UNIVERSITY OF
CAMBRIDGE
Department of Engineering

Generative Kernels and Score-Spaces for Classification of Speech: Progress Report III

R. C. van Dalen
rcv25@cam.ac.uk

J. Yang
jy308@cam.ac.uk

M. J. F. Gales
mjfg@eng.cam.ac.uk

Technical Report CUED/F-INFENG/TR.699

May 2015

This is the third and final progress report for EPSRC Project EP/I006583/1 (Generative Kernels and Score Spaces for Classification of Speech) within the Global Uncertainties Programme. This project combines the current generative models developed in the speech community with discriminative classifiers. An important aspect of the approach is that the generative models are used to define a score-space that can be used as features by the discriminative classifiers. This report discusses progress in three areas. First, efficiently computing segmental features for segments that begin *or* end at adjacent times. Second, finding the exact word error for discriminative training of acoustic models. Third, training infinite log-linear models with a criterion for the whole model. Experiments use a log-linear model with segmental features derived from a hidden Markov model with neural-network output distributions.

Contents

1 Introduction

This is the third and final progress report for EPSRC Project EP/I006583/1 (Generative Kernels and Score Spaces for Classification of Speech) within the Global Uncertainties Programme.

The aim of this project is to significantly improve the performance of automatic speech recognition systems across a wide-range of environments, speakers and speaking styles. The performance of state-of-the-art speech recognition systems is often acceptable under fairly controlled conditions and where the levels of background noise are low. However for many realistic situations there can be high levels of background noise, for example in-car navigation, or widely ranging channel conditions and speaking styles, such as observed on YouTube-style data. This fragility of speech recognition systems is one of the primary reasons that speech recognition systems are not more widely deployed and used. It limits the possible domains in which speech can be reliably used, and increases the cost of developing applications as systems must be tuned to limit the impact of this fragility. This includes collecting domain-specific data and significant tuning of the application itself.

The vast majority of research for speech recognition has concentrated on improving the performance of systems based on hidden Markov models (HMMs). HMMs are an example of a generative model and are currently used in state-of-the-art speech recognition systems. A wide number of approaches have been developed to improve the performance of these systems under changes of speaker and noise. Despite these approaches, systems are not sufficiently robust to allow speech recognition systems to achieve the level of impact that the naturalness of the interface should allow.

One of the major problems with applying traditional classifiers, such as support vector machines, to speech recognition is that data sequences of variable length must be classified. This project combines the current generative models developed in the speech recognition community with discriminative classifiers used both in speech processing and in machine learning. Figure 1 gives a schematic overview of the approach that this project takes. The shaded part of the diagram indicates the generative model of a state-of-the-art speech recogniser. In this project, the generative models are used to define a score-space. These scores then form features for the discriminative classifiers. This

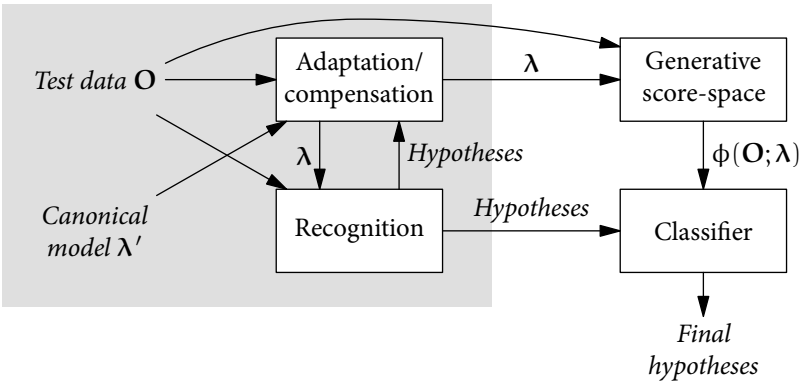


Figure 1 *Flow diagram of the project plan. The shaded region encompasses the components of a state-of-the-art speech recogniser.*

approach has a number of advantages. It is possible to use current state-of-the-art adaptation and robustness approaches to compensate the acoustic models for particular speakers and noise conditions. As well as enabling any advances in these approaches to be incorporated into the scheme, it is not necessary to develop approaches that adapt the discriminative classifiers to speakers, style and noise. Using generative models also allows the dynamic aspects of speech data to be handled without having to alter the discriminative classifier. The final advantage is the nature of the score-space obtained from the generative model. Generative models such as HMMs have underlying conditional independence assumptions that, whilst enabling them to efficiently represent data sequences, do not accurately represent the dependencies in data sequences such as speech. The score-space associated with a generative model does not have the same conditional independence assumptions as the original generative model. This allows more accurate modelling of the dependencies in the speech data.

While the project was taking place, the state of the art in speech recognition changed. A current-day speech recogniser still uses a hidden Markov model, but neural network are used in addition to or instead of the traditional Gaussian mixture models. In *tandem* systems, outputs from a neural networks are appended to each feature ector, and otherwise the system is a standard GMM-HMM system. This has the advantage that many methods for adaptation still apply. It also means that generative score-spaces designed for GMM-HMMs can be used. The second type of model is a *hybrid* system. Its output distributions are not Gaussian mixture models, but neural network outputs. The neural network maps from the acoustic input to a distribution over HMM states. This "posterior", given the acoustic input, is then converted to something resembling an HMM output density using the state prior. If is possible to extract some types of score-spaces from this sort of model, and this will be discussed. However, not all types of score-spaces translate. The conclusion will look forwards to work in this area.

This report will report on work performed since the previous two progress reports (van Dalen *et al.* 2012*b*; 2013*b*). These covered the three components in figure 1: adaptation/compensation, score-space computation, and classifiers.

In the first component, Progress Report I introduced a novel variational method for compensating HMM speech recognisers for noise, which does not assume that the resulting distribution is Gaussian (van Dalen and Gales 2011).

The second component is score-space computation, the efficiency of which is important. Progress Reports I and II presented aspects of a method to compute scores from generative models for all segmentations in amortised constant time (van Dalen *et al.* 2012*a*; 2013*a*). Section 2 of this report will generalise this method and show exactly what property makes it so efficient by analysing the feature extraction as a computation on a monoid (van Dalen and Gales 2013). It will then sketch how this insight can be exploited to extract features in score-spaces derived from HMM systems that use neural networks.

The third and main component is the discriminative classifiers. These are trained with a discriminative criterion. A popular criterion, the minimum Bayes' risk, is usually approximated using fixed segmentations of words or phones. This is impossible with segmental models where the criterion needs to marginalise out over all segmentations. Section 3 of this report therefore introduces a method to mark lattices with the exact word or phone error which is much more memory-efficient than the state of the art, and can error-mark much larger lattices.

The actual discriminative models are of interest in this project. Progress Report I

reported on log-linear models, both for acoustic modelling (work related to the project, Ragni and Gales 2011*b*;*a*) and language modelling. Another model of interest is large-margin classifiers. Progress Reports I and II discussed use of the structured support vector machine (SVM) (work related to the project, Zhang and Gales 2011*b*;*a*) and the kernelised structured SVM (Zhang and Gales 2013).

SVMs or log-linear models can be used as experts in a Bayesian mixture of experts. Progress Report I discussed Bayesian non-parametric approach to classification using infinite Gaussian mixture models. Progress Report II applied the infinite SVM to speech recognition (Yang *et al.* 2013). This report introduces sequence classifiers as experts in the infinite mixture. Section 4 of this report will discuss how to train the whole model with either a Bayesian criterion, or a Bayesian large-margin criterion (Yang *et al.* 2015). In practice, it is possible to train with a per-expert large-margin criterion, which leads to infinite structured SVMs (Yang *et al.* 2014).

1.1 Log-linear models

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.

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., Layton 2006; Zweig and Nguyen 2009; Gales and Flego 2010). 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.

Discriminative models (Gales *et al.* 2012) 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 $\boldsymbol{w}$ given the observation sequence $\mathbf{O}$. Here, each of the elements w_i of $\boldsymbol{w}$ is equal to one element v_j from the vocabulary $\boldsymbol{v}$. To enable compact discriminative models to be trained, the input sequence must be segmented into, e.g., words. Let $\mathbf{s} = \{s_i\}_{i=1}^{|\boldsymbol{w}|}$ denote a segmentation. This paper will use a log-linear model that gives the joint probability of the word sequence and segmentation given the observation:

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

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

The score function will in this report be assumed to contain acoustic features, extracted with function $\boldsymbol{\phi}_a$, and a language model score, extracted with function $\boldsymbol{\phi}_l$:

$$\boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s}) \triangleq \begin{bmatrix} \boldsymbol{\phi}_a(\mathbf{O}, \boldsymbol{w}, \mathbf{s}) \\ \boldsymbol{\phi}_l(\boldsymbol{w}) \end{bmatrix}. \tag{2}$$

The part of the distribution related to the acoustic model then factorises over the segments of the audio, i.e. the score function is a sum of scores for each segment:

$$\boldsymbol{\phi}_a(\mathbf{O}, \boldsymbol{w}, \mathbf{s}) \triangleq \sum_i \boldsymbol{\phi}_a(\mathbf{O}_{s_i}, w_i), \tag{3}$$

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. For clarity, assume there is no language model. Then,

$$\begin{aligned} \arg\max_{\boldsymbol{w},\mathbf{s}} P(\boldsymbol{w}, \mathbf{s}|\mathbf{O}; \boldsymbol{\alpha}) &= \arg\max_{\boldsymbol{w},\mathbf{s}} \frac{1}{Z(\mathbf{O}, \boldsymbol{\alpha})} \exp\left(\boldsymbol{\alpha}^T \boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})\right) \\ &= \arg\max_{\boldsymbol{w},\mathbf{s}} \left(\boldsymbol{\alpha}^T \boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})\right) \\ &= \arg\max_{\boldsymbol{w},\mathbf{s}} \sum_i \boldsymbol{\alpha}^T \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i). \end{aligned} \tag{4}$$

To perform this type of decoding, features for all possible segments must be available.

