Learning Morphology – Practice Makes Good

Alon Itai
Computer Science Department
Technion, Haifa, Israel
email: itai@cs.technion.ac.il

– Extended Abstract –

Abstract

A model for learning morphological transformations is given. The model is based upon Koskenniemi's Two Level Morphology model. It is shown that while the number of examples required for learning is polynomial, the computational problem associated with learning is intractable.

Learning homomorphisms is a simple special case of the general model. While a general method for learning homomorphisms is not known, it is shown that even if the target language is over a single letter alphabet, learning within the minimum number of examples is computationally intractable. If, however, additional examples are given, learning can be achieved in time polynomial in the size of the morphological transformation.

1 Introduction

In linguistics, morphology is the theory of the construction of words. Words are constructed from more basic units called *morphemes*. To construct a word the appropriate morphemes are concatenated then *morphological transformations* are applied to yield the final form. For example, in English the word *friendliness* is derived from the morphemes *friend*, *ly* and *ness*. A morphological transformation changes the morpheme *ly* to *li*.

English morphology is quite simple. However, other languages, such as Finnish and Hebrew, have very rich morphology. The richness can manifest itself in either the number and complexity of the morphemes involved in constructing a word (Finnish or Turkish), or the complexity of the transformations (Hebrew). We shall be concerned in the transformations, and more specifically in how they are learned.

In order to develop a theory for learning one must have a concrete model of the morphological process. We shall adopt the *Two Level Morphology* model suggested in 1983 by Koskenniemi, which has gained a lot of popularity and is the most widely accepted model for morphological transformations. The model is language independent, and permits an automatic generator and analyzer from language dependent transition rules. Koskenniemi constructed a analyzer and generator, KIMMO, and then specified the rules for Finnish

[10] and other languages [8] to obtain efficient and simple morphological tools. Several other authors used the same system for additional languages.

This approach has drawn criticism on two accounts: the computational complexity of the model [2, 7], and the appropriateness for languages whose morphology is not of a concatenative nature [11]. Rather than entering this controversy we wish to examine the learning problems associated with the model.

According to Koskenniemi's Two Level Morphology model, words are constructed in two phases: first, one constructs the lexical level consisting of a sequence of stems and affixes then morphological transformations are applied. Koskenniemi modelled these transformations by a finite state transducer (see formal definitions in Section 2.1).

Assuming that the two level model is indeed an appropriate model of morphological transformations, we concentrate on the problem of learning the transducer.

Our learning model is the bounded error model of Littlestone [12] (see Section 2.2). We feel that in the context of learning a language it is more natural than, say, the PAC model [16], since normally we receive only positive examples, and a probabilistic distribution does not seem natural. Moreover, learning according to this model implies PAC learnability [13], and learning in the limit [1].

We show that the general problem of learning a transducer is computationally infeasible. However, even for the case that the transducer has a single state, and the output language has only one letter, we show that minimizing the number of errors is an NP-Complete problem. However, in this case, if we allow more errors, then we can learn such that the number of errors is linear in the "size" of the transducer. The computation time is polynomial in the number of errors. Thus the computational problem is hard only if we insist on minimum information. If we allow redundant information, then the task becomes manageable. We show how to learn a homomorphism in time that is polynomial in the size of the training sequence, however, the length of the training sequence depends on the length of the longest example, and the degree of the polynomial is equal to the size of the input alphabet.

A similar situation occurs for learning general transducers: under reasonable cryptographic assumptions, there is no polynomial time algorithm that learns transducers from polynomial length training sequences. Oncina et al. [14] show how to learn another variant of transducers, subsequential transducers, in the limit, and each error requires work that is polynomial in the length of the current training sequence.

The rest of the paper is organized as follows: Section 2 describes the linguistic and computational learning models. Section 3 discusses learning homomorphisms, Section 4 learning transducers, and finally Section 5 contains the conclusions and further research areas.

2 The Model

This section consists of two parts. The first describes the linguistic background, and the second the Computational Learning model.

