# Inverting the Point Process Model for Fast Phonetic Keyword Search

*Keith Kintzley*[1], *Aren Jansen*[1,2], *Kenneth Church*[2,3], *Hynek Hermansky*[1,2]

[1]Dept. of Electrical and Computer Engineering, [2]HLT Center of Excellence
Johns Hopkins University, Baltimore, Maryland, USA
kintzley@jhu.edu, aren@jhu.edu, hynek@jhu.edu
[3]IBM T.J. Watson Research Center, Yorktown Heights, New York, USA
kwchurch@us.ibm.com

## Abstract

Normally, we represent speech as a long sequence of frames and model the keyword with a relatively small set of parameters, commonly with a hidden Markov model (HMM). However, since the input speech is much longer than the keyword, suppose instead that we represent the speech as a relatively sparse set of impulses (roughly one per phoneme) and model the keyword as a filter-bank where each filter's impulse response relates to the likelihood of a phone at a given position within a word. Evaluating keyword detections can then be seen as a convolution of an impulse train with an array of filters. This view enables huge speedups; runtime no longer depends on the frame rate and is instead linear in the number of events (impulses). We apply this intuition to redesign the runtime engine behind the point process model for keyword spotting. We demonstrate impressive real-time speedups (500,000x faster than real-time) with minimal loss in search accuracy.

**Index Terms**: keyword spotting, point process model

## 1. Introduction

State-of-the-art spoken term detection (STD) systems based on ASR lattice search offer very strong performance, but it comes at a high cost. Assuming the existence of an appropriate ASR pipeline, there is a non-trivial computational overhead associated with running full recognition, and after having processed the speech, we are confronted with the issue of storing and searching the resulting lattices. In order to provide reasonable access speed, it is typical to employ an inverted index the size of which can easily be on par with the word lattice itself. A recently published state-of-the-art STD system for Turkish Broadcast News in [1] using a finite-state transducer based index reports an average search time of 4 ms per query on 163 hours of audio (nearly 150,000,000x real-time). However, achieving this search speed requires an index more than twice the size of the corresponding word lattice. Handling out-of-vocabulary (OOV) terms poses a further challenge. By definition these terms will not be present in ASR word lattices, therefore many STD systems fall back on searching potentially larger phonetic lattices in order to handle these queries.

In addition to the size of the index, there is also significant processing overhead involved with index construction. The figures reported in [2] on the IBM system constructed for the NIST 2006 STD evaluation provides some insight. This system recorded an average query time of 0.0041 sec per hour of speech (878,000x real-time) and an index size of 0.327 MB per hour of speech. However, these numbers do not account for the index construction time (including audio processing, word/phonetic lattice generation and index creation) of 7.56 hours of processing per hour of speech (8 times *slower* than real-time). Furthermore, if we are dealing with a constant stream of audio, we need to consider the ease with which an index can be augmented with new data. Consider the task of spoken term detection applied to the intake of a typical day at YouTube. At the time of writing, YouTube reports that its users upload 60 hours of video on average every single minute (or equivalently, 9.9 years of content per day). Considering the volume of this stream, perhaps a less complex solution has merit for certain scenarios.

The dynamic match lattice spotting STD system presented in [3] is not based on an LVCSR system. It offers open vocabulary search on phonetic lattices and reports an average search speed of 2 sec per hour of speech (1,800x real-time). Certainly, older HMM-based keyword spotting systems, which predate ASR lattice-based approaches, have reasonable results with relatively low complexity, but search speed is lacking. The basic HMM-based system described in [4] was implemented in [3], which reported a search speed of roughly 33 times real-time. With these issues in mind, we begin by arguing the advantages of a sparse representation and outline a novel detection framework. After reviewing the point process model (PPM) for keyword spotting, we develop an efficient upper bound on the detection function consistent with our proposed approach. Finally, we verify these ideas in keyword spotting experiments.

## 2. Motivation

An ideal keyword search system is one which offers compact representation, provides fast query times, and can easily incorporate new data. Before addressing approaches to keyword search, we should pause to consider the representation of the signal upon which our decoder must operate. For a signal $T$ frames in duration and an HMM with $N$ states, the signal representation consists of the observation likelihoods at every state and every frame, $O(NT)$ real-valued quantities. It is on top of this representation which the Viterbi algorithm operates. Determining the most likely path requires that we consider all transitions into every state and thus decoding is constrained to proceed in time $O(N^2T)$.