2 Monoids for segmental features

Ragni and Gales (2012); Van Dalen *et al.* (2013*a*) proposed an efficient method for extracting features for one word (or other unit of speech) from every possible contiguous segment of audio. The specific features are in *generative score-spaces*. These contain log-likelihoods for word HMMs (Ragni and Gales 2012) and their derivatives (van Dalen *et al.* 2013*a*). The number of possible segments is quadratic in the length of the audio. By re-using part of the computation, the time needed to extract the features for all segments also becomes quadratic. If features for all segments are needed, feature extraction therefore takes average constant time.

However, a realistic speech recogniser will not need features for all these segments, because it will approximate its hypothesis space. One approximation scheme is to take hypotheses from an existing decoder and to compute features only for segments around ones contained in these hypotheses. It is therefore an interesting question what types of features can be efficiently computed for segments that can overlap in different ways. Of particular interest are segments that start and end around two times given by the arc in a lattice.

There are two aspects to decoding with a segmental log-linear model like in (5):

$$\arg\max_{\boldsymbol{w},\mathbf{s}} P(\boldsymbol{w}, \mathbf{s}|\mathbf{O}; \boldsymbol{\alpha}) = \arg\max_{\boldsymbol{w},\mathbf{s}} \sum_i \boldsymbol{\alpha}^T \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i). \tag{5}$$

First, the score $\boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i)$ must be extracted for all possible words and segments. Second, the best combination of word sequence and segmentation must be found. Assuming the language model constant, the latter task takes $\Theta(T^2)$ time Ragni and Gales

(2012). The former task, of extracting $\Theta\big(|\nu| \cdot T^2\big)$ scores, forms the bottleneck in performance.

This report will consider the score for one word (or sub-word unit). To highlight the aspects that influence the efficiency of computing these scores, ϕ will be decomposed into two parts. First, a segmental *feature* is computed, and then it is converted into a *score*. The features are representations of segments of observations that allow them to be re-used; the scores are (often trivially) derived from the features. This report therefore focuses on the structure of the features.

First, section 2.1 will re-analyse the method in Ragni and Gales (2012); van Dalen *et al.* (2013*a*) as computing a function of the form

$$\boldsymbol{\phi}(\mathbf{O}_{\tau:t}, w_i) = h\big(g(g(\ldots g(g(\lambda, \mathbf{o}_\tau), \mathbf{o}_{\tau+1}) \ldots, \mathbf{o}_{t-1}), \mathbf{o}_t)\big). \tag{6a}$$

Here, h is the word score function, and g a function that extends a feature by one observation $\mathbf{o}$. To compute the feature for a segment, g is called recursively $t-\tau$ times. λ, the feature for the empty segment, is the base case for the recursion. (This will be formulated in terms of higher-order function "fold".) This recursion limits re-use of computation to segments that start at the same time τ.

In contrast, section 2.2 will introduce a less general but more generally applicable type of feature. The computation is of the form

$$\boldsymbol{\phi}(\mathbf{O}_{\tau:t}, w_i) = h\big(f(\mathbf{o}_\tau) \odot f(\mathbf{o}_{\tau+1}) \odot \ldots \odot f(\mathbf{o}_t)\big), \tag{6b}$$

where f extracts a feature for one observation, and the operation $\odot$ combines two of these features. $\odot$ is required to be associative over the features, i.e., the result must be the same whichever order the features are combined in. This freedom allows many different pruning schemes.

This section will re-analyse the method in van Dalen *et al.* (2013*a*) to highlight what makes it efficient. Computing the feature for a word can be phrased in terms of the primitive that pure functional programming languages use for iteration: the higher-order function "fold" (section 2.1). The method assumes a schedule that progressively computes features for all segments that start at the same time. This paper will then introduce a related form of feature. Used with the same schedule, it is as time-efficient, but can also be used within different types of schedules (section 2.2). These features are in a *monoid*. This makes it possible to combine two features for any two consecutive segments into one value for the segment encompassing both.

2.1 Uni-directional segmental features

Ragni and Gales (2012); van Dalen *et al.* (2013*a*) proposed a method for computing segmental features in average constant time if features for all segments are needed. This section will discuss this method at a high level. It will use a form that highlights its limitations and allows section 2.2 to relate this to a more general form. Section 2.1.1 will explain how this description maps to the specific instantiation in Ragni and Gales (2012); van Dalen *et al.* (2013*a*).

Assume that the observations $\mathbf{o}_t \in \mathcal{O}$ (for example, vectors with Mel-frequency cepstral coefficients) for an utterance of length T are available as $[\mathbf{o}_1 \ldots \mathbf{o}_T]$. Segmental features are to be extracted from each possible segment of consecutive observations.

Two functions will act on this: first a function g that extends a feature by one observation, and a function h that computes a score given such a feature.

The features are in a feature space $\mathcal{F}$. The feature for the empty segment, which contains 0 observations, is defined as $\lambda \in \mathcal{F}$. Features can be extended in one direction. This is performed by the function

$$g : \mathcal{F} \times \mathcal{O} \to \mathcal{F}, \tag{7}$$

which takes a feature for a segment, and the next observation, and returns the feature for the segment extended with the observation.

Thus, the feature extracted from the segment containing just observation $\mathbf{o}_1$ extends the feature for the empty segment with

$$f_1 \triangleq g(\lambda, \mathbf{o}_1)\,. \tag{8a}$$

To compute the feature for observations $[\mathbf{o}_1, \mathbf{o}_2]$, the function is applied on this result again:

$$f_{1:2} \triangleq g(g(\lambda, \mathbf{o}_1), \mathbf{o}_2)\,. \tag{8b}$$

As an example, assume an observation sequence consisting of four elements $\mathbf{O} = [\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4]$. The feature for the whole sequence is computed with four consecutive applications of the function g:

$$f_{1:4} \triangleq g(g(g(g(\lambda, \mathbf{o}_1), \mathbf{o}_2), \mathbf{o}_3), \mathbf{o}_4) = g(g(f_{1:2}, \mathbf{o}_3), \mathbf{o}_4)\,. \tag{8c}$$

The interesting aspect that is clear from (8c) is that computation can be shared between segments. The feature for sub-segments, like $f_{1:2}$ in (8c), has already been computed in (8b).

To exploit this when features for all segments are required, it is useful to write them in terms of higher-order functions "fold" and "scan" (see van Dalen and Gales 2013). The feature for the whole sequence can be written as

$$f_{1:4} \triangleq \mathrm{fold}(g, \lambda, [\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4])\,. \tag{9}$$

The number of applications of the function g is equal to the length of the sequence.

It is possible to avoid duplication of computation by computing features for all segments starting at the same time at once. This can be expressed with the higher-order functional primitive "scan". It performs the same computation as "fold" but returns all intermediate results as a sequence:

$$\mathrm{scan}(g, \lambda, [\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4]) = [\lambda, f_1, f_{1:2}, f_{1:3}, f_{1:4}]\,. \tag{10}$$

The number of applications of g is, just like in (9), equal to the number of elements of the sequence. This means that the average number of applications of g to compute a feature vector for each segment starting at a given time is constant.

To compute features for all segments, the recursion is started at each possible start time:

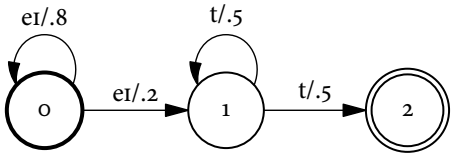
$$\mathrm{scan}(g, \lambda, [\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4]) = [\lambda, f_1, f_{1:2}, f_{1:3}, f_{1:4}]\,; \tag{11a}$$

$$\mathrm{scan}(g, \lambda, [\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4]) = [\lambda, f_2, f_{2:3}, f_{2:4}]\,; \tag{11b}$$

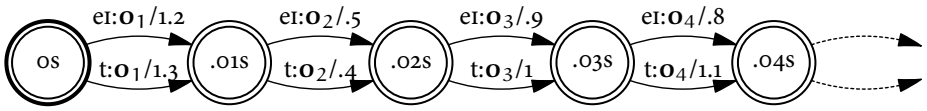
$$\mathrm{scan}(g, \lambda, [\mathbf{o}_3, \mathbf{o}_4]) = [\lambda, f_3, f_{3:4}]\,; \tag{11c}$$

$$\mathrm{scan}(g, \lambda, [\mathbf{o}_4]) = [\lambda, f_4]\,; \tag{11d}$$

$$\mathrm{scan}(g, \lambda, []) = [\lambda]\,. \tag{11e}$$



(a) An automaton that generates sub-word symbols.



(b) An automaton that converts sub-word symbols into observations, weighted by their output weights.

Figure 2 *The two automata that are composed to yield the automaton in figure 3.*

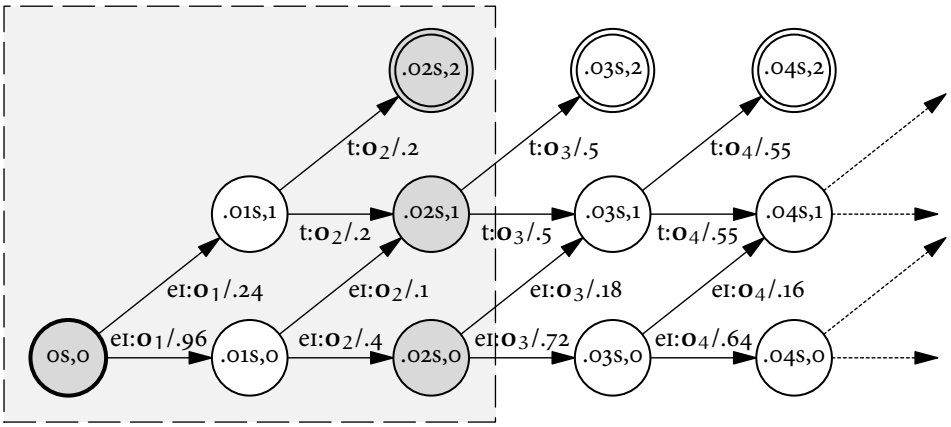


Figure 3 *A trellis expressed as a weighted finite-state automaton. The feature vector* $\mathbf{f}_{1:2}$ *contains the total weights from the state "os,0" to each state at 0,02 s (highlighted).*

This computes the features for all $\Theta(T^2)$ segments in $\Theta(T^2)$ time, which is average constant time.

