EFFICIENT DECODING WITH GENERATIVE SCORE-SPACES USING THE EXPECTATION SEMIRING

Rogier C. van Dalen, Anton Ragni, and Mark J. F. Gales

Department of Engineering
University of Cambridge
Cambridge, United Kingdom

ABSTRACT

State-of-the-art speech recognisers are usually based on hidden Markov models (HMMs). They model a hidden symbol sequence with a Markov process, with the observations independent given that sequence. These assumptions yield efficient algorithms, but limit the power of the model. An alternative model that allows a wide range of features, including word- and phone-level features, is a log-linear model. To handle, for example, word-level variable-length features, the original feature vectors must be segmented into words. Thus, decoding must find the optimal combination of segmentation of the utterance into words and word sequence. Features must therefore be extracted for each possible segment of audio. For many types of features, this becomes slow. In this paper, long-span features are derived from the likelihoods of word HMMs. Derivatives of the log-likelihoods, which break the Markov assumption, are appended. Previously, decoding with this model took cubic time in the length of the sequence, and longer for higher-order derivatives. This paper shows how to decode in quadratic time.

***Index Terms*—** Speech recognition, log-linear models, weighted finite-state transducers, expectation semiring.

1. INTRODUCTION

State-of-the-art speech recognisers are often based on hidden Markov models, which use a Markov process to model a hidden symbol sequence, and assume that the observations are conditionally independent given the symbol sequence. This restriction allows for fast algorithms, like the Viterbi and forward algorithms, but it is deemed to be one of the main impediments to improving recognition performance [1]. Recently, there has been interest in discriminative log-linear models that can deal with a wide range of features extracted from spans longer than one frame and of variable length (e.g. [2, 3, 4, 5]). When using longer-span features to recognise continuous speech, segmentations into, e.g., words must be found explicitly. Existing approaches for structured models often derive a segmentation from a conventional speech recogniser. However, these segmentations are not optimal for the structured model, and may limit the performance gains from moving to a more powerful model. It is therefore desirable for the decoding process to find the optimal combination of word sequence and segmentation.

Decoding with such a model requires two steps: extracting segmental features, and finding the joint optimal word sequence and segmentation. Keeping the language model fixed, the latter task can be performed in $\Theta(T^2)$ time, where T is the length of the utterance [2, 6]. The complexity of the former task depends on the nature of the features, but computing features for the $\Theta(T^2)$ segments must take at least $\Theta(T^2)$ time. This is therefore the bottleneck for performance. Previous work [6] has shown how to extract *generative scores* that include nth-order derivatives in $\Theta(T^{2+n})$ time. This paper will introduce a method to extract derivatives of any order for all segments in $\Theta(T^2)$ time, that is, amortised constant time.

Generative score-spaces, which generalise Fisher score-spaces [7], consist of log-likelihoods of generative models, and their derivatives. In this paper, segmental features for speech recognition are derived from likelihoods of word HMMs for audio segments. There are two reasons for using these scores as features for a classifier. With a zeroth-order generative score-space, which contains only the log-likelihood itself, classification is closely related to that of the original generative model. This allows state-of-the-art techniques for speech recognition, such as methods for noise-robustness, to be applied. Secondly, derivatives even of frame-level log-likelihoods are functions of all frames in the segment. Thus, the independence assumptions of the HMM are relaxed.

Since the derivatives in the generative score-space depend on all frames in a segment, a direct implementation would re-compute them completely for every hypothesised segment. This was the approach adopted in [3], using an algorithm with nested passes of forward–backward that was run separately for each hypothesised segmentation. This paper introduces a method that incrementally computes scores for all segmentations that share a common start time with only a forward pass. It augments the output and transition probabilities with their derivatives. As long as the HMMs have only few states, which for word HMMs is the case, this algorithm requires a modest amount of extra storage. When scores are required for all possible segments, they can be computed in amortised constant time.

