Resource Bounded Next Value and Explanatory Identification
Learning Automata, Patterns and Polynomials On-Line *

Susanne Kaufmann †
Universität Karlsruhe

Frank Stephan ‡
Universität Heidelberg

Abstract

This paper considers learning via predicting the next value – this concept is also known as "on-line learning" or "forecasting". The concept is combined with the limited memory model and has two variants: Exact NV-learning has a polynomial resource bound depending on the sizes of current input and the concept on long term memory and on working space (or time); in addition the number of errors is limited by a polynomial in the concept size. Independent NV-learning has polynomial resource bounds depending on the size of the current input only on long term memory and on working space (time). The following is shown: A class of functions is independently NV-learnable iff it is uniformly computable in PSPACE. Exact NV-learning is a proper restriction of independent NV-learning. For the well-known classes of pattern languages, regular languages and polynomials, it is investigated under which variations of the resource bounds they are learnable or not learnable. Also an explanatory version of resource bounded learning is defined. It is more powerful than next value learning. Furthermore while next value learning is closed under union, this type of explanatory learning has a proper team hierarchy which is the same as in the standard case of explanatory learning without resource bounds.

1 Introduction

Inductive inference [3, 8, 12] means to learn functions in the limit where the learner makes up the hypothesis form larger and larger prefixes $f(0)f(1)\ldots f(x)$ of the function f. There are two basic concepts how to make the conjecture. Either the learner predicts from $f(0)f(1)\ldots f(x)$ the next value $f(x{+}1)$ [4] or the learner computes from this input a hypothesis e_x [12] which is intended to be a program for f. In both cases, the learner is expected to be correct for almost all x. The present work deals with resource bounded versions of these two basic types of learning.

There are several approaches how to introduce resource bounds to learning. All these approaches have to deal with one difficulty which is connected with the data-presentation: the machine might receive so many redundant data, that the time to process all this input is much higher than the computational complexity of the learning process. For example in the case of learning finite automata, it might be necessary to read 2^n data-items until a word is found on which two given automata with n states are different. The natural assumption that the learner may use at least as much time for its learning algorithm as for reading the input gives the learner the ability to use an exponential time algorithm and so this model makes it very difficult to analyse subexponential complexity of learning.

One way to overcome this problem arising from the data distribution is Angluin's model of a teacher [2]: the learner and the teacher communicate via a dialogue of queries and answers and the learner has to succeed in polynomial time measured relative to the problem size and the answers of the teacher. Littlestone [17, 18] designed a model of next value learning where the learner receives the data in arbitrary order and has to make on each sequence of data at most polynomially many false predictions. In this community, the notion NV is

†Interactive Systems Laboratories, Am Fasanengarten 5, Universität Karlsruhe, 76128 Karlsruhe, Germany, electronic mail ⟨kaufmann@ira.uka.de⟩.

‡ Mathematisches Institut, Im Neuenheimer Feld 294, Universität Heidelberg, 69120 Heidelberg, Germany, electronic mail ⟨fstephan@math.uni-heidelberg.de⟩. Supported by the Deutsche Forschungsgemeinschaft (DFG) grant Am 60/9-1.

also called on-line learning or forecasting. Valiant [24] considered randomly presented data and expected the learner therefore only to succeed on most of the data (where "most" is measured with respect to a given probability distribution) and in addition the learning process is permitted to fail totally with a low probability. All these approaches have in common that they modify the mode to present the data to the learner.

Freivalds, Kinber and Smith [11] followed another direction. They did not change the mode of data presentation but restricted the ability of the learner to remember previous data and computations. This approach can be motivated from the behaviour of natural learners like animals or humans: A human or animal is lifelong learning, that means during a long period processing each day a new collection of data. But most of this data is forgotten after processing it. For example, a 10 year old child is not able to say what has happened 5 years + 3 month + 2 days + 5 hours ago. But the child remembers a choice of important or impressive events in the past and furthermore has a lot of skills and data derived from the experience of the last 10 years. An other example of such type of learning is reading a book: the readers (or to be more exact: most of them) process books by dealing with one page after the other so that every page in the book is read once. Nevertheless they do not memorize the whole text but keep in the brain only some small relevant information on the book.

So the main idea is that the prime restriction during this type of learning is not the time but the limited memory of the learner. Freivalds, Kinber and Smith [11] formalized this idea and obtained so an alternative approach to introduce resource bounds in inductive inference. The learner works in stages and processes in each stage one current argument x plus the corresponding value of f; thereby the learner can obtain information on the previous values only by that which is stored in the long term memory. This memory is too small to store all previous values, thus the management of the long term memory is an important part of each learning algorithm.

There is also some work in on-line learning keeping the default ordering. Vovk [25] looked upon learners, which predict statistically always the next value on an infinite binary string. Vovk considered statistical and deterministic learners, which try to predict the next value. He put his accent on minimizing the number of errors and showed that there is a universal learner which is almost as powerful as any specialized learner designed for a certain class of strings. His time constraints are polynomial in the length of the string $f(0)f(1)\ldots f(x-1)$ and thus exponential in the size n of x according the notion used in this paper.

In order to separate limitations of storage and computation, Freivalds, Kinber and Smith [11] offered to the learner an additional short term memory, which is cleared before reading each new word of the input. So the learner in stage x reads the argument x, the information $f(x-1)$ on the previous argument in order to verify or disprove the last conjecture, modifies the long term memory and outputs an hypothesis – a program for f in the case of explanatory learning, a guess for $f(x)$ in the case of next value learning. It is also reasonable to consider a bound on computation time instead of computation space: a natural example would for this scenario is a tennis player who during a long professional life collects the data how to play tennis but in each match has to react rapidly in order to return the upcoming ball. Nevertheless even during the game, the player learns something on the game or the actual opponent and so updates his strategy represented by the long term memory.

Long term memory and computations have resource bounds which depend polynomially on some parameters among which the size n of x is the most important one. Mathematically, the size of a number is just its logarithm. Roughly spoken, the dual number 10011 has the size 5 since it can be written using 5 digits (bits). The other parameters are the maximal size o of some previous seen value $f(y)$ $(y = 0, 1, \ldots, x-1)$ and the size m of the description (index) of the function to be learned. In addition to the two original bounds on long term memory and on working space (or time), a new bound is introduced which limits in the case of exact learning the number of false predictions. Now a formal description of the learning process and in particular of its resource bounds is presented.

- Long term memory: The learner predicts every value $f(x)$ using only the informations x, $f(x-1)$ and the content of the long term memory which was produced after predicting $f(x-1)$. The content of this long term memory is the only way for the learner to retrieve information on the previous values $f(0), f(1), \ldots, f(x-2)$. After making the prediction, the learner updates the long term memory and thereby respects the space bound $l(m,n)$ for it.

- Computation time or space: The time or the space to compute a prediction for $f(x)$ is bounded by a function $t(m,n)$ or $s(m,n)$, respectively.

- Number of errors: During the whole inference-process of f, the number of false predictions should not exceed some bound $b(m)$ where m is the size of the concept to be learned, that means the size of the smallest index of f relative to a given enumeration.

