Learning Unbiased Stochastic Edit Distance in the form of a Memoryless Finite-State Transducer[*]

Jose Oncina[†]

Dep. de Lenguajes y Sistemas Informáticos, Universidad de Alicante, E-03071 Alicante (Spain)

Marc Sebban[‡]

EURISE, Université de Saint-Etienne,
23 rue du Docteur Paul Michelon, 42023 Saint-Eienne (France)

Abstract

We aim at learning an *unbiased* stochastic edit distance in the form of a finite-state transducer from a corpus of (*input,output*) pairs of strings. Contrary to the other standard methods, which generally use the algorithm Expectation Maximization, our algorithm learns a transducer independently on the marginal probability distribution of the *input* strings. Such an unbiased way to proceed requires to optimize the parameters of a *conditional* transducer instead of a *joint* one. This transducer can be very useful in many domains of pattern recognition and machine learning, such as noise management, or DNA alignment. Several experiments are carried out with our algorithm showing that it is able to correctly assess theoretical target distributions.

1 Introduction

Many applications dealing with sequences require to compute the similarity of a pair (*input,output*) of strings. A widely-used similarity measure is the well known *edit distance*, which corresponds to the minimum number of operations, *i.e. insertions, deletions*, and *substitutions*, required to transform the *input* into the *output*. If this transformation is based on a random phenomenon and then on an underlying probability distribution, edit operations become random variables. We call then the resulting similarity measure, the *stochastic edit distance*.

An efficient way to model this distance consists in viewing it as a stochastic transduction between the input and output alphabets [Ristad and Yianilos, 1998]. In other words, it means that the relation constituted by the set of (*input,output*) strings

can be compiled in the form of a 2-tape automaton, called a *stochastic finite-state transducer*. Such a model is able to assign a probability at each new pair of strings, and could be then very useful to tackle many problems based on edit operations, such as segmentation, DNA alignment, classification, noisy channel decoding, or more generally to handle noise in sequences. Concerning this last case, note that Sakakibara and Siromomey have characterized in [Sakakibara and Siromoney, 1992] what they call *edit noise, i.e.* the result of the corruption of an input string (into an output one) by random errors of edit operations. In such a context, learning a transducer providing a probability to each couple (*input,output*) of sequences would be very useful in domains where the presence of noise has dramatic effects on the quality of the inferred models. This is the case in grammatical inference, for instance, which requires either to remove or correct noisy data to avoid overfitting phenomena. More generally, the main problem does not consist in finding domains where such a model of stochastic edit distance could be efficiently used, but rather in estimating the parameters of the transducer itself. Actually, stochastic finite-state transducers suffer from the lack of a training algorithm. To the best of our knowledge, the first published algorithm to automatically learn the parameters of a stochastic transducer has been proposed by Ristad and Yianilos [Ristad and Yianilos, 1996; 1998]. They provide a stochastic model which allows us to learn a stochastic edit distance, in the form of a memoryless transducer (*i.e.* with only one state), from a corpus of similar examples, using the Expectation Maximization (EM) algorithm. During the last few years, the algorithm EM has also been used for learning other transducer-based models [Casacuberta, 1995; Clark, 2002; Eisner, 2002].

Ristad and Yianilos define the stochastic edit distance between two strings x and y as (the minus logarithm of) the *joint* probability of the pair (x, y). In this paper, we claim that it would be much more relevant to express the stochastic edit distance from a *conditional* probability.

First, in order to correctly compute the edit distance, we think that the probabilities of edit operations over a symbol must be independent of those computed over another symbol. In other words, if the transformation of a string x into another one y does not require many edit operations, it is expected that the probability of the substitution of a symbol by itself should be high. But, as the sum of the probabilities of all edit

operations is one, then the probability of the substitution of another symbol by itself can not obviously be too large. Thus, by using a joint distribution (summing to 1), one generates an awkward dependence between edit operations.

