

Improving the Multigram Algorithm by using Lattices as Input

Joris Driesen, Hugo Van hamme

Dept. ESAT, K.U.Leuven, Leuven, Belgium

joris.driesen@esat.kuleuven.be, hugo.vanhamme@esat.kuleuven.be

Abstract

The multigram algorithm is a statistical technique that can be used for extracting recurring patterns from a sequential input. When provided with a symbol sequence representing a speech signal, it is able to extract word-like patterns from it, despite the large amount of subsequences that can represent a single word. For this, it uses statistical information derived from the entire input. However, due to the abstraction of speech to symbols, much of the information originally present in the signal is no longer available to the algorithm.

In this paper we propose a way of using a richer abstraction of the signal in the form of a lattice. Furthermore, a way of grounding recurring patterns to concepts in other modalities will be presented. Finally, the information learned by the algorithm using both kinds of input is tested in a recognition experiment. This will show that the use of lattices leads to a significant improvement in terms of recognition rate.

Index Terms: Machine Learning, Language Acquisition, Pattern Discovery, Lexical Models.

1. Introduction

Humans are all born without any knowledge about language. In the first years of their life, they learn their mother tongue without the use of any prior lexical knowledge, in a weakly supervised way compared to HMM training. Presumably, they do this by gradually remembering meaningful segments of acoustic information and linking them to concepts they observe in other modalities than speech.

When implementing such a mechanism in automatic systems, the first challenge is the pattern discovery step. This is not an easy task, since speech signals often have random variations like background noise, or variations that are hard to model, like speaker dependencies. Due to these difficulties, pattern discovery techniques that perform very well on cleaner signals, like music ([1]), are not suitable for speech. The multigram algorithm [2] is a classic pattern discovery algorithm that seems to meet all the requirements to a satisfactory degree. However, it can only detect meaningful patterns in a symbolic representation of the speech signal, in which the amount of variability is reduced, not directly in the speech signal itself. Even in this stable symbol sequence every single word is still represented by a large amount of possible symbol strings. The multigram algorithm has to determine which subsequences are generated by the same underlying word and which are not. This is done by making use of statistical information in the entire input. However, since the input sequence contains only a part of the statistical information present in the original speech signal, those decision making capabilities are not optimal.

The organisation of this paper is as follows: in section 2, a review of the multigram algorithm will be given. In section 3 we will introduce a way to cope with uncertainty in the input.

In section 4 we will propose a way of relating discovered patterns in the sequence to information present in other modalities than speech, a process that is also referred to as grounding. Finally, in section 5, we will describe the results of a recognition experiment using an automatically acquired lexicon.

2. The Multigram Algorithm

The multigram algorithm is based on the assumption that the input sequence is the product of a probabilistic generative framework, as depicted in figure 1. At the highest level, there is a sequence Z of word-like concepts z_i , also known as *multigrams*,

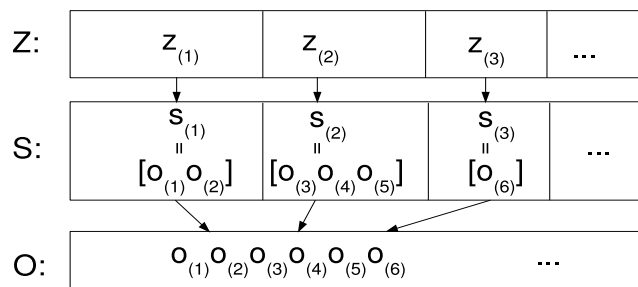


Figure 1: *The generative framework. Each element of sequence Z yields a symbol string in S . Together, they make up the observed input O .*

all of which are elements of a finite set $\mathcal{Z} = \{z_1, z_2, \dots, z_m\}$. Each of these multigrams generates symbol strings (of variable length) according to some random process. In this paper, this process will be described by an HMM. The sequence of symbol strings, each of which is generated by an element of Z , will be referred to as S . The input sequence, O , is then formed by removing the pattern boundaries from S . The goal of the multigram algorithm is to determine the set of multigrams \mathcal{Z} that maximizes the likelihood of the observation. Formally:

$$\mathcal{Z}^* = \arg \max_{\mathcal{Z}} \mathcal{L}(O|\mathcal{Z})\mathcal{L}(\mathcal{Z}) \quad (1)$$

in which $\mathcal{L}()$ is the likelihood.

