbb{N}$ by assigning probability 0 to it. Second, $p(\{x\}) \geq p(\{y\})$ for all $x, y \in \mathbb{N}$ with $x \leq y$. The motivation for the second assumption is as follows. Since $\mathbb{N}$ is infinite, the limit superior of $p(\{x\})$ (as x tends to infinite) has to be zero. That means, large numbers must have low probability and the larger x is the lower is the relevant probability.

As we shall see later, it will be more convenient to deal with the probability that some datum from the set $\{x, x+1, x+2, \ldots\}$ has been drawn rather than a particular one. Therefore, we specify the class $\mathcal{D}$ via functions f defined by $f(x) = p(\{x, x+1, \ldots\})$, i.e., f describes the probability to draw some datum from $\{x, x+1, x+2, \ldots\}$. Now, one easily verifies that f is decreasing, and the following properties are fulfilled:

(1) $f(0) = 1$,

(2) $\lim_{x\to\infty} f(x) = 0$,

(3) $f(x) > 0$ for all $x \in \mathbb{N}$, and

(4) $f(x) - f(x+1) \geq f(x+1) - f(x+2)$.

So f is very parallel to the parameter function f used in the next section. $\mathcal{D}$ is now the set of all probability distributions generated by a decreasing function f satisfying (1) through (4) above.

Interpolation is the best known method for learning polynomials. It returns to all data-items a hypothesis in polynomial time and its average-case complexity is a bit below $\frac{1}{f(0)\cdot f(1)\cdot\ldots\cdot f(n)}$ (cf. [20]):

- There are polynomial time algorithms to compute a formula for a polynomial g of least degree interpolating given data (x_0, y_0), (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n).
- It needs $n+1$ different examples for identifying a polynomial of degree n; the average example complexity is just the number of draws necessary until $n+1$ different values are obtained. An upper bound of the expected example complexity of learning polynomials by interpolation is then
$$\frac{1}{f(0)\cdot f(1)\cdot\ldots\cdot f(n)} .$$

Gold [9] as well as Jantke and Beick [13] showed that learning by enumeration is optimally data-efficient. The same is true also for interpolation.

PROPOSITION 2. *Interpolation is optimally data-efficient.*

Proof. Interpolation returns to any $n+1$ different data-items a polynomial of degree n and keeps that hypothesis until some data comes up showing that it is incorrect. Assume by way of contradiction that some other algorithm M would be strictly more data-efficient than interpolation. Then M learns some polynomial g before it is interpolated, that is, M outputs a correct hypothesis for g on some data (x_0, y_0), (x_1, y_1), $\ldots$, (x_n, y_n) although $\deg(g) > n$. As a consequence, M does not identify the polynomial g' interpolating this sequence from these $n+1$ data-items, which contradicts the fact that M is at least as data-efficient as interpolation. ∎

So alternatives to interpolation are more incomparable to it than better on all possible polynomials and data-sequences. The main axiom of interpolation is that the easiest way to describe a set of data is to take the polynomial of least degree interpolating it. That is, interpolation is based on the assumption that a data-item of the form
$$(10000, 100020001)$$
is more likely to describe the polynomial
$$g(x) = 100020001$$
than the polynomial
$$h(x) = x^2 + 2x + 1 ;$$
so the size of the coefficients is totally ignored. The subsequent model therefore tries to find for given data rather a polynomial having small absolute values of the coefficients rather than being of small degree. This approach leads us to a definition of the *size* of a polynomial that takes into account not only the degree but also the absolute value of the coefficients. Integer valued polynomials may have rational coefficients, for example the clearly integer-valued polynomial
$$x \to 0 + 1 + \ldots + x$$
has the formula
$$\frac{1}{2}x^2 + \frac{1}{2}x .$$
Every integer-valued polynomial of degree k is of the form
$$g(x) = a_0\binom{x}{0} + a_1\binom{x}{1} + \ldots + a_k\binom{x}{k},$$
where all $a_i \in \mathbb{Z}$, $a_k \neq 0$ and
$$\binom{x}{h} = \frac{x\cdot(x-1)\cdot\ldots\cdot(x-h+1)}{1\cdot 2\cdot\ldots\cdot h}$$
is the binomial coefficient for x and h. Note that
$$\binom{x}{h} = 0 \text{ for all } x < h .$$

For example,

$$g(x) = \frac{1}{2}x^2 + \frac{1}{2}x$$

has then the form

$$g(x) = \binom{x}{1} + \binom{x}{2} ,$$

i.e., $a_0 = 0$, $a_1 = a_2 = 1$.

Using this general form, a natural definition of the size of a polynomial g is

$$\mathrm{size}(g) = \max\{1, \deg(g), |a_0|, |a_1|, \dots, |a_k|\}.$$

Here $\deg(g) \le k$, and equality holds in the case that $a_k \ne 0$. The next theorem shows that — with high probability — every polynomial of size n can be learned within polynomial time from logarithmically many examples.

