Design of a Linguistic Postprocessor using Variable Memory Length Markov Models

I. Guyon and F. Pereira*
AT&T Bell Laboratories, Room 4g-324, Holmdel, NJ 07733, USA
Phone: (908) 949 3220. Fax: (908) 949 7722
* AT&T Bell Laboratories, Murray Hill, NJ 07974, USA
{isabelle,pereira}@research.att.com

Abstract

We present the design of a linguistic postprocessor for character recognizers. The central module of our system is a trainable variable memory length Markov model (VLMM) which predicts the next character given a variable length window of past characters. The overall system is composed of several finite state automata, including the main VLMM and a proper noun VLMM. The best model reported in the literature (Brown et al 1992) achieves 1.75 bits per character on the Brown corpus. On that same corpus, our model, trained on 10 times less data, reaches 2.19 bits per character and is 200 times smaller ($\simeq 160,000$ parameters). The model was designed for handwriting recognition applications but can be used for other OCR problems and speech recognition.

Keywords

Linguistics, finite state automata, probabilities, statistics, statistical languages, statistical grammars, grammar inference, regular languages, handwriting recognition, speech recognition, Markov models, hidden Markov models, n-grams.

1 Introduction

In a communication process, if the receiver has a powerful model of the information source, the sender needs only to transmit the information which cannot be predicted by the model (residuals). Some people exploit naturally this result of information theory when they communicate with each other with a cryptic handwriting. They rely on the fact that they are exchanging information with people who use the same language and share much of the same background knowledge, that is, who have a good "model" of the information source.

At AT&T Bell Laboratories, we are actively investigating the use of handwriting as a means of human-machine communication. However, handwriting recognition without contextual information is similar to trying to decode an encrypted message without a proper decoder. If machines had a language model that approximated that of humans, the odds of successful decoding might be better.

We present initial results on a statistical model of English trained on a large text corpus. Our model is an extension of the *variable-memory-length Markov models* (VLMMs) of Ron, Singer and Tishby [1]. Its basic components are probabilistic finite state automata equivalent to Markov models of "variable order." These automata are built from a set of *prefixes* which are variable length character strings appearing frequently in the training corpus and having a good predictive power for the next character. To avoid the overfitting problem, the capacity of the model is controlled with the method of *structural risk minimization* (SRM) [2, 3].

We take advantage of a modular decomposition of the model into cascaded probabilistic automata. To train the main module, we factor out capitalization, number strings, symbols and proper nouns. This allows us to use standard English corpora to train the main module while keeping the flexibility to modify the models for capitalization, numbers, symbols and proper nouns models, according to the needs of particular applications. The modules defined and composed use a special case of the approach of Pereira *et al*[4]. This allows us to use graph search techniques, including beam search, without having to build explicitly the whole composed automaton.

The classical measure of performance of language models is the *cross-entropy* on a test set, which estimates the per character average length of the shortest encoding of the test set, for an encoder-decoder pair with access to the model's probability estimates. Using cross-entropy as a performance measure allows us to make comparisons with the code lengths obtained with various coding schemes. For instance, the ISO-1 code requires 8 bits per character. On the Brown corpus [5], classical coding schemes like Huffman coding or Lempel-Ziv coding use between 4 and 4.5 bits/character. In the 50's, Shannon used human subjects and estimated the entropy of English to be between 0.6 and 1.3 bits per character.

Trained on 200 million characters from the AP news, our model, tested on the 6 million characters of the Brown corpus, reaches 2.19 bits/character, using a state machine with approximately 160,000 parameters. The performance achieved by the best model reported in the literature (Brown et al. 1992) is 1.75 bits per character on the Brown corpus. This

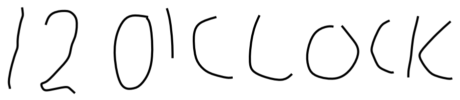

Figure 1: **Intrinsic ambiguity of handwriting:** The same symbols may have different meanings and symbol components may be grouped in different ways. Language constraints are needed to disambiguate handwriting.

model uses word trigrams with 27 million parameters estimated with a training set of 400 million words (approximately 2 billion characters). Therefore, our model, almost 200 times smaller and trained on 10 times less data, appears to have competitive performance.

2 Statement of the problem

A handwriting recognizer produces, for a given handwritten observation (figure 1), miscellaneous candidate interpretations: "IZ O'CCOCK", "12 0'CCOCK", "12 O'CLOCIC", "12 O'CLOCK", "I2 O'CCOCIC", "1201 CLOCK", etc.

In the absence of language knowledge, these interpretations are almost all equally likely. By querying a language model about the linguistic probability of each string, we can pick up the interpretation that is most consistent with the language: "12 O'CLOCK".

For a given observation o, the probability of an interpretation w is proportional to the product of the observation likelihood $P(o|w)$ and the linguistic probability $P(w)$:

$$P(w|o) = \frac{P(o|w)P(w)}{P(o)}. \tag{1}$$

An estimate of $P(o|w)$ is provided by the recognizer and an estimate of $P(w)$ is provided by the language model.

The literature offers a choice of statistical language models [6, 7] of various levels of complexity. Probabilistic finite state automata (PFSA) are the simplest statistical language models, the most widely used and, up to now, the most successful ones [8]. They include lexicon trees (or "tries"), n-grams and other Markov models. PFSAs are easily combined with any recognizer, including neural networks and hidden Markov models.

Our design addresses not only the problem of getting high prediction accuracy, but also that of flexibility and that of computational resources. Our modular approach provides flexibility to tailor the system to specific applications, and allows us to weigh the speed-accuracy tradeoffs using beam search techniques. Our use of variable-memory-length Markov models allows us to trade modeling accuracy for memory by varying a single complexity parameter.

Our goal is to approach and eventually outperform the best reported language models, while keeping our model to a practical size. For that reason, we report our results with respect to a widely available evaluation resource, the Brown corpus [5].

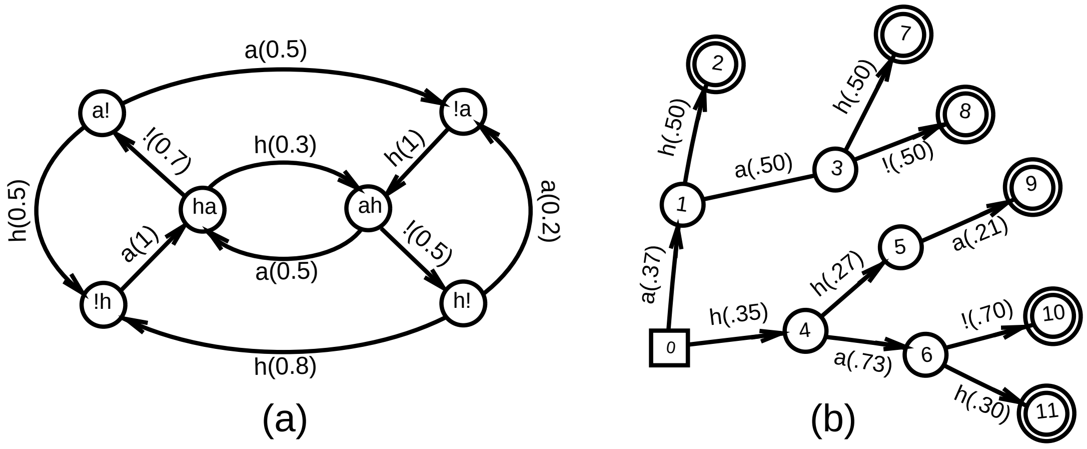

Figure 2: **Example of PFSAs with deterministic graph:** a - Cyclic graph of a tri-gram model predicting the next character given the two previous ones. Let to evolve from a random initial state, the machine produces: ah!ha!ha!ah!aha!ahaha!hah!ahah!ha!ah!ah!hah!hah!haha etc. b - Acyclic graph a lexicon tree containing the words $\{ah, aah, aa!, hha, ha!, hah\}$.