Assuming that the occurrence of z_i in Z does not depend on the context, but only on a prior probability $p(z_i)$, the first part of this equation can be rewritten as:

$$\mathcal{L}(O|\mathcal{Z}) = \sum_{(S,Z)} \mathcal{L}(O, S, Z|\mathcal{Z}) = \sum_{(S,Z)} \prod_{t=1}^{c(S,Z)} p(s_{(t)}|z_{i_t})p(z_{i_t}) \quad (2)$$

Here, t is the segment index, $c(S, Z)$ is the number of segments in S (fig. 1) and i_t is the multigram index of the t 'th segment. We maximize this likelihood by making use of the help function

(see [3]):

$$Q(k, k+1) = \sum_{(S,Z)} p^{(k)}(S, Z|O, \mathcal{Z}) \cdot \log \mathcal{L}^{(k+1)}(O, S, Z|\mathcal{Z})$$

in which k is the index of the current iteration. It is straightforward to see that this leads to two separate cost functions that can be iteratively optimized: one over $p(z_i)$, the other over $p(s_{(t)}|z_i)$. These cost functions are

$$\sum_{i=1}^m \sum_{(S,Z)} c(z_i|S, Z, \mathcal{Z}) p^{(k)}(S, Z|O, \mathcal{Z}) \log p^{(k+1)}(z_i) \quad (3)$$

and

$$\sum_{(S,Z)} p^{(k)}(S, Z|O, \mathcal{Z}) \sum_{t=1}^{c(S,Z)} \log p^{(k+1)}(s_{(t)}|z_{it}) \quad (4)$$

Here $c(z_i|S, Z)$ is the number of times the multigram z_i occurs in Z . Optimizing equation 3 yields an iterative update formula for $p(z_i)$:

$$p^{(k+1)}(z_i) = \frac{\sum_{(S,Z)} c(z_i|S, Z) p^{(k)}(S, Z|O, \mathcal{Z})}{\sum_j \sum_{(S,Z)} c(z_j|S, Z) p^{(k)}(S, Z|O, \mathcal{Z})} \approx \frac{c(z_i|S^*, Z^*)}{c(S^*, Z^*)} \quad (5)$$

in which S^* and Z^* are the most likely S and Z , given the observation. Optimizing equation 4 is tantamount to a training pass over the observed input sequence of the generative models associated with each multigram.

The second part of equation 1, $\mathcal{L}(\mathcal{Z})$, also needs to be optimized in every iteration. Since the exact value of this part cannot be calculated, this is not a trivial task. One possible way would be to find out how many patterns are present in the input and then determine the likelihood of the input, for every possible subset of the entire pattern set with the correct number of elements. The set that yields the highest likelihood is then the optimal one. Needless to say, this would be a process of prohibitive computational complexity. In most cases, the new pattern set is heuristically determined by removing all patterns whose probabilities $p(z_i)$ fall beneath a certain threshold.

3. Using Lattice Input

At any point in time the acoustic signal is only represented by a single symbol in the input. Often, it is better to have several alternative possibilities, accompanied by a confidence measure. Effectively, this means that the input of the algorithm should be a lattice. An example is shown in figure 2. We would like to

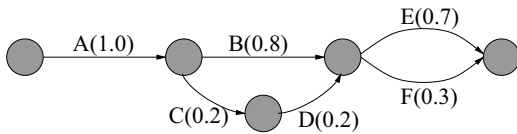


Figure 2: An example of a lattice. The symbols representing the speech signal, along with their probabilities, are on the arcs.

maximize the likelihood of the original acoustic waveform x . This waveform can be represented by a large number of symbol sequences, each of which can be seen as a path through the lattice. The chosen path then becomes another hidden variable of the model, which we will call P . We can then rewrite equation 2:

$$\begin{aligned} \mathcal{L}(x|\mathcal{Z}) &= \sum_{(S,Z,P)} \mathcal{L}(x, S, Z, P|\mathcal{Z}) \\ &= \sum_{(S,Z,P)} \mathcal{L}(x|S, Z, P, \mathcal{Z}) \mathcal{L}(S, Z, P|\mathcal{Z}) \\ &= \sum_P \mathcal{L}(x|P) \sum_{(S,Z)} \prod_{t=1}^{c(S,Z,P)} p(z_{it}) p(s_{(t)}|z_{it}) \end{aligned} \quad (6)$$

Using a similar reasoning as in section 2, we arrive at two cost functions:

$$\sum_{i=1}^m \sum_{(S,Z,P)} c(z_i|S, Z, P, \mathcal{Z}) \cdot p^{(k)}(S, Z, P|x, \mathcal{Z}) \log p^{(k+1)}(z_i) \quad (7)$$

and

$$\sum_{(S,Z,P)} \sum_{t=1}^{c(S,Z,P)} p^{(k)}(S, Z, P|x, \mathcal{Z}) \cdot \log p^{(k+1)}(s_{(t)}|z_{it}) \quad (8)$$

