Detection of Phoneme Boundaries Using Spiking Neurons

Gábor Gosztolya and László Tóth

MTA-SZTE Research Group on Artificial Intelligence
of the Hungarian Academy of Sciences and University of Szeged
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
{ggabor, tothl}@inf.u-szeged.hu

Abstract. Automatic speech recognition (ASR) is an area where the task is to assign the correct phoneme or word sequence to an utterance. The idea behind the ASR segment-based approach is to treat one phoneme as a whole unit in every respect, in contrast with the frame-based approach where it is divided into equal-sized, smaller chunks. Doing this has many advantages, but also gives rise to some new problems. One of these is the detection of potential bounds between phones, which has an effect on both the recognition accuracy and the speed of the speech recognition system. In this paper we present three ways of boundary detection: first two simple algorithms are tested, then we will concentrate on our novel method which incorporates a spiking neuron. On examining the test results we find that the latter algorithm indeed proves successful: we were able to speed up the recognition process by 35.72% while also slightly improving the recognition performance.

1 Introduction

In the problem of Automatic Speech Recognition (ASR) we have to map the correct phoneme or word sequence to a given speech signal. Most methods for this task are based on the frame-based notion, which treats the speech signal as a series of independent, equal-sized small units called frames. These frames are classified as phonemes, and then joined to make whole words or sentences, from which the most probable one is chosen. This scheme is based on the concept of combining multiple, consecutive frames into one phoneme.

The segment-based approach is founded on different principles. Here all the frames which make up a phoneme are treated together as one unit, and naturally their classification is also done together. This way we can extract more information from the context, and this leads to more precise phoneme-level identification, but doing this is not without its drawbacks. One, for instance, is that both ends of the phoneme segment become variables of the search, which raises the computational needs of the recognition process to a quadratic running time instead of the former linear one. To reduce this CPU requirement we can filter the phoneme boundary hypotheses before evaluating them. It is easy to see that this restriction has to be carried out very carefully: if we do not allow the kind

L. Rutkowski et al. (Eds.): ICAISC 2008, LNAI 5097, pp. 782–793, 2008.

of bound where there is actually a boundary between two phonemes, we will lose information, and this will probably lead to a decrease in the recognition performance. On the other hand if we allow possible boundaries where there is no need for them, we can significantly and unnecessarily slow down our speech recognition system. Hence a tradeoff should be made.

Here we will focus on the task of constructing a method for detecting probable segments in order to speed up the recognition process. To do this, we will investigate one simple method as a baseline, construct three novel segmentation algorithms, and then vary their parameters in order to make them work efficiently. In the experiments we found that we were able to not only speed up the recognition process significantly, but also improve the recognition performance, which was an unexpected bonus.

The structure of our paper is as follows. First we define the speech recognition task in a segment-based approach. Next we discuss the issue of segmentation, outlining the problem and our solutions. Then we describe the test environment and the tests we made. Lastly we present our results and draw some conclusions.

2 The Speech Recognition Problem

In speech recognition problems we have a speech signal represented by a series of observations (frames) $A = a_1 \dots a_t$, and a set of possible phoneme sequences (words) which will be denoted by W. Our task is to find the word $\hat{w} \in W$ defined by

$$\hat{w} = arg \max_{w \in W} P(w|A), \tag{1}$$

which, using Bayes' theorem, is equivalent to the maximization problem

$$\hat{w} = arg \max_{w \in W} \frac{P(A|w) \cdot P(w)}{P(A)}. \tag{2}$$

Further, noting the fact that $P(A)$ is the same for all $w \in W$, we have that

$$\hat{w} = arg \max_{w \in W} P(A|w)P(w). \tag{3}$$

Speech recognition models can be divided into two types – the discriminative and generative ones – depending on whether they use Eq. (1) or Eq. (3). In the experiments we restricted our investigations to the generative approach [1]. Speech recognition models can also be divided into two types, namely a frame-based one and a segment-based one. Here we will focus on the latter type. Note too that the factors $P(A|w)$ and $P(w)$ can be treated separately. From now on we will focus on the former one, and take $P(w)$ (the *language model*) as given.