While HMMs operate on a dense frame-based representation, one possible starting point for developing faster alternatives begins with a sparse representation of the speech signal. Consider a framework in which the speech input is reduced to a discrete set of impulses, each corresponding to a phonetic event. The density of this representation is solely a function of the phone production rate, significantly lower than the typical 100-Hz frame rate. Further, it is independent of the phone set or state space dimension $N$.

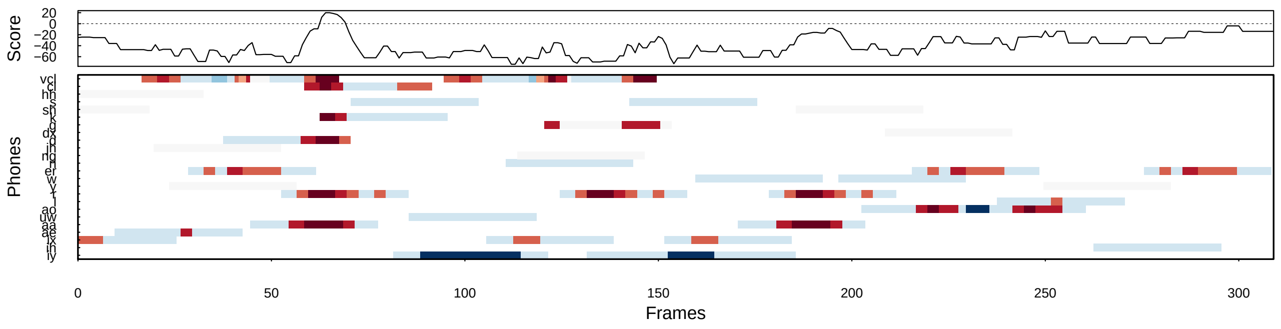

Figure 1: *Depiction of computing a detection function (top) by summing over scores from each phonetic event impulse response.*

The goal is to identify keywords. We desire a system which takes as input discrete phonetic events and efficiently produces a real-valued output where high values correspond to the presence of a keyword. The keyword can be modeled as an ensemble of filters $h_p(t)$ for each phone $p$ where the impulse response relates to the log likelihood of observing phone $p$ at each position within the word. Computing a detection function becomes a simple convolution of impulses with filters which can be made to run very fast and no longer depends on frame rate or alphabet size. An example of this operation in the point process keyword spotting framework is shown in Figure 1. In the following sections, we detail how the point process model decoding can be cast in this form.

## 3. Point process model for keyword spotting

The keyword spotting system used in this work is based upon a point process model described completely in [5]. In this framework, the input speech signal is represented by an extremely sparse set of *phonetic events*. We consider events taken as the maxima of a filtered posterior trajectory function which results in a single event per phone occurrence (a *posterior trajectory* refers to the posterior probability of a phone as function of time). Models built upon this representation take advantage of not only the identities of the phones detected, but also the sequence and relative timing between the events. Candidate keyword occurrences are identified from the detection function defined as the ratio of the likelihood of a set phonetic events under a keyword model relative to its likelihood under a background model. Formally, given a keyword $w$ and a set of observed phonetic events $O(t)$ beginning at time $t$, the detection function $d_w(t)$ is given by

$$d_w(t) = \log\left[\frac{P(O(t)|\theta_w)}{P(O(t)|\theta_{bg})}\right], \qquad (1)$$

where $\theta_w$ corresponds to the keyword-specific model parameters and $\theta_{bg}$ corresponds to background model parameters. This detection function is simply a log likelihood ratio evaluated at $t$ which takes large values when it is likely that the keyword occurred. For each phone $p$ in the set of all phones $\mathcal{P}$, we define $N_p = \{t_1, \cdots, t_{n_p}\}$, the set of points in time at which phone $p$ occurs relative to time $t$. The observation $O(t) = \{N_p\}_{p\in\mathcal{P}}$ is thus the collection of these sets of points. Assuming for the moment a fixed keyword duration $T$, we will now specify the form of the models which yield estimates of $P(O(t)|T, \theta_w)$ and $P(O(t)|T, \theta_{bg})$.