Maximizing for instance cost function 7 yields

$$\begin{aligned} p^{(k+1)}(z_i) &= \frac{\sum_{(S,Z,P)} c(z_i|S, Z, P, \mathcal{Z}) p^{(k)}(S, Z, P|x, \mathcal{Z})}{\sum_{j=1}^m \sum_{(S,Z,P)} c(z_j|S, Z, P, \mathcal{Z}) p^{(k)}(S, Z, P|x, \mathcal{Z})} \\ &\approx \frac{\sum_P c(z_i|S_P^*, Z_P^*, P) p(P|x, \mathcal{Z}) p^{(k)}(S^*, Z^*|P, x, \mathcal{Z})}{\sum_j \sum_P c(z_j|S_P^*, Z_P^*, P) p(P|x, \mathcal{Z}) p^{(k)}(S^*, Z^*|P, x, \mathcal{Z})} \\ &\approx \frac{\sum_P \delta_P \cdot c(z_i|S_P^*, Z_P^*, P) p(P|x, \mathcal{Z})}{\sum_j \sum_P \delta_P \cdot c(z_j|S_P^*, Z_P^*, P) p(P|x, \mathcal{Z})} \end{aligned} \quad (9)$$

with

$$\max_{(S,Z)} p^{(k)}(S, Z|P, x, \mathcal{Z}) = p^{(k)}(S^*, Z^*|P, x, \mathcal{Z}) \approx \delta_P$$

δ_P is assumed to equal either a constant independent of P , or zero. Zero occurs when the (z_i) -sequence cannot be aligned with P . Thus, δ_P can effectively be seen as a lexical constraint.

The remaining aligned paths through the symbol lattice can be represented as a new lattice in which every arc is a concatenation of arcs in the symbol lattice and corresponds to a single pattern z_i . The probability of such a pattern arc can be calculated:

$$p(n_k, n_{k+1}, \dots, n_{k+N}) = p(n_k, n_{k+1}) \cdot \frac{p(n_{k+1}, n_{k+2})}{\sum_i p(i, n_{k+1})} \cdot \dots \cdot \frac{p(n_{k+N-1}, n_{k+N})}{\sum_i p(i, n_{k+N-1})} \quad (10)$$

where the pattern arc starts at node n_k and ends at node n_{k+N} in the symbol lattice. $\sum_i p(i, n)$ is the sum of the probabilities of all symbol arcs arriving in node n and will be further referred to as its node weight. If we name a path through this pattern lattice Q , and an arc on this path a_Q , the numerator of equation 9 can be rewritten as

$$\begin{aligned} \sum_Q \sum_{a_Q} \delta_{i,a} p(Q) &= \sum_a \delta_{i,a} \sum_{Q_a} p(Q_a) \\ &= \sum_a \delta_{i,a} p(a) \left(\sum_{Q_{L,a}} p(Q_{L,a}) \right) \cdot \left(\sum_{Q_{R,a}} p(Q_{R,a}) \right) \\ &= \sum_a \delta_{i,a} p(a) p_L(a) p_R(a) \end{aligned} \quad (11)$$

In all probabilities, the condition on x and \mathcal{Z} has been left out for the sake of brevity. $\delta_{i,a}$ is 1 if arc a corresponds to z_i and 0 otherwise. $Q_{L,a}$ is a path from the start node of the pattern lattice to the start node of arc a , $Q_{R,a}$ a path starting from the end node of that arc, to the end node of the pattern lattice. Their respective summations $p_L(a)$ and $p_R(a)$ can be efficiently calculated using a forward-backward algorithm. Equation 9 can then finally be rewritten as

$$p^{(k+1)}(z_i) \approx \frac{\sum_a \delta_{i,a} p(a) p_L(a) p_R(a)}{\sum_a p(a) p_L(a) p_R(a)} \quad (12)$$

If we not only keep track of z_i on each of the arcs in the pattern lattice, but also of its symbol string $s_{(t)}$, i.e. the symbol arcs from which it was formed, we can maximize equation 8 in a similar way.