There is an influence on the choice of the hypothesis space since the parameter m depends on this choice. Therefore two cases are distinguished:

- exact NV-learning: the parameter m is chosen with respect to the given enumeration; m is the size (= logarithm) of the smallest index of f in the given enumeration.
- independent NV-learning: Here the learner has to keep its bounds only for the long term memory and the complexity of the computation of each next value. Both resource bounds have to be independent of m and depend only on n.

Many applications look only at $\{0,1\}$-valued functions so that the parameter o can just be ignored in these cases. In the other cases each of the two definitions above can be split into two versions: those where the parameters depend only on n and not on o and those where the size of the values of functions is taken into account and each parameter n in the definitions above is replaced by $n+o$. For certain classes like the class of all integer valued polynomials, it makes a difference whether $n+o$ or only n is considered as parameter for the resource bounds.

Lange and Zeugmann [16, 26] studied the influence of the space of hypotheses on learnability. They distinguish three types of learning: exact, class-preserving and class-comprising. In the first case, the learner has to output guesses using a given space of hypotheses, in the second the learner takes the hypotheses from a space which contains the same functions as the given one and in the third case, the learner might use any superclass. So it is natural to ask to which extent such models can also be applied to NV-learning. An NV-learner does not any longer output a hypothesis but the next value. Thus the hypothesis space is less relevant as in the case of explanatory learning. Exact NV-learning still needs the space of hypotheses since the parameter m depends on the size of the minimal index of the function to be learned. But independent learning is – as the name already indicates – absolutely independent of the space of hypotheses. That means an independent NV-learner either ignores hypotheses or uses them only internally which might make an algorithm more transparent. Class-preserving and class-comprising NV-learners have the ability to blow up the indices: if they use $g_{i'}$ instead of f_i where $g_{i'} = f_i$ for $i' = 2^i, 2^i+1, \ldots, 2^{i+1}-1$ then polynomially many errors in the size of the new index are exponentially many in the size of the old index. This is of course not what is intended and therefore in the field of resource bounded inference it is more suitable to consider only exact and independent learning but not the class-preserving and class-comprising models which are similar (but not identical) to independent learning.

In the present work only PSPACE-computable functions are considered and every class $S = \{f_0, f_1, \ldots\}$ has to be polynomial time m-reducible to the standard acceptable numbering with clocks – this numbering is given by the functions

$$\varphi_{e,p}(x) = \begin{cases} \varphi_e(x) & \text{if the computation terminates} \\ & \text{without using more than} \\ & p(m+n) \text{ bits working space;} \\ 0 & \text{otherwise;} \end{cases}$$

where p is a polynomial, m the size of the index e and n the size of the input x. Note that there is still a difference between uniformly PSPACE-computable classes of functions and the here introduced non-uniform enumeration of all PSPACE-computable functions.

2 Independent NV-Learning and PSPACE

The main result of this chapter is, that a class of functions is independently NV-learnable with polynomial space bounds on computation and long term memory iff it is uniformly in PSPACE. So this theorem is the direct analogue to Barzdins' and Freivalds' [6] result that a class of functions is NV-learnable (without resource bounds) iff it is contained in a uniformly recursive class of functions [9, Theorem 2.21]. A similar result for time bounded learners requires at least the unlikely assumption that every function computable with sublinear space is also computable with polynomial time; therefore only one direction from the result on space transfers to that for time.

Theorem 2.1 *A class S of $\{0,1\}$-valued functions is independently* NV*-learnable via a machine having polynomial bounds on working space and long term memory iff it is uniformly in* PSPACE.

Proof Before starting the proof, the statement is made more precise: The class S is uniformly PSPACE-computable iff there is an enumeration $f_0, f_1, \ldots$ containing each function from S and a polynomial p such that each computation $i, x \to f_i(x)$ can be executed in space $p(n+m)$ where n is the size of x and m that of i. It is now shown that both concepts can be translated into each other.

($\Rightarrow$): Let S be independently NV-learnable with a polynomial bound $p(n)$ on long term memory and computation space. The learner produces at each stage from the old content j of the long term memory, the value x and the value $f(x-1)$ a prediction $M(x, f(x-1), j)$ for $f(x)$ and a new content $J(x, f(x-1), j)$ for the long term memory. Now the following algorithm computes for any finite string σ the corresponding function f_σ:

If $x \in dom(\sigma)$ then $f_\sigma(x) = \sigma(x)$.
Otherwise $f_\sigma(x)$ is computed as follows:
j and z are initialized to 0.
For $y = 1, 2, \ldots, x$ the following is done:

If $y - 1 \in dom(\sigma)$ then $z' = \sigma(y-1)$
else $z' = M(y, z, j)$.
Let $j' = J(y, z, j)$.
Update $z = z'$, $j = j'$.

The loop terminates with $y = x$ and the program outputs $f_\sigma(x) = z$.

So f_σ does nothing else than following the function given by the NV-machine except that in the domain of σ the construction explicitly follows the values coded in this string. It follows from the construction that $f = f_\sigma$ provided that each of the finitely many x where M does not predict $f(x)$ is in $dom(\sigma)$.

Now the amount of space needed for the computation is determined. The values j and z are used to store the long term memory and the value $f(y-1)$ at the moment the program computes $f_\sigma(y)$. They might be needed twice during the update-time, so they use together space $4p(n+m)$. Furthermore the emulation of the learner might need a bit more space than the learner itself due to some control structures; thus the space needed to compute M and J can be estimated by the upper bound $2p(m+n)$. The variable y needs at most as much storage as x, that is space n. The algorithm has also to store σ which by definition needs space m. So the whole computation needs at most space $n + m + 6p(m+n)$ and is uniformly in PSPACE.

($\Leftarrow$): Let $S \subseteq \{f_0, f_1, \ldots\}$ and let $f_i(x)$ be computable using space $O(p(m+n))$. Now the following algorithm N with a space-bound in $O(p(n))$ and a long term memory of size n NV-learns each f_i. In the algorithm, j denotes always the current value of the long term memory and is used to store the last guess for an index of the function f.

If $x = 0$ then j is set to 0 and N predicts $f_0(0)$.
Otherwise $x > 0$ and N receives as additional input $f(x-1)$ and has the index j in the long term memory.
If $f_j(x-1) \neq f(x-1)$ then $j = j+1$.
N predicts $f_j(x)$ (for the updated new j).

After each false prediction, j is increased by one. Furthermore whenever j reaches i, all following predictions are correct. So there are at most i false predictions. The number of errors is finite for each $f \in S$. The most space in each step is used to compute $f_j(x)$ but this can be done using space $p(m+n)$. Since $j \leq x$ in each step, the amount of space necessary is bounded by $p(2n)$. The long term memory can even be restricted by n since j is the only content of the long term memory and $j \leq x$. So the polynomial $p(2n) + n$ is an upper bound for the size of both resources. ▮

Considering non-$\{0,1\}$-valued functions the question is to which extent the size o of the largest value $f(y)$ seen so far is taken into account. If one requests that these sizes are always bound by $p(m+n)$ then the result above can be kept, but the second direction might fail if the learner may use space polynomial in $n + o$. An example for this is the function given by $f(0) = 2$ and $f(x+1) = f(x)^2$. This function grows very rapidly, $f(x)$ needs size 2^x. Thus it is not longer computable with space polynomial in n since the output can not be coded in the working space. On the other hand, a NV-learner having space o^2 would just take $f(x-1)$ and compute its square, so $\{f\}$ is NV-learnable with space polynomial in o but f is not PSPACE-computable. In order to avoid such pathological situations, it is assumed from now on that the values of any considered class $\{f_0, f_1, \ldots\}$ have uniformly polynomial size. In particular, classes are taken to be $\{0,1\}$-valued whenever this is possible.