The word score function,

$$h : \mathcal{F} \to \mathcal{S}, \tag{12}$$

takes a feature and returns a score in $\mathcal{S}$ that can be integrated in, for example, a log-linear model. This function is applied separately to each feature in (11).

2.1.1 Instantiation: generative score-spaces

The original instantiation of the algorithm to extract segmental features (Ragni and Gales 2012; van Dalen *et al.* 2013*a*) computes features in a generative score-space. These contain log-likelihoods of the data with respect to the parameters of a generative model (an HMM, in this case) and their derivatives.

First consider the likelihood of an HMM. The following represents HMMs as weighted finite-state automata (WFSAs). A more traditional representation is possible, but requires additional matrix multiplications. The FSA representation also draws out the symmetry of the model, which will become important in section 2.2.1. Composing the weighted finite-state automata in figure 2 on the preceding page, one producing a weighted symbol sequence, and another converting symbols into observations, produces the trellis in figure 3 on the facing page. In this composed automaton, time proceeds from left to right; HMM states are laid out vertically. The weights in the trellis are products of transition weights in figure 2a and the output weights in figure 2b. The likelihood for a sequence of observations is the sum over all paths from the highlighted start state "os,o" to the final state (with double line) corresponding to the last observation in the segment. There is one start state, and many final states, because the start time of the segment is fixed, while the end time is flexible. (In section 2.2.1, both start and end times will be flexible.) To compute the likelihoods efficiently, the original method applies the forward algorithm. This algorithm takes a vector $\mathbf{f}$ of "forward weights" for all possible states of a finite-state automaton at time $t-1$.

The initial feature $\boldsymbol{\lambda}$, for the empty segment, contains 1 for the start state and 0 for all other states. For the three-state HMM in the example in figure 3,

$$\lambda \triangleq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \tag{13a}$$

Applying the function g on current weights $\mathbf{f}$ and new observation $\mathbf{o}_t$ is implemented as one step of the forward algorithm. One step of the forward algorithm takes the forward weights for time $t-1$ (say, 0.01 s) and computes the weights for time t (say, 0.02 s). The value of $\mathbf{o}_t$ is necessary to compute the weights on the transitions for the next step in the automaton. The first evaluation of g generates the forward weights at 0.01 s, and the second evaluation the weights at 0.02 s:

$$\mathbf{f}_1 = g(\lambda, \mathbf{o}_1) = \begin{bmatrix} 0.96 \\ 0.24 \\ 0 \end{bmatrix}; \qquad \mathbf{f}_{1:2} = g(\mathbf{f}_1, \mathbf{o}_2) = \begin{bmatrix} 0.384 \\ 0.144 \\ .048 \end{bmatrix}. \tag{13b}$$

The second result, $\mathbf{f}_{1:2}$, is indicated in the figure. This feature vector contains the sum of the weights up to each of the highlighted states at time 0.02 s.

The elements of $\mathbf{f}$ can be scalars, as in this example. In that case, the standard forward algorithm for computing the likelihood of a segment is used. This is exploited by Ragni and Gales (2012) to efficiently produce likelihoods for all segments of audio.

In van Dalen *et al.* (2013*a*), the elements of $\mathbf{f}$ are in the expectation semiring. By running the forward algorithm using generalised $+$ and $\times$ operations, not only the likelihoods but also their derivatives are produced.

The word score function h takes a vector $\mathbf{f}$ and extracts the element that indicates the weight in the final state (in the example, the third element of the vector). It then converts it into a score, which is then used in the log-linear model.

2.2 Monoid features

The previous section has discussed a general class of segmental features which can be efficiently computed if all features for all possible segments are required. Though fea-

ture extraction is then feasible for small-vocabulary recognisers, for larger systems additional approximations will be necessary. This could be done by producing a set of hypotheses with a faster recogniser and rescore them using segmental features. In that scenario, segmental features are required for segments with start and end times around those from the hypothesis set. The feature extraction process in section 2.1 allows flexibility only around the end time of segments, not around start times.[1]

This section will therefore propose a related class of features that are more flexible. Section 2.2.1 will show how scores in likelihood score-spaces can be computed in this framework. Section 2.2.2 will generalise this to HMM-like automata with weights in any semiring.

The requirement is made that features for two subsequent segments can be combined into the feature for the joint segment. The process of computing word scores from segments of observations is therefore split up into three functions that act on them consecutively.

First, the function $f(\mathbf{o}_t)$ converts the observation into a segmental feature for the segment $[\mathbf{o}_t]$. Second, the function $\odot$ combines two of these features into a feature representing the combined segment. Third, the function h computes a score for a word (or phone) given a segmental feature f.

The first step is to convert a sequence of observations into a sequence of features. This works by applying the function

$$f : \mathcal{O} \to \mathcal{F}. \tag{14}$$

Applying f to each observation can be formulated with the higher-order function "map" (see van Dalen and Gales 2013):

$$\mathrm{map}(f,[\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4]) = [f_1, f_2, f_3, f_4]\,, \tag{15}$$

where f_t is a shorthand for $f(\mathbf{o}_t)$. f_t is a feature for the segment $[\mathbf{o}_t]$, of length 1.

The second, and most interesting, step is to combine two features for consecutive segments with the function

$$\odot : \mathcal{F} \times \mathcal{F} \to \mathcal{F}. \tag{16}$$

The feature for the segment $[\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4]$ can then be computed as

$$f_1 \odot f_2 \odot f_3 \odot f_4. \tag{17}$$

To speed up the computation, it is useful for features for shared sub-segments to be reusable. The features in section 2.1 are only shared between features for segments with the same start time. For more flexible re-use to be possible, it is useful to allow the computation in (17) to be performed in any order. To see why the freedom of evaluation order is important, consider an example. If a segment is hypothesised to start at $\mathbf{o}_1$ or $\mathbf{o}_2$ and end at $\mathbf{o}_3$ or $\mathbf{o}_4$, and $f_{2:3}$ has been computed, the necessary features can be derived from it efficiently:

$$f_{1:3} = f_1 \odot f_{2:3}; \tag{18a}$$

$$f_{2:4} = f_{2:3} \odot f_4; \tag{18b}$$

$$f_{1:4} = (f_1 \odot f_{2:3}) \odot f_4 = f_{1:3} \odot f_4. \tag{18c}$$

[1] If the backward algorithm is used instead of the forward algorithm, the start time is flexible, but the end time fixed.

Here, $f_{2:3}$ can be cached and re-used in (18a) and (18b), and $f_{1:3}$ in (18c).

To ensure that the segmental feature is the same whatever order the sub-segments are combined in, $\odot$ must be associative. That is, for any $f, f', f'' \in \mathcal{F}$,

$$(f \odot f') \odot f'' = f \odot (f' \odot f''). \tag{19}$$

This implies that the set $\mathcal{F}$ of features is a *monoid* with $\odot$ as its operation.

In general, a segmental feature can be computed from the features for different sub-segments. One possible order of evaluation goes through the observations consecutively. This is the same order as the one in section 2.1. It can be written

$$f_{1:4} = \text{fold1}(\odot, [f_1, f_2, f_3, f_4])\,. \tag{20}$$

In section 2.1, the same order of evaluation was produced with the "fold" function (which requires an extra argument, the initial state). "fold1" computes a "skewed reduction". Since the $\odot$ operation is associative, the "fold1" function can be generalised to a function defined as computing the same value as "fold1", but with the evaluations of $\odot$ in any order. In this report, this function is called "reduce":

$$f_{1:4} = \text{reduce}(\odot, [f_1, f_2, f_3, f_4])\,. \tag{21}$$

If features for all segments are required, then expressions analogous to (11) can be used:

$$\text{scan1}(\odot,[f_1, f_2, f_3, f_4]) = [f_1, f_{1:2}, f_{1:3}, f_{1:4}]\,; \tag{22a}$$
$$\text{scan1}(\odot,[f_2, f_3, f_4]) = [f_2, f_{2:3}, f_{2:4}]\,; \tag{22b}$$
$$\text{scan1}(\odot,[f_3, f_4]) = [f_3, f_{3:4}]\,; \tag{22c}$$
$$\text{scan1}(\odot,[f_4]) = [f_4]\,. \tag{22d}$$

This still computes the feature values for all $\Theta(T^2)$ segments in $\Theta(T^2)$ time, which is constant time on average.

The third step is, as in section 2.1.1, to apply the word score function:

$$h : \mathcal{F} \to \mathcal{S}. \tag{23}$$

This function takes a feature and returns a score in $\mathcal{S}$ that can be integrated in, for example, a log-linear model. This function is applied separately to each feature in (22).

2.2.1 Instance: likelihood score-spaces

Section 2.1.1 has described a method introduced by Ragni and Gales (2012); van Dalen *et al.* (2013*a*) for computing features in "generative score-spaces" derived from HMMs. This section will sketch how this can be extended to monoid features, and how the observations and the functions f, $\odot$, and h are then defined.