Moreover, we think that the primitive edit costs of the edit distance must be independent of the *a priori* distribution $p(x)$ of the input strings. However, $p(x)$ can be directly deduced from the joint distribution $p(x, y)$, as follows: $p(x) = \sum_{y \in Y^*} p(x, y)$, where Y^* is the set of all finite strings over the output alphabet Y. This means that this information is totally included in the joint distribution. By defining the stochastic edit distance as a function of the joint probability, as done in [Ristad and Yianilos, 1998], the edit costs are then dependent of $p(x)$. However, if we use a conditional distribution, this dependence is removed, since it is impossible to obtain $p(x)$ from $p(y|x)$ alone.

Finally, although it is sensible and practical to model the stochastic edit distance by a memoryless transducer, it is possible that the *a priori* distribution $p(x)$ may not be modeled by such a very simple structure. Thus, by learning a transducer defining the joint distribution $p(x, y)$, its parameters can converge to compromise values and not to the true ones. This can have dramatic effects from an application standpoint. Actually, a widely-used solution to find an optimal output string y according to an input one x consists in first learning the joint distribution transducer and later deducing the conditional transducer dividing by $p(x)$ (more precisely by its estimates over the learning set). Such a strategy is then irrelevant for the reason we mentioned above.

In this paper we have developed a way to learn directly the conditional transducer. After some definitions and notations (Section 2), we introduce in Section 3 the learning principle of the stochastic edit distance proposed by Ristad and Yianilos [Ristad and Yianilos, 1996; 1998]. Then, by simulating different theoretical joint distributions, we show that the *unique way*, using their algorithm, to find them consists in sampling a learning set of (x, y) pairs according to the marginal distribution (*i.e.* over the input strings) of the target joint distribution itself. Moreover, we show that for all other *a priori* distribution, the difference between the target and the learned models increases. To free the method from this bias, one must *directly* learn at each iteration of the algorithm EM the conditional distribution $p(y|x)$. Achieving this task requires to modify Ristad and Yianilos's framework. That is the goal of Section 4. Then, we carry out experiments that show that it is possible to correctly estimate a target distribution whatever the *a priori* distribution we use.

2 Classic String Edit Distance

An alphabet X is a finite nonempty set of symbols. X^* denotes the set of all finite strings over X. Let $x \in X^*$ be an arbitrary string of length $|x|$ over the alphabet X. In the following, unless stated otherwise, symbols are indicated by $a, b, \ldots$, strings by $u, v, \ldots, z$, and the empty string by λ. $\mathbb{R}^+$ is the set of non negative reals. Let $f(\cdot)$ be a function, from which $[f(x)]_{\pi(x, \ldots)}$ is equal to $f(x)$ if the predicate $\pi(x, \ldots)$ holds and 0 otherwise, where x is a (set of) dummy variable(s).

A string edit distance is characterized by a triple (X, Y, c_e) consisting of the finite alphabets X and Y and the primitive cost function $c_e : E \to \mathbb{R}^+$ where $E = E_s \cup E_d \cup E_i$ is the alphabet of primitive edit operations, $E_s = X \times Y$, is the set of substitutions, $E_d = X \times \{\lambda\}$ is the set of deletions, $E_i = \{\lambda\} \times Y$ is the set of insertions. Each such triple (X, Y, c_e) induces a distance function $d : X^* \times Y^* \to \mathbb{R}^+$ that maps a pair of strings to a non negative real value. The edit distance $d(x, y)$ between two strings $x \in X$ and $y \in Y$ is defined recursively as:

$$d(x, y) = \min \begin{cases} [c_e(a, b) + d(x', y')]_{x = x'a \wedge y = y'b} \\ [c_e(a, \lambda) + d(x', y)]_{x = x'a} \\ [c_e(\lambda, y) + d_c(x, y')]_{y = y'b} \end{cases}$$

Note that $d(x, y)$ can be computed in $O(|x| \cdot |y|)$ time using dynamic programming.