2.1 The Two Level Model

Koskenniemi's two level model for morphology considers two representations of a word: the *lexical representation* consisting of a sequence of the morphemes that constitute the word, and the *surface representation* the written (or spoken) presentation. The relationship between these two presentations is governed by the *two level rules*. These rules may be used to transform the lexical representation to the surface representation, as well as to get the lexical representation from the surface one.

From a mathematical point of view, a lexical representation is a word over some finite alphabet Σ. Koskenniemi assumed that this representation can be constructed from a *lexicon* (or from several lexicons). In this paper we do not address the problem of this construction. We assume that the lexical representation is a word v over a finite alphabet Σ and that $v \in \Sigma^*$ is given. The surface representation is also a word over a (different) alphabet Γ.

The two level rules may be viewed as a finite state transducer [4], which is a generalization of a nondeterministic finite state acceptor. The transition function, δ, not only determines the next state, but also the output. Formally, a finite state transducer is a quintuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$, where Q is a finite set of *states*; Σ and Γ are finite alphabets, called the input alphabet and output alphabet respectively; δ, the transition function, is a subset of $Q \times \Sigma \times \Gamma^* \times Q$, it may be viewed as a partial function from $Q \times \Sigma$ to subsets of $\Gamma^* \times Q$, i.e., if $(q, a, w, q') \in \delta$ then if M is in state $q \in Q$ and received input $a \in \Sigma$ it may produce output $w \in \Gamma^*$ and move to state $q' \in Q$; $q_0 \in Q$ is the *initial state*.

Given a word $v = c_1...c_n \in \Sigma^*$, *$M$ transforms v to w* if there exits a sequence of states $s_0, ..., s_n$ and $w_1, ..., w_n \in \Gamma^*$ such that:

1. $s_0 = q_0$.
2. $(s_{i-1}, c_i, w_i, s_i) \in \delta$, (for $i = 1, ..., n$).
3. $w = w_1...w_n$ (the concatenation of all the outputs).

Note that since δ is nondeterministic, there may be inputs for which the transducer produces zero, one or more output words. The case that M produces no output corresponds to rejection by a finite state acceptor.

Example 1: *The following finite state transducer properly transforms the suffix LY to li when it occurs before the end of the word.*
$\Sigma = \{A, B, ..., Z, +, \$\}$.
$\Gamma = \{a, b, ..., z\}$.
$Q = \{q_0, q_1\}$

$\delta = \{(q_0, A, a, q_0), ..., (q_0, Z, z, q_0),$
$(q_0, A, a, q_1), ..., (q_0, X, x, q_1), (q_0, Y, i, q_1), (q_0, Z, z, q_1),$
$(q_0, \$, \wedge, q_2), (q_1, +, \wedge, q_0)\}$.

Take the lexical level word FRIEND+LY+NESS. *After D the output is* friend *and the transducer is in state q_1, the first + is therefore transformed to $\wedge$ and the transducer reverts to q_0. When seeing the Y, the transducer produces i and moves to q_1, as before, + is transformed to $\wedge$. (The transition after Y to q_0 involves transforming Y to y, but then*

*there will be no transition for +, so there is no sequence that produces (*)*friendlyness*).*
□

Koskonniemi described his system somewhat differently, and provided default rules for transforming the lexical level to the surface one. The current formulation is mathematically equivalent to his system, and generalizes his only in the respect that Koskonniemi has imposed the restriction that in δ the output consists of zero or one character, i.e., $w_i \in \Gamma \cup \{\wedge\}$, where $\wedge$ denotes the empty word (the word of zero length). This restriction is natural for Finnish and simplifies the the implementation, but is unnatural for other languages (Hebrew [11] and Arabic). (In these languages, one sometimes has to insert vowels that do not exist in the lexical level. To implement these languages in KIMMO, the lexical level has to have a place-holder in every place that such a vowel can be inserted, and the transition rules have to erase them most of the time. The resultant system is redundant and unnatural.)

A finite state transducer is *deterministic* if $(q, a, w, p), (q, a, w', p') \in \delta$ implies that $w = w'$ and $p = p'$. A *homomorphism* is a deterministic finite state transducer which has only one state.