For both cases, we model the set of points as having arisen from underlying Poisson processes, the background model being *homogeneous* (constant Poisson rate parameter) and the keyword model as *inhomogeneous* (time-varying Poisson rate parameter). First, we consider the background model. Under the simplifying assumption that the Poisson process for each phone $p$ is independent of other phones, we can express the likelihood of the entire collection of points $O(t)$ under the background model given $T$ as

$$P(O(t)|T, \theta_{bg}) = \prod_{p\in\mathcal{P}} (\mu_p)^{n_p} e^{-\mu_p T},$$

where $n_p$ is the number of events for phone $p$ in the interval $(0, T]$ and $\mu_p$ is the homogeneous Poisson rate parameter for phone $p$.

For the inhomogeneous Poisson process, the rate parameter is assumed to be a continuous function of time; however, we will consider approximating this as a piecewise constant function over $D$ uniformly spaced divisions in $(0, T]$, with the inhomogeneous rate parameters for phone $p$ denoted $\lambda_{p,d}$ for $d = 1, \ldots, D$. We make a corresponding subdivision in our collection of observations $N_p$ into $D$ equal-size partitions specified as $N_{p,d}$ for $d = 1, \ldots, D$. Thus, the likelihood of the entire collection of points $O(t)$ under the keyword model given $T$ can be expressed as

$$P(O(t)|T, \theta_w) = \prod_{p\in\mathcal{P}} \prod_{d=1}^{D} (\lambda_{p,d})^{n_{p,d}} e^{-\lambda_{p,d}\Delta T}, \qquad (2)$$

where $\Delta T = T/D$. We will now describe how keyword duration $T$ is incorporated into the model. Underlying our entire approach is the assertion that words are distinguished by a characteristic pattern of phones in time. We now make a further simplifying assumption that this representative pattern is independent of the actual keyword duration. In other words, multiple observations of the same keyword scaled to the interval $(0, 1]$ will result in the same pattern and thus can be modeled by the same set of inhomogeneous rate parameters. To incorporate this, we define a new set of points with respect to a normalized time scale as $N'_p = \{t'_i | t'_i = t_i/T,\ \forall t_i \in N_{p,d}\}$ with $O'(t) = \{N'_p\}_{p\in\mathcal{P}}$. After a change of variables, the likelihood in (2) with $O'(t)$ becomes

$$P(O'(t)|T, \theta_w) = \prod_{p\in\mathcal{P}} \prod_{d=1}^{D} (\lambda_{p,d})^{n_{p,d}} e^{-\lambda_{p,d}/D}.$$

Our estimates of $P(O'(t)|T, \theta_w)$ and $P(O(t)|T, \theta_{bg})$ are conditioned on the latent variable $T$, therefore we may compute the detection function (1) on an unknown utterance by integrating over $T$ in

$$d_w(t) = \log\left[\int_0^\infty \frac{P(O'(t)|T, \theta_w) P(T|\theta_w)}{T^{|O(t)|} P(O(t)|T, \theta_{bg})} dT\right]. \qquad (3)$$

In practice this integral is approximated by a summation over a discrete set $\mathcal{T}$ of candidate durations. We estimate $P(T|\theta_w)$

for each $T \in \mathcal{T}$ based upon keyword examples from training. After finding the parameters for $\theta_w$, $\theta_{bg}$ and $P(T|\theta_w)$, we can calculate $d_w(t)$ given an observation $O(t)$. Keyword detections are declared at local maxima of $d_w(t)$ above a given threshold.