3 Stochastic Edit Distance and Memoryless Transducers

If the edit operations are achieved according to a random process, the edit distance is then called the *stochastic edit distance*, and noted $d_s(x, y)$. Since the underlying probability distribution is unknown, one solution consists in learning the primitive edit costs by means of a suited model. In this paper, we used memoryless transducers. Transducers are currently used in many applications ranging from lexical analyzers, language and speech processing, etc. They are able to handle large amount of data, in the form of pairs of (x, y) sequences, in a reasonable time complexity. Moreover, assuming that edit operations are randomly and independently achieved (that is the case in the edit noise [Sakakibara and Siromoney, 1992]), a memoryless transducer is sufficient to model the stochastic edit distance.

3.1 Joint Memoryless Transducers

A joint memoryless transducer defines a joint probability distribution over the pairs of strings. It is denoted by a tuple (X, Y, c, γ) where X is the input alphabet, Y is the output alphabet, c is the *primitive* joint probability function, $c : E \to [0, 1]$ and γ is the probability of the termination symbol of a string. As $(\lambda, \lambda) \notin E$, in order to simplify the notations, we are going to use $c(\lambda, \lambda)$ and γ as synonyms.

Let us assume for the moment that we know the probability function c (in fact, we will learn it later). We are then able to compute the joint probability $p(x, y)$ of a pair of strings (x, y). Actually, the joint probability $p : X^* \times Y^* \to [0, 1]$ of the strings x, y can be recursively computed by means of an auxiliary function (forward) $\alpha : X^* \times Y^* \to \mathbb{R}^+$ as:

$$\alpha(x, y) = \quad [1]_{x = \lambda \wedge y = \lambda}$$
$$+ [c(a, b) \cdot \alpha(x', y')]_{x = x'a \wedge y = y'b}$$
$$+ [c(a, \lambda) \cdot \alpha(x', y)]_{x = x'a}$$
$$+ [c(\lambda, b) \cdot \alpha(x, y')]_{y = y'b}.$$

And then,

$$p(x, y) = \alpha(x, y)\gamma.$$

In a symmetric way, $p(x, y)$ can be recursively computed by means of an auxiliary function (backward) $\beta : X^* \times Y^* \to \mathbb{R}^+$ as:

$$\beta(x, y) = [1]_{x=\lambda \wedge y=\lambda}$$
$$+ [c(a, b) \cdot \beta(x', y')]_{x=ax' \wedge y=by'}$$
$$+ [c(a, \lambda) \cdot \beta(x', y)]_{x=ax'}$$
$$+ [c(\lambda, b) \cdot \beta(x, y')]_{y=by'}.$$

And then,

$$p(x, y) = \beta(x, y)\gamma.$$

Both functions (forward and backward) can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique. This model defines a probability distribution over the pairs (x, y) of strings. More precisely,

$$\sum_{x \in X^*} \sum_{y \in Y^*} p(x, y) = 1,$$

that is achieved if the following conditions are fulfilled [Ristad and Yianilos, 1998],

$$\gamma > 0, c(a, b), c(\lambda, b), c(a, \lambda) \geq 0 \quad \forall a \in X, b \in Y$$

$$\sum_{\substack{a \in X \cup \{\lambda\} \\ b \in Y \cup \{\lambda\}}} c(a, b) = 1$$

Given $p(x, y)$, we can then compute, as mentioned in [Ristad and Yianilos, 1998], the stochastic edit distance between x and y. Actually, the stochastic edit distance $d_s(x, y)$ is defined as being the negative logarithm of the probability of the string pair $p(x, y)$ according to the memoryless stochastic transducer.

$$d_s(x, y) = -\log p(x, y), \forall x \in X^*, \forall y \in Y^*$$

In order to compute $d_s(x, y)$, a remaining step consists in learning the parameters $c(a, b)$ of the memoryless transducer, *i.e.* the primitive edit costs.

3.2 Optimization of the parameters of the joint memoryless transducer

Let S be a finite set of (x, y) pairs of *similar* strings. Ristad and Yianilos [Ristad and Yianilos, 1998] propose to use the expectation-maximization (EM) algorithm to find the optimal joint stochastic transducer. The EM algorithm consists in two steps (expectation and maximization) that are repeated until a convergence criterion is achieved.