2.2 The Bounded Error Model

We now describe the learning model. Following Littlestone [12] we assume the following setting. A triple $L = (X, Y, \mathcal{F})$ is a *learning problem* if X and Y are sets and $\mathcal{F}$ a set of functions from X to Y. Learning is a game played by two agents, the *teacher* and the *student*. Initially, the teacher chooses a function $f \in \mathcal{F}$.

The game consists of an infinite number of rounds. In each round the teacher presents the student with an element $x \in X$, and the student responds with a hypothesis for $f(x)$. If the hypothesis is correct, the teacher indicates this, and otherwise, it provides $f(x)$. The goal of the teacher is to maximize the number of errors, and that of the student to minimize them.

Let b be a natural number, and $L = (X, Y, \mathcal{F})$ a learning problem. L can be *learned within b errors*, if there exists an algorithm such that for every $f \in \mathcal{F}$ and every training sequence $x_1, x_2, ... \in X$ when presented with $x_1, x_2, ...$, if the student applies the algorithm it will make at most b errors.

To address issues of computational complexity, we partition $\mathcal{F}$ into an infinite sequence of disjoint sets, i.e., $\mathcal{F} = \bigcup_n \mathcal{F}_n$. Thus, each target function $f \in \mathcal{F}$ belongs to a distinct $\mathcal{F}_n$. We allow the number of errors b to be a function of n, i.e., $b(\cdot)$ is an integer function. Then $\mathcal{F}$ can be *learned with bounded error $b(\cdot)$* if for all n, $\mathcal{F}_n$ can be learned within $b(n)$ errors. Learning is *polynomial* if $b(\cdot)$ is bounded by a polynomial. Learning is *polynomial time* if the time complexity of the student's algorithm, is also bounded by a polynomial in the bit length of the training sequence. Note that in this definition $b(n)$, the number of examples of the training sequence E, is polynomial in n, but $||E||$ the number of bits of E is not necessarily bounded by a polynomial of n. The reason for this definition is that the teacher might provide a very long example, then just reading it might take super-polynomial time.

In our setting, the set X is the lexical level, and Y the surface level. Thus, the student receives the surface level, and has to "guess" the lexical level. This models the situation

where a child understands the lexical level from the context, and tries to generate the surface level. Correct transformations are reinforced, while erroneous ones are corrected. In the more realistic context of unsupervised learning (no explicit corrections), the student would eventually hear the correct form, and thus receive the correction.

The partition of $\mathcal{F}$ can be done several ways, either according to the number of states of the transducer, or according to the size of its descriptions (its Kolmogorov complexity). We shall indicate the partition when appropriate.

3 Learning Homomorphisms

In Section 4 we'll show that learning a general transducer is hard. We first consider a simple case where the transducer is deterministic and consists of a single state. In this case the transducer is a homomorphism.

3.1 A Polynomial Algorithm for $|\Gamma| = 1$.

We first consider the simple case that the output alphabet Γ consists of a single letter, i.e., $\Sigma = \{c_1, c_2, ..., c_n\}$, $\Gamma = \{a\}$.

In this case, we can learn within n errors. Let $x_i = |h(c_i)|$ and $n_i(v)$ be the number of occurrences of c_i in v. Every example $(v_j, h(v_j))$ induces an equation in the unknowns $x_1, ..., x_n$:

$$\sum_{i=1}^{n} n_i(v_j)x_i = |h(v_j)|.$$

Thus, once we have n linearly independent equations, we can compute the x_i's.

Consider the kth round. If the equation associated with v_k is linearly dependent on those of $v_1, ..., v_{k-1}$, then we can calculate $h(v_k)$ from the available data, without introducing any additional errors. Otherwise, we might introduce an error, but this case can occur at most n times, since there can be at most n linearly independent equations in n unknowns. The computation time is that of solving a system of linear equations – $O(n^3 + ||E||)$, where $||E||$ is the bit length of the training sequence E.

Also n is a lower bound on the number of errors: For $i = 1, ..., n$, let $v_i = c_i$. There is no way to predict $h(v_i)$ before seeing it.

3.2 Learning within the Minimum Number of Errors

Consider the problem of learning a homomorphism. Sometimes after N rounds the student has enough information to avoid any further errors. However, the computational effort might be prohibitive.