Well-known classes of $\{0,1\}$-valued functions are the classes PSPACE, LOGSPACE and P. For these the following statements can be derived from Theorem 2.1: The whole class PSPACE can not be independently NV-learned since there is no universal function for it in PSPACE itself. On the other hand there is a PSPACE-computable universal function for LOGSPACE and so LOGSPACE can be independently NV-learned using polynomial bounds on working space and long term memory. The question whether the class P can be independently NV-learned can not be answered with today's knowledge on complexity theory since the following two hypotheses are both possible (but unlikely) and give answers in opposite directions: if P = LOGSPACE then P is "very small" and can be NV-learned; if P = PSPACE then P is "very big" and can not be NV-learned.

The next results deal with the case where instead of a resource bound on the working space a polynomial resource bound t on the time to compute each prediction is used. One direction of the previous result directly transfers to the corresponding concept of polynomial-time NV-learning.

Theorem 2.2 *If a class S of functions is uniformly computable in polynomial time then S can independently be* NV*-learned with a polynomial time-bound $p(n)$ on the computation of each prediction and a linear bound on the size of the long term memory.*

The reverse direction is likely to fail. Giving up the bound on the number of errors, it is possible to NV-learn every function which takes non-zero values only on the "tally" inputs $1, 2, 4, 8, 16, \ldots$ and which is computable with sublinear space. Under the assumption that these functions are not all contained in P – which is only slightly stronger than the assumption P $\neq$ PSPACE – the following theorem shows that there is an independently NV-learnable class which contains functions not computable in polynomial time.

Theorem 2.3 *It is possible to* NV*-learn independently* *with memory-bound $l(n) = 3n$ and time-bound $t(n) = n \cdot \log(n)$ all tally functions which are computable with sublinear space.*

Proof There is an enumeration $f_0, f_1, \ldots$ of all tally functions in sublinear space such that $f_i(x)$ can be computed with space $\max\{m, n\}$. The space bound can be obtained by choosing the index large enough for these functions.

The main idea of the proof is to use the numbers $x \in \{2^y + 1, 2^y + 2, \ldots, 2^{y+1} - 1\}$ in order to compute step by step the value $f_i(2^{y+1})$ where i is the current guess what the function f_i might be. The working-tape contains without loss of generality only strings of 0s and 1s. So for input 2^{y+1} and program $i \leq y+1$ the working-tape may contain only binary strings of length up to $y + 1$ and so take at most 2^{y+2} different values. Whenever a computation takes longer than 2^{y+2} stages then it diverges: Some configuration occurs twice and so the computation goes infinitely often through the loop defined by this configuration.

So for $x = 2^y + 1, 2^y + 2, \ldots, 2^{y+1} - 1$, the learner predicts 0 ($f(x) = 0$ since f is tally) and uses the time to simulate the stages $5(x - 2^y - 1) + z$ for $z = 0, 1, 2, 3, 4$ for the computation of $f_i(2^{y+1})$ where the configuration of the working-tape after this 5 stages is stored into the long term memory together with i and perhaps also some bits of output. On input $x = 2^{y+1}$ the learner reads the value from the long term memory, checks whether the computation has already converged and if so, outputs the value found on the working-tape. If this value coincides with that which the learner afterwards receives for $f(2^{y+1})$ then the learner keeps the guess i otherwise the learner continues to work with the guess $i + 1$.

It is easy to see that the learner for $f \in \{f_0, f_1, \ldots\}$ makes only finitely often a false prediction and is correct from that moment where the correct index i is found. Furthermore at any time during the above procedure the currently considered index is not larger than the argument and so all computations can be carried out without violating the space bound which depends only on the size of the input. ∎

Next value learning without resource bounds is closed under union, e.g., if S_1 and S_2 are NV-learnable classes so is $S_1 \cup S_2$. The same result holds also for independent NV-learning with resource bounds on long term memory and computation space (or time): the new learner just emulates both algorithms but uses only the prediction of one of them. Whenever this actual algorithm makes an error the learner starts to follow the predictions of the other algorithm. So one obtains:

Theorem 2.4 *If two classes are independently* NV-*learnable with polynomial resource bounds on long term memory and computation space so is also their union. The same holds for bounds on computation time instead of computation space.*

3 Exact NV-Learning

Exact learning requires in addition to the bounds on working-space and long term memory a bound on the number of false predictions. This last bound is not constant but is polynomial in the size of the description – which means logarithmical in the index – of the function to be learned. The first result is that every exactly NV-learnable class is also independently NV-learnable.

Theorem 3.1 *If a class $S = \{f_0, f_1, \ldots\}$ is exactly* NV-*learnable with polynomial bounds on long term memory and working space then S is also independently* NV-*learnable with polynomial bounds on the same resources.*

Proof This result can be obtained via just proving that the family is uniformly in PSPACE. Two modifications are necessary in order to adapt the proof of Theorem 2.1 ($\Rightarrow$): first it has to be added that the σ has to be chosen so large that σ has at least the same size as the index i of the function f_i in the given class $S = \{f_0, f_1, \ldots\}$ – it is easy to see that this is possible. Second the computation is stopped and 0 is output whenever the space bound $p(n + \mathrm{size}(\sigma))$ is violated. This will only happen if the σ is too small or does not belong to any function in S. ∎

The next Theorem shows that every family of uniformly PSPACE-computable $\{0, 1\}$-valued functions is exactly NV-learnable. The proof of this results is obtained by combining the methods from Theorem 2.1 with the halfing-algorithm as presented by Littlestone [17]. A similar approach was used by Barzdin and Freivalds [7]. They did not care on resource bounds and found in a similar setting a very restrictive bound for the number of mind changes which is a bit below $m + \log_2(m) + \log_2(\log_2(m)) + 2 \cdot \log_2(\log_2(\log_2(m)))$. The price they pay is high complexity: "It turns out that if a prediction strategy is error-optimal, then the time complexity of computation" of the prediction of the next value "may go up, in some sense," double-exponentially [10]. Since there are families which are not uniformly computable in PSPACE according to the given enumeration but only according to some different one, the result goes only in one direction.

Theorem 3.2 *Let $S = \{f_0, f_1, \ldots\}$ be $\{0, 1\}$-valued and uniformly* PSPACE-*computable. Then S is exactly* NV-*learnable.*

Proof The proof is very similar to that of Theorem 2.1 which has only the disadvantage that it makes too many errors. So the errors have to be brought down from a quantity exponential in m to a quantity polynomial in

m. The idea to do this is the halfing-algorithm [17] which always considers all still possible solutions and then predicts the next value according the majority. It follows that every error halfes the number of still valid hypotheses and so reduces the number of errors to the logarithm of the initial quantity of hypotheses. In the adapted setting the halfing algorithm is always used on finite groups of hypotheses which is already sufficient to scale down the number of errors. These groups are formed by all hypotheses which have the same size.