Given an auxiliary $(|X| + 1) \times (|Y| + 1)$ matrix δ, the expectation step can be described as follows: $\forall a \in X, b \in Y$,

$$\delta(a, b) = \sum_{(xax', yby') \in S} \frac{\alpha(x, y)c(a, b)\beta(x', y')\gamma}{p(xax', yby')}$$

$$\delta(\lambda, b) = \sum_{(xx', yby') \in S} \frac{\alpha(x, y)c(\lambda, b)\beta(x', y')\gamma}{p(xx', yby')}$$

$$\delta(a, \lambda) = \sum_{(xax', yy') \in S} \frac{\alpha(x, y)c(a, \lambda)\beta(x', y')\gamma}{p(xax', yy')}$$

$$\delta(\lambda, \lambda) = \sum_{(x, y) \in S} \frac{\alpha(x, y)\gamma}{p(x, y)} = |S|,$$

$c^*(a, b)$	λ	a	b	c	d	$c^*(a)$
λ	0.00	0.05	0.08	0.02	0.02	0.17
a	0.01	0.04	0.01	0.01	0.01	0.08
b	0.02	0.01	0.16	0.04	0.01	0.24
c	0.01	0.02	0.01	0.15	0.00	0.19
d	0.01	0.01	0.01	0.01	0.28	0.32

Table 1: Target joint distribution $c^*(a, b)$ and its corresponding marginal distribution $c^*(a)$.

and the maximization as:

$$c(a, b) = \frac{\delta(a, b)}{N} \quad \forall a \in X \cup \{\lambda\}, \forall b \in Y \cup \{\lambda\}$$

where

$$N = \sum_{\substack{a \in X \cup \{\lambda\} \\ b \in Y \cup \{\lambda\}}} \delta(a, b).$$

3.3 Limits of Ristad and Yianilos's algorithm

To analyze the ability of Ristad and Yianilos's algorithm to correctly estimate the parameters of a target joint memoryless transducer, we have implemented it and carried out a series of experiments. Since the joint distribution $p(x, y)$ is a function of the learned edit costs $c(a, b)$, we only focused here on the function c of the transducer.

The experimental setup was the following. We simulated a target joint memoryless transducer from the alphabets $X = Y = \{a, b, c, d\}$, such as $\forall a \in X \cup \{\lambda\}, \forall b \in Y \cup \{\lambda\}$, the target model is able to return the primitive theoretical joint probability $c^*(a, b)$. The target joint distribution we used is described in Table 1[1]. The marginal distribution $c^*(a)$ can be deduced from this target such that: $c^*(a) = \sum_{b \in X \cup \{\lambda\}} c^*(a, b)$.
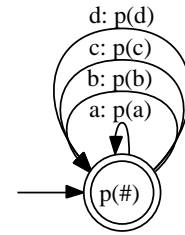

Figure 1: Automaton used for generating the input sequences. The probability $p(\#)$ corresponds to the probability of a termination symbol of a string, or in other words the probability of the state to be final.

---

[1]Note that we carried out many series of experiments with various target joint distributions, and all the results we obtained follow the same behavior as the one presented in this section.

Then, we sampled an increasing set of learning input strings (from 0 to 4000 sequences) of variable length generated from a given probability distribution $p(a)$ over the input alphabet X. In order to simplify, we modeled this distribution in the form of an automaton with only one state[2] and $|X|$ output transitions with randomly chosen probabilities satisfying that $\sum_{a \in X} p(a) + p(\#) = 1$, where $p(\#)$ corresponds to the probability of a termination symbol of a string (see Figure 1).

We used different settings for this automaton to analyze the impact of the input distribution $p(a)$ on the learned joint model. Then, given an input sequence x (generated from this automaton) and the target joint distribution $c^*(a, b)$, we sampled a corresponding output y. Finally, the set S of generated (x, y) pairs was used by Ristad and Yianilos's algorithm to learn an estimated primitive joint distribution $c(a, b)$.