2. LOG-LINEAR MODELS

Discriminative models [8] are probabilistic models that can operate on a wide range of features derived from the same segment of audio. Unlike a generative model, a discriminative model for speech recognition directly yields the posterior probability of the word sequence $\mathbf{w}$ given the observation sequence $\mathbf{O}$. Here, each of the elements w_i of $\mathbf{w}$ is equal to one element v_j from the vocabulary $\mathbf{v}$. To enable compact discriminative models to be trained, the input sequence must be segmented into, e.g., words. Let $\mathbf{s} = \{s_i\}_{i=1}^{|\mathbf{w}|}$ denote a segmentation. This paper will use a log-linear model:

$$P(\mathbf{w}, \mathbf{s}|\mathbf{O}; \boldsymbol{\alpha}) \triangleq \frac{1}{Z(\mathbf{O}, \boldsymbol{\alpha})} \exp\Big(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \mathbf{w}, \mathbf{s})\Big). \tag{1}$$

This work was partly supported by EPSRC Project EP/I006583/1 (Generative Kernels and Score-Spaces for Classification of Speech) within the Global Uncertainties Programme.

Here, $Z(\mathbf{O}, \mathbf{s})$ is the normalisation constant. $\boldsymbol{\phi}(\mathbf{O}, \mathbf{w}, \mathbf{s})$ is the feature function that returns a feature vector characterising the whole observation sequence. $\boldsymbol{\alpha}$ is the parameter vector.

For simplicity, this work will assume there is no language model. The distribution then factorises over the segments of the audio, i.e. the feature function is a sum of features for each segment:

$$\boldsymbol{\phi}(\mathbf{O}, \mathbf{w}, \mathbf{s}) \triangleq \sum_i \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i), \tag{2}$$

where $\mathbf{O}_{s_i}$ indicates the observations in segment s_i.

For decoding, it is in theory possible to marginalise out the segmentation. However, this is infeasible, so instead the segmentation and word sequence that maximise the posterior in (1) will be found:

$$\begin{aligned}\arg\max_{\mathbf{w},\mathbf{s}} P(\mathbf{w}, \mathbf{s}|\mathbf{O}; \boldsymbol{\alpha}) &= \arg\max_{\mathbf{w},\mathbf{s}} \frac{1}{Z(\mathbf{O}, \boldsymbol{\alpha})} \exp\left(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \mathbf{w}, \mathbf{s})\right) \\ &= \arg\max_{\mathbf{w},\mathbf{s}} \left(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \mathbf{w}, \mathbf{s})\right) = \arg\max_{\mathbf{w},\mathbf{s}} \sum_i \boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i). \end{aligned} \tag{3}$$

There are two aspects to performing this optimisation. First, the features $\boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i)$ must be extracted for all possible words and segments. Second, the best combination of word sequence and segmentation must be found. The latter task takes $\Theta(T^2)$ time [6]. The former task, of extracting $\Theta(|\mathbf{v}| \cdot T^2)$ features, forms the bottleneck in performance. The rest of this paper will be concerned with how to extract *generative scores* in amortised constant time per segment.

Generative scores use a joint feature space for all vocabulary entries v_j. The feature vector for one segment $\mathbf{O}_{\tau:t}$ is partitioned into features related to words, and the other dimensions are zero:

$$\boldsymbol{\phi}(\mathbf{O}_{\tau:t}, w) = \begin{bmatrix} \delta(w = v_1)\boldsymbol{\phi}_1(\mathbf{O}_{\tau:t}) \\ \vdots \\ \delta(w = v_{|\mathbf{v}|})\boldsymbol{\phi}_{|\mathbf{v}|}(\mathbf{O}_{\tau:t}) \end{bmatrix}, \tag{4a}$$

where $\delta(\cdot = \cdot)$ equals 1 if its argument is true, and 0 otherwise. This expression selects the feature vector $\boldsymbol{\phi}_v(\mathbf{O}_{\tau:t})$ for word v. In this work, this feature consists of two parts. One is the log-likelihood of the audio segment $\log l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v)$ given by a word HMM. The second is its derivative with respect to the parameters of the HMM:

$$\boldsymbol{\phi}_v(\mathbf{O}_{\tau:t}) = \begin{bmatrix} \log l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v) \\ \nabla_{\boldsymbol{\lambda}} \log l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v) \end{bmatrix}. \tag{4b}$$

It is well-known how to extract the HMM likelihood for one segment $\mathbf{O}_{\tau:t}$, but in this paper it must be computed for all possible segments. Also, the derivatives must be found.