THEOREM 3. *Let g be any target polynomial whose size is unknown to the learner. Then, an optimal algorithm learning g from data-items drawn at random with respect to f needs*

(1) *at least $(f(\mathrm{size}(g)))^{-1}$ and*

(2) *at most $(f(5 \cdot (\mathrm{size}(g))^2))^{-1}$ many examples until convergence.*

Proof. For proving Assertion (1), consider the two polynomials $x \to \binom{x}{n}$ and $x \to -\binom{x}{n}$. They have size n and do not differ at the places $0, 1 \dots, n-1$. Thus, the learner has to see some pair $(x, g(x))$ with $x \ge n$ to make up its mind. So, the expected number to draw examples until such a pair comes up is $\frac{1}{f(\mathrm{size}(g))}$.

For proving Assertion (2), consider the following algorithm:

```
Let $(x, g(x))$ be the data-item with the largest $x$ seen so far.
Let $n$ be the largest natural number such that $5n^2 \le x$.
Initialize $a_0, a_1, \dots, a_n$ with the value $-n$.
For $m = n, n-1, \dots, 0$ increment $a_m$ until
    either $a_m = n$ or $g'(x) + \binom{x}{m} > g(x)$ for the
    polynomial $g'$ defined by the current values of
    the coefficients $a_0, a_1, \dots, a_n$.

Output $g(x) = a_0 + a_1\binom{x}{1} + \dots + a_n\binom{x}{n}$.
```

It is easy to see, that this algorithm converges. Since each a_m is increased at most $2n$ times and since there are only n variables, the whole algorithm needs to compute the current values of $g'(x) + \binom{x}{m}$ at most $2n^2$ times.

Next, it is shown that the output is correct under the assumption that $\mathrm{size}(g) \le n$.

Assume now by way of contradiction, that the algorithm terminates with some g'' such that $g'' \ne g$. Let $b_0, b_1, \dots, b_n$ be the coefficients of g and $a_0, a_1, \dots, a_n$ those of g'' . There is a largest m such that $b_m \ne a_m$.

If $a_m < b_m$, then the algorithm stops at the m-th loop before incrementing a_m to b_m , in particular, $-n\binom{x}{0} - n\binom{x}{1} - \dots - n\binom{x}{m-1} + (a_m + 1)\binom{x}{m} + a_{m+1}\binom{x}{m+1} + \dots + a_n\binom{x}{n} > g(x)$. This implies by $a_k = b_k$ for $k > m$ that $-n\binom{x}{0} - n\binom{x}{1} - \dots - n\binom{x}{m-1} + (a_m + 1)\binom{x}{m} > b_0\binom{x}{0} + b_1\binom{x}{1} + \dots + b_{m-1}\binom{x}{m-1} + b_m\binom{x}{m}$. This assumption contradicts to the fact that the $a_m + 1 \le b_m$ and $-n \le b_k$ for all k . So this case does not occur.

Otherwise $a_m > b_m$. This implies that $-n\binom{x}{0} - n\binom{x}{1} - \dots - n\binom{x}{m-1} + (b_m + 1)\binom{x}{m} \le b_0\binom{x}{0} + b_1\binom{x}{1} + \dots + b_{m-1}\binom{x}{m-1} + b_m\binom{x}{m}$ since otherwise a_m could never been incremented to a value greater than b_m . So one gets that $\binom{x}{m} \le (b_0 + n)\binom{x}{0} + (b_1 + n)\binom{x}{1} + \dots + (b_{m-1} + n)\binom{x}{m-1}$. Using $b_k \le n$, $m \le \frac{x}{5}$ and that, for $k < \frac{x}{5}$, $\binom{x}{k} < \frac{1}{2}\binom{x}{k+1}$ one gets that $\binom{x}{m} \le 2n \cdot (2^{1-m} + 2^{2-m} + \dots + 2^0)\binom{x}{m-1} \le 4n \cdot \binom{x}{m-1}$. In particular $x + 1 - m \le 4nm$ which contradicts the requirements $m \le n$ and $x \ge 5n^2$ from the choice of m and n . Thus, $a_m > b_m$ is impossible either.

So, it follows from the case-distinction that $g'' = g$ whenever the parameter n is an upper bound for the size of g . In particular the example complexity of the algorithm is $(f(5 \cdot (\mathrm{size}(g))^2))^{-1}$. ∎

Before discussing further consequences of our Theorem 3, we shortly illustrate the learner described in the proof above.

EXAMPLE 4. Let g with $g(x) = 1$ for all x be the target polynomial to be learned by the algorithm given in the proof above. Thus, $\mathrm{size}(g) = 1$ and we therefore consider the data item $(5, 1)$ which has the smallest possible x .

Now, first the algorithm computes $n = 1$, and therefore $g'_1(x) = -x - 1$. During the first loop, i.e., $m = 1$, only the coefficient a_1 is possibly changed while the actual $a_0 = -1$ remains unchanged.