3 Overview of the design

3.1 Probabilistic Finite State Automata

Our system is a combination of several probabilistic finite state automata (PFSA). A PFSA $\mathcal{M}$ is conveniently represented by a graph. It has a finite number of states, represented by the nodes of the graph. Transitions between states, the arcs of the graph, are associated with the emission (or acceptance) of a character for a given finite alphabet with a certain probability

$$P_{\mathcal{M}}(s', \sigma|s) = P_{\mathcal{M}}(s'|\sigma, s)\ P_{\mathcal{M}}(\sigma|s) \quad . \tag{2}$$

where s and s' are states and σ the character emitted (accepted). The probabilities of all the outgoing arcs of a given node s sum up to one: $\sum_{s',\sigma} P_{\mathcal{M}}(s', \sigma|s) = 1$.

In the simplest case, two outgoing arcs from the same node cannot emit (or accept) the same character. A graph with such property is called *deterministic* because $P_{\mathcal{M}}(s'|\sigma, s) \doteq 1$ and thus the transition from one node to another is uniquely determined by the character which is emitted (or accepted), i. e. $P_{\mathcal{M}}(s', \sigma|s) = P_{\mathcal{M}}(\sigma|s)$. Given a starting state and a sequence of characters generated by the machine, it is possible to trace without ambiguity the full sequence of states that generated it. Markov models (or n-grams) and lexicon trees are examples of deterministic automata (see figure 2).

In this paper, our linguistic models have deterministic graphs. The probability of a string of characters is simply the product of the transition probabilities along the corresponding path:

$$P_{\mathcal{M}}(\sigma_1\sigma_2...\sigma_t) = \prod_i P_{\mathcal{M}}(\sigma_i|s_{i-1}) \quad , \tag{3}$$

with the convention $P_{\mathcal{M}}(\sigma_1|s_0) = P_{\mathcal{M}}(\sigma_1)$.

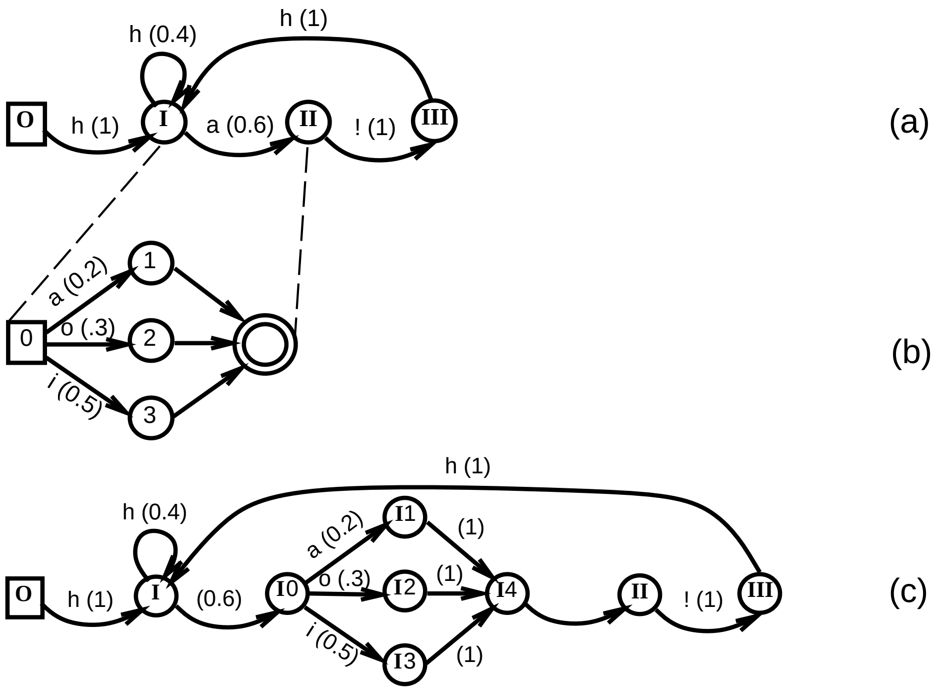

Figure 3: **Example of substitution:** We use a model $\mathcal{A}$ with an alphabet of three characters $\{a, h, !\}$, and a model $\mathcal{B}$ to substitute the character "a" produced by $\mathcal{A}$: a - PFSA $\mathcal{A}$. b - PFSA $\mathcal{B}$. c - Composed graph.

In the most general case, several outgoing arcs from the same node can emit (or accept) the same character. Discrete Hidden Markov Models (HMM), such as the ones used as speech or handwriting recognizers, correspond to this second kind of PFSA [7].

PFSA language models combine particularly easily with recognizers based on finite state automata, including recognizers of chains of presegmented characters and recognizers of unsegmented data based on dynamic time warping or hidden Markov models (see for instance examples in [7] and [9]). The probability of the best interpretation, given the recognizer scores and the language model (equation (1)), is given by the probability of the best path in the product graph of the two corresponding automata. The best path in the product graph is searched dynamically without actually building the product graph. We use a *beam search* to process "on-line" the characters to be recognized (see examples in [7]). At each step, a small number of candidate successor states are kept, those corresponding to the most probable interpretation up to then. The tradeoff between quality of the approximation, speed of calculation and memory allocation is monitored by the number of candidate paths kept at each step.

3.2 Modular Architecture

We split our language model into several independent modules, all of which are PFSAs. Two modules use the new technique of *variable memory length Markov models* (VLMM) [1] which allows training fairly compact probabilistic finite state machine with

Code or model	Code length or cross-entropy (bit/character)
ISO-1	8
Huffman	4
Lempel Ziv	4.5
Brown et al [5]	1.75
Humans	~ 1.25

Table 1: **Comparison of code length and cross-entropies on the Brown corpus.** All entropies are calculated on the Brown corpus, except for the last one which comes from the 1951 Shannon experiments on human subjects, using a book of D. Malone.

high predictive power.

The main module is a VLMM. It has an alphabet of only lowercase characters plus a few metacharacters which encompass broad categories of strings: **C** for proper nouns, **N** for numbers and **M** for symbols, etc. We have several sub-models, including a proper noun model (also a VLMM) and models for numbers, symbols, punctation, etc. When using the model, metacharacters are substituted by their corresponding sub-model. An example of model substitution is shown in figure 3.

Modularity is desirable for several reasons:

- sub-models can be modified depending on the application without having to retrain the main model;
- training is faster (smaller machines need to be trained);
- overfitting is avoided (fewer weights need to be estimated);
- less memory is required (fewer states and weights need to be stored);

When using the model, it is not necessary to actually build the graph of the composed automaton; beam search can be performed by querying the sub-models associated to metacharacters as needed.

3.3 Error measures, entropy

Consider a language source $\mathcal{L}$ that produces character strings w according to a probability distribution $P_{\mathcal{L}}(w)$, and a model $\mathcal{M}$ of $\mathcal{L}$, with underlying probability distribution $P_{\mathcal{M}}(w)$. The predictive power of model $\mathcal{M}$ is measured by its cross-entropy with respect to the actual distribution $P_{\mathcal{L}}(w)$, $H^{\mathcal{L}}_{\mathcal{M}} = -\sum_w P_{\mathcal{L}}(w) \log P_{\mathcal{M}}(w)$. The *intrinsic entropy* $H_{\mathcal{L}} = -\sum_w P_{\mathcal{L}}(w) \log P_{\mathcal{L}}(w)$ is a lower bound of $H^{\mathcal{L}}_{\mathcal{M}}$.