Now let us define the word w as a phoneme-sequence $o_1, \dots, o_n$. Next, let us divide A into non-overlapping segments $A_1, \dots, A_n$, each A_j belonging to the corresponding phoneme o_j. (As $A = a_1 \dots a_t$, we can define A_j as $a_{t_{j-1}} \dots a_{t_j}$

with $1 = t_0 < t_1 < \ldots < t_n = t$.) Then, making the common assumption that the phonemes are independent, we have

$$P(A|w) = \prod_{j=1}^{n} P(A_j|o_j). \tag{4}$$

In a segment-based model we calculate these $P(A_j|o_j)$ values by some machine-learning method using interval features (i.e. features like the mean, minimum or maximum, calculated on the whole $a_{t_{j-1}} \ldots a_{t_j}$ segment). In our system Artificial Neural Networks (ANNs) [2] are used, and their corresponding output serve as estimates for the $P(o_j|A_j)$ probability values after length normalization. From these, estimates of $P(A_j|o_j)$ can be readily obtained by a division by the priors.

2.1 Restricting the Space of Possible Segmentations

At this point we can assign a probability for a phoneme-sequence and segment-sequence pair, so in theory we can find the optimal (most probable) pairing. Unfortunately, it is practically impossible to examine all pairings, hence we have to introduce some further restrictions for technical reasons. Firstly, instead of referring to the segments as $A_1, \ldots, A_n$, we use their boundary elements, i.e. $I = [t_0, t_1, \ldots, t_n]$. This I will from now on be called a *segmentation*, and we can refer to an A_j segment with its starting and ending segmentation bounds as $[t_{j-1}, t_j]$. Next, we will limit the range of these t_j values: instead of taking any number between 1 and t, only the elements of some set T will be allowed. This set is called the set of possible segmentation bounds (or *possible segmentation* in short), while the algorithm which supplies these values for the given speech signal A is called the *segmentation algorithm*. In this paper we will focus on this issue.

It is not hard to find reasons for the existence of this T set. It is very unlikely that every value between 1 and t will be needed (partly because it would mean that all phonemes are then of identical length). On the other hand, keeping values in T which will obviously not be used in any segmentation makes speech recognition much more time-consuming without any gain. Constructing a good T also seems possible because in the ideal case it coincides with the real boundary between phones, which can hopefully be predicted by signal processing techniques. We will discuss this issue later in detail, but first we need to describe the search algorithm used.

2.2 The Search Process

The task of a search algorithm is to find the word and segmentation pair with the highest probability for the given speech signal A. Many search methods are available in the literature to do this: perhaps the most widely-used one is the Viterbi beam search [3], but here we opted for the multi-stack decoding algorithm [4].

These two methods are similar in the way they work: they handle prefixes of phoneme sequence-segmentation pairs which are called *hypotheses*. The hypotheses are stored in *priority queues* assigned to each possible phoneme boundary,

which automatically discard hypotheses considered improbable relative to the other hypotheses kept there. During a search these queues are visited in increasing time order. At each step all hypotheses are taken from the actual queue and they are expanded in every possible way: a new valid phoneme is attached to the end with a new end time. Then the new hypotheses have their probabilities calculated, and are inserted into the queue belonging to their new end time. Of course these queues automatically sort the hypotheses they store, and discard the most improbable ones. In the end the result will be the most probable hypothesis belonging to the final queue which has a correct phoneme-sequence (i.e. not ending in the middle of a word).

The difference between the two search methods is how they decide whether a hypothesis in a priority queue is improbable. The Viterbi beam search discards those whose probability does not lie within a given interval relative to the best one, this threshold being called the *beam width* parameter. The multi-stack decoding algorithm, however, always keeps a fixed number of best hypotheses, and discards the rest. The corresponding constant is the *stack size*. Since we used the multi-stack decoding algorithm, we shall concentrate on it, but the following statements also apply to the Viterbi beam search method.

Choosing the right stack size parameter is of course of great importance. If it happens to be too low, the correct word-segmentation pair may be lost, because a hypothesis leading to it is discarded due to its temporary improbability. If, however, the stack size chosen is too big, it greatly slows down the search process, forcing the method to needlessly investigate and expand many non-promising hypotheses.