4. Grounding

Knowing the patterns that make up the acoustic signal is only the first step in language acquisition. In order to receive the information that a speaker is trying to convey, it is necessary to know what every spoken word or word group means. In order to learn this without the aid of prior knowledge, a form of associative learning can be applied, like a baby presumably does. For instance, if a baby is often presented a ball upon hearing the word “ball”, it will gradually learn its meaning. Of course, this requires the baby to visually detect the ball and pay attention to it, to separate it from all the other stimuli it receives. Since attention mechanisms and sensory detection of different stimuli are difficult problems in their own right, we will make abstraction of them here, and simply have multimodal information accompany each speech utterance in the form of a number of relevant so-called multimodal tags. There is no one-to-one link between the speech signal and the multimodal information, so any connection between the two must be probabilistic, which can be written either as $p(z_i | tag_k)$ or as $p(tag_k | z_i)$. The calculation of these measures can only be accomplished by making use of cooccurrences of patterns in the segmented input and tags. Since in every utterance there are many possible pattern-to-tag mappings, only one of which is correct, both of these measures are distorted by noise. However, since the average utterance contains more words than multimodal tags, because of filler words that have no clear ecological significance, $p(z_i | tag_k)$ is noisier than $p(tag_k | z_i)$. It is therefore more sensible to use the latter. Moreover, if the assumption is made that multimodal tags are generated by the multigrams in \mathcal{Z} , the application of the maximum likelihood criterion straightforwardly leads to simple formulas to calculate this probability, the derivation of which will not be given here due to lack of space. For

the single sequence input:

$$p(tag_k | z_i) = \frac{\sum_{l=1}^T \delta_{k,l} c(z_i | S, Z, \mathcal{Z})}{\sum_{l=1}^T \delta_{k,l} \sum_{j=1}^m c(z_j | S, Z, \mathcal{Z})} \quad (13)$$

and for the lattice input:

$$p(tag_k | z_i) = \frac{\sum_{l=1}^T \delta_{k,l} \sum_a \delta_{i,a} p(a) p_L(a) p_R(a)}{\sum_{l=1}^T \sum_a \delta_{i,a} p(a) p_L(a) p_R(a)} \quad (14)$$

in which T is the number of different tags and $\delta_{k,l}$ is equal to 1 if the utterance is accompanied by tag_k , 0 otherwise. The appropriate tag for every discovered pattern then becomes

$$tag_i = \arg \max_k p(tag_k | z_i) \quad (15)$$

5. Experiments

To test the multigram algorithm in a language learning context, a simple language acquisition agent was built, which is schematically shown in figure 3. The input consists of utter-

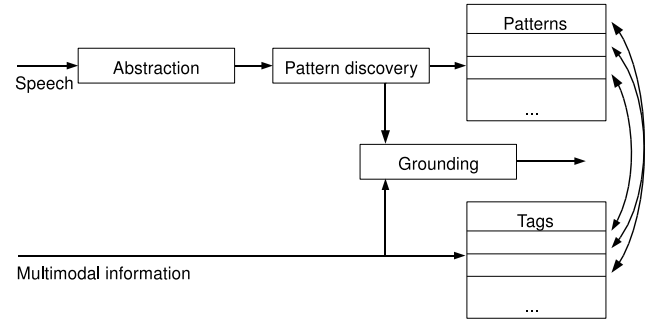


Figure 3: A schematic overview of a simple word learning agent

ances from the TIDIGITS database which contains a total number of eleven different words. Utterances that contain only a single word are left out, leaving a total of 6159 in the train set and 6214 in the test set. Each utterance is then augmented with an unordered set of multimodal tags, each of which corresponds with a single word in that utterance. Every utterance is abstracted to a single phone sequence by using a phone recognizer, yielding an average phone error rate of approximately 25%. Lattices on the same speech input are deduced by the first layer of the FLAVOR framework [4]¹. They are the same lattices that were used in [5]. When processing a corpus of utterances, equations 5, 12, 13 and 14 require summing over all utterances in their numerator and denominator.

5.1. Training

The pattern set is initialized by taking all subsequences present in all input sequences of the train set, up to a maximum length of five phones. The probability of each pattern is initialized as its relative number of occurrences. Only if this initial probability is above 0.001, it is added to the initial set. The generative model associated with each of these patterns is then initialized as follows: each phone is converted to an HMM-state in which the

¹The phone recognizer is used to abstract the speech signal into a symbol sequence. Since we want to concentrate on the lexical discovery problem, the choice of phones is for convenience of interpretation. Other abstractions could be envisaged (e.g. VQ or temporal decomposition)

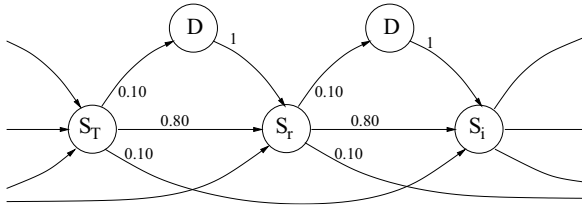


Figure 4: The model associated with the pattern “Tri”. The dummy-states are labelled with a ‘D’.

probability of emitting that phone is 0.5. The rest of the probability mass is evenly distributed over all other phones. From each state there is a transition with probability 0.80 to go to the next state. It is also possible to skip the next state with a probability of 0.10. Finally, it is possible with a probability of 0.10 to jump to a so-called dummy state before going to the next state. The emission distribution of the dummy states are initialized uniformly. Initializing the models like this not only accounts for substitutions but also single insertions and deletions. An example is shown in figure 4.