We compared the target and the learned distributions to analyze the behavior of the algorithm to correctly assess the parameters of the target joint distribution. We computed an average difference between the both, defined as follows:

$$d(c, c^*) = \frac{\sum_{a \in X \cup \{\lambda\}} \sum_{b \in Y \cup \{\lambda\}} |c(a, b) - c^*(a, b)|}{2}$$

Normalized in this way, $d(c, c^*)$ is a value in the range $[0, 1]$. Figure 2 shows the behavior of this difference according to various configurations of the automaton of Figure 1. We can note that the unique way to converge towards a difference near from 0 consists in using the marginal distribution $c^*(a)$ of the target for generating the input strings. For all the other ways, the difference becomes very large.

As we said at the beginning of this article, we can easily explain this behavior. By learning the primitive joint probability function $c(a, b)$, Ristad and Yianilos learn at the same time the marginal distribution $c(a)$. The learned edit costs (and the stochastic edit distance) are then dependent of the *a priori* distribution of the input strings, that is obviously awkward. To free of this statistical bias, we have to learn the primitive conditional probability function independently of the marginal distribution. That is the goal of the next section.

4 Unbiased Learning of a Conditional Memoryless Transducer

A conditional memoryless transducer is denoted by a tuple (X, Y, c, γ) where X is the input alphabet, Y is the output alphabet, c is the primitive conditional probability function $c : E \to [0, 1]$ and γ is the probability of the termination symbol of a string. As in the joint case, since $(\lambda, \lambda) \notin E$, in order to simplify the notation we use γ and $c(\lambda|\lambda)$ as synonyms.

The probability $p : X^* \times Y^* \to [0, 1]$ of the string y assuming the input one was a x (noted $p(y|x)$) can be recursively computed by means of an auxiliary function (forward)

$\alpha : X^* \times Y^* \to \mathbb{R}^+$ as:

$$\begin{aligned}
\alpha(y|x) = {} & [1]_{x=\lambda \wedge y=\lambda} \\
& + [c(b|a) \cdot \alpha(y'|x')]_{x=x'a \wedge y=y'b} \\
& + [c(\lambda|a) \cdot \alpha(y|x')]_{x=x'a} \\
& + [c(b|\lambda) \cdot \alpha(y'|x)]_{y=y'b}.
\end{aligned}$$

And then,

$$p(y|x) = \alpha(y|x)\gamma.$$

In a symmetric way, $p(y|x)$ can be recursively computed by means of an auxiliary function (backward) $\beta : X^* \times Y^* \to \mathbb{R}^+$ as:

$$\begin{aligned}
\beta(y|x) = {} & [1]_{x=\lambda \wedge y=\lambda} \\
& + [c(b|a) \cdot \beta(y'|x')]_{x=ax' \wedge y=by'} \\
& + [c(\lambda|a) \cdot \beta(y|x')]_{x=ax'} \\
& + [c(b|\lambda) \cdot \beta(y'|x)]_{y=by'}.
\end{aligned}$$

And then,

$$p(y|x) = \beta(y|x)\gamma.$$

As in the joint case, both functions can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique. In this model a probability distribution is assigned conditionally to each input string. Then

$$\sum_{y \in Y^*} p(y|x) \in \{1, 0\} \quad \forall x \in X^*.$$

The 0 is in the case the input string x is not in the domain of the function[3]. It can be show (see Annex) that the normalization of each conditional distribution can be achieved if the following conditions over the function c and the parameter γ are fulfilled,

$$\gamma > 0, c(b|a), c(b|\lambda), c(\lambda|a) \geq 0 \quad \forall a \in X, b \in Y \quad (1)$$

$$\sum_{b \in Y} c(b|\lambda) + \sum_{b \in Y} c(b|a) + c(\lambda|a) = 1 \quad \forall a \in X \quad (2)$$

$$\sum_{b \in Y} c(b|\lambda) + \gamma = 1 \quad (3)$$

As in the joint case, the expectation-maximization algorithm can be used in order the find the optimal parameters. The expectation step deals with the computation of the matrix δ:

$$\delta(b|a) = \sum_{(xax', yby') \in S} \frac{\alpha(y|x)c(b|a)\beta(y'|x')\gamma}{p(yby'|xax')}$$

$$\delta(b|\lambda) = \sum_{(xx', yby') \in S} \frac{\alpha(y|x)c(b|\lambda)\beta(y'|x')\gamma}{p(yby'|xx')}$$