Since $a_1 = -1 \ne 1 = n$, the algorithm then tests $-5 - 1 + 5 > 1$ which is false. Therefore, $g'_2(x) = -1$. Since $0 \ne 1$, it tests $-1 + 5 > 1$ which is true. This finishes the loop for a_1 with $a_1 = 0$, and the second loop, i.e., $m = 0$, is started with $g'_2(x) = -1$. Now, the condition to be checked is $-1 = 1$ or $-1 + 1 > 1$ which returns false. Thus, $g'_3(x) = 0$. Since $0 \ne 1$, it tests $0 + 1 > 1$ which is false. Finally, $g'_4(x) = 1$, and the algorithm terminates, since $1 = 1$. Hence, it has correctly learned the target g .

Note that the time-complexity of the algorithm presented in the proof of Theorem 3 could be improved by searching the a_m via interval search; the main reason for giving the algorithm as above was to get an easier verification. The following example gives concrete bounds for the case that the distribution is given by

$$f(n) = \frac{1}{\log(n+2)} \ .$$

Example 5. *There is an algorithm which learns polynomials with example complexity*

$$\log(2 + 5 \cdot (\mathrm{size}(g))^2)$$

where the distribution is given by $f(n) = \frac{1}{\log(n+2)}$.

Next, we compare the latter result to learning by interpolation. Let $n > 1$; there are $(2n+1) \cdot (2n+1)^n$ many polynomials having size at most n . Among these polynomials, there are $2n \cdot (2n+1)^n$ many polynomials that have degree n . For these polynomials of degree n the standard interpolation procedure requires $n+1$ different data-items. On the other hand, for the distribution described by $f(n) = \frac{1}{\log(n+2)}$, our algorithm needs for all polynomials of size at most n only $\log(5n+2)^2$ many examples. So the example complexity of the algorithm presented above is for the great majority of the polynomials of size at most n better than standard interpolation when measured with respect to the distribution described by $f(n) = \frac{1}{\log(n+2)}$ and taking the size instead of the degree as a key parameter.

On the other hand, interpolation beats this algorithm if the underlying probability distribution is given by $f(n) = 2^{-n}$. Then the example complexity to get all values $g(0), g(1), \ldots, g(n)$ is $2^{n(n+1)/2}$ while the example complexity to get one value $g(x)$ with $x \geq 5n^2$ is 2^{5n^2} .

Next, we turn our attention to the problem that analyzing only the expected number of examples needed until convergence is not so interesting. As often criticized by statisticians, expected values alone are not so informative. Thus, we are interested in knowing on how often the example complexity exceeds the average substantially.

For that purpose, we first note that the learner presented in Theorem 3 has two important properties, i.e., it is conservative and set-driven. A learner is set-driven, if its output depends only on the range of its input (cf. Wexler and Culicover [22]), while conservative learners maintain their actual hypotheses at least as long as they have not seen data contradicting them (cf. Angluin [1]). Now, let X be a random variable standing for the example complexity of the algorithm in the proof of Theorem 3, and let $E[X]$ denote the expectation of X . Then, using a result by Rossmanith and Zeugmann [19], we directly obtain the following corollary.

Corollary 6. $\Pr(X \geq 2tE[X]) \leq 2^{-t}$ *for all* $t \in \mathbb{N}$.

Consequently, our learner additionally possesses *exponentially* shrinking tail bounds for the example complexity. We can therefore transform the above algorithm into one that probably *exactly* identifies every target polynomial of size at most n provided a computable upper bound $\hat{f}$ for f is available to the learner. Probably exact learning is defined in the same way as PAC learning, except that the hypothesis output has to be correct rather than approximately correct. Additionally, it assumes a bit knowledge concerning the underlying distributions.

Theorem 7. *Let* $n \in \mathbb{N}$ *, and let* $\hat{f}$ *be a computable upper bound for* f *. Then there is an algorithm probably exactly learning every target polynomial of size at most* n *for every distribution* $f \in \mathcal{D}$ *with* $f(x) \leq \hat{f}(x)$ *for all* $x \in \mathbb{N}$ *using* $(2\hat{f}(5n^2))^{-1} \log \frac{1}{\delta}$ *many examples.*

Proof. The desired learner takes any $\delta \in (0,1)$ as additional input. Now, using the algorithm defined in the proof of Theorem 3, the learner performs $\log(1/\delta)$ many rounds.

In each round it requests $(2\hat{f}(5n^2))^{-1}$ many examples. During the first round, it computes a hypothesis by using the learner from Theorem 3. By Corollary 6, this hypothesis is already correct with probability at least $1/2$. In the remaining rounds, a new hypothesis is only computed if a data item $(x, g(x))$ is received such that $x > y$ for all y in the data items $(y, g(y))$ received so far. The last hypothesis computed is then output.

Therefore, the probability that none of the computed hypotheses is correct can be bounded by δ , i.e., with probability $1 - \delta$ the learner outputs a correct hypothesis after having processed $(2\hat{f}(5n^2))^{-1} \log(1/\delta)$ many examples. Since $\hat{f}$ is an upper bound for f , the theorem follows. ▮

We finish this section by comparing the latter result to PAC learning. Clearly, the PAC model is distribution free, and therefore, the best one can hope for is to get better bounds on the number of examples needed for special classes of distributions. This is indeed the case as we shall see.