Upper bounds on the intrinsic entropy of English have ranged from 1.3 bits/character, obtained by Shannon from human subjects' guesses of the next letter given a prefix in literary text [10] to Cover and King's gambling estimate of 1.25 bits/character [11]. In table 1 we compare these results with average code lengths per character for well-known compression schemes [12] and the cross-entropy of the best published model, all measured

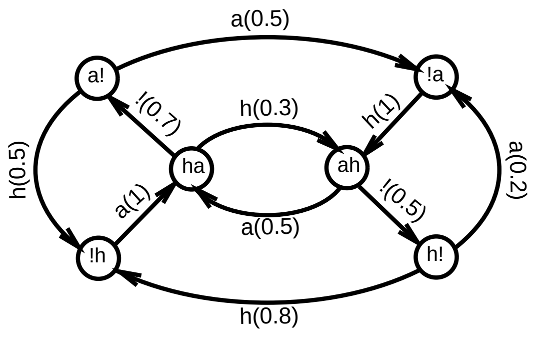

Figure 4: **Laughing finite state machine:** This finite state machine is a Markov model of order 2 (trigram model). We use it in our examples as language source.

on the same test set (Brown corpus) [5]. We use these values as points of reference for the performance of our language model.

If $\mathcal{M}$ is a stationary ergodic Markov process with states s which emits characters σ, the following expression for the per character entropy rate is valid [12]:

$$H^{\mathcal{L}}_{\mathcal{M}}\ (per\ \ char) = -\sum_{s,\sigma} P_{\mathcal{L}}(s,\sigma)\ \log P_{\mathcal{M}}(\sigma|s) \quad , \tag{4}$$

To measure the cross-entropy on a finite corpus of n characters, we use the following empirical estimate, where σ_t is the t^{th} character in the corpus:

$$\hat{H}^{\mathcal{L}}_{\mathcal{M}}\ (per\ \ char) = -(1/n)\sum_{t=1}^{n} \log P_{\mathcal{M}}(\sigma_t|s_{t-1}) \quad . \tag{5}$$

For a discussion on the validity of this estimate, see [12].

To train our language models, we follow the paradigm of *structural risk minimization* (SRM) [2, 3] with cross-validation. The available corpora are divided into 3 disjoint sets: a *training* set, a *validation* set and a *test* set. The model class is organized into **nested subsets** of models $\mathcal{E}_n$:

$$\mathcal{E}_1 \subset \mathcal{E}_2 \subset ... \subset \mathcal{E}_n \subset ... \tag{6}$$

For example, in the model class of Markov models, a structure is obtained by organizing the models in nested subsets of increasing Markov order. This guarantees that we vary the capacity monotonically, without having to actually calculate it:

$$c_1 \leq c_2 \leq ... \leq c_n \leq ... \tag{7}$$

Within each subset, the model with minimum cross-entropy on the training set is selected. Among the models thus retained, the model with smallest cross-entropy on the validation set is finally chosen. The quality of the chosen model is then evaluated by its test-set cross-entropy.

```
haha!ha!ha!ah!haha!ha!ha!haha!ha!ha!ah!hah!hahaha!aha!ha!ahah!ha!ah!hah
ha!ah!haha!ah!ah!ha!ah!ah!ha!ha!ha!ah!haha!ha!ha!haha!aha!ahaha!hah!ha!
!aha!ha!ah!ha!ah!hah!ha!ah!ha!ha!ahaha!ha!aha!ah!hahahah!hah!hah!aha!ah
h!ha!ahaha!ah!aha!aha!aha!ha!ha!ahah!ah!ah!ah!ha!aha!ha!aha!ha!ah!ah!ha
ha!ah!ha!ah!hah!hah!ha!ha!hah!ha!ha!ah!ha!ah!ha!aha!aha!ha!aha!haha!ha!
ha!haha!ha!aha!aha!ha!ah!hah!ahaha!ahaha!ah!haha!ha!ha!ah!ha!ha!aha!ha!
```

Figure 5: Extract from the 100,000 character long training sequence generated by the automaton of figure 4.

4 Variable Memory Length Markov Models

4.1 Principle of the method

Optimizing the length of the prefixes and capacity control

All Markov models determine the probability of the next character from a window of past characters or *prefix*. Standard Markov models of order $(n-1)$, or n-gram models, use a fixed size prefix of $(n-1)$ characters, while *variable memory length Markov models* (VLMM) use variable length prefixes. Training a Markov model of order $(n-1)$ simply amounts to computing the frequencies of all strings of length n. In the framework of *structural risk minimization*, varying n is a capacity control mechanism (see section 3.3). In the present section, we explain the principle of the method used to train VLMMs and its capacity control mechanism.

Assume that w is a prefix of length $(n-1)$ used to predict the next character σ', according to an estimate $\hat{P}(\sigma'|w)$ of $P(\sigma'|w)$. If $\hat{P}(\sigma'|\sigma w)$ differs significantly from $\hat{P}(\sigma'|w)$, adding one character σ in the past helps predicting better σ'. A possible decision criterion [1] is the Kullback-Leibler divergence between the next-character distributions for different prefixes, weighted by the prior distribution of σw:

$$\Delta H(\sigma w, w) = \hat{P}(\sigma w) \sum_{\sigma'} \hat{P}(\sigma'|\sigma w) \log \frac{\hat{P}(\sigma'|\sigma w)}{\hat{P}(\sigma'|w)} \tag{8}$$

If $\Delta H(\sigma w, w)$ exceeds a given threshold ε, then the longer prefix σw is accepted. Otherwise the prefix w is retained.

It is easily shown that the expression $\Delta H(\sigma w, w)$ is nothing but the training cross-entropy increase $\Delta H(\mathcal{M}', \mathcal{M})$ incurred by going from a model $\mathcal{M}'$ using a prefix $s = \sigma w$ to a simpler model $\mathcal{M}$ using $s = w$.

To perform *structural risk minimization*, one can use ε to induce a structure in the class $\mathcal{E}$ of VLMMs:

ε	# states	$H_{\mathcal{M}}$	$H_{\mathcal{M}}^{\mathcal{L}[train]}$	$H_{\mathcal{M}}^{\mathcal{L}[test]}$	Randomly generated text
0.5	1	1.574	1.574	1.576	`hha!!hhhhahah!!!aa!h!ahaaa!!ah!`
0.1	4	0.933	0.933	0.939	`ahaha!ha!ahaha!a!h!haha!aha!a!h`
0.05	6	0.732	0.752	0.756	`haha!aha!ha!ha!ha!ahahahahaha!h`
0.01	8	0.685	0.684	0.683	`ha!ha!aha!ha!hah!ha!aha!ha!ha!h`
0	10	0.666	0.668	0.666	`ha!ah!ha!ha!ha!haha!aha!ha!aha!`

Table 2: **Entropies of the models of figure 6.** To estimate $H_{\mathcal{M}}$, the intrinsic entropy of model $\mathcal{M}$, we generated a sequence of 200,000 characters from model $\mathcal{M}$. $H_{\mathcal{M}}$ assymptotically reaches $H_{\mathcal{L}}$ for $\varepsilon = 0$, in the limit of very large training sets. We use, a training set of size 100,000 and a test set of size 10,000, both generated independently by the reference model $\mathcal{L}$ of figure 4. The best model is obtained for $\varepsilon = 0$. It does not show any significant difference between $H_{\mathcal{M}}$, $H_{\mathcal{M}}^{\mathcal{L}[train]}$ and $H_{\mathcal{M}}^{\mathcal{L}[test]}$, therefore suggesting that the training set size is large enough to avoid overlearning.

$$\mathcal{E}_1 \subset \mathcal{E}_2 \subset \ldots \subset \mathcal{E}_l \subset \ldots \subset \mathcal{E} \tag{9}$$

$$\varepsilon_1 > \varepsilon_1 > \ldots > \varepsilon_l > \ldots \tag{10}$$

$$-\log \varepsilon_1 < -\log \varepsilon_1 < \ldots < -\log \varepsilon_l < \ldots \tag{11}$$

$$c_1 < c_2 < \ldots < c_l < \ldots \tag{12}$$

Varying ε is a capacity control mechanism which is superior to varying the order n of regular Markov models (see section 4.2). For a given number of parameters of the model, better performance is achieved; for a given performance, fewer parameters are needed.