At first, assume that log-likelihood score-spaces are used. This means that the word scores are given by the log-likelihoods of HMMs. The following discussion will again represent HMMs as weighted finite-state automata (WFSAs). This draws out the symmetry of the model, which is important since the features must be extensible in two directions. The rows and columns of the matrix correspond to all states in a word (or sub-word) HMM.

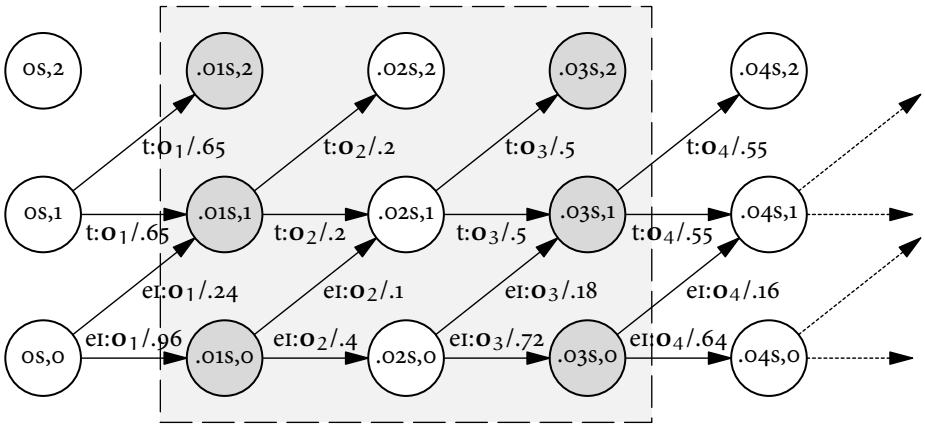


Figure 4 *A trellis like in figure 3, but without a specific start state. The feature* $f_{2:3}$ *contains total weights between each pair of states at 0.01 s and at 0.03 s (highlighted).*

Figure 4 contains a trellis similar to the one in figure 3 on page 8. However, this one is not specific to a start time. Each feature f is a square matrix indicating the total weight going from the state indicated by the row index at one time to the state indicated by the column index at one other time. The entries of matrix $f(\mathbf{o}_t)$ correspond to the weights on the transitions between two consecutive times. For example, the feature $f_2 = f(\mathbf{o}_2)$ is derived from the weights between 0.01 s and 0.02 s:

$$f(\mathbf{o}_2) = \begin{bmatrix} 0.4 & 0.1 & 0 \\ 0 & 0.2 & 0.2 \\ 0 & 0 & 0 \end{bmatrix}. \tag{24a}$$

The way to read this matrix is

		To state		
		0	1	2
From state	0	0.4	0.1	0
	1	0	0.2	0.2
	2	0	0	0

. (24b)

In this particular example, the feature matrix is for one observation. In general, this matrix gives the weights for combinations of start and end states for a specific segment, starting at the start time and at the end time. The trellis diagram in figure 4 illustrates $f_{2:3}$, which contains weights for each pair of states at 0.01 s and states at 0.03 s. To derive the total weight between each of these pairs from f_2 (in (24a)) and f_3, the connecting states, at 0.02 s, must be summed out. Expressed in terms of the elements of the matrices,

$$(f_2 \odot f_3)_{ij} = \sum_k f_{2,ik} \cdot f_{3,kj}. \tag{25a}$$

In general, the monoid operation on two features is therefore a matrix multiplication:

$$f \odot f' \triangleq f \cdot f'. \tag{25b}$$

Matrix multiplication is associative, so this adheres to the requirement for being a monoid.

To compute the word score for any feature f, the element of the matrix in (24a) representing the start and end state, in the example at position $(0, 2)$, must be selected. In general, word w will have a vector of WFSA start weights $\boldsymbol{\lambda}$ and a vector of end weights ρ. For the WFSA in figure 2a on page 8, they are

$$\boldsymbol{\lambda} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}; \qquad \rho = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{26}$$

The word score can then be computed as

$$h(f) \triangleq \boldsymbol{\lambda}^{\mathsf{T}} \cdot f \cdot \rho. \tag{27}$$

2.2.2 Features from finite-state models

So far, the entries of the matrices have been assumed to be just likelihoods. However, it is possible to accumulate other types of features. One example was shown in section 2.1.1, where the score required not only likelihoods, but also their derivatives (van Dalen *et al.* 2013*a*). Accumulating these worked with the forward algorithm, but the weights on the finite-state automata were in the expectation semiring, instead of being just scalars. The expectation semiring, as does every other semiring, replaces normal addition and multiplication by operations $\oplus$ and $\otimes$.

The monoid features discussed so far have been a matrix with scalar entries derived from an HMM. Now the entries can be generalised to be in the expectation semiring. The monoid operation between two features then is a generalised matrix multiplication, where the entries of the matrix are not just scalars as in (25a), but in a semiring:

$$(f \odot f')_{ij} = \bigoplus_k f_{ik} \otimes f'_{kj}. \tag{28}$$

The function $f(\cdot)$, which computes a feature from one observation, produces a matrix with entries in the expectation semiring. For generative score-spaces, the first element of each of the values in the semiring is the same likelihood, and the second element the partial derivatives with respect to the parameters of the generative model. However, other types of entries are also possible.

It is well-known (and shown in van Dalen and Gales 2013) that square matrices form a monoid under matrix multiplication iff its elements are in a semiring. Therefore, any finite-state model with the same shape as an HMM — one state model generating symbols, and one linear transducer from symbols to the observations — with weights in any semiring can be used to extract features in a monoid.

This opens up the possibility for many types of segmental features. There is no requirement for the semiring to contain a likelihood or anything similar. Of particular interest are features based on neural networks. Features that, just the likelihood of HMMs, are additive over paths of the finite-state automaton can then be extracted in average constant time if they are computed for all possible segments of audio. Additionally, monoid features for consecutive segments can be combined straightforwardly, which allows flexibility when pruning is used during decoding.

2.3 Conclusion

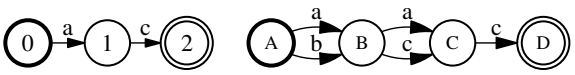
This section has discussed a general class of segmental features for speech recognition that are efficient to compute. The new features are as efficient as features used in Ragni and Gales (2012); van Dalen *et al.* (2013*a*) when features for all possible segments are required. However, they are more flexible in re-using features for sub-segments. This is done by requiring that the features are in a monoid. Because monoids are associative, features for two consecutive segments can be combined to form a feature for the union of the segments. This will allow more flexibility in integrating this type of feature in speech recognisers that perform pruning to obtain good performance. A type of features that is of particular interest for future work can be derived from HMM-like finite-state automata with weights in any semiring.

3 The exact word error over a lattice

Many operations in speech recognition can be elegantly described in terms of finite-state automata (Mohri *et al.* 2008; Hoffmeister *et al.* 2012; Povey *et al.* 2012; van Dalen *et al.* 2013*a*). However, some optimisation algorithms do not always create the desired results. This section focuses on determinisation, which has exponential space complexity. Even when the output automaton fits in memory, the intermediate representation may not. This section therefore proposes an algorithm to incrementally determinise and minimise an acyclic automaton. It uses less memory by keeping intermediate automata minimised at all times.

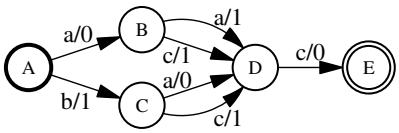
The example that this section will consider is that of annotating lattices with the exact word (or phone) error. This problem comes up in training acoustic models. A commonly-used criterion is the minimum Bayes' risk criterion (Povey 2003) (MBR, which aims to minimised the expected word (or phone) error. This involves a marginalisation over all word sequences, usually approximated with a lattice, of the weighted error of each of those word sequences. However, the algorithm for computing the word error for all word sequences in a lattice (Mohri 2003; Heigold *et al.* 2005) uses determinisation and in practice often runs out of memory. An approximation is therefore used that is related to the word error but uses a fixed alignment of the words in the hypothesis and the recognition lattice. This may limit the applicability of the criterion used to optimising the criterion it purports to minimise. Also, as new models for speech recognition are becoming more complex and features richer, the fixed time alignments could restrict performance. For example, recent work has focussed on the need to realign lattices every few iterations of training for HMMs with neural networks (Su *et al.* 2013) and structured SVMs (Zhang and Gales 2011*b*) or, for log-linear models, to use dense lattices to represent many alignments (Ragni and Gales 2012; van Dalen *et al.* 2013*a*). This section will therefore use the novel determinisation algorithm to compute the exact word or phone error for all the paths in a lattice.

The word error between two sequences is defined as the minimum edit distance between the two. The *word error rate* used to assess speech recognisers is the word error divided by the length of the reference. There is a well-known algorithm for computing the minimum edit distance, which performs a search for the lowest-cost path in a two-dimensional state space. This algorithm can be expressed in terms of finite-state automata, as a shortest-distance calculation in a graph whose states are in the Cartesian product of the automata representing the two sequences. This algorithm can be gener-



(a) The reference.

(b) The hypothesis lattice.



(c) The hypothesis lattice from figure 5b, annotated with the error. The topology has changed.

Figure 5 *Example reference and hypothesis automata.*

alised in two ways. One way is to find the minimum edit distance for any path in one automaton and any path in another automaton (Mohri 2003).

This section will concentrate on the other way to generalise the minimum edit distance algorithm: finding the error not for one sequence, but for a whole lattice. Instead of a shortest-distance algorithm, this requires determinisation. An example is given in figure 5: the reference sequence is "a c" and the hypothesis lattice has four sequences. The word error for three of those sequence is 1 ("a a c", "a c c", "b a c"), and for the other sequence it is 2 ("b c c"). The automaton in figure 5c assigns to the same four sequences the error. The topology of the automaton has changed, so that each of the symbol sequences has a different path. This illustrates that in general, determinisation can lead to an exponential number of transitions. The standard determinisation algorithm shows exponential behaviour on normal lattices used for training speech recognisers.