Assuming n to be again a bound on the *size* of all target polynomials, one can easily determine the VC dimension of the class of all these polynomials to be n . Thus, a PAC learner needs at least $O(\frac{1}{\varepsilon} \log \frac{1}{\delta} + \frac{n}{\varepsilon})$ many examples. Compared to the bound in Theorem 7, this bound is worse provided $\hat{f}(n) < \frac{1}{\sqrt{n}+1}$.

Further generalizations are possible by removing the parameter n for the upper bound of the size. For the PAC model, this has been shown in [11]. Their technique can be generalized to the setting considered in Theorem 7.

3 Learning from Remainder Sequences

Next, we turn our attention to learning univariate polynomials over finite fields. As already mentioned in the introduction, now the situation is slightly different from that studied in the previous subsection, since even complete sequences of pairs argument-value do not provide enough information to learn the target polynomial.

Therefore, we propose a new source of information, i.e., learning from remainder sequences. That is, now the learner has access to pairs (a, b), where a is any polynomial and b is the remainder of the target polynomial when divided by a. Again, we require the learner to infer the target in the *limit* from any such sequence.

This information shares some nice properties with the standard source of information given by sequences of argument-value. First, it is also easy to compute (only a bit more complicated than polynomial evaluation). Second, it contains enough information to learn the target polynomial. Third, though one can learn the target from this information, learning does not become easier in the sense that learning in the limit could be replaced by finite learning. Here, finite learning means that the learner can decide whether or not it has already successfully finished its learning task (cf. Gold [9]).

In some sense, learning even becomes harder. As we have already said, in the standard case it is always sufficient to have $n+1$ data items argument-value to learn any polynomial of degree n. This nice property is lost in our new model. Taking into consideration that there are $(q-1)q^m$ different polynomials in $\mathbb{F}_q[x]$ of degree precisely m, it is easy to see that it may take exponentially many examples (in the degree n of the target polynomial) until convergence. On the other hand, a single data item is sufficient in the best-case (if $\deg(a) > \deg(g)$, where g is the target polynomial).

Consequently, in the new setting the best-case and worst-case are overly optimistic and pessimistic, respectively. Therefore, we study the average-case complexity of this learning task. This requires the introduction of a class of probability distributions over the whole set $\mathbb{F}_q[x]$ of polynomials. Since $\mathbb{F}_q[x]$ is infinite, there is no uniform distribution. We therefore consider a rather rich class of quasi uniform distributions over $\mathbb{F}_q[x]$ defined as follows.

Definition 8. For every finite field $\mathbb{F}_q$ we define the class $\mathcal{D}$ of *quasi uniform distributions* over $\mathbb{F}_q[x]$ to be the set of all probability distributions generated by any decreasing function f satisfying

(1) $f(0) = 1$,

(2) $\lim\limits_{n\to\infty} f(n) = 0$,

(3) $f(n) \neq 0$ for all $n \geq 1$, and

(4) $f(n) - f(n+1) \geq f(n+1) - f(n+2)$.

The probability of a non-zero polynomial a is then just

$$(f(\deg(a)) - f(\deg(a)+1)) \cdot \frac{1}{(q-1)\cdot q^{\deg(a)}}$$

Conditions (1) and (2) in Definition 8 are necessary to obtain a probability distribution. For seeing this, note that there are precisely $(q-1)\cdot q^m$ many polynomials of degree m. Thus, the probability to draw some polynomial a of degree d is $(f(\deg(a)) - f(\deg(a)+1))$. Now, looking at the sequence $(S_m)_{m\in\mathbb{N}}$ of partial sums, we get that

$$S_m = \sum_{d=0}^{m} (f(d) - f(d+1)) = f(0) - f(d+1) ,$$

and hence $\lim_{m\to\infty} S_m = f(0) = 1$.

Furthermore, Condition (3) ensures that all polynomials have a non-zero probability. Finally, Condition (4) formalizes the requirement that all polynomials of degree $n+1$ have a lower probability than those of degree n.

First, we establish lower and upper bounds for learning from remainder sequences. In the following, whenever talking about learning algorithms for polynomials from $\mathbb{F}_q[x]$, it is assumed that all these algorithms infer to *whole* target class $\mathbb{F}_q[x]$.

Theorem 9. *Let $g \in \mathbb{F}_q$ be any target polynomial of degree n (where n is unknown to the learner). Then we have:*

(1) *Every algorithm learning the target polynomial g from remainder sequences drawn at random with respect to f needs at least*
$$(f(\log_q(n) - 2\log_q\log_q(n)))^{-1}$$
many examples until convergence.

(2) *There exists an algorithm learning g from remainder sequences drawn at random with respect to f that needs at most $(f(n+1))^{-1}$ many examples until convergence.*

Proof. For proving Assertion (1), choose the unique m such that $mq^{m+1} < n \leq (m+1)q^{m+2}$. Let g_1 be the product of all polynomials of degree up to m. There are precisely $(q-1)\cdot q^d$ many polynomials of degree d; thus the degree of g_1 can be upper bounded by

$$m\cdot(q-1)\cdot\sum_{d=1}^{m} q^d \leq m\cdot q^{m+1} .$$

Let g_2 be any polynomial of degree $n - \deg(g_1)$. Now the lower bound is obtained by analyzing the expected number of examples necessary for learning the polynomial $g = g_1 g_2$. Any polynomial of degree up to m is a divisor of g_1. Therefore, no data (a, b) with $\deg(a) \leq m$ gives any information on the particular form of g_2. Hence, the learner cannot succeed before seeing at least one data item (a, b) with $\deg(a) > m$. Such a polynomial a occurs with probability at most $f(m+1)$ and therefore, the example complexity is at least $(f(m+1))^{-1}$ which is an upper bound for $(f(\log_q(n) - 2\log_q\log_q(n)))^{-1}$.