Let TWO HOMOMORPHISMS be the following decision problem:
INPUT: $(v_1, w_1), ..., (v_N, w_N) \in \Sigma^* \times \Gamma^*$, $v_{N+1} \in \Sigma^*$.
QUESTION:Do there exist homomorphisms h_1, h_2, such that for $i = 1, ..., N$, $h_1(v_i) = h_2(v_i) = w_i$, and $h_1(v_{N+1}) \neq h_2(v_{N+1})$.

Theorem 1: *TWO HOMOMORPHISMS is NP-Complete, even if* Γ *consists of a single letter.*

Proof: We show a reduction from 3-Exact Cover (3XC):
INPUT: Sets $S_1, ..., S_m \subseteq \{1, ..., 3n\}$, each set S_i has cardinality 3.
QUESTION: Does there exist a subset of indices $I \subseteq \{1, ..., n\}$, such that every element is covered exactly once by $\bigcup_{i \in I} S_i$, i.e., for all $j \in \{1, ..., 3n\}$ there exist a unique $i \in I$ for which $j \in S_i$.

Given an instance of 3XC, we construct an instance of TWO HOMOMORPHISM as follows: $\Sigma = \{c_0, c_1, ..., c_n\}$, $\Gamma = \{a\}$. For $j = 1, ..., 3n$ let $w_j = a$, and if $S_{j_1} S_{j_2} ... S_{j_{k_i}}$ are all the sets that contain j then $v_j = c_0 c_{j_1} c_{j_2} ... c_{j_{k_i}}$. Finally, let $v_{3n+1} = c_0$.

Let $I = \{i_1, ..., i_n\}$ be a solution to 3XC. Then define h_1, as follows

$$h_1(c_i) = \begin{cases} a & \text{if } i \in I \\ \wedge & \text{otherwise.} \end{cases}$$

(In particular $h_1(c_0) = \wedge$.) The assumption that every j is covered by exactly one $S_i, i \in I$, implies that $h_1(v_j) = a = w_j$. Thus h_1 satisfies all the $N = 3n$ constraints $h(v_j) = a$.

Define h_2:

$$h_2(c_i) = \begin{cases} a & \text{if } i = 0 \\ \wedge & \text{otherwise.} \end{cases}$$

However, $h_1(c_0) = \wedge \neq a = h_2(c_0)$. Thus, h_1 and h_2 differ on v_{N+1}.

In the other direction: Obviously, h_2, as defined above, satisfies all the constriants. Let h_1 be another homomorphism that satisfies the constraints. Since c_0 appears in all the v_i's and $h(v_i) = a$ $(i \leq 3n)$, $|h_1(c_0)| \leq 1$. Thus, since $h_1(c_0) \neq h_2(c_0)$, we must have $h_1(c_0) = \wedge$.

Define $I = \{i : h_1(c_i) \neq \wedge\}$. Since for all $j \leq 3n$, $h_1(v_j) = a$, for each j, v_j contains a letter c_{i_j} $(i_j \geq 1)$ for which $h_1(c_{i_j}) = a$. Thus, $i_j \in I$. Also, we cannot have $j \in S_i, S_{i'}$ for distinct $i, i' \in I$, since then $h_1(v_j)$ would contain at least two a's. □

Thus if there is only one homomorphism, the student need not make an error after the first N rounds. And if there is more than one homomorphism, an adversary may force the student to make an error in round $N + 1$. However, the student must solve an NP-complete problem in order to know that there is only one answer.

3.3 Discussion

The proofs of Sections 3.1 and 3.2 seem to contradict each other. The first states that learning requires polynomial time, and the second that learning a homomorphism is NP-Complete. The difference is that the positive result allows more examples than necessary from an Information Theoretic point of view, while the negative result holds only when we insist on the minimum number of examples. Thus, if we allow more examples, learning becomes feasible, i.e., practice makes good.

3.4 Larger Output Alphabets

When $|\Gamma| = 1$ every error introduced a new independent linear equation. This does not hold for larger output alphabets.