$$\delta(\lambda|a) = \sum_{(xax', yy') \in S} \frac{\alpha(y|x)c(\lambda|a)\beta(y'|x')\gamma}{p(yy'|xax')}$$

$$\delta(\lambda|\lambda) = \sum_{(x,y) \in S} \frac{\alpha(y|x)\gamma}{p(y|x)} = |S|.$$
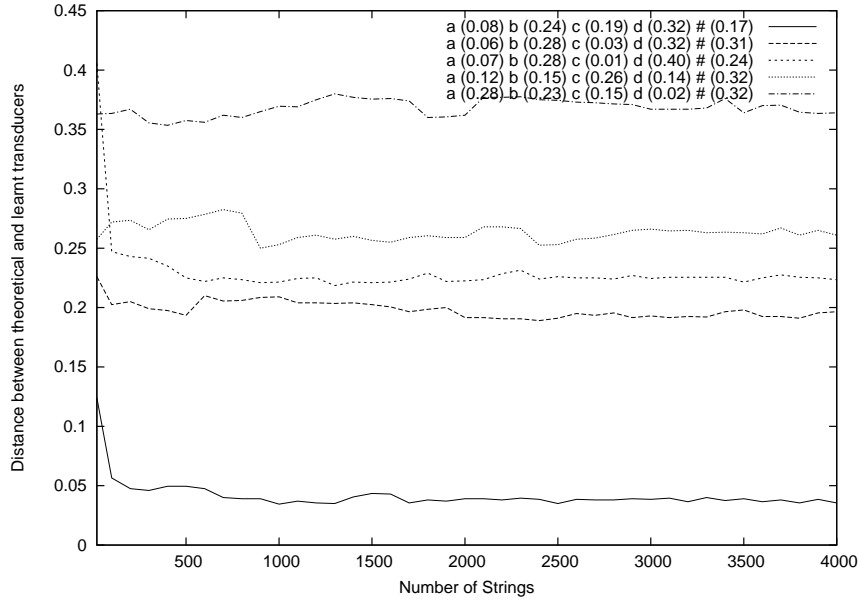

Figure 2: Average difference between the target and the learned distributions according to various generations of the input strings.

In order to do the maximization step, we begin by normalizing the insertion cost because it appears in both normalization equations (eq. 2 and eq. 3). Then:

$$c(b|\lambda) = \frac{\delta(b|\lambda)}{N}$$

where

$$N = \sum_{\substack{a \in X \cup \{\lambda\} \\ b \in Y \cup \{\lambda\}}} \delta(b|a)$$

The value of γ is now fixed by eq. 3 as:

$$\gamma = \frac{N - N(\lambda)}{N}$$

where

$$N(\lambda) = \sum_{b \in Y} \delta(b|\lambda)$$

and $c(b|a)$ and $c(\lambda|a)$ are obtained working out the values in eq. 2 and distributing the probability proportionally to their respective expectations $\delta(b|a)$ and $\delta(\lambda|a)$. Then

$$c(b|a) = \frac{\delta(b|a)}{N(a)} \frac{N - N(\lambda)}{N} \quad c(\lambda|a) = \frac{\delta(\lambda|a)}{N(a)} \frac{N - N(\lambda)}{N}$$

where

$$N(a) = \sum_{b \in Y \cup \{\lambda\}} \delta(b|a).$$

We carried out experiments to assess the relevance of our new learning algorithm to correctly estimate the parameters of target transducers. We followed exactly the same experimental setup as the one of Section 3.3, except to the definition of our difference $d(c, c^*)$. Actually, as we said before,

our new framework estimates $|X|$ conditional distributions. So $d(c, c^*)$ is defined as follows :

$$d(c, c^*) = \frac{(A + B|X|)}{2|X|}$$

where

$$A = \sum_{a \in X} \sum_{b \in Y \cup \{\lambda\}} |c(b|a) - c^*(b|a)|$$

and

$$B = \sum_{b \in Y \cup \{\lambda\}} |c(b|\lambda) - c^*(b|\lambda)|$$

The results are shown in Figure 3. We can make the two following remarks. First, the different curves clearly show that the convergence toward the target distribution is independent of the distribution of the input strings. Using different parameter configurations of the automaton of Figure 1, the behavior of our algorithm remains the same, *i.e* the difference between the learned and the target conditional distributions tends to 0. Second, we can note that $d(c, c^*)$ rapidly decreases, *i.e.* the algorithm requires few learning examples to learn the target.