Example

We illustrate the principle of the method in figures 4, 5 and 6.

In Figure 4, we show the automaton $\mathcal{L}$ which is used as a reference language to generate training and test data. State 0 (the one depicted with a square box) is the initial state of each PFSA. The dashed-line arcs correspond to the initial transient where not enough context is available, while the solid-line transitions correspond to the stationary part of the PFSA. We show in Figure 5 the beginning of the sequence of 100,000 characters generated by $\mathcal{L}$ that we used for training. In figure 6, we show the automata $\mathcal{M}$ obtained for various values of ε. In table 2, we show the corresponding entropies and a small sequence of character they produced.

The automaton which achieves best performance to predict new sequences produced by $\mathcal{L}$ is the largest one, that is the one with smallest ε, i.e. with largest capacity. This demonstrates that our training sequence is long enough to avoid the overlearning problem. For short training sequences, smaller models work better because it is not possible to obtain accurate probability estimates for unfrequent prefixes.

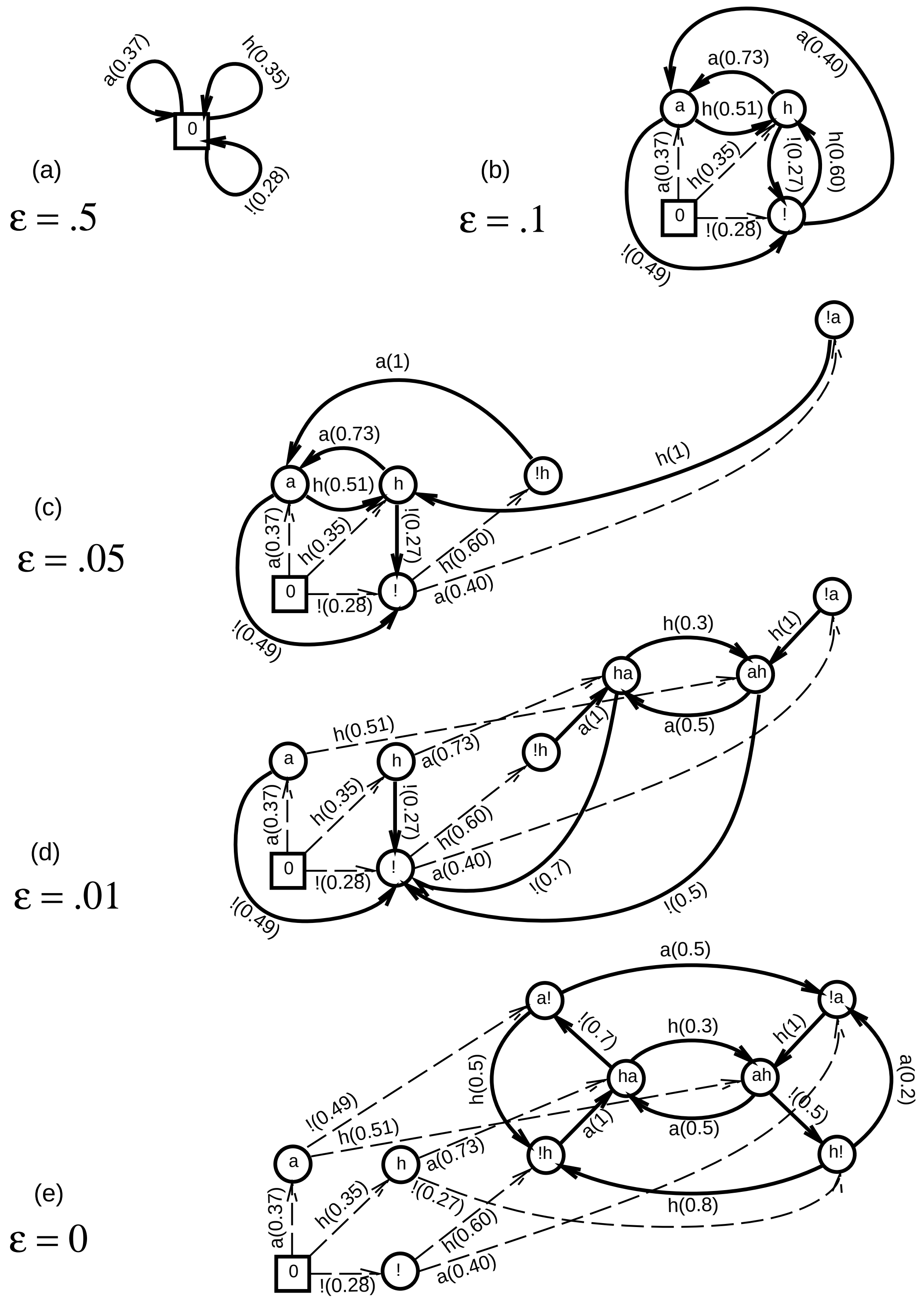

Figure 6: **VLMMs learned from data:** The machines are obtained by training on the data of figure 5, for various values of ε.

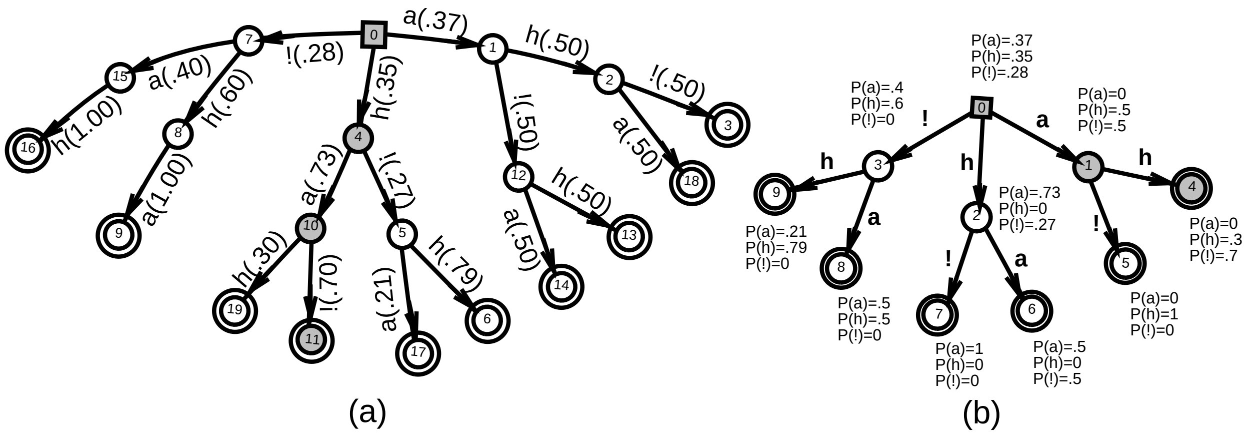

Figure 7: **Example of a prefix tree and its corresponding** *prediction suffix tree* **(PST):** a - Prefix tree of depth $d_p = 3$ built from the training data of figure 5. In the process of training, strings of length 3 are entered in the tree from the root to the leaves. New branches are grown when needed. The word "ha!" is highlighted. b - PST of depth $d_s = 2$ is built from the prefix tree with $\varepsilon_s = 0$. The prefix "ha" is highlighted. It is entered backwards in the PST. The probabilities shown at node 4 of the PST correspond to the prefix tree probabilities of figure 7-a: $P(11, !|10) = 0.7$, $P(11, h|10) = 0.3$ and $P(11, a|10) = 0$.