The adapted algorithm needs $(m+n)^3$ long term memory, polynomial working space and makes at most $(m+1)^2$ errors. The algorithm deals for $k = 0, 1, \ldots, m$ with the groups of all indices of size k and applies on each group the halfing algorithm. When all indices inside a group fail, the algorithm takes the next group until the correct m-th group is reached. In the long term memory each incorrect prediction with the correct value is stored – thus the algorithm can simulate all previous steps and recover the correct values by either using the prediction in the case it was correct or looking up in the long term memory in the case it was incorrect. So the long term memory can work with the size $(m+2)^2(n+1)$ where $(m+1)^2$ is the upper bound of the number of errors and $n+1$ is the space to store at each error the correct previous value (one bit) and the argument where the error occurred (up to n bits). Furthermore, k is stored in the long term memory. So it is now sufficient to describe the algorithm without worrying on the long term memory but only on the number of errors.

Input x and $f(x-1)$.
For $a = 0, 1$ compute the number c_a of all indices of length k which coincide with $f(y)$ for all $y < x$ and which output a at argument x.
If $c_0 > c_1$ then predict 0 else predict 1.
If $c_0 + c_1 = 0$ then replace k by $k+1$.

The verification is based on the idea, that for every k at most $k+1$ errors are made and so at most $(m+1)^2$ errors are made until the m-th group with the correct f_i is found. From then on only m errors are made inside the group and so the error bound is kept. It is easy to see that this computations can be done with space $(m+n)^4 \cdot p(m+n)$ provided that every value $f_i(x)$ can be computed with space $p(m+n)$. ∎

This result requires that the functions f_i are uniformly PSPACE-computable. So the result fails if only nonuniformly PSPACE-computability is required: Let M_0, M_1, ... be an enumeration of all PSPACE-computable NV-learners and let

$$f_i(x) = \begin{cases} 1 & \text{if } x \leq i \text{ and } i = (2k+1)2^h \text{ for} \\ & \text{some } h, k \text{ and } M_h \text{ predicts } 0; \\ 0 & \text{otherwise.} \end{cases}$$

Each machine M_h makes on each f_i at least $i/2^{h+1} = 2^{m-h-1}$ mistakes and so the polynomial error bound can not be kept by any machine M_h. But note, that the class is still NV-learnable simply by always predicting 0 and so one sees that just dropping the requirement that the f_i are uniformly in PSPACE allows to find a class which is independently but not exactly NV-learnable.

The next investigations return to classes which are uniformly in PSPACE. They are dealing with natural examples and try to get the resource bounds as restrictive as possible. In particular they try to replace the bound on computation space by one on computation time whenever this is possible.

Somehow the first example, the non-erasing pattern languages [1], still needs polynomial computation space (or, alternatively, nondeterministic computations). The difficulty is not finding the correct pattern but for evaluating it in each stage. But they are still well-behaved in some other sense: the long term memory and the computation space depend only on the size n of the current x and not on the size m of the pattern to be learned.

In order to make technical definitions easier, numbers and binary words are identified as follows: A string $a_0a_1 \ldots a_n$ corresponds to the number i with the binary representation $1a_0a_1 \ldots a_n$. So when learning functions with the domain $\{0,1\}^*$, the data is represented in the order of the corresponding numbers: $f(\lambda)$, $f(0)$, $f(1)$, $f(00)$, $f(01)$, $f(10)$, $f(11)$, $f(000)$, $f(001)$ and so on. In particular a string σ is before a string τ iff 1σ defines a binary number less than 1τ.

Example 3.3 *The class of all languages generated by non-erasing patterns can be exactly* NV*-learned with* $m+1$ *errors,* $2n \cdot \log(n)$ *bits long term memory and computation space* $3n$*.*

Proof The learning algorithm is now explained for the example $10x_2x_2x_0x_1x_0$. The learner predicts always 0 until this prediction is false the first time. This happens exactly at the pattern 1000000 where all variables are mapped to 0. The learner now stores the pattern as 1000000 into the long-term memory and predicts from now on always the current value of the stored pattern.

Each time the learner errs, a new variable has to be added. By the ordering of the strings, the first time a until now undiscovered variable causes a problem is when this variable is set to 1 while all other variables are set to 0. So it is easy to identify the variable: in the given case, x_0 is identified by comparing the pattern 1000101 with the stored pattern 1000000, so the stored pattern is corrected to $1000x_00x_0$. x_1 is discovered when the learner erroneously predicts 0 for 1000010. The pattern is then corrected to $1000x_0x_1x_0$ and after failing at the input 1011000 the learner has to do the third update to $10x_2x_2x_0x_1x_0$. So the learner identifies the pattern with 1 error at the beginning followed by 3 errors where each variable causes exactly one error.

To see that the other two resource-bounds are satisfied one should note that the pattern can be stored with $2n \cdot \log(n)$ bits if n exceeds the length of the pattern. But since the first error occurs when this length is reached, this is no problem. The term $\log(n)$ is due to the fact that the string to represent the pattern does not only consist of the constants 0 and 1 but also of codes for the variables. The easiest way to do it is to use the alphabet $0, 1, 2$ where 2 is used as a separator and the binary sequences $00, 01, 10, 11, 100, 101, \ldots$ are used to represent the variables. So the pattern above could be stored as 21202102102002012002. The space bound $3n$ is enough to evaluate the pattern. ∎

While pattern languages can be identified with working space and long term memory only depending on the size of the current value x, there are other exactly learnable classes where the resource bounds also must take in account the size of the index of the function to be learned. An example for such a class are the periodic functions. The periodic function f_i is defined as follows: let $1a_1a_2\ldots a_m$ be the binary representation of i. Then $f_i(x) = a_k$ for that k with $k \equiv x$ modulo m.

Example 3.4 *The class of all periodic $\{0,1\}$-valued functions is*
(a) *exactly* NV*-learnable with parameters $b = 2m+1$, $l = m + \log(m)$, $t = (m+n)^2$;*
(b) *not exactly* NV*-learnable with parameters $b = p(m)$ and $l = p(n)$ for any polynomial bound p.*

Proof (a): The learner keeps always in its long term memory the following data: an estimation h for m and the period $a_1a_2\ldots a_h$ representing the values $f(1), f(2), \ldots, f(h)$.

> Input: x, $f(x-1)$;
> Long term memory: h and $a_1a_2\ldots a_h$.
> Compute $k \in \{1, 2, \ldots, h\}$
>   such that $k \equiv x-1$ modulo h.
> Check whether $f(x-1) \neq a_k$.
> If so, then the storage has to be adapted:
>
>   Search for the first $h' \leq x$ such that
> $f(y) = f(y+h')$ for all $y \leq x - h'$.
>   Let $b_k = f(k)$ for $k = 1, 2, \ldots, h'$.
>   Replace $a_1a_2\ldots a_h$ by $b_1b_2\ldots b_{h'}$ and
> h by h'.
>
> Compute $k \in \{1, 2, \ldots, h\}$
>   such that $k \equiv x$ modulo h.
> Output a_k as guess for $f(x)$.