3. FEATURE EXTRACTION

This section will discuss how to extract the features in (4b) from a word HMM and a sequence of observations. The following will first recall how the HMM likelihood can be computed for a given segment of observations. Then, it will explain how to extend this to compute the likelihood for a range of segments at the same time, with the method introduced in [6]. After that, it will discuss how first- or higher-order derivatives can be found with the same time complexity (but with a greater constant factor).

Fig. 1 contains a finite-state automaton representing a trellis.[1]

[1]Since the trellis is represented as a Mealy machine, it combines contributions from the HMM output distributions and transition matrices. This will be useful in section 3.1. This trellis automaton can be produced by composing an automaton representing the output probabilities for consecutive observations with one that produces all possible HMM state sequences $\mathbf{o}_1 \ldots \mathbf{o}_T$. See [9] for more details, or [10] for a more general discussion.

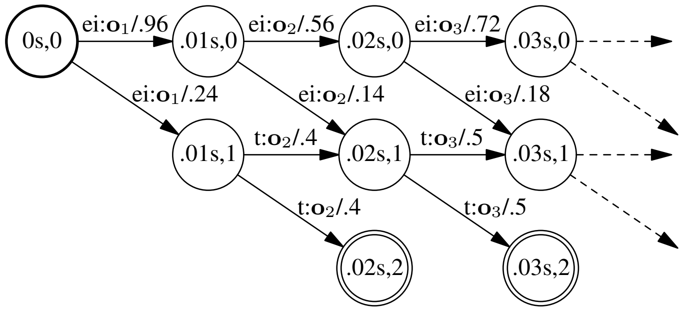

Fig. 1. Weighted finite-state transducer $\mathcal{T}$ representing a trellis.

Time goes from left to right and discrete HMM symbols from top to bottom. Each successful combination of an HMM symbol sequence (e.g. "ei ei t") and an observation sequence (e.g. $(\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3)$) corresponds to a path from a bold circle to a double circle. The corresponding likelihood is given by the product of the weight (after the slash) along the edge. The likelihood for the whole HMM for, e.g., observations $\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3$, is given by the summed likelihood over all paths from (0s,0) to (.03s,2). In general,

$$l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v) \triangleq \sum_{\substack{\pi: p[\pi] = (\tau, \text{start}) \\ \wedge n[\pi] = (t, \text{end})}} \prod_{e \in \pi} \sigma[e; \boldsymbol{\lambda}_v], \tag{5}$$

where π is a path with start state $p[\pi]$ and end state $n[\pi]$, and $e \in \pi$ are the edges along the path. $\sigma[e; \boldsymbol{\lambda}_v]$ is the weight on arc e.

It is well-known how to compute (5) efficiently: with either the forward or the backward algorithm. Both are "single-source shortest-distance" algorithms (as they are called in the literature about finite-state automata, e.g., [11]) that exploit the trellis structure. Both compute the sum of the paths from node s_1 to node s_2:[2]

$$\text{forwd}(s_1, s_2, \boldsymbol{\lambda}_v) \triangleq \text{backwd}(s_1, s_2, \boldsymbol{\lambda}_v) \triangleq \sum_{\substack{\pi: p[\pi] = s_1 \\ \wedge n[\pi] = s_2}} \prod_{e \in \pi} \sigma[e; \boldsymbol{\lambda}_v]. \tag{6}$$

The forward algorithm computes forward weights from one start node to many end nodes incrementally. Similarly, the backward algorithm computes backward weights from one end node to many start nodes. (6) can be rewritten recursively, where $p[e]$ denotes the source node of an edge, and $n[e]$ the destination:

$$\text{forwd}(s_1, s_2, \boldsymbol{\lambda}_v) = \sum_{\substack{s', e: p[e] = s' \\ \wedge n[e] = s_2}} \text{forwd}(s_1, s', \boldsymbol{\lambda}_v) \times \sigma[e; \boldsymbol{\lambda}_v]; \tag{7a}$$

$$\text{backwd}(s_1, s_2, \boldsymbol{\lambda}_v) = \sum_{\substack{s', e: p[e] = s_1 \\ \wedge n[e] = s'}} \sigma[e; \boldsymbol{\lambda}_v] \times \text{backwd}(s', s_2, \boldsymbol{\lambda}_v); \tag{7b}$$

$$\text{forwd}(s, s, \boldsymbol{\lambda}_v) = \text{backwd}(s, s, \boldsymbol{\lambda}_v) \triangleq 1. \tag{7c}$$