A few words about training

In this section we briefly outline the training algorithm. Further details can be found in [1].

The transition probabilities of the VLMM $P_{\mathcal{M}}(\sigma|s)$ and the decision criterion $\Delta H(\sigma w, w)$ are derived from estimates of $P(\sigma_n|\sigma_1\sigma_2...\sigma_{n-1})$ and $P(\sigma_1\sigma_2...\sigma_n)$ calculated for various values of n. The simplest such estimates are given by:

$$\hat{P}(\sigma_n|\sigma_1\sigma_2...\sigma_{n-1}) = \frac{\nu(\sigma_1\sigma_2...\sigma_n)}{\nu(\sigma_1\sigma_2...\sigma_{n-1})} \tag{13}$$

and

$$\hat{P}(\sigma_1\sigma_2...\sigma_n) = \frac{\nu(\sigma_1\sigma_2...\sigma_n)}{\nu_0} \tag{14}$$

where $\nu(\sigma_1\sigma_2...\sigma_n)$ is the number of times string $\sigma_1\sigma_2...\sigma_n$ appears in the training data and ν_0 is the total length of the training sequence.

To count strings of length n ($n = 1, 2, ...d_p$), we grow a so-called *prefix* tree of depth d_p, similar to a lexicon tree (or *trie*). A example of such a tree is shown in figure 7-a.

To build the tree, a window of fixed length d_p is slid along the text of the training corpus. Strings $\sigma_1\sigma_2, ...\sigma_{d_p}$ appearing in the window are added to the tree, from the root to the leaves. Every time a given branch is attained, by entering a given string $\sigma_1\sigma_2...\sigma_n$, the counter $\nu(\sigma_1\sigma_2...\sigma_n)$ associated to that branch is incremented.

The memory limitations of the computer are eventually reached when training on large corpora of natural language. This imposes to the tree a maximum number of

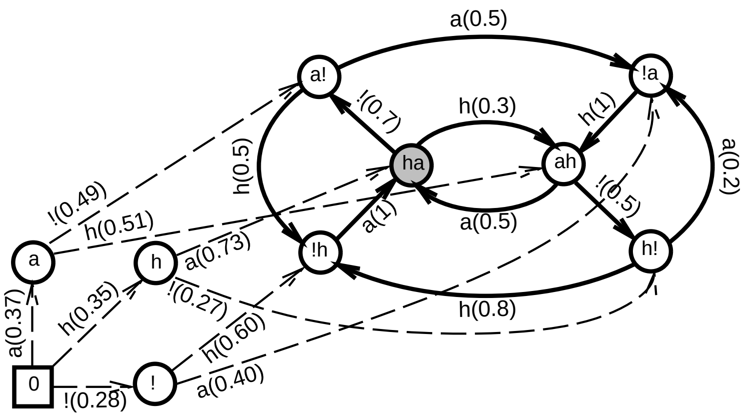

Figure 8: **Example of VLMM PFSA:** This PFSA is obtained from the PST of figure 7-b. It is identical to the original finite state machine of figure 4, except for the transient states. These states are useful when stationarity cannot be assumed.

nodes. When that number is reached, the branches which have been least visited need to be pruned. Branch i will be pruned if:

$$\nu_i/\nu_0 \leq \varepsilon_p \quad , \tag{15}$$

where ν_i is the counter of branch i, ν_0 is the root counter and ε_p is a threshold value.

The prefix tree contains all the information necessary to build a VLMM. However, the VLMM training is facilitated by building first an intermediate structure called a *prediction suffix tree* (PST) [1]. In a PST (figure 7-b), strings are entered in reverse order. But one should emphasize that a PST is not a prefix tree which would be trained with a text read backwards. No weights are associated to the branches of the tree. The node reached by accepting string $\sigma_{n-1}\sigma_{n-2}...\sigma_1$ is associated with the prefix tree probability $\hat{P}(\sigma_n|\sigma_1, \sigma_2...\sigma_{n-1})$. Thus for a string of any given length $\sigma_{-\infty}...\sigma_0\sigma_1\sigma_2...\sigma_{n-1}$, it is possible to walk "backwards" through the suffix tree to get its longest suffix $\mathit{Suffix}(sigma_{-\infty}...\sigma_0\sigma_1\sigma_2...\sigma_{n-1}) = \sigma_1\sigma_2...\sigma_{n-1}$ which provides the best available probability estimates for the next character σ_n.

In our example of figure 7-b, assume that we want to predict probabilities for the character which follows the string "ah!haha". The highlighted path is the longuest suffix "ha" of "ah!haha" found by the tree. It corresponds to the highlighted path in figure 7-a which provides probabilities for the next character.

The following algorithm [1] is used to train the PST:

- Initialize the PST and the candidate strings W: the PST is a single root node and $W = \{\sigma | \sigma \in \Sigma \text{ and } \hat{P}(\sigma) > \varepsilon_s\}$.
- While $W \neq \emptyset$, do:
  1. Pick any $w \in W$ and remove it from W.
  2. If $\Delta H(w, \mathit{suffix}(w)) \geq \varepsilon_s$ then add w to the PST by growing all the necessary nodes.
  3. If $|w| < d_s$ then, for every $\sigma \in \Sigma$, if $\hat{P}(\sigma w) > \varepsilon_p$ add σw to W.

Corpus	number of characters
AP news (training set)	215,320,418
Brown (test set)	6,044,101
AP news (train0, subset of training)	771,529
Alice in Wonderland (valid1)	138,463
Email (valid2)	182,762
Unix man1 (valid3)	689,702
Unix man3 (valid4)	578,027
Tex files (valid5)	389,590

Table 3: **Text corpora used for training and testing.** The Associated Press (AP) news corpus is used for training and the Brown corpus for testing. A subset of the AP corpus (train0) is used to estimate the performance on the training set. Sets valid1 through valid5 are used as validation sets.

where, for string $w = \sigma_1\sigma_2, ...\sigma_n$, $\mathit{suffix}(w) = \sigma_2, ...\sigma_n$, $|w|$ is the length of w, $d_s \leq d_p - 1$ is the PST depth, d_p is the prefix tree depth, ε_s is PST pruning threshold, ε_p is prefix tree pruning threshold and Σ is the alphabet.

Notice that the tree is grown from the root to the leaves. If a string w does not meet the criterion $\Delta H(w, \mathit{suffix}(w)) \geq \varepsilon_s$, it is not definitively ruled out, since its descendents σw are added to W in step 3. With this precaution, we keep a provision for future descendents of w which might meet the selection criterion.

Although it is possible to emulate a VLMM with a PST, it is important in practice to actually build the corresponding PFSA to gain processing speed. In a PFSA, the longest matching suffixes are precomputed into the states, whereas in a PST the longest suffixes must be dynamically determined.