Note that the computation of $f(y)$ for $y < x-1$ can be done via the old long term memory and that the new one (whether changed or not) then codes all values $f(y)$ with $y < x$. The guess a_k at the end of the algorithm is of course always based on the updated storage. Since $f(y) = f(y+m)$ for all y, the update of the long term memory never produces a $h > m$ and so the requirement on the size of the long term memory is kept: the binary string $a_1a_2\ldots a_h$ needs at most space m; its length h needs at most space $\log(m)$. Furthermore if $x > 2m$ and $f(y) = f(y+h)$ for some $h < m$ and all $x < 2m - h$ then $f(y) = f(y+h)$ for all y, thus any error occurring beyond $2m$ is the last and the algorithm makes at most $2m+1$ false predictions. The bound on the computation-time is due to the fact that the loop in the algorithm is only used for $x \leq 2m+1$ and the other part is at most quadratic in n, the size of x.

(b): Assume that M is a learner which identifies every periodic function with long term memory $p(n)$ and $p(m)$ errors. Then for each string $a_1a_2\ldots a_m$ there is a number $x \in \{0, 1, 2, \ldots, p(m)\}$ such that M correctly predicts the values $f(mx+1), f(mx+2), \ldots, f(mx+m)$. Since $f(mx+k) = a_k$ it is possible to compute the sequence $a_1a_2\ldots a_m$ from this value x and from the value of the long term memory which existed before predicting $f(mx+1)$. Now x can be coded with $\log(p(m))$ bits and the long term memory before predicting $f(mx+1)$ has the size $p(\log(xm+1)) \leq p(\log(p(m)m+1))$. Thus there is a discription for $a_1a_2\ldots a_m$ which needs at most $p(\log(p(m)m+1))$ bits. On the other hand for each length m there is a string which can not be represented with less than m bits by the standard Kolmogorov-complexity argument. So for each m, the relation

$$\log(p(m)) + p(\log(p(m)m+1)) \geq m$$

holds and it follows that p can not be a polynomial since then the expression on the left-hand side of the relation would be bounded by a polynomial in the logarithm of m – but such a function does not dominate the identity. ∎

Note that the part (b) does not require any bounds on computation time or space but only on the number of errors and the size of the long term memory. The result can be generalized to learning regular languages. Regular languages are represented by deterministic finite state automata and the size m of such a representation can be estimated by $2k \cdot \log(k)$ where k is the number of states of the automaton. Adapting a result of Ibarra and Jiang [13] it is shown that the class of regular languages can be NV-learned with polynomial resource bounds depending on $n+m$ but not with these bounds depending on n alone.

Example 3.5 *The class of all regular languages is*
(a) *exactly* NV*-learnable with parameters $b = p(m)$, $l = p(m+n)$, $t = p(m+n)$ for some polynomial bound p;*

(b) *not exactly* NV*-learnable with parameters* $b = q(m)$ *and* $l = q(n)$ *for any polynomial bound* q.

Proof (a): Without loss of generality, the languages are subsets of $\{0,1\}^*$ where each string $a_0 a_1 \ldots a_n$ corresponds to the binary number $x = 1a_0 a_1 \ldots a_n$. The learning algorithm is then the result of Ibarra and Jiang [13, Theorem 5] that regular languages can be learned in time polynomial in m using equivalence queries for which in the case of disagreement the least counterexample is returned; least means here according to the order given by the above coding of words into the numbers $\{1, 2, \ldots\}$. The translation of their algorithm goes as follows:

- It can be checked whether two finite automata behave different below a value x using polynomial time in the size n of x and the size of the two finite automata. Therefore it can be assumed without loss of generality that the algorithm makes only such queries which coincide with the values seen so far. Thus the counterexamples are strictly increasing during the learning process.
- The long term memory always carries the last conjecture and the internal data of the algorithm when this conjecture is made. On input x, $f(x-1)$ it is checked whether the conjectured automaton evaluates the word coded by $x-1$ to $f(x-1)$. If not, the word coded by $f(x-1)$ is supplied as the least counterexample and the algorithm is simulated until a next conjecture comes up which is now consistent with $f(0), f(1), \ldots, f(x-1)$. Then all internal data and the new conjecture are stored again in the long term memory. In both cases (whether $f(x-1)$ was false or correct) the algorithm evaluates the current automaton for x and outputs the result as a prediction for $f(x)$.

The number of false predictions does not exceed the number of equivalence queries of the algorithm of Ibarra and Jiang. Thus it is bounded by a polynomial. Similar the running time in each step and the long term memory are bounded by a polynomial in m and n; the parameter n can not be dropped since at any prediction, the automaton has to evaluate two words of length n.

(b): This part follows from the fact that every periodic $\{0,1\}$-valued periodic function can be identified with a regular language. The corresponding deterministic finite automaton has the states $1, 2, \ldots, m$ such that 1 is the initial state, state i is accepting iff $f(i) = 1$ and whenever the automaton is in state i and reads the symbol a from the input then it goes to that state which is equivalent to $2i + a$ modulo m. So the periodic functions form a subclass to the regular languages and the non-learnability of them transfers to the non-learnability of the automata since the size of representation grows only from m bits at the functions to $m \cdot \log(m)$ bits at the automata which is still polynomially bounded in the old size. ∎

For $\{0,1\}$-valued functions the size of the input depends only on the size n of x since the value $f(x-1)$ can be represented by only one bit. This is different for arbitrary functions. Using the example of the integer-valued polynomials it is shown that besides the size of x it is also necessary to consider also the parameter o which is the maximum size of some $f(y)$ with $y < x$ or the parameter m which is the size of the concept to be learned. It is natural to code the polynomials such that $m \geq k$ for the degree k of the polynomials to be learned. Let p denote any given polynomial resource bound.

Example 3.6 *The* NV*-learnability of the class of integer-valued polynomials depends on the choice of the parameters:*
(a) NV*-learnable if* $b = m+1$, $l = m$, $t = (m+n)^2$;
(b) NV*-learnable if* $b = 5m^2$, $l = t = 2(m+o)^2$;
(c) *not* NV*-learnable if* $b = p(m)$ *and* $l = p(n)$.

Proof (a): The learning algorithm stores in the long-term memory only those values of f which are not predicted correctly by the interpolation algorithm working with all previous data – thus with this knowledge it is always possible to recover all old values. Since for each $x > k$ the value $f(x)$ can be computed by only knowing $f(0), f(1), \ldots, f(k)$, the long term memory has always a prefix of the string $f(0)f(1)\ldots f(k)$ in it and so does by definition not use more space than the bound m. Interpolating a polynomial never produces more errors than its degree plus 1 as long as all previous values can be reconstructed from the long term memory. Thus the error bound is kept and also the bound on the computation time can be satisfied since evaluating a polynomial at x using some interpolation procedure with pairs $(0, f(0))$, $(1, f(1))$, $\ldots$, $(k, f(k))$ is a algorithm quadratic in the size of x and the data.