For proving Assertion (2) consider the learning algorithm that always conjectures b for the pair (a, b) seen so far where a has the highest degree. Note that $b = p$ whenever $\deg(a) > \deg(p)$. Since such an a has to be drawn up eventually, the algorithm converges. It remains to show the example bound given by this algorithm matches the bound given in (2). The event that $\deg(a) > n$ has probability $f(n+1)$ and thus, by Proposition 1, the expected number of examples is $(f(n+1))^{-1}$ until such a polynomial a shows up. ∎

Next, we discuss the bounds obtained in Theorem 9 by looking at very slowly and very rapidly decreasing functions f. For the former, let $f(2^n) = \frac{1}{2}f(n)$ and require f to be uniformly decreasing for all arguments inbetween two consecutive powers of 2. That is, $f(0) = 1$ by Condition (1) in Definition 8, $f(1) = 1/2$, $f(2) = 1/4$ $f(3) = 3/16$, $f(4) = 1/8$, $f(5) = 15/128$, $f(6) = 14/128$, $f(7) = 13/128$, $f(8) = 3/32$, ..., $f(16) = 1/16$, ..., and so on. Now, it is easy to see that the difference between the bounds given in Theorem 9 is only a constant factor.

But for distributions given by other functions f like $f(n) = q^{-n}$ the gap is large. Now, the upper bound

$$q^{n+1}$$

and the lower bound

$$q^{\log(n)-2\log\log(n)} = n \cdot (\log(n))^{-2}$$

differ exponentially.

Consequently, while the gap between lower and upper bound in Theorem 9 can be large for some distribution in $\mathcal{D}$, one can expect better bounds for particular distributions. So the next two results improve the bounds for the distribution given by $f(n) = q^{-n}$.

THEOREM 10. *Let $g \in \mathbb{F}_q[x]$ be any target polynomial of degree n (n again unknown to the learner). Then every algorithm learning g from remainder sequences drawn at random with respect to the distribution generated by $f(n) = q^{-n}$ has at least example complexity $q^{-4} \cdot \frac{n^2}{\log(n)}$ for sufficiently large n.*

Proof. (Sketch) This lower bound is obtained by adapting the previous proof and by exploiting special knowledge on the distribution. Let m be the unique number with $(q+m+1)q^{m+1} < n \leq (q+m+2)q^{m+2}$. As in the proof of Theorem 9 let g_1 be the product of all monic polynomials of degree less than or equal to m and g_2 is some polynomial of degree $n - \deg(g_1)$. Since the degree of g_1 is at most mq^{m+1}, the degree of g_2 is at least q^{m+2}. As in the previous lower bound proof, again only polynomials whose degree is at least $m+1$ contribute to some knowledge on g_2. The expected degree of these polynomials can be upper bounded by $m+3$. In order to reconstruct g_2 the sum of the degrees of these polynomials must be q^{m+2} and so at least $\frac{q^{m+2}}{m+3}$ examples are needed to do the job. Also only one out of q^m examples qualifies to have at least degree m, so a lower bound for the example complexity is $\frac{q^{2m+2}}{m+3}$. Since $n \leq (q+m+2)q^{m+2}$ and $\log(n) \geq m$, one can estimate $\frac{q^{2m+2}}{m+3} \geq n^2 \cdot q^{-3} \cdot (q+m+2)^{-1} \geq q^{-4} \cdot \frac{n^2}{\log(n)}$ and get that, for sufficiently large n, $q^{-4} \cdot \frac{n^2}{\log(n)}$ examples are necessary to learn any given polynomial of degree n. ∎

Theorem 9 gives for $f(n) = (n+1)^{-k}$, $k \in \mathbb{N}^+$, directly an algorithm which learns every polynomial g of degree n with average example complexity $(n+2)^k$. Considering that every example has to be read through once, the time complexity of the algorithm is $(n+2)^{k+1}$.

This does not longer work for distributions like $f(n) = q^{-n}$. Here the lower bound is polynomial, but the upper bound is exponential. Therefore, the next theorem shows how to improve the upper bound from exponential to polynomial time. So both bounds are under this particular distribution quadratic modulo some poly-logarithmic term.