This section is organised as follows. Section 3.1 will discuss the minimum edit distance problem in terms of finite-state automata. Section 3.2 will introduce a semiring whose members are acyclic automata that are always determinised and minimised. This allows them to be used a weights of another automaton. Then a general algorithm will be introduced for determinising and minimising acyclic automata, which has the property that intermediate results are always minimised and thus take as little memory as possible.

3.1 The minimum edit distance

The edit distance measures the similarity between two sequences of symbols as the number of operations required to transform one string into another. The Levenshtein distance, used to evaluate speech recognisers, allows as operations the deletion, insertion or substitution of one symbol. The optimal sequence of operations is found by determining the best path through both sequences at once. There is a standard dynamic programming algorithm for solving this (see e.g. Cormen *et al.* 2009). To be able to generalise it, the problem and algorithm will here be expressed in terms of finite-state automata (Mohri 2003).

Figure 6 shows an example automaton for determining the edit distance between "a c" (along the vertical axis) and "a c c" (along the horizontal axis). The states are in

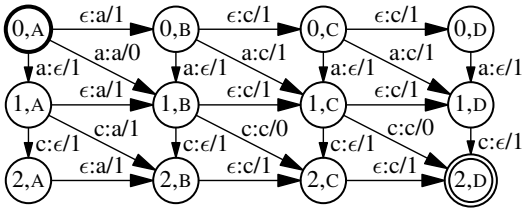


Figure 6 *The edit distance automaton for "a c" to "a c c". The shortest distance in this automaton is the edit distance.*

a product space of the states in figure 5. Each transition stands for an edit operation. A vertical transition moves in the reference but not in the hypothesis: a deletion. The label is e.g. "a:ϵ/1", with "a" in the reference, no symbol ("ϵ") in the hypothesis, and a cost of 1 for the deletion. The opposite, an insertion, is represented by a horizontal transition, with e.g. "ϵ:a/1". Diagonal transitions indicate a substitution (e.g. "c:a/1"), with a cost of 1 or a correct symbol (e.g. "a:a/1"), with a cost of 0. This automaton can be produced with a 3-way composition of the two input automata and a special transducer (Mohri 2003; Heigold *et al.* 2005), or with an ad hoc algorithm.

The lowest-cost path in this automaton corresponds to the minimum edit distance (Mohri 2003). Weights on finite-state automata must be in a *semiring* (Mohri 2002), here representing the edit cost. The semiring operation $\otimes$ concatenates the weights on two paths into that of a longer path, and the operation $\oplus$ combines two paths. Between paths, the lowest cost should be selected: $x \oplus y \triangleq \min(x, y)$; along paths, edit distances should add up: $x \otimes y \triangleq x + y$. Using this semiring, the cost semiring (sometimes called the "tropical" semiring), any shortest-distance algorithm will find the minimum edit distance between two sequences.

One generalisation of this algorithm, which this paper will not focus on, is to make the hypothesis a lattice (as in figure 5b). The same shortest-distance algorithm then results in the minimum edit distance between a reference and any one path in a hypothesis lattice: the oracle error.

3.2 Incremental determinisation and minimisation

To find the minmum edit distance for all paths in a lattice, the edit distance automaton in figure 6 must be determinised (Mohri 2003; Heigold *et al.* 2005). The standard algorithm for determinisation of weighted finite-state automaton (Mohri 2009) uses a powerset construction: each state in the resulting automaton is a set of states in the original automaton with a weight for each of them. Each transition represents all transitions with the same symbol out of each of these states. The destination state is instantiated as the set of the original destination states and weights. The weights are normalised so that states are more likely to be re-used: if the state representing the exact same set of states and weights has been seen before, it is mapped to the same state in the resulting automaton.

The problem with applying this general algorithm for the edit distance problem is memory use. Because all states in the original automaton are inserted into at least one state in the resulting automaton, the amount of memory required is the size of the automaton in figure 6. In theory, it would be possible to use the knowledge that

the resulting automaton is acyclic to reduce this, but in general this is a hard problem, requiring garbage collection that can deal with cycles as well as problem-specific reference counting to determine whether a state in the resulting automaton may yet be revisited.

An option that may seem attractive is to prune the edit distance automaton. It is true that many paths turn out to be redundant. However, a pruning algorithm that removes paths with high cost will prune not only redundant paths, but also legitimate paths in the lattice where the recogniser was badly wrong. That is not a good idea for discriminative training. Therefore, this paper aims to annotate lattices with the exact error.

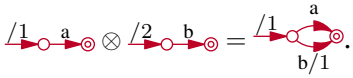
Instead of optimising standard determinisation to acyclic automata, this paper uses a simpler approach: the weights are replaced by equivalent weights in a different semiring. This semiring is the space of determinised and minimised acyclic automata. A standard shortest-distance algorithm, then recreates the original automaton in determinised and minimised form.

3.2.1 The automaton semiring

Minimisation of weighted automata (Revuz 1992; Mohri 2000; Eisner 2003) works by normalising the weights (also known as "weight pushing"), and then merging states with the same *suffix*, the same labels and weights following. Assuming that there are no arcs with empty symbol sequences, there is only one possible topology for one suffix, and the state with this suffix can be shared. An important insight since many values in the automaton semiring will be in memory at once, they can all share the same states, so they are jointly minimised. Each automaton is also kept determinised. This means that each state has at most one outgoing transition with a given symbol, and therefore that for each symbol sequence there is only one path. Since deterministic automata have only one start state, and most automata will have a start weight, it will be indicate on an unconnected arrow: 

is an automaton that assigns 1 to the symbol sequence "a" (as usual, a weight /0 is not written). To prevent confusion, the rest of this paper will use the cost semiring and operations specific to this. However, the automaton semiring can carry any weakly left divisible semiring (Mohri 2002).

Like the cost semiring, the automaton semiring must have operations $\otimes$ and $\oplus$, and identities $\overline{1}$ and $\overline{0}$ defined. Since the sum over all paths of the automaton semiring should recreate its host automaton, the operations are defined as follows. The operation $\oplus$ is used to combine the weights of two competing paths. It is therefore defined as the union of its arguments, which assigns any sequence the $\oplus$-sum of what its arguments assign to that sequence. For example,

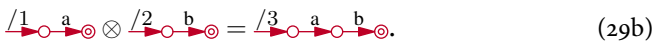


(29a)

The left argument assigns 1 to "a", and the right one 2 to "b". The result assigns those weights to both. To keep the automaton normalised, its weights have been pushed to the front.

The operation $\otimes$, which is used on consecutive arcs along a path, concatenates two automata. This means that the concatenation of each string of the left-hand automaton

with each string of the right-hand automaton is assigned the $\otimes$-product of the weights of the two automata. For example,



(29b)

The resulting automaton assigns a weight of 3 to "a b". To keep the automaton normalised, the weights are pushed to the front.

The values $\overline{0}$ and $\overline{1}$ must be defined so that adding $\overline{0}$ does nothing, and multiplying by $\overline{1}$ does nothing. They are therefore defined as $\overline{1} \triangleq$  and $\overline{0} \triangleq$  .

3.2.1.1 Implementation The data structures used to implement the automata and the operations on them are best expressed mathematically. An automaton $a = (s, s)$ consists of an initial weight s, which can be extracted with $s(a)$, and a state s, which can be extracted with $q(a)$. A state $s = (f, \mathcal{T})$ is defined by a final weight f, which can be extracted with $f(s)$, and a set of outgoing transitions $\mathcal{T}$, which can be extracted with $n(s)$. Each transition $t = (k, a)$ consists of a symbol k, which can be extracted with $k(t)$ and an automaton a attached to it, which can be extracted with $a(t)$. For example, the automaton  that only assigns 0 to empty sequences can be written $a_f = (0, (0, \emptyset))$. The automaton  can be expressed as $(2, (\infty, \{(b, a_f)\}))$.

The following will discuss the operations $\textsc{Union}(a_l, a_r)$, which implements $\oplus$, and $\textsc{Concatenate}(a_l, a_r)$, which implements $\otimes$, but first the building blocks $\textsc{Normalise}(a)$ and $\textsc{Denormalise}(a)$.

Since the states should be re-used as much as possible, they should be stored normalised. The standard normalisation used in minimisation algorithms is to push weights from all paths forward (Mohri 2000; Eisner 2003). The function $\textsc{Normalise}(a)$ ensures that the outer level of weights from state $q(a)$ is normalised. In the cost semiring, the minimum of the start weight and the weights of the automata following the state is made 0 by extracting the residual c from the weights inside the state. For readability, the following definition expands the argument:

$$\textsc{Normalise}((s, (f, \mathcal{T}))) = \big(c, (f - c, \mathcal{T}')\big), \tag{30a}$$

$$\text{where } \mathcal{T}' \triangleq \{(k, (s - c, s)) \mid (k, (s, s)) \in \mathcal{T}\};$$

$$c \triangleq \min(\{f\} \cup \{s \mid (k, (s, s)) \in \mathcal{T}\});$$

$$\textsc{Normalise}(\overline{0}) = \overline{0}. \tag{30b}$$

The resulting automaton assigns the same weights to the same sequences as the original automaton does.

The operation $\textsc{Denormalise}(a)$ performs the opposite operation: it returns an automaton $a' = (0, s')$, with the start weight 0, that is equivalent to a.