Example 2: *Consider the training sequence* $((c_1c_2, ababab), (c_2c_1, ababab)$ *. Both examples yield the the same equation:* $x_1 + x_2 = 6$*. There are 7 homomorphisms consistent with the first example, (define* h_i *by* $|h_i(c_1)| = i$*,* $i = 0, ...6$*) and four homomorphism consistent with the second one (*h_i*,* $i = 0, 2, 4, 6$*.) Thus, in this training sequence we might make two errors, and still not find* h*.*

Thus this case seems more difficult, and we have not found a bounded error learning algorithm. We will resort to finding algorithms that are polynomial in the size of the training sequence but exponential in the size of the input alphabet.

Let $|\Gamma| \geq 1$ and let n be the size of the input alphabet Σ. It is trivial to learn a homomorphism within n errors if in every example (v_j, w_j), v_j consists of a single letter. Therefore, we shall consider the the number of examples in the training sequence as a function of the length (in characters) of the *longest* example. A learning algorithm A is *example sensitive* if there exists a function $f(\cdot)$ such that if given r examples the longest of which is of length k and $r \geq f(k)$, then A learns. Learning is polynomial if f is a polynomial. As before, the learning algorithm is polynomial time if the time complexity is bounded by a polynomial in the (number of bits of the) training sequence. This definition is weaker than the one we had before, since now the number of errors depends on the size of the examples, while beforehand the number of errors depended only on the parameter n, (in this case the size of the input alphabet Σ).

In this section we show an example sensitive algorithm to learn homomorphisms, within $O(k^n)$ errors, the time complexity is polynomial in $||E||k^n$, where $||E||$ is the bit complexity of the training sequence. For fixed sized alphabets this learning algorithm is polynomial in the size of the training sequence.

In the course of the learning process, the student is presented with a training sequence $(v_1, w_1), (v_2, w_2), ...$ Let us order the letters of $\Sigma = \{c_1, ..., c_n\}$ by their order of occurrence in $v_1v_2v_3...$; i.e., c_1 appeared first, c_2 second etc. Let (v_{q_i}, w_{q_i}) be the first example containing c_i, and $\ell_i = |w_{q_i}|/n_i(v_{q_i})$. (Recall that $n_i(v)$ denotes the number of occurrences of c_i in v.) Obviously, $0 \leq |h(v_{c_i})| \leq \ell_i$. And hence the number of homomorphisms consistent with $E_i = ((v_1, w_1), ..., (v_{q_i}, w_{q_i}))$ is at most $L = \prod_{i=1}^{n}(1 + \ell_i)$. The hypothesis of our algorithm is the homomorphism consistent with all the examples seen so far and whose lengths are lexicographically minimum, i.e., the consistent homomorphism for which $(|h(c_1)|, |h(c_2)|, ..., |h(c_i)|)$ is lexicographically minimum.

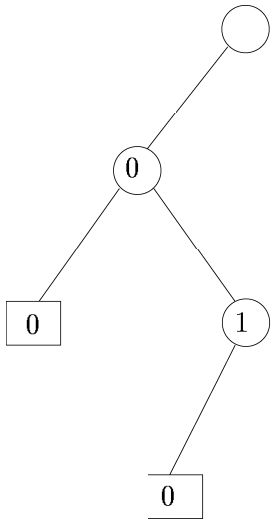
Since a hypothesis is updated only on making an error or seeing a new letter, the number of errors is also bounded by L. Consequently, if k is the length of the longest example v_q, then the number of errors is $b \leq L \leq (1 + k)^n = O(k^n)$.

A straightforward implementation of the above ideas requires $O(||E|^2k^n)$ time. However, with suitible data structures we illustrate how this can be done in $O(||E||k^n)$ time.