3 Phoneme Boundaries

In a segment-based speech recognition task it is vital that all actual phoneme boundaries (or at least frames in their immediate neighbourhood) be in the set of possible segmentation bounds T. On the other hand we should limit the size of this set, otherwise it would lead to an unacceptable loss in speed. This makes the choice of the elements of this T quite important.

A baseline phoneme boundary assignment can be simply to draw one possible phoneme boundary at every k^{th} frame. It is a brute force solution, but it makes quite satisfactory recognition scores possible. However, using this, during a search we have to consider lots of hypotheses, which naturally slows down the search process. On the other hand, this way we cannot miss any real phoneme boundary. But this method is by no means optimal, so next we will describe the algorithms we constructed for phoneme boundary detection.

3.1 Phoneme Boundary Detector Algorithms

All our algorithms will take as input a function which estimates for each point the probability of this point being a boundary position. One may think that supplying this value for all frames (all a_i values) solves the entire segmentation

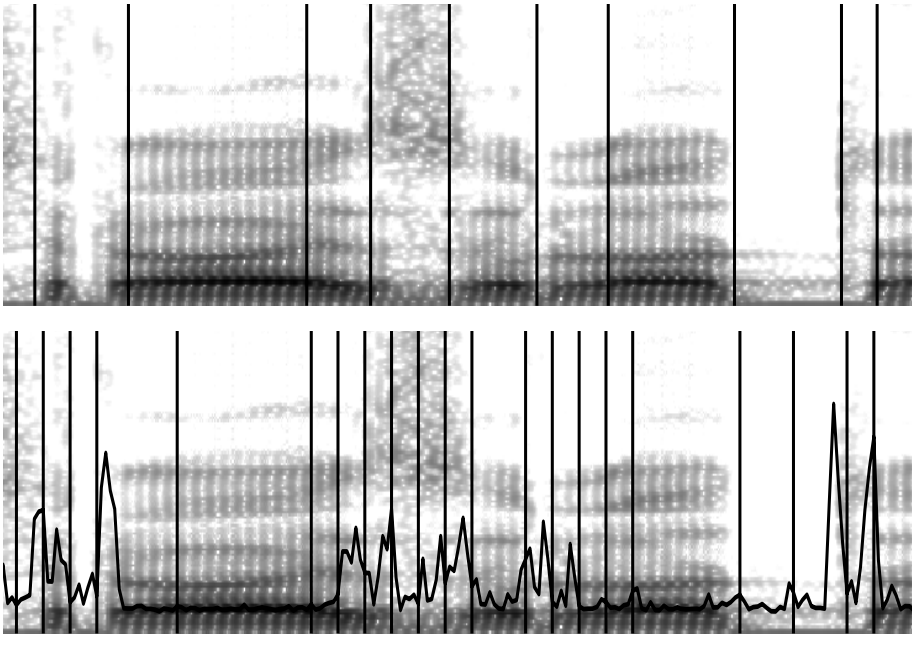

Fig. 1. Up: A spectrum with its correct segmentation. Down: the same spectrum with the b_i values and the output of the Thresholding Algorithm.

problem, but unfortunately this is not the case. While it is indeed a very important factor, in our experiments we found that determining the actual position of the segment bounds is by no means straightforward. Hence in the following we will assume that for each frame a_i we have a value b_i which tells us the probability of a phoneme boundary being at that particular place. To get these b_i values we applied a neural net, which will be described in detail in the Experiments section.

Thresholding Algorithm. This algorithm improves the default brute force segmentation method by applying a threshold. That is, we still draw a possible phoneme boundary at every k^{th} frame, but only if the corresponding b_i value is greater or equal to some minimum probability p_{min}. The lower panel of Figure 1 shows how this method works. One can see that it draws equidistant boundaries everywhere, apart from two intervals where the b_i values are permanently low. Compared to the correct segmentation shown in the upper panel, it is clear that this method draws lots of unnecessary boundaries.