In figure 8, we show a VLMM built from the PST of figure 7-b. Each node in the VLMM corresponds to a node in the PST. States in the PFSA are labeled by a string, or *prefix*, which is read backwards, from the corresponding PST node, to the root. For example, the shaded state in figure 8 corresponds to the highlighted path in figure 7-b. An outgoing arc from a state w which emits a character σ' with probability $\hat{P}(\sigma'|w)$ ends up in a state labeled by the longuest suffix of $w\sigma'$ found in the PST (e.g. $\mathit{Suffix}(hah) = ah$).

If the VLMM has a state w which emits a character σ' with probability zero, we do not build the connection from state w to the state labeled by the longuest suffix of $w\sigma'$ found in the PST. When matching against an input text, if the character σ' is presented to the machine when it is in state w, we return to the initial state, loose all the past context and predict σ' with the prior probability $\hat{P}(\sigma')$. Better backoff techniques may be worth exploring in this situation [13, 14].

4.2 Experimental results

In this section, we present a detailed study of the training of the main VLMM module in our system.

Number of characters	1	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	$2\ 10^8$
Number of nodes	7	58	494	3432	21082	102896	319215	139127	198653	195890

Table 4: **Prefix trees trained on the AP news corpus** for various sizes of the training set (in number of characters), with $d_p = 6$ and $\varepsilon_p = 10^{-6}$.

The corpora used for training and testing are listed in table 3. They have been chosen to illustrate a wide range of difficulties.

The data is preprocessed to substitute some strings by metacharacters:
. = { . ! ? },
, = { , ; : },
(= { ({ [},
) = {) }] },
’ = { “ ” ‘ ’ },
N = strings of contiguous characters containing at least one number,
M = strings of contiguous characters containing symbols (not from the above sets), but no numbers,
C = proper nouns.

After substitution, the remaining uppercase characters are turned to lowercase. The resulting alphabet has 37 characters, including space: {a b c d e f g h i j k l m n o p q r s t u v w x y z “space” - I **. , ( ) ’ N M C**}.

Leaning curves

A prefix tree of depth $d_p = 6$ was trained on the entire training set and a pruning threshold $\varepsilon_p = 10^{-6}$. Results are summarized in table 4.

For $\varepsilon_s = \varepsilon_s^{min} = 0$ and $d_s = d_s^{max} = 5$, we varied the size of the training set to obtain the learning curves shown in figure 9. Model performance is given by cross-entropy in bits/character. The intrinsic entropy of the automaton is approximated by the cross-entropy on a text of 200,000 characters generated at random by the automaton itself, let to evolve according to its own dynamics. “Validation” refers to the average cross-entropy on the validation sets (see table 3) and the grey shading indicates the standard deviation.

Our estimate of the intrinsic entropy starts at zero and increases rapidly to reach a maximum around 1000 characters then decreases again. Two factors are competing: the dominant effect in the first part of the curve is an increase in entropy due to an increase in the number of states of the machine; in the second part of the curve, the entropy decreases because better estimates of the PFSA weights are obtained. The cross-entropy of the validation set begins at infinity and decreases rapidly. It starts leveling at 10^8 characters, suggesting that training on more data will not improve the model substantially, at least for the given prefix tree depth.

In Figure 9, we also show the corresponding evolution of the size of the PFSA. It stops

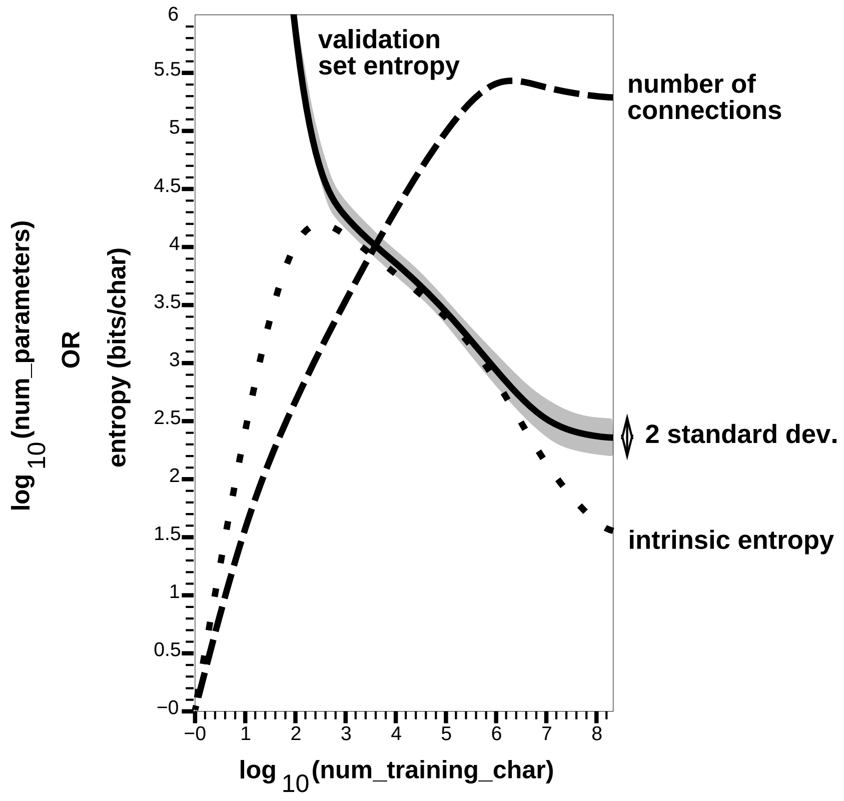

Figure 9: **Variations of the entropy as a function of the number of training examples:** The VLMMs are trained on the AP news data with an alphabet of 37 characters and parameters $d_p = 6$, $\varepsilon_p = 10^{-6}$, $d_s = d_s^{max} = 5$ and $\varepsilon_s = \varepsilon_s^{min} = 0$. Training set and validation sets are those of table 3. The size of the machine is shown on the same scale, but in different units.

increasing around 10^6 characters because the maximum number of nodes in the prefix tree is reached and the pruning algorithm starts eliminating nodes. This does not seem to affect immediately the test cross-entropy which continues decreasing steadily until 10^7 characters. Therefore, we are confident that the prefix tree pruning, necessary because of memory limitations, is not the dominant capacity control mechanism.

Model selection by varying ε_s

In figure 10-b and 10-d, we show the results obtained by varying ε_s. The depth of the suffix tree is set to its maximum $d_s = d_p - 1 = 5$. "Train0" refers to a fixed size subset of the training set (see table 3). As previously, "validation" refers to the average cross-entropy on the validation sets (see table 3) and the grey shading indicates the standard deviation.

The validation curve does not give a clear indication of overlearning. It starts leveling for a value of ε_s around 10^{-5}. At $\varepsilon_s = 10^{-5}$, the cross-entropy on the validation set is $H_{\mathcal{M}}^{valid} = 2.3$. The number of states of the PFSA is 14149 and the number of connections 76907.