THEOREM 11. *There is an algorithm that learns every polynomial from $\mathbb{F}_q[x]$ from remainder sequences drawn at random with respect to the probability distribution generated by $f(n) = q^{-n}$ that needs on average at most $q^7 n^2$ many examples and time $O(n^2 \log(n))$ until convergence on target polynomials of degree n.*

Proof. The learner does not use all data-items (a, b) but only those which belong to irreducible polynomials a. Furthermore, one can divide the polynomial a by its leading coefficient. Note that this division does not change the remainder b. Thus, we can assume a to be monic without changing the remainder b. This normization enforces that equivalent irreducible polynomials like x^2+1 and $2x^2+2$ occur only in the form x^2+1. So one makes the polynomials monic and performs a test for irreducibility before further processing. As a consequence the learning algorithm avoids factoring and similar work. Furthermore before updating its hypothesis, the learner first checks whether it has already seen the data coming in. This check allows to process each irreducible polynomial at most once and to establish the low time complexity of the learning process. All data, which pass these two checks, are then used in order to construct a hypothesis about the polynomial to be learned by applying the Chinese Remainder Theorem. As soon as the learner has accumulated a sufficient large basis of coprime (even irreducible) polynomials, the target is fully described by its corresponding remainders. We continue with a formal description.

Within the algorithm, (a, b) denotes the current input data, h is a variable standing for the hypothesis about the polynomial to be learned, L is just the set of all irreducible polynomials for which the corresponding remainder of the target is known and d is the product of all polynomials in L. The variable d just keeps this product in order to avoid doing the same multiplication several times. Now the formal algorithm is presented. Note that this algorithm outputs only finitely many hypotheses the last of which will correctly describe the target to be learned.

```
Initialize $h = 0$, $d = 1$ and $L = \emptyset$ and do
the following for ever:
Read $(a, b)$.
Make $a$ monic, that is, divide $a$ by its leading
coefficient.
If $a$ is irreducible and $a \notin L$ then
    Compute the remainder $c$ of $h$ modulo $a$.
    Let $L = L \cup \{a\}$.
    If $b \neq c$ then
        Compute the smallest $e$ such
        that $e \cdot d$ has the remainder
        $b - c$ modulo $a$.
        Let $h = h + e \cdot d$.
        Output the new hypothesis $h$.
    Let $d = a \cdot d$.
Continue the loop with reading the next data-
items.
```

First note that the polynomials in L are all monic and irreducible and thus co-prime. Furthermore, d is always the product of these polynomials and h satisfies $(\exists q)\,[h = aq + b]$ for all irreducible polynomials a and the corresponding b seen so far. This can be proved, for every $a \in L$ and the corresponding b, as follows: At some step, the algorithm processes (a, b) in the loop and the innermost loop guarantees that the new h has the correct remainder by choosing e such that $e \cdot d$ has the remainder $b - c$; then the sum $h + e \cdot d$, which gives the new h, has the remainder $c + (b - c) = b$ modulo a. In all later steps, a divides d and only terms having the remainder 0 modulo a are added to h so that the correct remainder modulo a is preserved.

Second, it is an invariant of the construction that $\deg(h) < \deg(d)$: It holds at the initialization (by defining $\deg(0) < 0$) and whenever the values of h and d are updated to $h + ed$ and ad, respectively, then the property is preserved since $\deg(h+ed) < \deg(ad)$. This can be seen as follows: The degree of ad is the sum $\deg(a) + \deg(d)$. Furthermore, the degree of $h + ed$ is bounded by $\max\{\deg(h), \deg(ed)\}$ where $\deg(h) < \deg(d) < \deg(a) + \deg(d)$ and $\deg(ed) = \deg(e) + \deg(d) < \deg(a) + \deg(d)$. This last relation is based on the fact that $\deg(e) < \deg(a)$ and this fact is the consequence of the observations, that a is irreducible and so e can be chosen as the smallest representative of the quotient $(b - c)/d$ in the field of polynomials modulo a; this quotient is well-defined since a and d are co-prime. Note that $e = 0$ is the solution for the case $b - c = 0$, therefore one can abstain from processing the inner-most loop in this case.

Third, if h' is the target polynomial and $\deg(d) > \deg(h')$, that is, $\deg(d) > n$, then $h = h'$ since, by the Chinese Remainder Theorem, h and h' are both the unique polynomial of degree below $\deg(d)$ which has, for all $a \in L$, the corresponding remainder b when divided by a.

Fourth, there are infinitely many irreducible monic polynomials a and each a is eventually presented together with the corresponding remainder b. So the event $\deg(d) > n$ must happen eventually and the algorithm converges to the correct hypothesis.

It remains to verify the time bound given and to estimate the expected number of examples necessary until convergence.

First, we compute the number of examples necessary. For that purpose it suffices to ask how many examples are necessary until d has reached some degree greater than n. To get such an estimation, let m be the least degree such that there are $\frac{2n}{m}$ irreducible polynomials of degree m in $\mathbb{F}_q[x]$. Using the lower bound

$$I_m > \frac{q^m - q^{m/2+1}}{m}$$

on the number I_m of monic irreducible polynomials of degree m in $\mathbb{F}_q[x]$ (cf. Berlekamp [3]), we obtain that $2n$ is near to q^m, more precisely, that $q^{m-2} < 2n < q^{m+2}$. As long as less than n of the irreducible polynomials of degree m have occurred in the data seen so far, the probability of getting one further one of them is $\frac{n}{m} \cdot q^{-2m-1}$ where q^{-m-1} is the probability of getting a polynomial of degree at least m and $\frac{n}{m} q^{-m}$ is a lower bound for the probability that this polynomial is among the still unseen irreducible polynomials of degree at least m. The expected number of examples necessary to receive $\frac{n}{m}$ different irreducible polynomials of degree at least m can therefore be estimated by the upper bound q^{2m+1}. The degree of their product is n so that after q^{2m+1} many examples the size of d is n. Note that $2n > q^{m-2}$ and thus $q^{2m+1} \leq 4n^2 \cdot q^5 \leq q^7 n^2$. So $q^7 n^2$ is an upper bound on the expected number of examples needed until convergence.