These can be computed efficiently with dynamic programming. The forward weights for one time in a trellis like Fig. 1 depend only on those of the previous time. The algorithm therefore progresses time instance by time instance. For a fixed HMM, the forward algorithm takes linear time in the length of the observation segment.

[2]Note that these definitions of the forward and backward algorithms are different from the ones sometimes given, for HMMs, where the emission probabilities are on states. Here, weights are only on edges, so that the two algorithms are symmetrical.

However, the objective in this paper is to consider all possible segmentations. Therefore, the likelihood for all segments $\mathbf{O}_{\tau:t}$ must be computed. In an utterance $\mathbf{O} = (\mathbf{o}_1, \dots, \mathbf{o}_T)$, with T observations, there are $\Theta(T^2)$ possible segments. The segments have an average length of $\Theta(T)$. Applying the forward algorithm separately for all possible segments would therefore take $\Theta(T^3)$ time.

Instead, it can be noted that the forward algorithm from time τ to end time T generates likelihoods for segments starting at time τ and ending at all times $t = \tau \dots T$ [6]. By applying the forward algorithm for each time step $\tau = 1 \dots T-1$, likelihoods for all $\Theta(T^2)$ segments can be found in $\Theta(T^2)$ time. Per segment, this therefore takes amortised constant time.

3.1. Computing derivatives with the expectation semiring

In addition to the log-likelihood, which the method discussed above finds efficiently, (4b) contains its derivative with respect to the parameters $\boldsymbol{\lambda}_v$. This section will first explain why applying the same strategy as above speeds up the standard method for computing the derivatives merely by a constant factor. It will then introduce a different strategy, attaching derivatives to all weights, to improve the time complexity. For simplicity, this section will find the derivative of the likelihood, not of the log-likelihood. The latter is straightforward to compute by dividing the former by the likelihood.

The weight $\sigma[e; \boldsymbol{\lambda}_v]$ on each arc in Fig. 1 is a product of a transition weight and an HMM output probability. Computing its derivative $\nabla_{\boldsymbol{\lambda}} \sigma[e; \boldsymbol{\lambda}_v]$ is therefore straightforward. The standard way of computing the derivative of the likelihood of an HMM is by using unnormalised arc posteriors $\gamma_{\tau:t}(e)$:

$$\nabla_{\boldsymbol{\lambda}} l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v) = \sum_e \gamma_{\tau:t}(e) \frac{\nabla_{\boldsymbol{\lambda}} \sigma[e; \boldsymbol{\lambda}_v]}{\sigma[e; \boldsymbol{\lambda}_v]}, \tag{8a}$$

where $\gamma_{\tau:t}(e)$ is the fraction of the total weight through edge e:

$$\gamma_{\tau:t}(e) \triangleq \sum_{\substack{\pi : p[\pi] = (\tau, \text{start}) \\ \wedge n[\pi] = (t, \text{end}) \wedge e \in \pi}} \prod_{e' \in \pi} \sigma[e'; \boldsymbol{\lambda}_v]. \tag{8b}$$

It is clear from (8b) that $\gamma_{\tau:t}(e)$ depends on the weights on edges before and after e, so that the derivative, in (8a), relaxes the Markov property (see [3] for more detail). (8b) is normally computed with the forward–backward algorithm. This combines the weight from the start node (τ, start) up to the source of e, $p[e]$, and the backward weight from the end node (t, end) to the destination of e, $n[e]$:

$$\begin{aligned} \gamma_{\tau:t}(e) = {} & \text{forwd}((\tau, \text{start}), p[e], \boldsymbol{\lambda}_v) \times \sigma[e; \boldsymbol{\lambda}_v] \\ & \times \text{backwd}(n[e], (t, \text{end}), \boldsymbol{\lambda}_v). \end{aligned} \tag{8c}$$