## 4. Bounding the detection function

To simplify the evaluation of (3), we will consider the approximate detection function

$$d_w(t) \approx \max_T \log \left[ \frac{P(O'(t)|T,\theta_w)P(T|\theta_w)}{T^{|O(t)|}P(O(t)|T,\theta_{bg})} \right] \quad (4)$$

and show how terms can be combined. First, note that $|O(t)| = \sum_p n_p$ and $n_p = \sum_d n_{p,d}$. Also, using $\mu_p = \sum_d \mu_p/D$, we may rewrite the argument of the log in (4) as

$$P(T|\theta_w) \prod_{p \in \mathcal{P}} \prod_{d=1}^{D} \left( \frac{\lambda_{p,d}}{\mu_p T} \right)^{n_{p,d}} e^{\mu_p T/D - \lambda_{p,d}/D}.$$

After taking the log, we find the approximate detection function consists of three terms,

$$d_w(t) \approx \max_T \left\{ \log P(T|\theta_w) + \sum_{p\in\mathcal{P}} \left( \mu_p T - \frac{1}{D} \sum_{d=1}^{D} \lambda_{p,d} \right) + \sum_{p\in\mathcal{P}} \sum_{d=1}^{D} n_{p,d}\phi_{p,d} \right\} \quad (5)$$

where $\phi_{p,d} \triangleq \log(\lambda_{p,d}/\mu_p T)$. Note that the first two terms depend only on the word duration model $P(T|\theta_w)$ and the Poisson rate parameters ($\mu_p$, $\lambda_{p,d}$) but are independent of observed phonetic events. Thus the detection function depends on the total event count $n_{p,d}$ for each phone $p$ and word division $d$ times a weighting factor $\phi_{p,d}$. Direct evaluation of this function proceeds as follows: (i) for each time $t$ and sample keyword duration $T$, determine $O(t)$, the set of phonetic events which occur in the interval $(t, t+T]$; (ii) from $O(t)$ accumulate the total count $n_{p,d}$ for each phone and word division; (iii) sum over the product of $n_{p,d}$ and its corresponding weighting factor $\phi_{p,d}$; and, (iv) repeat for each candidate duration $T$ and take the max.

### 4.1. Simple upper bound on detection function

Although computationally simple, the direct implementation of (5) requires that for each event we determine the word division $d$ to which it belongs as the window of length $T$ slides right one frame at a time. Our first approach to speeding up the computation of the detection function was to replace full evaluation of (5) with a simple upper bound. It is easy to see that using $\phi_{\max,p} \triangleq \max_d \phi_{p,d}$, the maximum weighting factor over all divisions $d$, provides an upper bound on $d_w(t)$ and liberates us from evaluating $n_{p,d}$ (i.e. $n_p$ suffices). Importantly, note that changes in $d_w(t)$ only occur when an event enters or exits a window of duration $T$. Instead of storing the value of the detection function at each frame, we instead retain the much sparser set of *changes* to the detection function. With delta coding we maintain one accumulator array and only perform two additions for each phonetic event. On average, we observe 16 phonetic events per second, a factor of 6 lower than the frame rate of 100 Hz. Examples of the detection functions and simple upper bounds for the keyword "greasy" are shown in Figure 2 using two different sources of phonetic event data. Oracle phonetic events represent perfect phonetic information. The

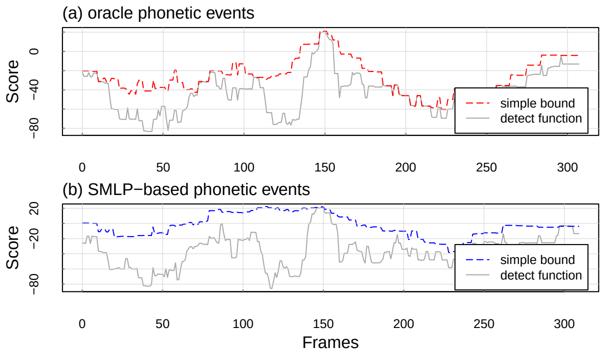

Figure 2: *Detection function $d_w(t)$ and detection function simple upperbound for keyword "greasy" using oracle and SMLP-based phonetic events.*

SMLP-based events are derived from a sparse multilayer perceptron (SMLP) phone recognizer [6] with phonetic matched filtering [7]. Since this simplification reduces the model to a "bag of phones," it is not surprising that the upper bound is loose. This was corroborated by poor performance observed in keyword spotting experiments.