At the optimum of the cross-entropy on the validation set (i. e. for $\varepsilon_s = 10^{-5}$), the cross-entropy on the test set (Brown corpus) is:

$$H_{\mathcal{M}}^{test} = 2.11 \; bits/character.$$

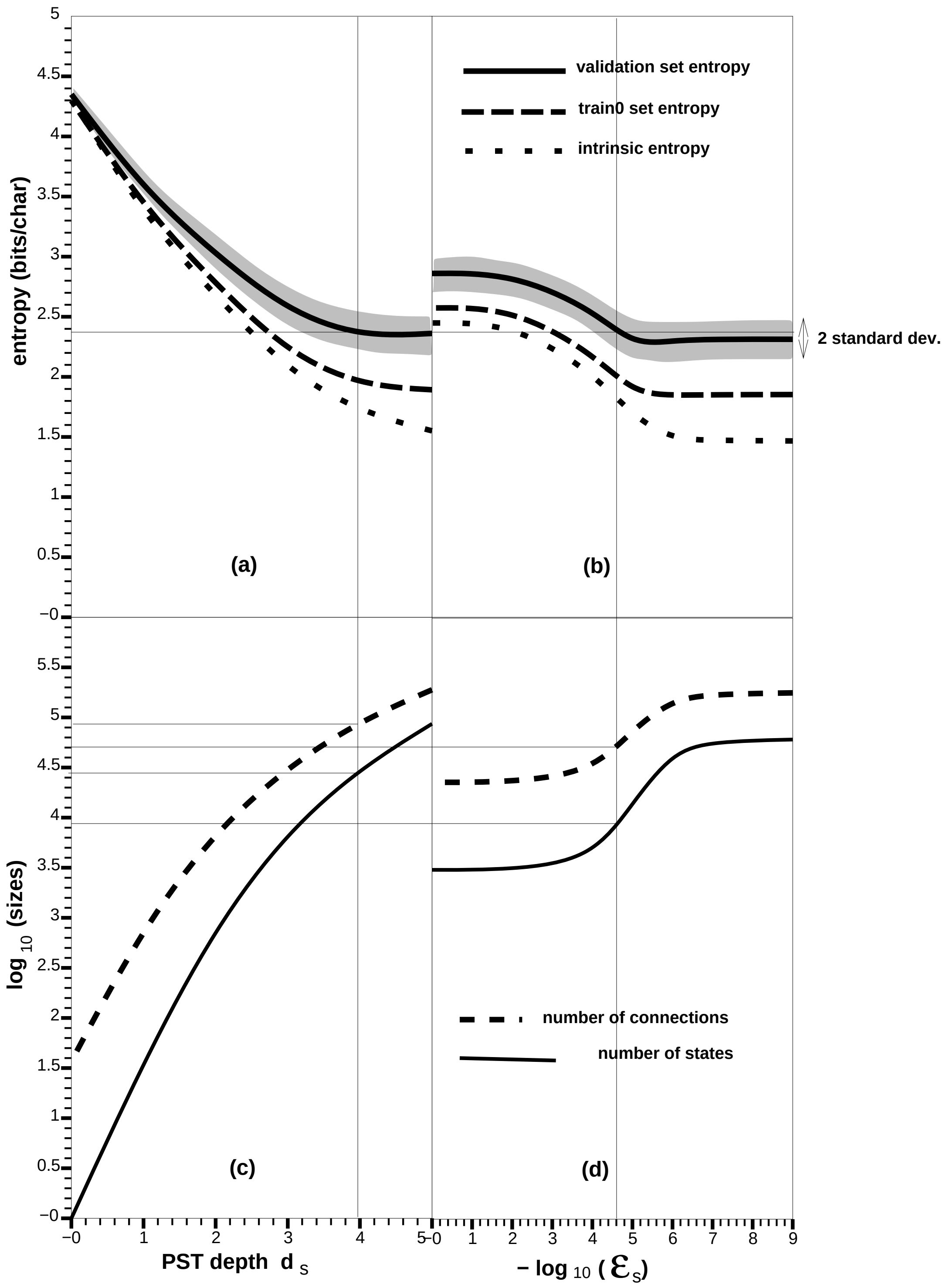

Figure 10: **Comparison of two capacity control methods:** The prefix tree is trained on all the AP news data with an alphabet of 37 characters and parameters $d_p = 6$ and $\varepsilon_p = 10^{-6}$. Either ε_s or d_s are allowed to vary. a - Entropy *vs* d_s for $\varepsilon_s = \varepsilon_s^{min} = 0$. b - Entropy *vs* ε_s for $d_s = d_s^{max} = 5$. c - Machine sizes *vs* d_s for $\varepsilon_s = \varepsilon_s^{min} = 0$. d - Machine sizes *vs* ε_s for $d_s = d_s^{max} = 5$.

```
liferator member of flight since N. a report the managical including from
C all N months after dispute. C and declaracter leaders first to do a lot
of though a ground out and C C pairs due to each planner of the lux said
the C nailed by the defender begin about in N. the spokesman standards of
the arms responded victory the side honored by the accustomers was arrest
two mentalisting the romatory accustomers of ethnic C C. the procedure'.
```

Figure 11: **Text generated by a VLMM** trained on the AP news data with parameters $d_p = 6$, $\varepsilon_p = 10^{-6}$, $d_s = 5$ and $\varepsilon_s = 10^{-5}$.

In figure 11, we show a text that was generated at random with the machine just described. Although the text generated is obviously not grammatical English, most words are actual English words. Some invented words are quite amusing:

```
liferator
managical
declaracter
mentalisting
accustomers
```

If somebody reads the text of figure 11 aloud when you are not quite paying attention, it really sounds like radio news!

Model selection by varying d_s

We performed a control experiment by fixing ε_s and allowing d_s to vary. Varying d_s is the classical means of regularization used for n-gram models. The results of varying d_s for $\varepsilon_s = 0$ are shown in figure 10-a. The validation curve starts leveling for depth $d_s = 4$. At this point, the validation performance is approximately $H_{\mathcal{M}}^{valid} = 2.4$. The number of cells of the associated PFSA is 31070 and the number of connections 92427 (Figure 10-c).

The same value of $H_{\mathcal{M}}^{valid} = 2.4$ is obtained for $d_s = 4$ in figure 10-a and $\varepsilon_s = 10^{-4.6}$, in figure 10-b. The corresponding number of states and number of connections are respectively: $10^{4.45}$ and $10^{4.95}$ in figure 10-c and $10^{3.95}$ and $10^{4.70}$ in figure 10-d. Therefore, for the same performance of $H_{\mathcal{M}}^{valid} = 2.4$, the machine obtained by using ε_s as capacity control criterion has 3 times less states and almost 2 times less connections than the machine obtained by using d_s as capacity control criterion.

The difference between using d_s and ε_s is maybe not as large as one could have had expected. At first glance, it seems that n-gram models are difficult to use for large values of n because the number of distinct strings grows exponentially with n. However, the number of distinct strings whose corpus frequency exceeds ε_p (the pruning threshold of the prefix tree) is manageable even without limiting n. The principal capacity control criterion is thus ε_p.

Metacharacter (**M**)	Corresponding characters (x)	Posterior proba ($\times 10^{-2}$) $\hat{P}(x\|M)$	Test set prior proba $\hat{P}(\mathbf{M})$	Test set cross-entropy (bits/char.) $\hat{H}(\mathbf{M})$
.	.	95.07	0.010	0.32
	!	3.68		
	?	1.25		
, (coma)	,	92.56	0.010	0.45
	;	4.37		
	:	3.07		
(	(	99.26	0.0004	0.06
	{	0.66		
	[	0.08		
)	)	99.28	0.0004	0.06
	}	0.64		
	]	0.08		
' (quote)	'	30.95	0.005	1.96
	"	30.95		
	`	19.05		
	``	19.05		
$\sum_{\mathbf{M}} \hat{P}(\mathbf{M})\hat{H}(\mathbf{M})$				0.017

Table 5: **Parameters and performance of the punctuation sub-models.**

Metacharacter (**M**)	Test set prior proba $\hat{P}(\mathbf{M})$	Test set cross-entropy (bits/char.) $\hat{H}(\mathbf{M})$
N	0.0016	4.01
M	0.00006	5.22
C	0.008	2.09

Table 6: **Parameters and performance of the numbers (N), symbols (M) and proper noun (C) sub-models.**

5 Training the sub-models

For each metacharacter of the main model, we trained a sub-model.

The punctuation sub-models are zero order Markov models (or uni-gram models) such as the one in figure 8-a. Their parameters and performances are reported in table 5.