Now the complexity of each step is analyzed. There are two parts, which have to be dealt separately with:

(a) the part which is done for every data-item, and

(b) the part within the first "if"-statement after the "then" which is not executed for most of the data-items.

The part (a) consists of reading the hypothesis of (a, b), the check whether a is irreducible and the test whether $a \in L$. The test whether $a \in L$ has the time complexity $\deg(a) \log(|L|)$ by keeping L as an ordered list. The size of L does not exceed n. So for each data-item, the computations of type (a) need with probability q^{-k-1} the time $p(k) \log(n)$ for some polynomial p. Since the sum over $q^{-k-1} p(k)$ converges to some constant r, the step has time complexity $r \log(n)$ for each single data item and time complexity $rq^7n^2\log(n)$ in total.

In part (b) let k denote the degree of a. The degree of d and the size of L do not exceed n when entering this part of the algorithm. Now computing the remainder c needs $O(nk)$ time steps and gives a polynomial of degree below k. $b-c$ is computed in $O(k)$ time steps, e is computed in $O(nk)$ steps where e is obtained by taking the smallest representative of the quotient $\frac{b-c}{d}$ in the field generated by the irreducible polynomial a, the multiplication $e\cdot d$ needs $O(nk)$ time steps, the addition to and update of h needs $O(n)$ steps, the product ad is computed within $O(nk)$ time steps, the update of d to ad needs $O(n)$ time steps and the update of L needs $O(n)$ time steps (cf. [4]). So a single run through part (b) needs $r'nk$ time steps where r' is some constant. During the whole time, the degrees of the polynomials summed up satisfy $k_1+k_2+\ldots+k_{last}\leq n+k_{last}$. Hereby it can be estimated that k_{last} does not exceed n with reasonable probability. So one has that all runs through part (b) together have time complexity $r'(k_1+k_2+\ldots+k_{last})n\leq 2r'n^2$.

The total example complexity of the algorithm has the upper bound q^7n^2 and the time complexity has upper bound $(rq^7\log(n)+2r')n^2$, that is, $O(n^2\log(n))$. ∎

The next example illustrates the learning algorithm of Theorem 11 for a concrete polynomial over the finite field with three elements.

Example 12. Assume that the polynomial x^3 in $\mathbb{F}_3[x]$ should be learned; note that $-1=2$ in $\mathbb{F}_3$. Consider some data-sequence starting with $(x^2,0)$, $(x^2+2,2x)$, $(x+1,2)$, $(x+2,1)$, $(x^2+1,2x)$, $(x+1,2)$, $(x,0)$, $(a(x),b(x))$ would be the initial part of the data-sequence where $a(x)$ is an irreducible and monic polynomial not seen before. From these data-items, $(x^2,0)$ and $(x^2+2,2x)$ do not qualify since x^2 and x^2+2 are not irreducible in $\mathbb{F}_3$. Furthermore, the second occurrence of $(x+1,2)$ also does not qualify, since $x+1\in L$ after the first occurrence of $(x+1,2)$. The following table gives now an overview on the values of the other variables of the algorithm after executing the interior loop where data, which did not qualify, is omitted. The current value of h is also always the current hypothesis.

data-item	c	e	d	h
$-$	$-$	$-$	1	0
$(x+1,2)$	0	2	$x+1$	2
$(x+2,1)$	2	1	x^2+2	x
$(x^2+1,2x)$	x	x	x^4+2	x^3
$(x,0)$	0	0	x^5+2x	x^3
$(a(x),b(x))$	$b(x)$	0	$(x^5+2x)b(x)$	x^3

If d' and h' are the values of d and h from the previous row, then the update rule for these two variables is $d=a\cdot d'$ and $h=h'+e\cdot d'$. So, the updates of h are from 0 to $0+2\cdot 1=2$, then from 2 to $2+1\cdot(x+1)=x$ and finally from x to $(x+1)+x\cdot(x^2+2)=x^3$. From then on, $b-c$ and thus also e are always 0 and no further updates are done, that is, the learner has stabilized on the correct hypothesis x^3. After processing some data-item, L contains all monic and irreducible polynomials processed from the beginning up to the current data-item, so after processing $(a(x),b(x))$, the content of L are the polynomials $x+1$, $x+2$, x^2+1, x and $a(x)$.

Since $q=3$ and $n=3$, the average number of examples needed until successful learning has the upper bound $3^9=19683$. On the one hand, this bound is not optimal, but on the other hand, the above sample sequence was also a bit unrealistic in the sense that it contained much more useful data than a randomly distributed sequence of this length would give.