### 4.2. Tighter upper bound on detection function

What accounts for the difference in the two bounds in Figure 2? The matrix of weights $\phi_{p,d}$ is computed from the homogeneous and inhomogeneous Poisson rate parameters. Intuitively, phonetic events consistent (correct phone and relative timing) with the keyword result in positive weights, and those which are inconsistent with the model have large negative weights. Keyword detections are marked at the peaks of the detection function and occur when phonetic events are maximally aligned with positive weights $\phi_{p,d}$. By considering only the maximal score $\phi_{\max,p}$ for each phone and ignoring intra-word phonetic timing, we purchase computational simplicity at the cost of a model's ability to discriminate based on timing.

This approximation is particularly detrimental with non-oracle phonetic data which contains phone detector confusions. Comparing two models, the more discriminative one will contain a higher fraction of large negative values in the $\phi_{p,d}$ terms. Large negative $\phi_{p,d}$ terms are produced when the total number of events observed in keyword training examples corresponding to phone $p$ and division $d$ is zero. Phonetic confusions result in non-zero counts, which is exacerbated by taking the maximum $\phi_{p,d}$ for all $d$. Keyword models derived from oracle data do not exhibit the errors present in real phone detectors, and this results in relatively few phones with positive scores. This accounts for the difference in the tightness of the bounds seen in Figure 2 and it also offers insight on how to improve the bound.

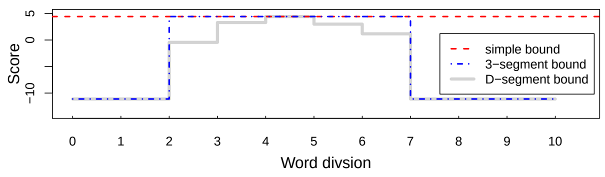

Figure 3: *Various bounds on the score vector $\phi_{p,d}$ for the phone /aa/ in the keyword model for "dark".*

The simple upper bound $\phi_{\max,p}$ is fast because it requires that we encode only 2 score changes per event. To tighten the bound, we can instead consider multi-segment, piecewise constant upper bounds as illustrated in Figure 3. The 3-segment bound may contain up to 4 score changes, but we permit fewer depending on the score vector. Likewise, the $D$-segment bound may contain as many segments as word subdivisions. By considering multi-segment bounds, we significantly improve the ability to discriminate by phonetic event position, while retaining most of the computational advantage of only evaluating changes in the detection function.

### 4.3. Detection function as a convolution

Another way to envision the computation of the detection function is to recognize it as the summation over all phones of a sequence of phonetic events (i.e., an impulse train for phone $p$) convolved with its corresponding score vector (i.e. a filter impulse response, $h_p(t)$). This is depicted more clearly in Figure 1. The convolution operation alone does not suggest any computaional savings, but because the input is a sparse set of impulses, we are liberated from shifting and multiplying. Direct implement of (4) requires computing an event's position within a sliding window $(t, t+T]$ for each frame $t$. The alternate view just described allows us to *invert* the process; we proceed event-by-event and immediately record incremental contributions to the detection function using the time-reversed score vector. Thus, we make the calculation of the detection function linear in the number of phonetic events, rather than the number of frames.

## 5. Experiments

In this section we report results of keyword spotting experiments using various detection function bounds. We utilized the Wall Street Journal (WSJ0 and WSJ1) datasets which were partitioned into two folds of 23 hours of speech. The audio data was processed into perceptual linear prediction features and then transformed into a phone posteriorgram representation using a hierarchical MLP with 9 context frames. We then extracted phonetic events from posteriorgram data using phonetic matched filters as described in [7] with a threshold of $\delta = 0.24$.

We assembled a list of 1521 keywords from the WSJ corpus with minimum average duration of 200 ms, a minimum of 4 phones, and which occurred at least 10 times in each data fold. For each keyword and each data fold, we computed keyword model parameters $\theta_w$ using all available keyword examples in that fold. The models in these experiments were based on MLE parameter estimates so performance depended on the number of keyword examples, but we have recently demonstrated MAP-based models which require significantly less training data [8]. Models from one fold were used to search for keywords in the other fold, and detections were declared at local maxima of $d_w(t)$ above threshold $\delta_w$. Detections within 100 ms of the beginning of the keyword in the transcript were marked as correct. Multiple correct detections of the same keyword were discarded, and all other detections were recorded as false alarms. For the results listed in Table 1, we calculated average figure of merit (FOM), the mean detection rate given $1, 2, \ldots, 10$ false alarms per keyword per hour as the threshold $\delta_w$ was varied.