Maximum Algorithm. This algorithm simply looks for positions where the b_i values take their local maxima over a given interval of neighbouring points. The advantage of this method is that the boundary positions suggested by it in many cases fall pretty close to the ones proposed by humans, as those human experts who manually process such databases also place the boundaries at

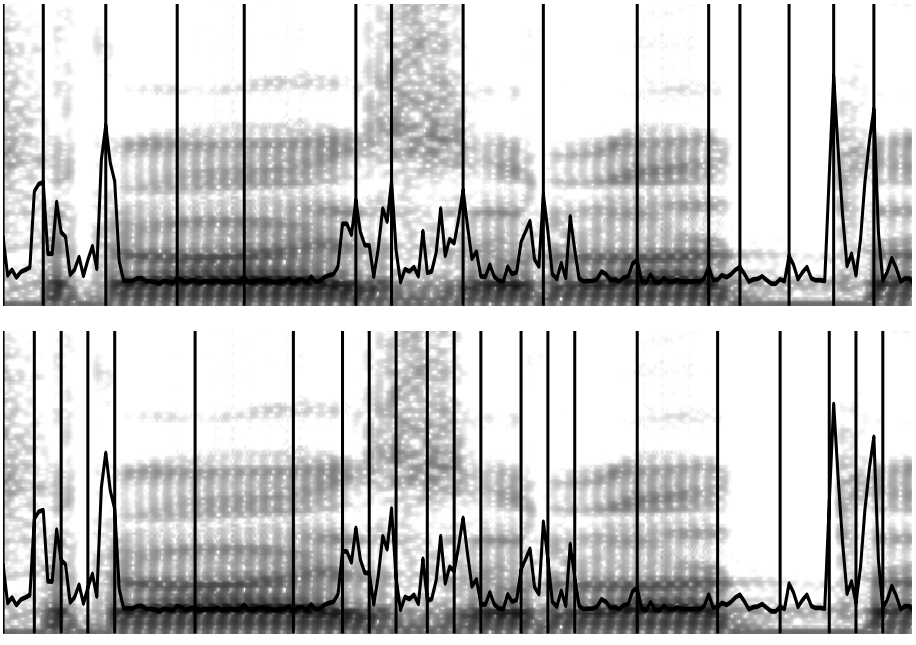

Fig. 2. Up: the same spectrum as in Figure 1 with the b_i values and the output of the Maximum Algorithm. Down: the same spectrum with the b_i values and the output of the LIF Algorithm.

the points where the spectral change is the largest. By adjusting the size k of the neighbourhood examined by the algorithm we can easily guarantee a minimal distance between the hypothesized boundaries. The algorithm can also be combined with the thresholding method described above. The behaviour of this method is illustrated in Figure 2 (upper panel).

LIF Algorithm. The problem with both the former algorithms is that we have no direct control over the density of the boundary hypotheses, so in certain cases they may not draw any boundary for a 'long' period of time. A less risky solution would be a modification of the baseline algorithm which puts boundary lines everywhere along the time axis, but dynamically adjusts the density of the markers in such a way that it is proportional to the local probability values b_i. A self-evident biologically motivated solution for this is to apply a leaky integrate-and-fire (LIF) neuron model [5]. This construct is the best-known example of the family of spiking neuron models, and its operation is very simple: the neuron integrates its input until the sum reaches a certain threshold. Then the neuron fires (emits a spike), its potential is reset, and the whole process starts again. The model can be refined by making the integration "leaky", in which case the membrane potential decays with a characteristic time constant when no input is present. The model may also incorporate an absolute refractory period, over which it is not excitable.

In our segmentation algorithm the b_i values are used as input for the integrate-and-fire neuron. Obviously, where these values are higher, their sum reaches the activation threshold sooner, and the spikes become more dense. With the tuning of the threshold one can control the density of the spikes, while by setting the refractory period a minimum distance between the neighbouring spikes can be easily guaranteed. In our experiments these parameters (e and k, respectively) were tuned manually. Obviously, we tried to set the parameters so that at the most dense areas the density of the markers were similar to those of the baseline algorithm, while at other places the boundary markers were more spaced out. The behaviour of this method is illustrated in Figure 2 (lower panel).