The computation of all required forward weights can be done efficiently as described above, as can the computation of all backward weights. However, for each segment $\tau : t$, $\gamma_{\tau:t}(e)$ derives from different forward and backward passes and must be recomputed. The average time to compute (8a) for one segment is therefore $\Theta(T)$. Finding the derivatives for all segments in this way then takes $\Theta(T^3)$ time. Extending this strategy to second-order derivatives, posteriors for pairs of arcs must be computed, and the algorithm will take $\Theta(T^4)$ time, et cetera [6]. Taking $\Theta(T^3)$ time or more to find features becomes prohibitive for speech recognition.

The following will view HMM derivatives in a different way. It is possible to append derivatives to all initial weights, and then propagate them throughout the computation of the likelihood. Any weight l is replaced with

$$\sigma \triangleq (l, \nabla_{\boldsymbol{\lambda}} l). \tag{9}$$

This starts from the arcs weights in Fig. 1. Original weights $l[e; \boldsymbol{\lambda}]$ on arc e, dependent on parameters $\boldsymbol{\lambda}$, are replaced with

$$\sigma[e; \boldsymbol{\lambda}] \triangleq (l[e; \boldsymbol{\lambda}], \nabla_{\boldsymbol{\lambda}} l[e; \boldsymbol{\lambda}]). \tag{10}$$

Now the forward algorithm must be extended so that the weight in each end state becomes a tuple of the likelihood and its derivative:

$$\text{forwd}((\tau, \text{start}), (t, \text{end}), \boldsymbol{\lambda}_v) = (l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v), \nabla_{\boldsymbol{\lambda}} l(\mathbf{O}_{\tau:t}; \boldsymbol{\lambda}_v)). \tag{11}$$

Then, finding the first-order score in (4b) with the derivative of the log-likelihood the values in (11) is straightforward.

This can be done by generalising the operations on weights so that they maintain the invariant that the second element is the derivative of the first element, as in (9). This is called "automatic differentiation" using dual numbers [12]. Given values for weights l_1 and l_2, the derivatives of their sum and product can be found with

$$\nabla_{\boldsymbol{\lambda}}(l_1 + l_2) = \nabla_{\boldsymbol{\lambda}} l_1 + \nabla_{\boldsymbol{\lambda}} l_2; \tag{12a}$$

$$\nabla_{\boldsymbol{\lambda}}(l_1 \cdot l_2) = l_1 \cdot \nabla_{\boldsymbol{\lambda}} l_2 + l_2 \cdot \nabla_{\boldsymbol{\lambda}} l_1. \tag{12b}$$

The trellis in Fig. 1 is a weighted finite-state automaton. Requirements on the types of weights that can be used in weights automata are well-established [13]. For many algorithms, including the forward algorithm, the requirement is that the weights are in a semiring. Semirings define operations $\oplus$ and $\otimes$, which generalise $+$ and $\times$ on scalars, and identities $\overline{0}$ and $\overline{1}$, which generalise 0 and 1. An important property for a, b, c to be in a semiring is that multiplication must distribute over addition, that is, $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$. This makes it possible to rewrite (6) to (7).

It turns out that the tuple in (9) is in the *expectation semiring* [14], so that it can be used in weighted finite-state automata. Denoting the weights with $(l, \nabla_{\boldsymbol{\lambda}} l)$, it follows from (12) that the semiring operations must be defined as

$$(l_1, \nabla_{\boldsymbol{\lambda}} l_1) \oplus (l_2, \nabla_{\boldsymbol{\lambda}} l_2) \triangleq (l_1 + l_2, \nabla_{\boldsymbol{\lambda}} l_1 + \nabla_{\boldsymbol{\lambda}} l_2); \tag{13a}$$

$$(l_1, \nabla_{\boldsymbol{\lambda}} l_1) \otimes (l_2, \nabla_{\boldsymbol{\lambda}} l_2) \triangleq (l_1 \cdot l_2, l_1 \cdot \nabla_{\boldsymbol{\lambda}} l_2 + l_2 \cdot \nabla_{\boldsymbol{\lambda}} l_1), \tag{13b}$$

and the additive and multiplicative identities are defined as

$$\overline{0} \triangleq (0, \mathbf{0}); \qquad \overline{1} \triangleq (1, \mathbf{0}). \tag{13c}$$

This definition ensures that both operations $\oplus$ and $\otimes$ propagate tuples with as the second element the derivative of the first element. It is also possible to extend this to higher-order derivatives to find higher-order generative scores [15].