(b): The basic idea of the algorithm is to keep the last values $f(a), f(a+1), \ldots, f(x-2)$ in the long term memory and to interpolate $f(x-1)$ from them. The update rule for the long term memory is to compute the least $b \geq a$ such that $f(b), f(b+1), \ldots, f(x-1)$ do not use more space than $2(o+n)^2 - \log(o)$ where the $\log(o)$ bits are necessary to store o whose new value is always the maximum of the size of $f(x-1)$ and the old value.

So the main task is that within $4m^2 + m$ the size of the parameter o has become large enough to store the information necessary to interpolate the next value. Since each of the values $f(b), f(b+1), \ldots, f(x-1)$ needs at most space o it is sufficient to show that there is some number $y \leq 4m^2$ for which $f(y) \geq 2^k$ so that from then on $o \geq k$ and for $x \geq k + z$ the last k values are in the long term memory and for no $x \geq z + k$ a false prediction is made. Note that $k \leq m$ and thus it is sufficient to show that at most $4k^2 + k$ errors are made. There are two equivalent ways to represent a polynomial of degree k:

$$\begin{aligned} f(x) &= a \cdot (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_k); \\ f(x) &= a_0 + a_1 \cdot \tfrac{x-1}{1} + a_2 \cdot \tfrac{x-1}{1} \cdot \tfrac{x-2}{2} + \ldots \\ &\quad + a_k \cdot \tfrac{x-1}{1} \cdot \tfrac{x-2}{2} \cdot \ldots \cdot \tfrac{x-k}{k}. \end{aligned}$$

In the first representation, the x_i are the complex numbers where f takes the value 0. In the second all the a_i are integers and each a_i is the difference between $f(i)$ and the value obtained via interpolating the data $(x, f(x))$ for $x = 1, 2, \ldots, i-1$. So it follows that $|a| = |a_k| \frac{1}{1 \cdot 2 \cdot \ldots \cdot k} \geq k^{-k}$ since a_k is an integer. For each x_i there are at most $4k$ natural numbers x with $|x - x_i| < 2k$. Thus some $x \in \{0, 1, \ldots, 4k^2\}$ satisfies $|x - x_i| \geq 2k$ for all i. At this number x, $|f(x)| \geq |a| \cdot (2k)^k \geq 2^k$ and so $f(x)$ needs at least k bits for its binary representation. It follows that $o \geq k$ at $x = 4k^2$ and $f(x)$ is predicted correctly for all $x \geq 5k^2$. So the number of errors is polynomially bounded in k and thus also in m.

(c): The Kolmogorov-argument for the unlearnability must be adapted since in contrast to the case of periodic functions, the input $f(x-1)$ might contribute much more information than a single bit. Now assume that $k+1$ numbers $a_0, a_1, \ldots, a_k \in \{2^k, 2^k+1, \ldots, 2^{k+1}-1\}$ are given. In each of these numbers can be coded k bits and so the parameters can be chosen such that they code a string with Kolmogorov complexity $k \cdot (k+1)$.

Assume now that f can be learned with at most $p(m) = q(k)$ errors where q is a polynomial which is obtained using the fact that the size of the representations of the functions to be learned depends polynomially in k. Thus there is a number $x \leq q(k)(k+1)$ for which the algorithm predicts all values $f(y)$ correctly from $f(y-1)$ and y for $y = x, x+1, \ldots, x+k$. These predictions allow to interpolate the polynomial and to compute $a_0, a_1, \ldots, a_k$ from the following data: $f(x-1)$, x and the content of the long term memory after processing $x - 1$. $f(x-1)$ is bounded by $2^{k+1}(k+1)(q(k)(k+1))^k$ and can thus be represented by $(k+1) \cdot \log(2(k+1)q(k)) \leq c \cdot k \cdot \log(k)$ bits where c is a sufficiently large constant. x is bounded by $q(k)(k+1)$, thus n is bounded by $\log(q(k)) + \log(k+1)$ and so $p(n)$ is bounded by $p(\log(q(k)) + \log(k+1))$. So the whole number of bits available to compute $a_0, a_1, \ldots, a_k$ is a product of k and a function $r(k)$ which is polynomial in $\log(k)$. For sufficiently large k the now obtained relation $k \cdot (k+1) \leq k \cdot r(k)$, that is $k+1 \leq r(k)$ fails since r is a polynomial in $\log(k)$. This contradiction gives that the class of these f can not be learned under the considered criterion. ∎

4 Resource-Bounded Explanatory Learning

One nice result of inductive inference without resource-bounds is the Theorem of Barzdins and van Leeuwen [9, Theorem 2.19] which states that predicting the next value is the same concept as explanatory learning via a machine, which never outputs "buggy" programs. This means in particular that every NV-learnable class is Ex-learnable but some Ex-learnable classes are not NV-learnable. Somehow the inclusion needs the absence of a resource bound on the long term memory: Let S be the class of all functions which take only finitely often a value different from 0. They can be independently NV-learned by always guessing 0 but they can not be independently Ex-learned since for sufficiently difficult functions the bound of the long term memory makes it impossible to remember all the arguments at which the function differs from 0. Therefore it is necessary to work with Ex* instead of Ex, that means the final hypothesis may differ from the function to be learned at finitely many arguments.

Definition 4.1 M resource-bounded Ex*-learns a class S of functions iff for every M outputs an index from a given space of hypotheses for every input $a_0 a_1 \ldots a_x$ and these hypotheses converge on ever function f to a fixed index which computes f with at most finitely many errors. The learner has to satisfy for any $f \in S$ the following resource bounds.

- Long term memory: this bound l is defined as in the case of NV.
- Computational complexity: the computation of every hypothesis has to satisfy a polynomial resource bound on space (s) or time (t).
- Number of mind changes: the number of the arguments x where M outputs a different hypothesis as previously at $x - 1$.
- Number of errors: the bound b on the number of arguments $x-1$ where M outputs a guess e such that the e-th hypothesis with input x does not compute $f(x)$.

The first two bounds l and s (l and t, respectively) have always to be a polynomial in the given parameters n, m and o. The last two bounds have only to be finite for each $f \in S$ for the case of independent Ex*-learning and be a polynomial in m for the case of exact Ex*-learning. While an exact Ex*-learner has to use a given space of hypothesis, an independent Ex*-learner may choose the space of hypothesis by itself. So independent learning corresponds to the class comprising learning as used by Lange and Zeugmann [16].

concept	time	memory	mind changes	errors
pattern languages	$c \cdot n \cdot \log(n)$	$3n \cdot \log(n)$	$m+1$	$m+1$
regular languages	$\text{poly}(n+m)$	$\text{poly}(n+m)$	$\text{poly}(m)$	$\text{poly}(m)$
polynomials 1	$c \cdot (n+m)^2$	m	$m+1$	$m+1$
polynomials 2	$c \cdot (n+o)^2$	$5(n+o)^2$	$5m^2$	$5m^2$

Table 1: Resource-Bounds for Prominent Classes

The last two bounds have to be polynomial in the size m of the smallest index in the case of exact Ex*-learning and have to be only finite in the case of independent Ex*-learning. The first two bounds are polynomial in n, o and m in the exact case and polynomial only in n and o in the second case.