4 Experiments and Results

The experiments were carried out within the framework of the OASIS speech recognition system, which, due to its module-based structure and its flexible script system, provides a good basis for experimentation [6]. We used the multi-stack decoding algorithm for searching with a constant big stack size. While it is true that this parameter also affects both the running speed and the recognition scores, here we sought to test just the various segmentation algorithms.

There are many ways of measuring the correctness of a possible segmentation (and hence the algorithm which produced it). Perhaps the easiest one is to compare T with the actual phoneme bounds. To do this, first the elements of T and the actual phoneme bounds have to be mapped to each other somehow. After this step one can easily see how many boundaries are missing from T and how many of its elements are redundant. This test can be carried out in a very short time, but it is not without its drawbacks: the actual phoneme boundaries have to be known beforehand, and it is not easy to say how many missing bounds should be considered "too few". (Similarly, it is not known how many needless bounds should be judged as "too many".) Moreover, it is also not clear when two phoneme boundaries can be mapped to each other.

Since this testing method had quite a few drawbacks, we eventually chose the other option: we examined how a possible segmentation T works in practice; that is, how it affects the actual recognition scores on real data. For this we performed a thorough test involving sentence recognition, and examined how the recognition scores varied. But to do this we first have to introduce the way these two recognition measures can be calculated.

We cannot simply compare the original and the resultant sentences because this way even one badly identified word would ruin the whole sentence. We cannot compare the two sentences word for a word either, because one incorrectly inserted or omitted word would also ruin the calculated performance ratio. For this reason, usually the edit distance of the two sentences is calculated; that is, we construct the resulting sentence from the original using the following operations: inserting and deleting words, and replacing one word with another one. These operations have some cost (in our case the common values of 3, 3 and 4,

respectively), then we choose an operation series with the lowest cost. Finally we can calculate the following measures:

$$Correctness = \frac{N - S - D}{N} \tag{5}$$

and

$$Accuracy = \frac{N - S - D - I}{N}, \tag{6}$$

where N is the total number of words in all the original sentences, S is the number of substitutions, D is the number of deletions and I is the number of insertions. We will employ these two formulas throughout our tests. As for the speed-up values, we always express the running speed of the recognition using T as the percentage of that of the baseline.

4.1 The Baseline Values

The next step is to construct a working speech recognizer setup to serve as our baseline configuration. First we have to solve the problem of phoneme recognition (see Eq. (4)), for which we will use ANNs. The features used were the typical segment-based ones (for details see [7]). We had a large, general database for training: 332 people of various ages spoke 12 sentences and 12 words each, which were recorded on different computers and sound cards via different microphones [8]. To get a segment-boundary hypothesis we used the brute force method with $k = 8$.

The tests were performed on sentences from the field of medical reports; for this purpose 150 randomly selected sentences were recorded. The language model was a simple word 3-gram; i.e. the probability of the next word depended just on the last two words spoken, and it was calculated by a statistical examination of texts in a similar field. We attained, at the word-level, correctness and accuracy scores of 96.42% and 95.34%, respectively, which then served as our baseline.

4.2 Detecting Phoneme Boundaries

For phoneme boundary detection, first a function which generates the b_i values is needed. As the phoneme boundary positions correlate well with the local spectral changes, we used the first derivates ("Δ values") of the mel-frequency cepstral coefficients (MFCC) [9] as features. For each phoneme in the train database we assigned a value of 1 to the first and last frames, a value of 0 to the middle frame, and a value between 0 and 1 to the rest of the frames depending on how close they were to the sides. Next an ANN was trained for these frames and target values in regression mode. This way evaluating this ANN on any frame a_i, say, generated the corresponding phoneme bound probability value b_i.