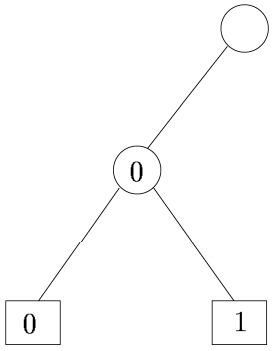
A *length vector* is a sequence of length $i \leq n$ of nonnegative integers. Given a finite training sequence , $E = ((v_1, w_1), ...(v_q, w_q))$ over $\{c_1, ..., c_i\} \subseteq \Sigma$, a length vector $\lambda^i = (\lambda_1, ..., \lambda_i)$ is *consistent* with E if there exists a homomorphism h consistent with E such that $|h(c_j)| = \lambda_j$, $(j = 1, ..., i)$. We can check if a length vector λ^i is consistent with a training sequence E in $O(||E||)$ time.

During the course of the algorithm we check length vectors for consistency. However, not all vectors need be checked separately, since if $\lambda^i = (\lambda_1, ..., \lambda_i)$ is inconsistent then for all $j \geq i$ so is $\lambda^j = (\lambda_1, ..., \lambda_j)$. Thus we need not keep inconsistent length vectors

that have a prefix which is also inconsistent. This observation suggests that we keep the inconsistent length vector in a *trie*. In our context, a trie is a labelled tree of depth $\leq n$. Its root is labelled by the empty word, all vertices at depth i are labelled with a value for λ_i. Thus a vertex of depth $i-1$ can have at most $1+\ell_i$ children, labelled $0, ..., \ell_i$, (in the sequal we will bound the maximum degree by ℓ_i). Therefore, each path from the root to a leaf at level i spells out an inconsistent length vector of length i (see Figure 1 (a)).



(a) Trie for
(0,0) (0,1,0)



(b) Trie for
(0,0) (0,1,0) (0,1,1)

Figure 1: Tries with upper bounds: $\ell_1 = 1$, $\ell_2 = 2$, $\ell_3 = 1$.

Instead of looking for a homomorphism directly, we look for a consistent length vector. Thus when we find that a length vector $\lambda = (\lambda_1, ..., \lambda_i)$ has become inconsistent, we insert it into the trie. Let $\pi = (\lambda_1, ..., \lambda_{i-1})$ be the parent of λ. If π has $1 + \ell_{i-1}$ children, then π itself is inconsistent, so we can remove all its children and turn π into a leaf (see Figure 1 (b)). Otherwise, we check $\lambda' = (\lambda_1, ..., 1 + \lambda_i)$ for consistency.

This process ensures that every length vector considered is either inconsistent or corresponds to the lexicographically minimum consistent homomorphism. Since checking whether a length vector is consistent requires at most time proportional to $O(||E||)$, and there are at most L infeasible length vectors, the entire process requires $O(||E||L) = O(||E||k^n)$ time.

4 Learning Transducers

In general, learning transducers is difficult, since it generalizes learning finite state machines.

The following problem has been shown by Gold [6], (see also [5, AL8]) to be NP-complete:

MINIMUM INFERRED FINITE STATE AUTOMATON
INSTANCE: Finite alphabet Σ, two finite subsets $S, T \subseteq \Sigma^*$, positive integer K.
QUESTION: Is there a K-state deterministic finite automaton A that recognizes a language $L \subseteq \Sigma^*$ such that $S \subseteq L$ and $T \subseteq \Sigma^* - L$?

Theorem 2: *The problem of learning a transducer from m positive and negative examples is NP-Complete.*

Since the problem is obviously in NP, it suffices to show a reduction from MINIMUM INFERRED FINITE STATE AUTOMATON.

Let $+, -, @ \notin \Sigma$, $\Sigma' = \Sigma \cup \{@\}$ and $\Gamma = \Sigma \cup \{+, -\}$. For each $s \in S$ give the pair $(s@, s+)$, and every in $t \in T$ the pair $(t@, t-)$, i.e., @ is transformed to either + or − depending on whether the preceding word belongs to L.

Let $M_R = (Q, \Sigma, \delta_R, q_0, F)$ be a finite state machine that accepts all words of S and rejects all words of T. We construct a finite state transducer $M_T = (Q, \Sigma', \Gamma, \delta_T, q_0)$. We define δ_T as follows: for every triple $(q, a, q') \in \delta_R$, δ_T contains the quadruple (q, a, q', a). In addition δ_T contains the quadruples $(q, @, q, +)$ for all $q \in F$ and $(q, @, q, -)$ for $q \notin F$. Both M_R and M_T have the same number of states, and M_T performs the correct transformation, iff M_R accepts S and rejects T.