Theorem 4.2 *In the case of a resource bound on computation space, every independent* NV*-learnable class is also independent* Ex**-learnable.*

Proof Theorem 2.1 also holds when the functions take values which are uniformly polynomially bounded in size (instead of only the values 0 and 1); following the convention that only such classes are considered to be independently NV-learnable, one can adapt the proof as follows: every independently NV-learnable class is contained in a uniformly PSPACE-computable class. Now using the other direction of the proof, the learner witnessing that every uniformly PSPACE-computable class of functions is independent NV-learnable is modified into one which witnesses that the class is also independent Ex*-learnable: instead of outputting $f_i(x)$ it outputs the index i with respect to the hypothesis space $\{f_0, f_1, \ldots\}$. ∎

Example 4.3 *Table 1 gives an overview on the examples on the last section and states under which resource bounds it is possible to exactly* Ex**-learn the classes from Section 2. In the table, c is a sufficiently large constant which also depends on the chosen machine model.*

So pattern languages can be learned via a machine which uses in each step time and long term memory linear in $n \cdot \log(n)$, changes its mind at most $m+1$ times and makes at most $m+1$ many errors. For polynomials, there are two possibilities: either the bounds on computation time and long term memory are polynomial in $n+m$ or in $n+o$.

All these concepts can also be independently Ex**-learned with bounds polynomial in n on computation time and long term memory.*

The next theorem establishs several basic facts on Ex*-learning provided one follows the definition that a class $S \subseteq$ PSPACE to be learned needs not to be uniformly PSPACE-computable and that it is sufficient if the learner provides any PSPACE-algorithm for functions in this class. For this proof, let $M_0, M_1, \ldots$ be an enumeration of all PSPACE-computable Ex*-learners.

Theorem 4.4 *There are classes $S_0 = \{f_0, f_2, f_4, \ldots\}$ and $S_1 = \{f_1, f_3, f_5, \ldots\}$ such that:*
(a) *S_0 and S_1 are nonuniformly contained in* PSPACE.
(b) *S_0 and S_1 are independently* Ex**-learnable.*
(c) *$S_0 \cup S_1$ is not independently* Ex**-learnable.*

Proof First the construction of f_{2k} and f_{2k+1} is given inductively. $f_{2k,0}$ and $f_{2k+1,0}$ are initialized as 0^k1, that means that the first k arguments are mapped to 0 and the next one to 1. Now these functions are extended as follows:

The invariant of the definition is that in each stage s the strings $f_{2k,s}$ and $f_{2k+1,s}$ have the same length and M_k has the same long term memory j after processing $f_{2k,s}$ and $f_{2k+1,s}$. Now one looks for the first different strings σ, τ of the same length such that M_k takes again the same long term memory after processing $f_{2k,s}\sigma$ and $f_{2k+1,s}\tau$. Now one extends these functions by $f_{2k,s+1} = f_{2k,s}\sigma 0$ and $f_{2k+1,s+1} = f_{2k+1,s}\tau 0$.

Now one verifies the constructions. First the 0 of the extensions $\sigma 0$ and $\tau 0$ has in both cases the same effect on the long term memory and so the induction invariant is kept. Furthermore the guess after reading $f_{2k,s}\sigma 0$ and $f_{2k+1,s}\tau 0$ is the same since in both cases the machine M_k uses the long term memory j and receives as input in both cases the same argument x plus value 0. So provided that M_k converges on both functions, M_k converges on them to the same index. Since they differ in each step of the construction at least once (σ and τ are not equal), f_{2k} and f_{2k+1} differ at the end at infinitely many arguments and so M_k fails to infer at least one of these two functions. So (c) holds.

Such strings σ, τ are found since M_k has a polynomial resource bound $p(n)$ on its long term memory and thus has to take the same value for two strings $\sigma, \tau \in \{0,1\}^{p(n)+1}$. It follows that f_{2k} and f_{2k+1} can be computed with computation space $(p(n)+n)^2$, that means in PSPACE. Thus (a) holds.

Furthermore PSPACE-programs for f_{2k} and f_{2k+1} can be computed from k. So a learner which has just seen the input 0^k1 can generate two programs where the program e_{2k} computes f_{2k} and the program e_{2k+1} computes f_{2k+1}. From then on the Ex^*-learner for S_0 keeps the guess e_{2k} and that for S_1 keeps the guess S_1.

It follows that (b) is satisfied. ▮

While the result that explanatory learning is not closed under union is parallel to the result for learning without resource bounds the way how it is obtained is different: in the resource bounded case the problem is not that one learner may not converge but the problem is that it is impossible to make an amalgamation or more directly said, if an amalgamated program turns out to derive from inconsistent data it is impossible for the learner to check which of the two programs is incorrect – the learner just has forgotten the information necessary to detect this. As a corollary one obtains for both variants, independent and exact learning:

Corollary 4.5 Ex^* *is strictly more powerful than* NV. PSPACE *can not be* Ex^**-learned.*

The fact that Ex^* is not closed under union shows that it makes sense to consider teams. The next result states somehow that such teams can always be taken in the form "one out of a". So the team-hierarchy is exactly the same as the one obtained by Pitt and Smith [22] for Ex-learning without resource bounds.

Theorem 4.6 *A team of size (c, d) is as powerful as a team of size $(1, a)$ for independent Ex^*-learning iff $\frac{1}{a+1} < \frac{c}{d} \leq \frac{1}{a}$. In particular a team of size $(1, a + 1)$ is more powerful than a team of size $(1, a)$ for independent Ex^* learners.*

Proof A modification of the proof of Theorem 4.4 shows that each $(1, a + 1)$-team is more powerful than a $(1, a)$-team. Furthermore if S can be learned via an $(1, a)$-team and $\frac{c}{d} \leq \frac{1}{a}$ then one can obtain the (c, d)-team simply by using each machine from the $(1, a)$-team m times and adding for the rest arbitrary learners. It is clear that on any $f \in S$ at least m machines succeed. So it remains only to show that every (c, d)-team can be replaced by an $(1, a)$-team where a is the least integer such that $\frac{1}{a} \geq \frac{c}{d}$.

Let a resource bounded (c, d)-team $M_1, M_2, \ldots, M_d$ learn a given class S. For each set $D \subseteq \{1, 2, \ldots, d\}$ of cardinality c, a combined learner H_D is defined. H_D simulates all learners M_e with $e \in D$ and outputs always the smallest hypothesis of this team. Furthermore H_D runs a statistic which is intended to approximate the last error. This is stored in an extra variable $q_{D,x}$ in the long term memory and whenever the machine H_D finds an error at input x, it updates $q_{D,x+1} = x$ and keeps $q_{D,x+1} = q_{D,x}$ otherwise. Here H_D looks for two types of errors: first some M_e with $e \in D$ makes a mind change. Second H_D simulates for all input y which is above $q_{D,x}$ and below a function $u(x)$ depending on the resource bounds all programs output by the learners at x with input y: if they disagree on some of these input y then this indicates that the simulated learners still do not have converged to a reasonable hypothesis. Here the function u is just the largest z which allows to evaluate all the hypotheses for $y = 0, 1, \ldots, z$ within the given resource bounds; u is not bounded by a constant and monotone, but u might be extremely sublinear.