The expectation semiring can be used to accumulate any statistic that is additive along the edges of a path and is weighted by the original weight along the path [16, 17]. It was proposed for the expectation step of expectation–maximisation in probabilistic transducers. In theory, training a speech recogniser would be possible with just a forward pass and the expectation semiring. However, this would entail keeping statistics for all speech recogniser parameters relevant to the utterance for all states in the HMM, which requires much memory, and the operations in (13) become expensive. By using the forward–backward algorithm, the order of the semiring can be one lower than using just the forward algorithm [16, 17]. Since in normal speech recogniser training the start and end times are known, the sensible trade-off is to use the forward–backward algorithm.

However, for computing generative scores, this trade-off works out differently. Firstly, the scores for all start and end times are required. Secondly, the generative model for one segment has a small and fixed number of parameters, so memory use is not an issue.

3.2. Implementation

The likelihoods, and their derivatives, have a large dynamic range. It is usual for likelihoods in sequential probabilistic models to be represented by their logarithms. This is possible because the likelihoods are always non-negative. The additional statistics, on the other hand, can be positive or negative. It is possible to separately store the logarithm of the absolute value and the sign [15].

However, in this paper the statistics are derivatives $\nabla_{\boldsymbol{\lambda}} l$ of the weight. Their values are known to be in the order of the weight l itself. If the derivatives are divided by the weights, therefore, they can be represented directly as floating-point numbers without risk of overflow or underflow. The operations $\oplus$ and $\otimes$ must be performed in terms of normalised derivatives as well. Assume two values in this normalised expectation semiring:

$$\sigma_1 \triangleq \left(\log l_1, \frac{\nabla_{\boldsymbol{\lambda}} l_1}{l_1}\right); \qquad \sigma_2 \triangleq \left(\log l_2, \frac{\nabla_{\boldsymbol{\lambda}} l_2}{l_2}\right). \tag{14}$$

How to perform addition and multiplication in the log-domain, for the first elements, is well-known. The second element of $\sigma_1 \otimes \sigma_2$ can be expressed in terms of the elements of σ_1 and σ_2 as

$$\frac{\nabla_{\boldsymbol{\lambda}}(l_1 l_2)}{l_1 l_2} = \frac{l_2 \nabla_{\boldsymbol{\lambda}} l_1 + l_1 \nabla_{\boldsymbol{\lambda}} l_2}{l_1 l_2} = \frac{\nabla_{\boldsymbol{\lambda}} l_1}{l_1} + \frac{\nabla_{\boldsymbol{\lambda}} l_2}{l_2}. \tag{15a}$$

The second element of $\sigma_1 \oplus \sigma_2$ is

$$\frac{\nabla_{\boldsymbol{\lambda}}(l_1 + l_2)}{l_1 + l_2} = \frac{l_1}{l_1 + l_2}\left(\frac{\nabla_{\boldsymbol{\lambda}} l_1}{l_1}\right) + \frac{l_2}{l_1 + l_2}\left(\frac{\nabla_{\boldsymbol{\lambda}} l_2}{l_2}\right). \tag{15b}$$

In the log-domain, a slightly more numerically stable way to compute $l_1/(l_1+l_2)$ is $1/(1+l_2/l_1)$. The result of this can be converted to the normal domain and be used to scale $\nabla_{\boldsymbol{\lambda}} l_1/l_1$, and similar for $\nabla_{\boldsymbol{\lambda}} l_2/l_2$. Then, (15b) can be computed. The resulting weights are tuples of the log-likelihood (not the likelihood) and its derivative, so they can be used directly as the scores in (4b).