Since the automata are acyclic, the operations $\textsc{Union}$ and $\textsc{Concatenate}$ can be implemented recursively. $\textsc{Union}((s_l, s_l), (s_r, s_r))$ returns an automaton that assigns to each symbol sequence the minimum of the weights that the two arguments assign. Without loss of generality, the initial weights of the automata can be assumed 0, and the states unnormalised. Otherwise, $\textsc{Denormalise}$ can be used. Then,

$$\textsc{Union}((0, (f_l, \mathcal{T}_l)), (0, (f_r, \mathcal{T}_r))) = \textsc{Normalise}((0, \min(f_l, f_r), \mathcal{T}')), \tag{31a}$$

where $\mathcal{T}' \triangleq \{(k, a_k) \mid (k, a) \in \mathcal{T}\}$ and

$$a_k \triangleq \begin{cases} \textsc{Union}(a(t), a(u)) & \text{if } \exists t \in \mathcal{T}_l, \exists u \in \mathcal{T}_l \text{ s.t. } k(t) = k(u) = k; \\ a(t), & \text{if } \exists t \in \mathcal{T}_l \text{ s.t. } k(t) = k; \\ a(u), & \text{if } \exists u \in \mathcal{T}_l \text{ s.t. } k(u) = k. \end{cases} \tag{31b}$$

The $\textsc{Concatenate}(a_l, a_r)$ operation returns an automaton that assigns to the concatenation of each string of the left-hand automaton with each string of the right-hand automaton the sum of the weights of the two automata, like in (29b).

The strategy for finding this automaton is to treat two parts of the left automaton separately: the final weight and the outgoing arcs. The left automaton assigns empty sequences weight $s_l + f_l$ (which may be ∞), which is multiplied by the cost that the right automaton assigns to sequences. The outgoing arcs of the left automaton are traversed recursively. The results of the two parts is then summed by calling $\textsc{Union}$. Expanding the arguments, $\textsc{Concatenate}(a_l, a_r)$ can be written as

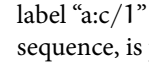
$$\begin{aligned} &\textsc{Concatenate}((s_l, (f_l, \mathcal{T}_l)), (s_r, s_r)) \\ &= \textsc{Normalise}\big(\textsc{Union}\big((s_l + f_l + s_r, s_r), (s_l, (0, \mathcal{T}'))\big)\big), \\ &\text{where } \mathcal{T}' \triangleq \{(k, \textsc{Concatenate}(a, a_r)) \mid (k, a) \in \mathcal{T}_l\}. \end{aligned} \tag{32}$$

The expressions in (31a) and (32) are recursive, and in an eagerly-evaluated language (like C++) they will take time in the order of the number of paths of the input automata. However, the fact that these functions do not have any side effects can be exploited. A sensible implementation uses *memoisation*, which stores the result of a function the first time it is called, and afterwards returns the stored value. Using memoisation lowers the time complexity of $\textsc{Union}$ and $\textsc{Concatenate}$ to the order of the number of states in the automata.

Another opportunity for optimisation is when either the left or the right argument is returned exactly. By memoising weight thresholds for when this happens, $\textsc{Union}$ can be made to run faster, but using no less memory.

3.2.2 Determinisation and minimisation

The complete algorithm for determinisation and minimisation works as follows. An automaton is constructed with values in the automaton semiring as its labels, representing the symbols and weights of interest. Then, a shortest-distance algorithm is applied. Since the semiring has been described as modelling the suffixes of states, not the prefixes, the shortest-distance algorithm should work backwards from the final state.

Figure 7 illustrates the normal weighted automaton in figure 6 can be converted into one with weights in the automaton semiring. Since the interest here is in the symbols in the hypothesis lattice, the "output" symbols are used, with their weights. For example, a label "a:c/1" is converted into  , which assigns a weight only to the empty sequence, is produced for a label without a symbol, like "a:ϵ/1".

The shortest-distance algorithm in progress is shown in figure 8. For simplicity, the example has only one path in the hypothesis lattice, so the resulting automaton will be simple. The states from the original automaton whose transitions have been processed have been removed from the graph. For the states on the frontier, the automaton-valued

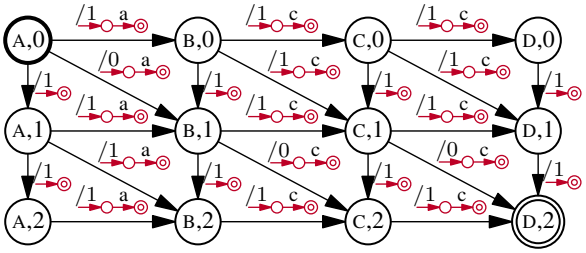


Figure 7 *The graph from figure 6 with labels in the automaton semiring.*

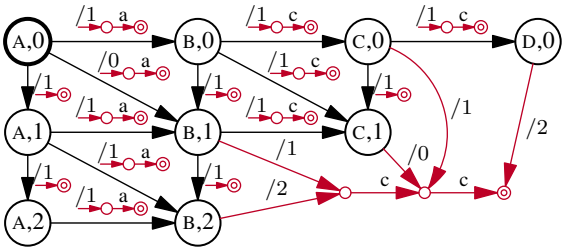


Figure 8 *A shortest-distance algorithm on the automaton in figure 7. The resulting determinised and minimised automaton is forming as the accumulated weights of the algorithm.*

shortest distances to the final state have been computed. They are drawn as small automata starting from the states on the frontier. For example, from state "B,1" the automaton semiring has cost 1 for "c c". Once the algorithm has completed, the shortest distance to "A,0" will assign a cost of 1 to "a c c", which is the correct result.

In general, this algorithm finds a determinised and minimal equivalent automaton for any acyclic automaton. The advantage compared to the standard determinisation algorithm is that by computing the output incrementally, peak memory use is lower.

4 Bayesian log-linear models

This section will explore the application of the ideas of Bayesian models to conditional (or "discriminative") models. Bayesian training will be re-analysed as minimising a criterion. This makes it possible to replace part of the criterion to perform maximum-margin training.

4.1 Bayesian models

Standard Bayesian models assume that the observations can be explained by a model that some parameters Θ. However, the setting of the parameters is unknown; only the training data is known. Therefore, Bayesian models propagate the uncertainty about the parameters to the inference stage on a test point.

Define the probability model for an observation $\mathbf{o}$ as

$$p(\mathbf{o}|\Theta)\,, \tag{33}$$

where Θ are the parameters for the distribution whose form depends on the problem. In maximum-likelihood training, a point estimate of Θ is found:

$$\Theta^* \triangleq \arg\max_{\Theta} p(\mathbf{o}|\Theta) . \tag{34}$$

However, it is also possible to define a distribution q, whose form again depends on the problem, over the parameters, and use the uncertainty in testing:

$$\int p(\mathbf{o}|\Theta)\, q(\Theta) d\Theta. \tag{35}$$

In standard Bayesian methods, q is found from the data $\mathcal{D} = \{\mathbf{o}_i\}_{i=1\ldots|\mathcal{D}|}$ by assuming the exact same distribution for the observations in training as in testing, and performing probabilistic inference. This can be expressed as one expression:

$$\int p(\mathbf{o}|\Theta)\, p(\Theta|\mathcal{D}) d\Theta. \tag{36}$$

This expression is the same as (35), with the posterior $p(\Theta|\mathcal{D})$ of the parameters given the data as the distribution q. There is then no choice in the distribution over the parameters; it must be found with Bayes' rule:

$$q(\Theta) \triangleq p(\Theta|\mathcal{D}) = \frac{p(\mathcal{D}|\Theta)\, p(\Theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\Theta)\, p(\Theta) . \tag{37}$$

$p(\Theta)$ is the prior over the parameters, which must be set in Bayesian methods. $p(\mathcal{D}|\Theta)$ is the likelihood of the training data. Assuming all observations are independent and identically distributed, it is

$$p(\mathcal{D}|\Theta) = \prod_{\mathbf{o}\in\mathcal{D}} p(\mathbf{o}|\Theta) . \tag{38}$$

In conclusion, after deciding the form of p and a prior over the parameters Θ, the posterior distribution over the parameters (after seeing the data) is

$$q(\Theta) \propto p(\Theta) \prod_{\mathbf{o}\in\mathcal{D}} p(\mathbf{o}|\Theta) . \tag{39}$$

The distribution over a new observation can then be modelled as in (35).

4.2 Bayesian conditional models

Section 4 has discussed pure Bayesian models. These assume that the real distribution of the observations is known, but the parameters are unknown. If the real distribution of the observations is not known, then it is sometimes possible to use a conditional model. A conditional model does not model the joint distribution of all the variables, but instead conditions on variables that are always given (like observations) and then models the distribution of other variables given those.

Discriminative training of generative models, often used for training speech recognisers, can be argued to result in a conditional model: the distribution over the observations, after all, does not represent the actual observations. In some sense, the

parameters that, in a proper generative model, model the observation are co-opted to help model the conditional distribution (Minka 2005; Lasserre *et al.* 2006).

If $\mathbf{o}$ is the observation that needs to be classified into a class w, then a conditional model could be

$$p(w|\mathbf{o}, \Theta)\,, \tag{40}$$

where, like in section 4.1, Θ are the parameters.

This type of model can be trained with maximum-likelihood training, where w_{ref} is the correct classification:

$$\Theta^* \triangleq \arg\max_{\Theta} p(w_{\text{ref}}|\mathbf{o}, \Theta)\,. \tag{41}$$

This is sometimes called "conditional maximum likelihood" (CML) training.