We evaluated four versions of the decoding algorithm: (i) the frame-by-frame, direct implementation of (4); (ii) the simple (1-segment) upper bound using $\phi_{\max,p}$; (iii) a 3-segment upper bound; and, (iv) a $D$-segment upper bound where the number of word divisions $D = 10$. All versions were coded in Java, but we also include results for a C++ version of (iv). In computing the real-time speedup (RTS), we included in the processing time the overhead of reading phonetic event data, scoring the detections, and saving the results. These results represent processing on a single-core of a 2.66-GHz Intel E5430 Xeon processor.

Table 1: *Comparison of median FOM and search speed on 1521 keyword set using various decoding algorithms.*

| algorithm | FOM | ΔFOM | Speed (RTS) |
|---|---|---|---|
| direct implementation (java) | 70.9 | – | 7,483 |
| simple upper bound (java) | 17.9 | -74.7% | 431,594 |
| 3-segment bound (java) | 68.4 | -3.5% | 397,752 |
| $D$-segment bound (java) | 70.5 | -0.5% | 374,195 |
| $D$-segment bound (C++) | 70.5 | -0.5% | 524,189 |

First, in Table 1 we observe that compared to the direct implementation of the detection function, computing the simple upper bound is 57 times faster, but results in a 75% relative decrease in average FOM. A simple 3-segment upper bound on $\phi_{p,d}$ reduces speed by only 8% yet recovers almost all of the previous loss in FOM. Finally, a $D$-segment bound is 13% slower than the simple bound, but offers virtually identical FOM performance as the direct implementation. Finally, we note that with a C++ implementation of the $D$-segment bound, we can obtain search speeds in excess of 500,000x faster than real-time.

## 6. Conclusions

We have presented a novel framework for keyword search in which speech is represented as a sparse set of phonetic impulses and keyword detection is implemented as convolution with an ensemble of filters. By approximation with an upper bound, we have shown that the point process model keyword detection function can be cast in this framework. Finally, we have demonstrated keyword search experiments which averaged better than 500,000x faster than real-time with only negligible loss in FOM.

## 7. Acknowledgments

The research presented in this paper was partially funded by DARPA RATS program under D10PC20015.

## 8. References

[1] Can, D., Saraclar, M., "Lattice Indexing for Spoken Term Detection,", in *IEEE Trans. Audio, Speech and Language Proc.,* 19(8):2338–2347, 2011.

[2] Mamou, J., Ramabhadran, B. and Siohan, O.,"Vocabulary independent spoken term detection" in *Proc. of SIGIR*, pp.615–622, 2007.

[3] Kishan Thambiratnam, K. and Sridharan, S.,"Automatic recognition of keywords in unconstrained speech using hidden Markov models," in *IEEE Trans. Audio, Speech and Language Proc.,* 15(1):346–357, 2007.

[4] Wilpon, J.G., et al.,"Automatic recognition of keywords in unconstrained speech using hidden Markov models," in *IEEE Trans. Acoustics, Speech and Signal Proc.,* 38(11):1870–1878, 1990.

[5] Jansen, A. and Niyogi, P.,"Point Process Models for Spotting Keywords in Continuous Speech", *IEEE Trans. Audio, Speech and Language Proc.,* 17(8):1457–1470, 2009.

[6] G.S.V.S. Sivaram and Hermansky, H., "Multilayer Perceptron with Sparse Hidden Outputs for Phoneme Recognition," in *Proc. of ICASSP,* 2011.

[7] Kintzley, K., Jansen A. and H. Hermansky, "Event Selection from Phone Posteriorgrams Using Matched Filters," in *Proc. of INTERSPEECH,* pp.1905-1908, 2011.

[8] Kintzley, K., Jansen A. and H. Hermansky, "MAP Estimation of Whole-Word Dictionary Priors Event Selection from Phone Posteriorgrams Using Matched Filters," to appear in *Proc. of INTERSPEECH,* 2012.