Decoding (but not parameter estimation) can be sped up by a constant factor. The trick is to apply the dot product of the score with the parameters in (3) within the semiring. Denote the part of the parameter vector that applies to the derivative (in general, the highest-order derivative) with $\boldsymbol{\alpha}_{\nabla}$. Instead of the tuple as in (14), with the normalised derivative, the tuple becomes

$$\sigma \triangleq \left(\log l, \frac{\boldsymbol{\alpha}_{\nabla}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}} l}{l}\right). \tag{16}$$

Notice that the last element of the tuple now is a scalar value, which makes the operations $\oplus$ and $\otimes$ (which are similar to (15) and straightforward to derive) a constant factor faster.

4. EXPERIMENTS

The feature extraction process described in this paper was tested in a log-linear model on a small, noise-corrupted corpus: AURORA 2. This makes it possible to test the interaction with noise compensation methods. The task uses a small vocabulary and no language model, which makes experiments without such optimisations as pruning possible. AURORA 2 [18] is a standard digit string recognition task. The generative model has whole-word HMMs with 16 states and 3 components per state. The number of HMM parameters is 46 732. The HMMs are compensated with unsupervised vector Taylor series (VTS) compensation as in [5]. The HMM parameters to derive features for the discriminative model are trained on clean data.

SNR	Test set								
	A			B			C		
(dB)	HMM	l	$l, \nabla l$	HMM	l	$l, \nabla l$	HMM	l	$l, \nabla l$
20	1.69	1.43	1.01	1.46	1.20	0.80	1.57	1.42	0.96
15	2.36	1.95	1.28	2.37	1.82	1.34	2.47	2.18	1.72
10	4.39	3.62	2.62	4.12	3.22	2.53	4.49	3.82	2.80
05	11.20	8.94	7.48	10.05	7.89	6.74	10.69	8.76	7.86
00	29.54	23.25	21.57	27.54	22.18	20.84	28.41	23.96	23.31
00–20	9.84	7.84	6.79	9.11	7.26	6.45	9.53	8.03	7.33

Table 1. WERs for decoding with the expectation semiring.

With features consisting of just word log-likelihoods, the discriminative model has 13 parameters, corresponding to the log-likelihoods of the 13 words (11 digits plus "sil" and "sp"). In first-order score-spaces the derivatives of the log-likelihood are computed as in section 3.1 and appended (like in [5] the data are whitened separately for each Gaussian before computing the derivative). Only derivatives of the compensated means are used, since including variances led to rapid over-fitting. The number of parameters was 21 554. Second-order score-spaces resulted in generalisation problems because of the small training set, and initial experiments did not yield improvements over first-order score-spaces.

The discriminative models were initialised to use just the likelihoods from the generative model. They were then trained with a minimum Bayes risk criterion as in [19]. This used a large lattice with many, but not all, segmentations for the numerator and denominator. Test set A was used as the validation set to stop training.

Word error rates for the experiments are in Table 4. Apart from numerical differences, they are the same as in [6]. However, there the derivatives were recomputed for every segment, whereas in this paper they are found with the expectation semiring as in section 3.1, which even with the unoptimised implementation is much faster.

With just likelihood features ("l"), the log-linear model is closely related to the HMM. The difference is merely that within words, all paths are taken into account, and that there are word-specific discriminatively-trained parameters, effectively scaling factors. This yields consistent improvements of 10–15 % on test sets A and B, with greater improvements at lower signal-to-noise ratios. Adding derivative features ("$l, \nabla l$") introduces longer-range dependencies that break the Markov assumption. This improves recognition consistently. Where discrimination relies most on modelling the speech accurately, at higher signal-to-noise ratios, this helps most, with 18–33 % relative improvement at 10–20 dB.

Using the optimal segmentation instead of the HMM segmentation accounts for around 5 % of the improvement. Features for large-vocabulary systems will be extracted per phone, like in [20], so that the segmentation is likely to have greater impact on performance.