5 Conclusion

In this paper, we proposed a relevant approach for learning the stochastic edit distance in the form of a memoryless transducer. While the standard techniques aim at learning a joint distribution over the edit operations, we showed that such a strategy induces a bias in the form of a statistical dependence on the input string distribution. We overcame this drawback by directly learning a conditional distribution of the primitive edit costs. The experimental results bring to the fore the
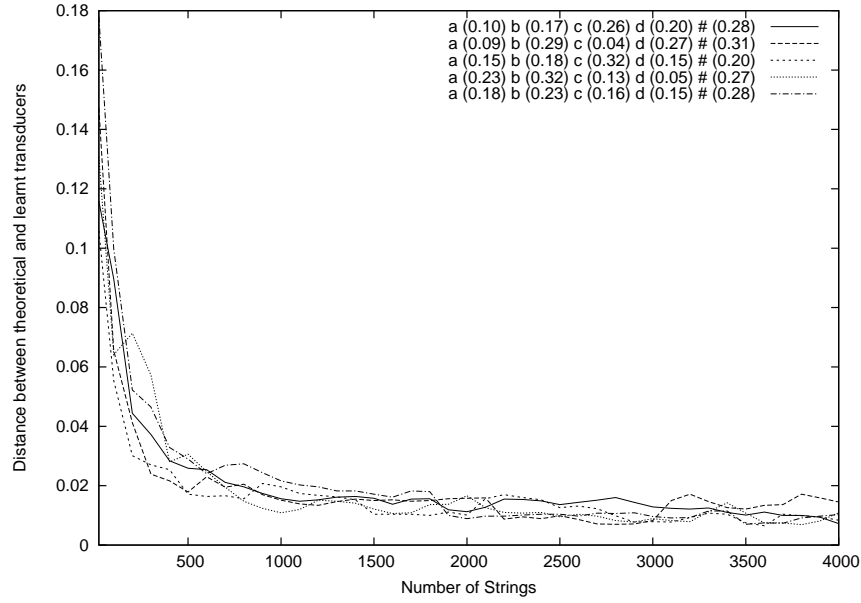
Figure 3: Average difference between the target and the learned conditional distributions according to various generations of the input strings.

interest of our approach. We think that our model is particularly suited for dealing with noisy data. For this reason, we now plan to use it on real world applications which often are subject to the presence of noisy data.

6 Annex

We are going to show that eq. 1, 2 and 3 are sufficient to satisfy

$$\sum_{y \in Y^*} p(y|x) = 1.$$

Let us first consider the case when $x = \lambda$.

$$\sum_{y \in Y^*} \alpha(y|\lambda) = 1 + \sum_{yb \in Y^*} \alpha(yb|\lambda)$$
$$= 1 + \sum_{yb \in Y^*} c(b|\lambda)\alpha(y|\lambda)$$
$$= 1 + \sum_{b \in Y} c(b|\lambda) \sum_{y \in Y^*} \alpha(y|\lambda)$$

then

$$\sum_{y \in Y^*} \alpha(y|\lambda)(1 - \sum_{b \in Y} c(b|\lambda)) = 1$$

$$\sum_{y \in Y^*} \alpha(y|\lambda) = \left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1}$$