The number and symbol models are first order Markov models (figures 12-a and 12-b). Their performances are reported in table 6.

The contribution to the overall entropy of a sub-model corresponding to a metacharacter **M** is:

$$\hat{P}(\mathbf{M})\hat{H}(\mathbf{M}),$$

where the cross-entropy $\hat{H}(\mathbf{M})$ is calculated on the test data restricted to the alphabet of the sub-model, and $\hat{P}(\mathbf{M})$ is the frequency of metacharacter **M** in that same test set. The

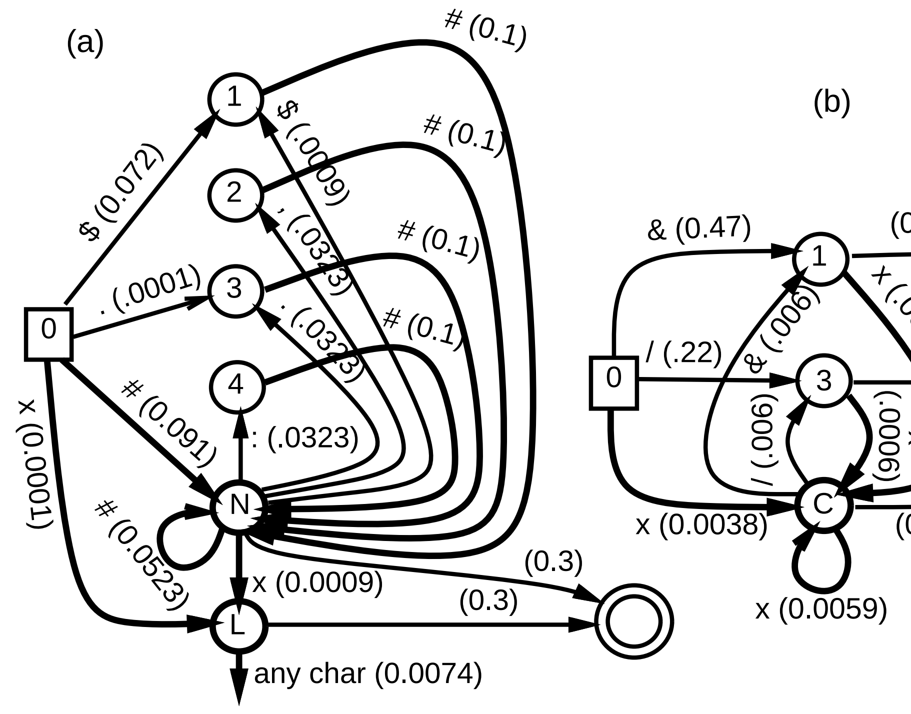

Figure 12: **PFSA model of numbers and symbols.** a - Number model: character "#" represents any numbers, and N represents all the states associated; letter "x" represents any letter and L represents all the states associated. b - Symbol model: letter "x" represents numbers, letters or symbols other than $\{\&, /\}$ and C represents all the states associated.

contributions of all the modules are summarized in table 7. For the capitalization model, which we have not yet implemented, we take the cross-entropy of the capitalization model of Brown et al [15].

Thus, we estimate the overall entropy of our models to be:

$$H_{\mathcal{M}}^{test} = 2.19 \ bits/character.$$

for a total of $\simeq 160,000$ parameters (weights of connections between states).

We performed a control experiment by training a VLMM on raw text. The cross-entropy of the model thus obtained is 2.32 bits/character. Therefore, our modular approach provides increased flexibility without sacrificing performance.

6 Conclusion and future work

Word trigram models have been for many years the leading method for designing efficient language models. In this paper, we demonstrate that VLMMs built at the character level are a viable alternative. They allow building finite state automata of rather modest sizes, compared to word trigram models, and have competitive performance in terms of entropy on a reference corpus, the Brown corpus. The use of structural risk minimization allows us to control the tradeoffs among memory, speed and accuracy by varying a single continuous parameter. Finite state automata composition allows us to reconfigure flexibly the language model according to the needs of different applications, without having to

Model	Contribution to the test set cross-entropy (bits/char.)	number of states	number of connections
Main model	2.11	14526	77570
Punctuation	0.017	5	16
N	0.006	96	8685
M	0.0003	86	7225
C	0.017	13043	65560
Capitalization	0.04	-	-
Total	2.19	27756	159056

Table 7: **Summary of the contributions to the test cross-entropy of all the modules.**

retrain the core of the language model. Our design is targeted towards on-line handwriting recognition. Combining the model with neural-network handwriting recognizers [16, 17] is under way. It is possible that the modeling techniques described here may also be relevant to speech recognition, using phonetic units instead of characters.

Acknowledgments

We gratefully acknowledge discussions with our colleagues at AT&T, Y. Bengio, Y. Le Cun, C. Nohl, S. Solla, E. Pednault and Y. Freud and with A. Stolke and D. Jurafsky of ICSI and Y. Singer and D. Ron of the Hebrew University of Jurusalem.

References

[1] D. Ron, S. Singer, and N. Tishby. The power of amnesia. In J. Cowan et al, editor, *Advances in neural information processing systems*, volume 6. Morgan Kauffmann, 1994.

[2] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

[3] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S.A. Solla. Structural risk minimization for character recognition. In J. E. Moody et al., editor, *Advances in neural information processing systems*, volume 4, pages 471–479. Morgan Kaufmann, 1992.

[4] F. Pereira, M. Riley, and R. Sproat. Weighted rational transductions and their application to human language processing. In *ARPA Natural Language Processing workshop*, 1994.

[5] H. Kucera and W. Francis. *Computational analysis of present day American English.* Brown University Press, Providence, RI, 1967.

[6] H. Ney. Stochastic grammars and pattern recognition. In P. Laplace and R. De Mori, editors, *Speech recognition and understanding, recent advances*, volume F75 of *NATO*

ASI Series, Berlin, Heidelberg, 1992. Springer Verlag.

[7] S. Furui and M. M. Sondhi, editors. *Advances in speech signal processing.* Marcel Dekker, New York, NY, USA, 1992.

[8] F. Jelinek. Up from trigrams! In *Eurospeech'91*, 1991.

[9] I. Guyon and P.S.P. Wang, editors. *Advances in Pattern Recognition Systems using Neural Network Technologies*, volume 7 (4). World Scientific, Singapore, August 1993 (appeared also as a book chapter in Series in Machine Perception and Artificial Intelligence, Vol. 7, World Scientific, Singapore, 1994).

[10] C. E. Shannon. Prediction and entropy of printed english. *The Bell system technical journal*, January 1951.

[11] T. M. Cover and R. C. King. A convergent gambling estimate of the entropy of English. *IEEE Trans. Information Theory*, IT-24(4):413–421, July 1978.

[12] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley series in telecommunications. Wiley, New York, 1991.

[13] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speeech and Signal Processing*, 35(3):400–401, 1987.

[14] F. Jelinek. Self-organized language modeling for speech recognition. In Alex Weibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*, chapter 8, pages 450–506. Morgan Kaufmann, San Mateo, California, 1990.

[15] V. J. Brown, P. F. an Della Pietra, R. L. Mercer, S. A. Della Pietra, and J. C. Lai. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18 (1), 1992.

[16] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using neural networks and hidden mar kov models. In *ICASSP 1994*, Adelaide, Australia, 1994.

[17] Y. Bengio, Y. Le Cun, and D. Henderson. Globally trained handwritten word recognition using spatial representation, space displacement neural networks and hidden markov models. In *Advances in Neural Information Processing Systems 6.* Morgan Kauffmann, 1994.