The reverse construction of M_R from M_T is similar. Thus finding a K state transducer is equivalent to finding a K state finite state machine. □

We can use the above construction to show that learning transducers is difficult in the following sense: Let R_n be the set of regular languages accepted by n-state finite automata. Since the VC-dimension of R_n is polynomial [3], then from an information theoretic point of view, R_n can be PAC-learned within polynomially many examples.

Kearns and Valiant [9] have shown that subject to reasonable cryptographic assumptions, there exist learning problems that cannot be learned in polynomial time, even if randomized algorithms are allowed.

Warmuth and Pitt [15] have shown that regular languages are the most hard to learn. Thus, subject to the same cryptographic assumptions, there does not exist an algorithm to learn a the language of an arbitrary n-state finite automaton whose computation time is polynomial in n. This results holds even if we allow an arbitrary number of examples.

The transformation used in the previous theorem shows that learning the transformation of a transducer is just as hard. Thus there probably is no polynomial time algorithm to PAC-learn the transformation of a transducer. Since bounded error learning implies PAC learning [13], there is no polynomial time algorithm to learn transducers in the bounded error model.

5 Conclusions

We have shown a model to discuss the problem of learning morphology. However, the general model (transducers) is computationally intractable. We therefore considered a special case, homomorphisms into a single letter alphabet. In this context we exhibited a tradeoff between computational time and the number of examples.

More work should be done on learning homomorphisms into larger alphabets. More importantly, it would be interesting to find natural linguistic limitations of the type of morphological transformations such that the resultant learning problem would become tractable.

References

[1] Angluin, D. and C. Smith, "Inductive inference: theory and methods". Comput. Surveys, 15, 237-269, (1983)

[2] Barton, G. E., "Computational complexity in two-level morphology". ACL '86, 53-59.

[3] Blumer A., A. Ehrenfeucht, D. Haussler and M. Warmuth, *Learnability and the Vapnik-Chervonenkis dimension*, J. ACM, 36(4), 929-965, (1989).

[4] Harrison, M. A., *Introduction to Formal Language Theory*, Addison-Wesley, 1978.

[5] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, (1979).

[6] Gold, E. M., "Complexity of automaton identification from given data". Information and Control, 37, 302-320, (1978).

[7] Koskenniemi, K. and K. W. Church. "Complexity, Two-level morphology, and Finnish". COLING '88.

[8] Kataja, L., and K. Koskenniemi, "Finite-state description of Semitic morphology: A case study of Ancient Akkadian".

[9] Kearns, M., and L. Valiant, "Cryptographic limitations on learning Boolean formulae and finite automata". Proc. 21st STPC, 433-444, (1989).

[10] Koskenniemi, K., "Two-level morphology: A general computational model for word-form recognition and production". Publication No. 11, University of Helsinki, Dept. of General Linguistics, Hallituskata 11-33, SF-00100 Helsinki 10, Finland.

[11] A. Itai, A. Lavie, U. Ornan and M. Rimon, "On the applicability of Two Level morphology to the inflection of Hebrew verbs". ALLC June 1988, Jerusalem, (TR 513, CS Department Technion).

[12] Littlestone, N., "Learning quickly when irrelevant attributes abound: a new linear threshold algorithm". Machine Learning 2, 285-318, (1987).

[13] Littlestone, N., "Mistake bounds and logarithmic linear-threshold learning algorithms", Ph.D. Thesis, U.C. Santa Cruz, March 1989.

[14] Oncina, J., P.García, and E. Vidal, Learning subsequential transducers for pattern recognition interpretation tasks". IEEE Trans. Pattern Anal. and Machine Intell., 15, 448-458, (1993)

[15] Pitt, L., and M. K. Warmuth, "Reduction among prediction problems: on the difficulty of predicting automata". Proc. of 3rd annual structure in complexity theory, (1988).

[16] Valiant L.G., *A Theory of the Learnable*, Comm. ACM, 27(11), 1134-42, (1984).