4.3 Test Results for the Thresholding Algorithm

Surprisingly this algorithm did not produce good results. Although with some parameters (like $k = 6$, $p_{min} = 0.065$) we were able to get a small speed-up, it

also caused a decrease in the recognition scores. With some other parameters (like $k = 6$, $p_{min} = 0.070$) the recognition performance was better than that of the baseline, but this configuration was slower as well. The next table shows the interesting test results we obtained:

k	p_{min}	Correctness	Accuracy	Relative speed
6	0.065	96.06%	94.94%	90.23%
6	0.070	97.49%	95.70%	114.02%
baseline		96.42%	95.34%	100.00%

Usually the method was just too slow, or when this was not the case, the two recognition percentages fell dramatically. These results are probably due to the fact that the phoneme boundary probability estimator ANN was trained to detect the amount of change in the spectrum. Sometimes, however, this change is not abrupt, but occurs over a long time, resulting in just slightly higher b_i values over the longer period. The Thresholding Algorithm was not able to detect these changes, unlike the other two which, in contrast, could take the context of a given frame into consideration.

4.4 Test Results for the Maximum Algorithm

The overall results of this algorithm are somewhat mixed. Although visually inspecting the possible segmentations it produced should be the most promising method of the three, in practice this was not so. The explanation is that although the method usually inserted very few needless possible bounds, it also skipped some necessary ones, and in practice failing to find phoneme bounds is a very serious mistake. The results can be seen in the next table.

k	Correctness	Accuracy	Relative speed
3	93.90%	90.68%	127.73%
4	95.34%	92.83%	72.98%
5	93.19%	89.96%	48.83%
6	82.80%	78.49%	35.70%
baseline	96.42%	95.34%	100.00%

Moreover, the algorithm was tuneable only to a very limited extent: its parameter had to be a small integer, so while with the value of 3 it was slower than the basic method, even with 5 it produced a much lower accuracy value. With the only value left in between (4) the accuracy was also somewhat low (92.83% vs. the baseline 95.34%; a 53% increase in the relative error).

4.5 Test Results for the LIF Algorithm

This algorithm, unlike the others, achieved a satisfactory improvement in speed with no loss in accuracy (see Figure 3). Moreover, it was able to raise the recognition figures along with a decent gain in speed. This was probably due to the fact that the algorithm was better than the other two in some respects. While the Maximum Algorithm was very good at finding possible phoneme bounds,

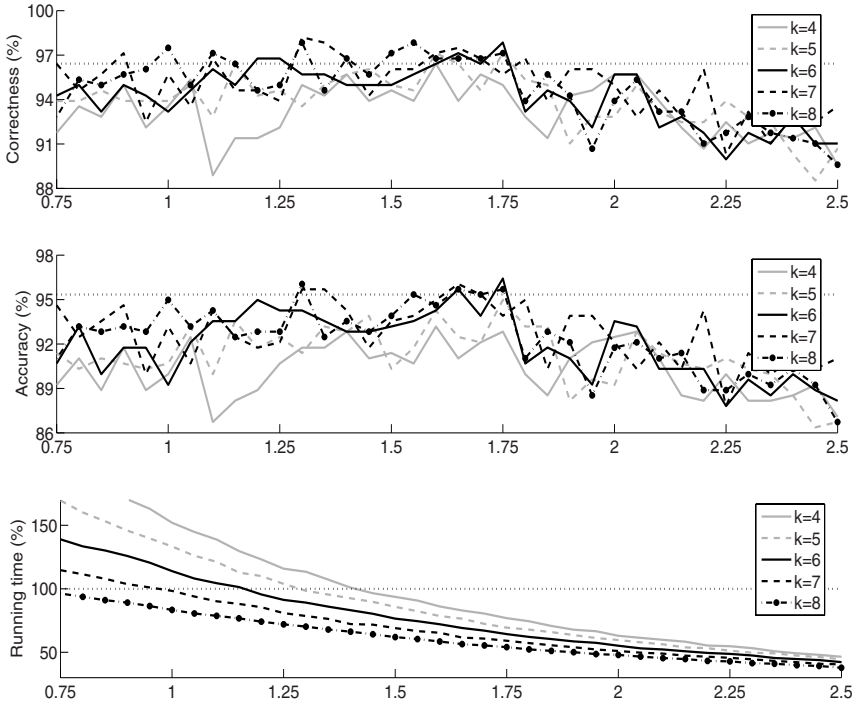

Fig. 3. The correctness, accuracy and running time results of the LIF algorithm with different k and e values; the baseline values are shown by a dotted line

it could not be tuned. On the other hand, the Thresholding Algorithm could be fine-tuned with its two parameters, but the method itself was too simple to produce really good results. The LIF algorithm, however, seems to be the golden mean between the two: it also has two parameters for tuning, and suggests phoneme bounds at much smarter places than the Thresholding Algorithm does, and it takes the context into account.