Next, a total of 10 training passes is performed on these newly generated HMM’s. For the single sequence input this means we will just perform 10 Viterbi passes over the inputs and update the HMM’s and the prior probabilities of the known patterns accordingly. For lattice input, a training pass entails the calculation of a pattern lattice in each utterance, and an update of the HMM’s and prior probabilities as explained above. In the end the set of multigrams is once more pruned with a probability threshold of 0.001, after which we can start from the beginning. This is repeated 4 times, after which recognition performance is unchanged. Grounding is performed at the same time. This means that in every iteration, we apply equation 13 for the single sequence input, and equation 14 for the lattice input. The end result is a set of trained HMM’s, all of which are statistically linked to a multimodal tag.

5.2. Testing

Finally, speech input from the test set is presented to the learning agent, without multimodal tags. The learning agent will make use of the information acquired in the training to generate the tag sequence it has understood from the input. With the single sequence input this is simply a Viterbi alignment of the input sequence in which every segmented pattern is substituted for its corresponding tag as determined in section 4. For the lattice input, we derive pattern lattices from the symbol lattices as explained in section 3 after which we seek the best path through them. This, too, provides us with a segmentation that can be converted to a tag sequence. By comparing this tag sequence to the oracle solution, which in this case is no longer an unordered bag, but a strictly ordered sequence, it is easy to determine the tag error rate as

$$TER = TDR + TIR + TSR = \frac{D + I + S}{T} \cdot 100\%$$

with D , I and S respectively the total number of deleted, inserted and substituted tags, and T the total number of all tags. The results of this experiment are given in table 1. It is clear that the use of lattices yields a substantial improvement over the single sequence input.

	sequences	lattices
substitutions	2.58	3.32
deletions	2.11	0.59
insertions	5.25	3.09
total	9.94	7.00

Table 1: The tag error rate, given in percents.

6. Conclusion and future work

In this paper we have presented an extension to the multigram algorithm, that allows it to utilize uncertainties in the input. We also described a way of mapping patterns discovered by this algorithm to information from other modalities which is symbolized by tags presented along with each speech utterance. This mechanism was cascaded with the multigram algorithm to form a rudimentary framework for modeling language acquisition. By training and testing this framework on the TIDIGITS database, we have shown that the use of lattices significantly improves its performance.

Future work will include an attempt at making an abstraction of the acoustic signal without resorting to an acoustic model or any other knowledge of prelexical level. Also, where the present paper focuses on the influence of lattices on recognition accuracy, we would also like to explore its influence on the noise robustness of the algorithm. The experiment presented here already showed that a symbol error rate of 25% can be dealt with quite easily, but these limits need to be explored further. Furthermore, the influence of the initialization parameters will be investigated. Finally, an analysis of several HMM initializations is in order. The present models, which account for single insertions and single deletions, are adequate for processing phone recognition results, but this needs to be verified for other kinds of input.

7. Acknowledgments

This research is part of the ACORNS project, funded by the European Commission under contract number FP6-034362.

8. References

- [1] J.-L. Hsu, C.-C. Liu, and L. Chen, “Discovering nontrivial repeating patterns in music data,” *IEEE Transactions on Multimedia*, vol. 3, no. 3, pp. 311–325, 2001.
- [2] S. Deligne and F. Bimbot, “Inference of variable-length linguistic and acoustic units by multigrams,” *Speech Communication*, vol. 23, pp. 223–241, 1997.
- [3] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum-likelihood from incomplete data via the em algorithm,” *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.
- [4] K. Demuyne, T. Laureys, D. Van Compernelle, and H. Van hamme, “Flavor: a flexible architecture for lvcsr,” in *Proc. Eurospeech*, pp. 1973–1976, 2003.
- [5] V. Stouten, K. Demuyne, and H. Van hamme, “Automatically learning the units of speech by non-negative matrix factorisation,” in *Proc. European Conference on Speech Communication and Technology*, pp. 1937–1940, 2007.