5. CONCLUSION

This paper has introduced a general method for introducing longer-range features from segments of sequence data originally modelled with a Markov model. It uses the log-likelihood and appends the derivatives with respect to the parameters. By formulating the Markov model as a weighted finite-state automaton with weights in the *expectation semiring*, the features can be extracted efficiently. For speech recognition, with a decoding algorithm that finds the optimal segmentation into words, the features are found in amortised constant time. On the standard noise-corrupted AURORA 2 task, this leads to substantial improvements.

6. REFERENCES

[1] Dan Gillick, Larry Gillick, and Steven Wegmann, "Don't multiply lightly: Quantifying problems with the acoustic model assumptions in speech recognition," in *Proceedings of ASRU*, 2011.

[2] Sunita Sarawagi and William W. Cohen, "Semi-markov conditional random fields for information extraction," in *Proceedings of NIPS*, 2004.

[3] Martin Layton, *Augmented Statistical Models for Classifying Sequence Data*, Ph.D. thesis, Cambridge University, 2006.

[4] Geoffrey Zweig and Patrick Nguyen, "A segmental CRF approach to large vocabulary continuous speech recognition," in *Proceedings of ASRU*, 2009.

[5] M. J. F. Gales and F. Flego, "Discriminative classifiers with adaptive kernels for noise robust speech recognition," *Computer Speech and Language*, vol. 24, no. 4, pp. 648–662, 2010.

[6] A. Ragni and M. J. F. Gales, "Inference algorithms for generative score-spaces," in *Proceedings of ICASSP*, 2012, pp. 4149–4152.

[7] Tommi Jaakkola and David Haussler, "Exploiting generative models in discriminative classifiers," in *Proceedings of NIPS*, 1998.

[8] M. J. F. Gales, S. Watanabe, and E. Fosler-Lussier, "Structured discriminative models for speech recognition: An overview," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 70–81, Nov 2012.

[9] R. C. van Dalen, A. Ragni, and M. J. F. Gales, "Efficient decoding with continuous rational kernels using the expectation semiring," Tech. Rep. CUED/F-INFENG/TR.674, Cambridge University Engineering Department, Feb 2012.

[10] Björn Hoffmeister, Georg Heigold, Ralf Schlüter, and Hermann Ney, "WFST enabled solutions to ASR problems: Beyond HMM decoding," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 551–564, Feb 2012.

[11] Mehryar Mohri, "Semiring frameworks and algorithms for shortest-distance problems," *Journal of Automata, Languages and Combinatorics*, vol. 7, no. 3, pp. 321–350, 2002.

[12] Barak A. Pearlmutter and Jeffrey Mark Siskind, "Lazy multivariate higher-order forward-mode AD," in *Proceedings of the annual symposium on Principles of programming languages*, 2007, pp. 155–160.

[13] Mehryar Mohri, "Weighted automata algorithms," in *Handbook of Weighted Automata*, Manfred Droste, Werner Kuich, and Heiko Vogler, Eds., pp. 213–254. Springer, 2009.

[14] Jason Eisner, "Parameter estimation for probabilistic finite-state transducers," in *Proceedings of the Annual Meeting on Association for Computational Linguistics*, 2002, pp. 1–8.

[15] Zhifei Li and Jason Eisner, "First- and second-order expectation semirings with applications to minimum-risk training on translation forests," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[16] Jason Eisner, "Expectation semirings: Flexible EM for finite-state transducers," in *Proceedings of the ESSLLI Workshop on Finite-State Methods in Natural Language Processing (FSMNLP)*, 2001.

[17] Georg Heigold, Thomas Deselaers, Ralf Schlüter, and Hermann Ney, "Modified MMI/MPE: A direct evaluation of the margin in speech recognition," in *International Conference on Machine Learning*, Helsinki, Finland, July 2008, pp. 384–391.

[18] Hans-Günter Hirsch and David Pearce, "The AURORA experimental framework for the performance evaluation of speech recognition systems under noise conditions," in *Proceedings of ASR*, 2000, pp. 181–188.

[19] A. Ragni and M.J.F. Gales, "Structured discriminative models for noise robust continuous speech recognition," in *Proceedings of ICASSP*, 2011.

[20] A. Ragni and M.J.F. Gales, "Derivative kernels for noise robust ASR," in *Proceedings of ASRU*, 2011.