As for the increase of the recognition scores, it was a surprising result, which was probably due to the combined effect of many factors. Firstly, the phoneme bounds were placed at more accurate positions than they were with the basic method. Secondly, fewer possible phoneme bounds were generated, thus fewer unusable hypotheses were generated, which did not fill the priority queues that much (as the stack size was the same in all cases). And thirdly, due to the fewer possible phoneme bounds being used, it was possible to draw them more closely to each other (i.e. to use a smaller k value) than in the basic method. (The last statement, of course, is true for all three segmentation algorithms.) Some interesting values are shown in the next table, the best values being presented in **bold**.

k	e	Correctness	Accuracy	Relative speed
5	1.75	97.13%	94.98%	69.37%
6	1.75	97.85%	**96.42%**	64.28%
7	1.30	**98.20%**	95.69%	78.58%
7	2.20	96.05%	94.26%	**46.48%**
8	1.30	97.85%	96.05%	70.23%
baseline		96.42%	95.34%	100.00%

From the values the tradeoff between the recognition scores and the running speed can clearly be seen: if we aim for a 50% plus reduction in the running times, it will eventually lead to a small decrease in both correctness and accuracy. Since accuracy is more important than correctness, we chose the configuration with the values $k = 6$ and $e = 1.75$. With these parameters we were able to achieve correctness and accuracy scores of 97.85% and 96.42%, respectively, with the running time being just 64.28% of the original one. These recognition performances mean a 40% and 23.28% improvement in relative error terms, while the 35.72% reduction in running times is also significant. Of course, it is not really the actual parameters which are important, but rather the fact that this method could be fine-tuned to our needs.

5 Conclusions

In this paper we investigated the issue of detecting potential phoneme bounds, which is an important task in segment-based speech recognition. Since the position of the probable phoneme bounds correlate well with local spectral changes, we were able to construct a bound probability estimator function based on them. Then we introduced three novel algorithms for phoneme bound detection instead of the basic, functional-but-slow brute force method, which were all based on this estimator function. The three algorithms were all quite different: the first was a thresholding version of the basic brute force procedure, the second looked for local maxima of the probability estimator function, while the third incorporated a special spiking neuron. Out of these three the last proved to be surprisingly good for the task: not only were we able to speed up the recognition process by 35%, but we could also significantly raise the recognition scores at the same time. These points, we think, make the algorithm worthy of both further study and putting it into practice.

References

1. Jelinek, F.: Statistical Methods for Speech Recognition. MIT Press, Cambridge (1997)
2. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995)
3. Hart, P.E., Nilsson, N.J., Raphael, B.: Correction to a formal basis for the heuristic determination of minimum cost paths. SIGART Newsletter (37), 28–29 (1972)

4. Bahl, L.R., Gopalakrishnan, P.S., Mercer, R.L.: Search issues in large vocabulary speech recognition. In: Proceedings of the 1993 IEEE Workshop on Automatic Speech Recognition, Snowbird, UT (1993)
5. Gerstner, W., Kistler, W.M.: Spiking Neuron Models. Cambridge University Press, Cambridge (2002)
6. Kocsor, A., Tóth, L., Kuba Jr., A.: An overview of the oasis speech recognition project. In: Proceedings of ICAI 1999, Eger-Noszvaj, Hungary (1999)
7. Glass, J., Chang, J., McCandless, M.: A probabilistic framework for feature-based speech recognition. In: Proceedings of ICSLP 1996, Philadelphia, PA, vol. 4, pp. 2277–2280 (1996)
8. Vicsi, K., Kocsor, A., Teleki, C., Tóth, L.: Hungarian speech database for computer-using environments in offices (in hungarian). In: Proceedings of MSZNY 2004, Szeged, Hungary, pp. 315–318 (2004)
9. Huang, X., Acero, A., Hon, H.-W.: Spoken Language Processing. Prentice-Hall, Englewood Cliffs (2001)