Similarly to the standard Bayesian framework in (35), it is possible to instead use a distribution over parameters:

$$\int p(w_{\text{ref}}|\mathbf{o}, \Theta)\, q(\Theta)\, d\Theta. \tag{42}$$

The standard Bayesian method, again, would be to use $p(\Theta|\mathcal{D})$ for q. The training data must be split up in variables that will be given, and variables that inference will be performed on. The variables that are given are observations $\mathbf{o}$, and each of these is connected with a class w: $\mathcal{D} = \{(\mathbf{o}_i, w_i)\}_{i=1\ldots|\mathcal{D}|}$. Analogously with (39), then, q must be set to

$$q(\Theta) = p(\Theta|\mathcal{D}) \propto p(\Theta) \prod_{(\mathbf{o},w)\in\mathcal{D}} p(w|\mathbf{o}, \Theta)\,. \tag{43}$$

4.3 A criterion for Bayesian models

To be able to generalise Bayesian inference to other training criteria, it is necessary to describe it as a criterion first. Unlike well-known criteria like the maximum-likelihood criterion, which is a function of the parameters, this criterion must be a function of the distribution q over parameters. Since it will be required to manipulate distributions, it is easy to give distributions names that identify their parameters explicitly:

$$p_{\mathcal{D}|\Theta}(\mathcal{D}|\Theta) \triangleq p(\mathcal{D}|\Theta)\,; \qquad p_{\Theta}(\Theta) \triangleq p(\Theta)\,; \qquad p_{\Theta|\mathcal{D}}(\Theta|\mathcal{D}) \triangleq p(\Theta|\mathcal{D})\,. \tag{44}$$

To compare distributions, the Kullback-Leibler (KL) divergence is often used. In general, to compare distributions q and p over the variable x, giving the variable in the subscript,

$$\mathcal{KL}_x(q\|p) \triangleq \mathcal{H}_x(q\|p) - \mathcal{H}_x(q) = \int q(x) \log \frac{q(x)}{p(x)} dx, \tag{45a}$$

where $\mathcal{H}_x(\cdot\|\cdot)$ is the cross-entropy between two distributions, and $\mathcal{H}_x(\cdot)$ is the entropy of one distribution:

$$\mathcal{H}_x(q\|p) \triangleq -\int q(x) \log p(x) dx; \tag{45b}$$

$$\mathcal{H}_x(q) \triangleq \mathcal{H}_x(q\|q) = -\int q(x) \log q(x) dx. \tag{45c}$$

It can be shown that, as long as the distributions are normalised, the cross-entropy is a lower bound on the entropy. The KL divergence is therefore always non-negative. It can also be shown that the KL divergence is only 0 if its two arguments are the same distribution.

The criterion $\mathcal{F}$ An obvious generalised criterion is the Kullback-Leibler (KL) divergence between the distribution q and the distribution that Bayesian theory says it ought to model.

$$\mathcal{F}(q) \triangleq \mathcal{KL}_{\Theta}\big(q \big\| p_{\Theta|\mathcal{D}}\big) + \text{constant}. \tag{46}$$

Since the KL divergence is 0 only if the distributions are the same, the minimum of this criterion is exactly when q is equal to $p_{\Theta|\mathcal{D}}$. Minimising this criterion therefore is equivalent to finding the distribution q^* over the parameters in a fully probabilistic manner, as in (43):

$$q^* \triangleq \arg\max_q \mathcal{F}(q) = \arg\max_q \mathcal{KL}_{\Theta}\big(q \big\| p_{\Theta|\mathcal{D}}\big) = p_{\Theta|\mathcal{D}}. \tag{47}$$

The reason for expressing this part of Bayesian inference as a criterion was to apply the idea of uncertainty over parameters and copy machinery used in Bayesian inference. Crucial for this is the Bayesian prior. To retain the prior in the optimisation, it must be written explicitly. The parameter posterior $p_{\Theta|\mathcal{D}}$ in (46) implicitly contains the prior. This becomes clear when the posterior is rewritten using Bayes' rule. Repeating (37) with the notation in (44) yields

$$p_{\Theta|\mathcal{D}}(\Theta) = \frac{p_{\mathcal{D}|\Theta}(\Theta)\, p_{\Theta}(\Theta)}{p(\mathcal{D})} \propto p_{\mathcal{D}|\Theta}(\Theta)\, p_{\Theta}(\Theta)\,. \tag{48}$$

The $\propto$ indicates that the distribution on the left is proportional to the distribution on the right, i.e. they are equal after multiplying with the correct constant with respect to Θ. This constant, $p(\mathcal{D})$, is redundant since it merely makes sure that all distributions are normalised. In the criterion (46), the posterior is inside a log term, and the log-constant should be added. Bayes' rule then yields

$$\begin{aligned}
\mathcal{F}(q) &= \mathcal{KL}_{\Theta}\big(q \big\| p_{\Theta|\mathcal{D}}\big) - \log p(\mathcal{D}) \\
&= \int q(\Theta) \log \frac{q(\Theta)}{p_{\Theta|\mathcal{D}}(\Theta)} \mathrm{d}\Theta - \log p(\mathcal{D}) \\
&= \int q(\Theta) \log \frac{q(\Theta)}{p_{\Theta}(\Theta)\, p_{\mathcal{D}|\Theta}(\Theta)} \mathrm{d}\Theta \\
&= \int q(\Theta) \log \frac{q(\Theta)}{p_{\Theta}(\Theta)} \mathrm{d}\Theta - \int q(\Theta) \log p_{\mathcal{D}|\Theta}(\Theta) \mathrm{d}\Theta && \text{(49a)} \\
&= \underbrace{\mathcal{KL}_{\Theta}(q \| p_{\Theta})}_{\text{prior}} + \underbrace{\mathcal{H}_{\Theta}\big(q \big\| p_{\mathcal{D}|\Theta}\big)}_{\text{evidence}}. && \text{(49b)}
\end{aligned}$$

This rewritten criterion separates the influence of the prior, in the left-hand term, and the influence of the evidence, in the right-hand term. That the left-hand term is a KL divergence and the right-hand term is a cross-entropy is entirely accidental; they could have been swapped around. However, since the right-hand term will be generalised, the formulation in (49b) is more convenient.

This report will generalise the criterion by replacing the term $\log \mathrm{p}_{\mathcal{D}|\Theta}(\Theta)$ in (49a) by $\mathcal{C}$:

$$\mathcal{F}(\mathrm{q}) \triangleq \mathcal{KL}_{\Theta}(\mathrm{q}\|\mathrm{p}_{\Theta}) + \int \mathrm{q}(\Theta)\mathcal{C}(\Theta)\mathrm{d}\Theta \tag{50a}$$

$$= \mathcal{KL}_{\Theta}(\mathrm{q}\|\mathrm{p}_{\Theta}) + \mathcal{H}_{\Theta}(\mathrm{q}\|\mathcal{C}^{\exp}), \tag{50b}$$

where $\mathcal{C}^{\exp}$ is defined as a distribution, so that

$$\mathcal{C}^{\exp}(\Theta) \propto \exp(\mathcal{C}(\Theta)); \tag{50c}$$

$$\int \mathcal{C}^{\exp}(\Theta)\mathrm{d}\Theta = 1. \tag{50d}$$

Note that in the generalised criterion, where the probabilistic meaning has been let go, the two terms in (50a) can be weighted relative to each other. This is equivalent to weighting $\mathcal{C}$. This, in turn, is equivalent to taking $\exp(\mathcal{C}(\Theta))$ to the power of that weight, making $\mathcal{C}^{\exp}$ sharper if the weight is greater than 1.

One way of using a Bayesian criterion is to approximate the right-hand argument to the KL divergence and to use that approximation as q. Gibbs sampling, a Markov chain Monte Carlo algorithm, is one way of doing this. This iteratively samples each variable from its conditional distribution given the setting of all its neighbours. After many iterations, the joint setting of all variables can be used as a sample from the overall distribution. Usefully, Gibbs sampling does not defined on the distribution's normalisation constant being known, or it being normalised at all. It is therefore possible to apply Gibbs sampling to find q in (50), by using $\mathcal{C}^{\exp}$ where $\mathrm{p}_{\mathcal{D}|\Theta}$ would be used normally (Yang *et al.* 2015). Whether this is straightforward does depend on the form of $\mathcal{C}^{\exp}$.

4.4 Large-margin training

The traditional form of training conditional models uses the conditional maximum likelihood criterion, as in (41). The criterion for training a Bayesian classifier, in (49b), can be generalised to other forms of training. (49b) maximises the Bayesian conditional likelihood. Probabilistically training a conditional model is to CML as standard Bayesian training is to normal maximum-likelihood training. However, for discriminative models there are other forms of criteria that can be used. A large-margin criterion is of particular interest, because of its theoretical guarantees, and its performance on previous tasks (Zhang and Gales 2012; van Dalen *et al.* 2013*b*).

A large-margin criterion that has been used successfully for speech recognition is one that provides a trade-off between the ratio of the posteriors of the reference and the competing hypothesis, and a loss function that compares the words in the reference, $\boldsymbol{w}_{\mathrm{ref}}$, and the words in the hypothesis, $\boldsymbol{w}$ (Zhang *et al.* 2010; van Dalen *et al.* 2013*b*):

$$\mathcal{C}(\Theta) = \mathrm{C} \sum_{(\mathbf{o},\boldsymbol{w}_{\mathrm{ref}})\in\mathcal{D}} \left[\max_{\boldsymbol{w}\neq\boldsymbol{w}_{\mathrm{ref}}} \mathcal{L}(\boldsymbol{w}_{\mathrm{ref}},\boldsymbol{w}) - \log\left(\frac{\mathrm{P}(\boldsymbol{w}_{\mathrm{ref}}|\mathbf{O};\Theta)}{\mathrm{P}(\boldsymbol{w}|\mathbf{O};\Theta)}\right)\right]_{+}, \tag{51a}$$

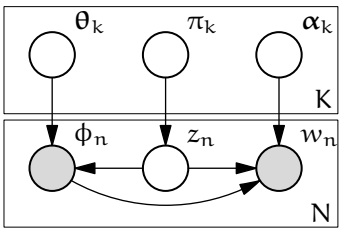


Figure 9 *The graphical model for a mixture of experts.*

where C is a hyperparameter that controls the balance between the parameter prior and the evidence, and $[\cdot]_+$ is the hinge-loss function, defined as

$$[x]_+ \triangleq \begin{cases} 0 & x < 0; \\ x & x >= 0. \end{cases} \tag{51b}$$

It is possible to retrieve the exact same criterion as in Zhang *et al.* (2010) by finding a point estimate for the parameters instead of a distribution (or, equivalently, find a distribution in the space of delta spikes on parameters). In essence, $\mathcal{C}(\Theta)$ itself, plus a regularisation term, is maximised. This will be used in this report for structured SVMs.