4 Conclusions

The learnability of univariate integer valued polynomials over the natural numbers and univariate polynomials over finite fields has been investigated. For both cases, we gave lower and upper bounds of the average example complexity. Measuring the convergence not relative to the degree of a polynomial but relative to a measure which takes into account also the size of the coefficients, standard interpolation is not any more the best possible algorithm. We found a quite natural distribution where the new learning algorithm gives a speed-up from polynomial to logarithmic example complexity on polynomials with small coefficients. Nevertheless, we show that interpolation is an optimally data-efficient strategy; so no other learning algorithm behaves on all input-sequences better than interpolation.

Since polynomials over $\mathbb{F}_q$ are not uniquely specified by their input-output-behavior, we chose as data-representation the remainder modulo other polynomials. In this model, the general gap between lower and upper bound of the example complexity obtained by optimal learning still is large for many distributions — we hope that future work might narrow this gap. But we could obtain much tighter results for the concrete distribution on the remainders induced by $f(n)=q^{-n}$.

References

[1] Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control, 45*, 117–135.

[2] D. Angluin and C.H. Smith. Inductive inference: theory and methods. *Computing Surveys* 15:237–269, 1983.

[3] E.R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, New York, 1968.

[4] D. Bini and V. Pan. *Polynomial and Matrix Computations: Volume 1, Fundamental Algorithms.* Birkhäuser, Boston, 1994.

[5] N.H. Bshouty. On learning multivariate polynomials under the uniform distribution. *Information Processing Letters* 61(6):303–309, 1997.

[6] N.H. Bshouty and Y. Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *in* "Proc. 36th Annual Symposium on Foundations of Computer Science," pp. 304–311, IEEE Press 1995.

[7] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger and T. Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *in* "Proc. 8th International Workshop on Algorithmic Learning Theory - ALT'97," (M. Li and A. Maruoka, Eds.), Lecture Notes in Artificial Intelligence 1316, pp. 260–276, Springer-Verlag 1997.

[8] W. Feller. *An Introduction to Probability Theory and its Applications, Vol. 1.* John Wiley & Sons, New York, 1968.

[9] M.E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[10] M. Golea. Average case analysis of a learning algorithm for μ-DNF expressions. *in* "Proc. 2nd European Conference on Computational Learning Theory - EuroColt'95," (P. Vitanyi, Ed.), Lecture Notes in Artificial Intelligence 904, pp. 342–356, Springer-Verlag 1995.

[11] D. Haussler, M. Kearns, N. Littlestone and M.K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95:129–161, 1991.

[12] D.S. Hirschberg, M.J. Pazzani and K.M. Ali. Average case analysis of k-CNF and k-DNF learning algorithms. *in* Computational Learning Theory and Natural Learning Systems, Vol. II: Intersections Between Theory and Experiment, pp. 15–28, MIT Press 1994.

[13] K.P. Jantke and H.R. Beick. Combining postulates of naturalness in inductive inference. *Elektronische Informationsverarbeitung und Kybernetik (EIK)* 17:465–484, 1981.

[14] P. Langley and S. Sage. Tractable average-case analysis of naive Bayesian classifiers. *in* "Proc. 16th International Conference on Machine Learning, pp. 220–228, Morgan Kaufmann 1999.

[15] S. Okamoto and K. Satoh. An average-case analysis of k-nearest neighbor classifier. *in* "Proc. 1st International Conference on Case-Based Reasoning Research and Development," Lecture Notes in Computer Science 1010, 253–264, Springer-Verlag, 1995.

[16] M.J. Pazzani and W. Sarrett, A framework for average case analysis of conjunctive learning algorithms. *Machine Learning* 9:349–372, 1992.

[17] R. Reischuk and T. Zeugmann. Learning one-variable pattern languages in linear average time. *in* "Proc. 11th Annual Conference on Computational Learning Theory - COLT'98," pp. 198–208, ACM Press 1998.

[18] R. Reischuk and T. Zeugmann. A complete and tight average-case analysis of learning monomials. *in* "Proc. 16th International Symposium on Theoretical Aspects of Computer Science - STACS'99," Trier, March 1999, (C. Meinel and S. Tison, Eds.), Lecture Notes in Computer Science 1563, pp. 414 - 423, Springer-Verlag 1999.

[19] P. Rossmanith and T. Zeugmann. Learning k-variable pattern languages efficiently stochastically finite on average from positive data. *in* "Proc. 4th International Colloquium on Grammatical Inference - ICGI'98," (V. Honavar and G. Slutzki, Eds.), Lecture Notes in Artificial Intelligence 1433, pp. 13–24, Springer-Verlag 1998.

[20] J. Stoer. *Numerische Mathematik, Vol. 1.* Springer-Verlag, Berlin 1989.

[21] L.G. Valiant. A theory of the learnable. *Commun. ACM* 27:1134-1142, 1984.

[22] K. Wexler and P. Culicover. *Formal Principles of Language Acquisition.* MIT Press, Cambridge, Mass., 1980.