Let us now consider the complete case

$$\sum_{y \in Y^*} \alpha(y|xa) =$$
$$\alpha(\lambda|xa) + \sum_{yb \in Y^*} \alpha(yb|xa) =$$
$$c(\lambda|a)\alpha(\lambda|x) + \sum_{yb \in Y^*} c(b|a)\alpha(y|x)$$
$$+ c(b|\lambda)\alpha(y|xa) + c(\lambda|a)\alpha(yb|x) =$$
$$\sum_{y \in Y^*} c(\lambda|a)\alpha(y|x) + \sum_{b \in Y} c(b|a) \sum_{y \in Y^*} \alpha(y|x)$$
$$+ \sum_{b \in Y} c(b|\lambda) \sum_{y \in Y^*} \alpha(y|xa) =$$
$$\left(c(\lambda|a) + \sum_{b \in Y} c(b|a)\right) \sum_{y \in Y^*} \alpha(y|x)$$
$$+ \sum_{b \in Y} c(b|\lambda) \sum_{y \in Y^*} \alpha(y|xa)$$

then

$$\sum_{y \in Y^*} \alpha(y|xa) \left(1 - \sum_{b \in Y} c(b|\lambda)\right) =$$
$$\left(c(\lambda|a) + \sum_{b \in Y} c(b|a)\right) \sum_{y \in Y^*} \alpha(y|x)$$

and

$$\sum_{y \in Y^*} \alpha(y|xa) =$$

$$\left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1} \left(c(\lambda|a) + \sum_{b \in Y} c(b|a)\right) \sum_{y \in Y^*} \alpha(y|x)$$

Applying this equation recursively on the length of x and taking in account that the base case is

$$\sum_{y \in Y^*} \alpha(y|\lambda) = \left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1}$$

we have

$$\sum_{y \in Y^*} \alpha(y|a_1 \dots a_n) =$$

$$\prod_{i=1}^{n} \left[\left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1} \left(c(\lambda|a_i) + \sum_{b \in Y} c(b|a_i)\right) \right]$$

$$\cdot \left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1}$$

and

$$\sum_{y \in Y^*} p(y|a_1 \dots a_n) =$$

$$\prod_{i=1}^{n} \left[\left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1} \left(c(\lambda|a_i) + \sum_{b \in Y} c(b|a_i)\right) \right]$$

$$\cdot \left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1} \gamma$$

A sufficient condition for $\sum_{y \in Y^*} p(y|a_1 \dots a_n) = 1$ is that each of the terms that appear in the productory is equal to 1 and that the final product is also 1. Then,

$$\left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1} \left(c(\lambda|a_i) + \sum_{b \in Y} c(b|a_i)\right) = 1$$

$$1 - \sum_{b \in Y} c(b|\lambda) = c(\lambda|a_i) + \sum_{b \in Y} c(b|a_i)$$

$$\sum_{b \in Y} c(b|\lambda) + c(\lambda|a_i) + \sum_{b \in Y} c(b|a_i) = 1$$

and we have equation 2, and

$$\left(1 - \sum_{b \in Y} c(b|\lambda)\right)^{-1} \gamma = 1$$

$$1 - \sum_{b \in Y} c(b|\lambda) = \gamma$$

$$\gamma + \sum_{b \in Y} c(b|\lambda) = 1$$

and we have equation 3.

Note that these equations are not valid if $\sum_{b \in Y} c(b|\lambda) = 1$ but this is impossible since $\gamma > 0$.

References

[Casacuberta, 1995] Francisco Casacuberta. Probabilistic estimation of stochastic regular syntax-directed translation schemes. In *Proceedings of th VIth Spanish Symposium on Pattern Recognition and Image Analysis*, pages 201–207, 1995.

[Clark, 2002] Alexander Clark. Memory-based learning of morphology with stochastic transducers. In *Proceedings of the Annual meeting of the association for computational linguistic*, 2002.

[Eisner, 2002] Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the Annual meeting of the association for computational linguistic*, pages 1–8, 2002.

[Ristad and Yianilos, 1996] Eric Sven Ristad and Peter N. Yianilos. Finite growth models. Technical Report CS-TR-533-96, Princeton University Computer Science Department, 1996.

[Ristad and Yianilos, 1998] Eric Sven Ristad and Peter N. Yianilos. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):522–532, 1998.

[Sakakibara and Siromoney, 1992] Yasubumi Sakakibara and Rani Siromoney. A noise model on learning sets of strings. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 295–302, 1992.