The team $N_1, N_2, \ldots, N_a$ is chosen as follows: each learner follows one guess of some machine H_D where the machine tries on one hand to get the guess of a learner H_D with very high confidence and on the other hand not to interfere with the guesses of the other machines such that as many guesses are covered as possible. So at each x each learner N_h chooses a D_h such that the following holds:

- $D_h \subseteq \{1, 2, \ldots, d\}$ and $|D_h| = c$.
- $D_h \cap D_l = \emptyset$ for all $l < h$.
- $q_{D_h,x}$ is minimal among all $q_{D,x}$ where $D \cap D_l = \emptyset$ for all $l < h$, $D \subseteq \{1, 2, \ldots, d\}$ and $|D| = c$.

Furthermore the set D_h is never replaced by some other if this is not necessary. This means: if there are two sets D_h and $D_{h'}$ such that both are disjoint to the D_l with $l < h$ and $q_{D_h,x} = q_{D_{h'},x}$ then the learner N_h does not oscillate between them but remains at the same set which N_h has already used before.

A machine N_h is said to converge if it almost always takes the same output as a fixed machine H_D whose value $q_{D,x}$ does never go above a certain bound r_D. The way the machine N_h is constructed implies that all machines N_l with $l < h$ also converge if N_h does. So assume that the machines $N_1, N_2, \ldots, N_h$ converge and $N_{h+1}, N_{h+2}, \ldots, N_a$ diverge. Let $D_1, D_2, \ldots, D_h$ denote the sets which are used by $N_1, N_2, \ldots, N_h$ almost everywhere. Now there are two cases:

(I): Some set D_l with $l < h$ contains a correct machine, that is there is an $e \in D_l$ which is an index of some M_e which succeeds on f. Since the value $q_{D_l,x}$ always stays below r_{D_l} it follows that after processing $x = r_{D_l}$ the learner N_e does not make any further mind change and that the index output by H_{D_l} does not differ from the program computed by the last hypothesis of N_e above r_{D_l}. Since M_e converges to an index which differs only finitely often from the function to be learned, so does N_l.

(II): No set D_l with $l < h$ contains a correct machine, that is there is no $e \in D_l$ which is an index of some M_e which infers f. Since there are c such indices and since $(a + 1)c > d$ it follows that $h < a$, thus N_{h+1} diverges. But there is a set D of cardinality c which contains only indices of machines which infer f. Every machine M_e with $e \in D$ changes its mind only finitely often on f and its last guess is always a finite variant of f. In particular the last guesses of the machines M_e with $e \in D$ are almost everywhere equal. It follows that the value $q_{D,x}$ is increased only finitely often and so remains below some bound r_D. Since D is disjoint to the

D_l at almost all x it follows that N_{h+1} has to take some D' with $q_{D',x} \leq r_D$ at almost all x. This would imply that N_{h+1} converges in contrary to the assumption and thus case (II) does not hold.

It follows that always case (I) holds and thus the $(1, a)$-team $N_1, N_2, \ldots, N_a$ learns S under the criterion of resource bounded Ex*. Here the proof is independent on the actual choice of the parameters and it is sufficient to permit the $(1, a)$-team to use resource bounds which are increased by some suitable constant factor. ∎

Acknowledgment We would like to thank Efim Kinber and Matthias Ott for proofreading and commentary. Also we are grateful to the anonymous referees for several suggestions and references.

References

[1] Dana Angluin (1980): Finding patterns common to a set of strings. Journal of Computer and System Sciences 21:46–62.

[2] Dana Angluin (1987): Learning Regular Sets From Queries and Counterexamples. Information and Computation 75:87–106.

[3] Dana Angluin and Carl Smith (1983): Inductive inference: theory and methods. Computing Surveys 15:237–269.

[4] Janis Barzdins (1971): Prognostication of automata and functions. Information Processing '71 (1) 81–84. Edited by C. P. Freiman, North-Holland, Amsterdam.

[5] Janis Barzdins (1974): Two theorems on the limiting synthesis of functions. In: Theory of Algorithms and Programs, vol. 1, 82–88. Edited by Janis Barzdins, Latvian State University, Riga.

[6] Janis Barzdins and Rusins Freivalds (1972): On the prediction of general recursive functions. Soviet Math. Doklady 13:1224–1228.

[7] Janis Barzdins and Rusins Freivalds (1974): Prediction and limiting synthesis of effectively enumerable classes of functions. Theory of Algorithms and Programs, vol. 1, Latvia State University 101–111 (in Russian).

[8] Leonard Blum and Manuel Blum (1975): Towards a Mathematical Theory of Inductive Inference. Information and Control 28:125–155.

[9] John Case and Carl Smith (1983): Comparison of Identification Criteria for Machine Inductive Inference. Theoretical Computer Science 25:193–220.

[10] Rusins Freivalds, Janis Barzdins and Karlis Podnieks (1991): Inductive Inference of Recursive Functions: Complexity Bounds. in: Baltic Computer Science: Selected Papers, edited by Janis Barzdins, Springer, Heidelberg, Lecture Notes to Computer Science 502:111–155.

[11] Rusins Freivalds, Efim Kinber and Carl H. Smith (1993): On the impact of forgetting on learning machines. Proceedings of the 6th Annual ACM Conference on Computational Learning Theory (COLT) 165–174.

[12] Mark Gold (1967): Language Identification in the Limit. Information and Control 10:447–474.

[13] Oscar H. Ibarra and Tao Jiang (1988): Learning Regular Languages from Counterexamples. Proceedings of the First Annual Conference on Computational Learning Theory (COLT) 371–385.

[14] Efim Kinber (1994): Monotonicity versus Efficiency for Learning Languages from Texts, Proceedings of the Fifth Workshop on Algorithmic Learning Theory (ALT) 395–406.

[15] Efim Kinber and Frank Stephan (1995): Language Learning from Texts: Mind Changes, Limited Memory and Monotonicity. Information and Computation, 123:224–241.

[16] Steffen Lange and Thomas Zeugmann (1993): Language learning in dependence on the space of hypotheses. Proceedings of the Sixth Conference on Computational Learning Theory (COLT) 127–136.

[17] Nick Littlestone (1988): Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Machine Learning, 2: 285–318.

[18] Nick Littlestone and Manfred Warmuth (1994): The weighted majority algorithm. Information and Computation 108:212–261.

[19] Hartley Rogers, Jr. (1967): Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Company, New York.

[20] Piergiorgio Odifreddi (1989): Classical Recursion Theory. North-Holland.

[21] Daniel Osherson, Michael Stob and Scott Weinstein (1986): Systems that Learn. Bradford – The MIT Press, Cambridge, Massachusetts.

[22] L. Pitt and Carl Smith (1988): Probability and plurality for aggregations of learning machines. Information and Computation 77:77–92.

[23] Robert Soare (1987): Recursively Enumerable Sets and Degrees. Springer-Verlag Heidelberg.

[24] Leslie Valiant (1984): A Theory of the Learnable. Communications of the Association of Computing Machinery 27:1134–1142.

[25] Vladimir Vovk (1992) Universal Forecasting Algorithms. Information and Computation 96:245–277.

[26] Thomas Zeugmann (1993): Algorithmisches Lernen von Funktionen und Sprachen. Habilitationsschrift an der Technischen Hochschule Darmstadt.