5 Infinite support vector machines

Kernelising support vector machines is one way of using linear classifiers to model non-linear decision boundaries; using a mixture of SVM experts is another. This allows the classification of experts that have been trained on different parts of the feature space to be interpolated depending on the position in space of the observation. The following will first discuss the mixture of experts, for which the number of experts needs to be pre-specified. It will then discuss the Bayesian non-parametric variant, which integrates out over all possible mixtures and partitions of the training data.

5.1 The mixture of experts

An alternative way of modelling non-linearities in the input is using a mixture of experts. This model first decides on the weighting between experts given the region of input space, and then interpolates between the classification of these experts.

Figure 9 shows the graphical model for a Bayesian mixture of expects. In the middle of the graph there are the component priors as a vector $\boldsymbol{\pi}$. Unusually, the plate at the bottom considers all N observations separately. (This will be exploited in section 5.2 to deal with an infinite number of experts.) Each observation is assigned a component z_n. The observation ϕ_n depends on the component, and the parameters θ_{z_n} of that component. The feature is assumed generated by a mixture of Gaussians. This is often unrealistic. However, this is not a problem, since the model will not be used to generate data, and the observation is always given. This part of the model is often called the *gating network*: all it does is assign data to experts.

The right-hand part of the graphical model is the *expert model*. Each component has one expert. The experts are discriminative: they directly model the class given the

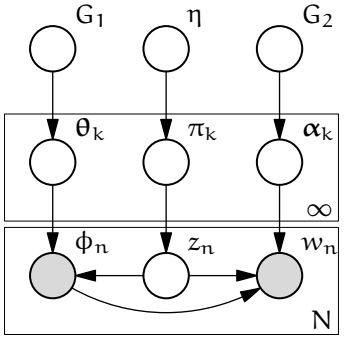


Figure 10 *The graphical model for an infinite mixture of experts.*

observation. The class w_n that the observation is assigned to depends on the component and the observation, and on the parameter $\boldsymbol{\alpha}_{z_n}$ of the expert.

If the parameters θ, $\boldsymbol{\alpha}$ are given, classification in this model works as follows:

$$P(w|\phi, \theta, \boldsymbol{\alpha}) = \sum_k P(z = k|\phi, \theta)\, P(w|\phi, \boldsymbol{\alpha}, z = k)\,. \tag{52}$$

5.2 The infinite mixture of experts

The mixture-of-experts model is parametric: the number of experts K must be fixed in advance. In order to circumvent this requirement, a Bayesian non-parametric version of the mixture of experts will be used here. The Dirichlet process mixture of experts (Rasmussen and Ghahramani 2001) instead uses an infinite number of components. In theory, it therefore considers all possible partitions of the training data and associated components. However, it is impossible to deal with an infinite number of components.

The trick, common in Bayesian non-parametrics, is that the posterior distribution of the components given the data is approximated with a Monte Carlo scheme, in this work, Gibbs sampling. For any one sample, the number of components assigned any observation is then at most the number of observations. To allow inference, all components that have not been assigned to any observation must be marginalised out. Each sample acquired from the infinite mixture of experts then has exactly the form of a normal mixture of experts in figure 9.

The graphical model is given in figure 10. The number of components K is set to ∞. The distributions for the mixture of experts must be chosen so that it is practical to represent a finite subset of the infinite number of experts. They are therefore (again as usual in Bayesian nonparametrics) chosen

$$\boldsymbol{\pi} \sim \text{Dirichlet}(\eta)\,; \tag{53}$$

$$z_n \sim \text{Categorical}[\boldsymbol{\pi}]\,. \tag{54}$$

By making the vector of priors of experts, $\boldsymbol{\pi}$, infinite-dimensional, the mixture model is given by a *Dirichlet process* (Rasmussen 1999).

Instead of performing inference over the infinite-dimensional vector π when $K \to \infty$, it is marginalised out. The distribution of the assignment of the observations to

experts is then given by the *Chinese restaurant process* (Aldous 1985; Pitman 2002). (In the metaphor, the experts are tables; the observations customers who sit at a table as they enter the restaurant.) Because the components are *exchangeable*, it is possible to just consider the ones that have at least one observation assigned to them. Only for those components z are the parameters θ_z and $\boldsymbol{\alpha}_z$ kept in memory. Since Gibbs sampling is used, at any time it is only necessary to re-draw the assignment of one observation to an expert. Because of the nature of the Dirichlet process, the posterior distribution of the assignment of one observation given all other ones is split between the existing components and the new ones:

$$P\big(z_n = k \big| \boldsymbol{z}_{\setminus n}, \eta\big) \propto \begin{cases} N_k, & \text{if } k \text{ is an existing expert;} \\ \eta, & \text{if } k \text{ is a new expert.} \end{cases} \tag{55}$$

During Gibbs sampling, of course, for sampling the assignment of one observation to an expert, the complete posterior of z_n is used. This uses the component priors, in (55), but also the current parameters of the components and the experts. Because of this, observations are more likely to move to an expert that classifies them correctly than to an expert that does not. During the training process, this creates an interaction between components' positions and experts' performance that is a great strength of this model.

The procedure for Gibbs sampling is as follows. The parameters $\Theta = \{\theta, \boldsymbol{\alpha}\}$ and component assignments $\boldsymbol{z}$ are sampled iteratively. While the value of one variable, e.g. $\boldsymbol{z}$, is sampled, all other variables, e.g. θ, $\boldsymbol{\alpha}$ remain the same. Because of the exchangeability of $\boldsymbol{z} = \{z_1, \dots, z_N\}$, the posterior distribution of z_n given the value of its neighbours, according to which it is sampled, is

$$P\big(z_n \big| \boldsymbol{z}_{\setminus n}, \phi_n, \theta_k, w_n, \boldsymbol{\alpha}_k\big) \propto P\big(z_n \big| \boldsymbol{z}_{\setminus n}\big)\, p(\phi_n | \theta_{z_n})\, P(w_n | \phi_n, \boldsymbol{\alpha}_{z_n})\,, \tag{56}$$

where the first term $P\big(z_n \big| \boldsymbol{z}_{\setminus n}\big)$ is given according to (55). When k is a represented component, z_n can be directly sampled from equation (56). When k is an unrepresented component, however, the term $p(\phi_n | \theta_{z_n})$ is given by $\int p(\phi_n|\theta)\, G_1(\theta) \mathrm{d}\theta$; and $P(w_n | \phi_n, \boldsymbol{\alpha}_{z_n})$, is given by $\int P(w_n | \phi_n, \boldsymbol{\alpha})\, , G_2(\boldsymbol{\alpha}) \mathrm{d}\boldsymbol{\alpha}$.

After obtaining the assignments of observations to components, first the number of represented components is updated. Then the parameters of the components θ are sampled, after which the parameters of each expert $\boldsymbol{\alpha}$ are updated.

Classification with this Bayesian model in theory would use all components of the mixture model weighted by their posterior from the training data:

$$P(w | \phi, \mathcal{D}) = \int P(w | \phi, \Theta)\, p(\Theta | \mathcal{D}) \mathrm{d}\Theta, \tag{57a}$$

where $\Theta = \{\theta_k, \boldsymbol{\alpha}_k\}_{k=1\dots K}$ are all the parameters of the mixture of experts. However, since the number of components is infinite, (57a) cannot be computed. Instead, the samples acquired with Gibbs sampling are used. Denote with $\Theta^{(l)}$ the lth draw from the posterior. As usual in Gibbs sampling, the draws are taken sufficiently far apart that independence between them can be assumed:

$$\Theta^{(l)} \sim p(\Theta | \mathcal{D})\,. \tag{57b}$$

Each of these draws contains a finite number $K^{(l)}$ of active components. Classification then uses all components from all samples:

$$\begin{aligned} P(w|\phi, \mathcal{D}) &\simeq \frac{1}{L} \sum_{l=1}^{L} P\left(w\middle|\phi, \Theta^{(l)}\right) \\ &= \frac{1}{L} \sum_{l=1}^{L} \sum_{k=1}^{K^{(l)}} P\left(z = k\middle|\phi, \theta^{(l)}\right) P\left(w\middle|\phi, \alpha_k^{(l)}\right). \end{aligned} \tag{57c}$$

This form of classification is equivalent to classification with a mixture of experts consisting of all experts from all draws.

5.3 Infinite support vector machines

If the experts are multi-class support vector machines, then the resulting model is an infinite support vector machine (infinite SVM) (Zhu *et al.* 2011). A multi-class SVM is a special case of a structured SVM where the classes have no structure. Unlike standard SVMs, however, there are more than two classes, so that the large-margin criterion in (51a) is required.

To use the infinite SVM, however, SVMs must be interpreted as probabilistic models. This is necessary both for classification (in (57c)), because the experts' distributions must be summed, but also for Gibbs sampling. The interpretation as log-linear models will be used here. When the assignment to an expert for an observation is sampled, therefore, the distribution of the class given its parents is

$$P(w|\phi, z, \alpha) \propto \exp\left(\alpha_z^{\mathsf{T}} \mathbf{\Phi}(\phi, w)\right), \tag{58}$$

where α_z is the parameter vector for expert z, and $\mathbf{\Phi}(\phi, w)$, gives a vector which contains zeros except for the dimensions indicated by w, where it has a copy of ϕ.

Instead of sampling the parameters of experts within Gibbs sampling, the standard multi-class SVM training procedure is run. A potential problem in training the SVM experts is that of overfitting. This problem is much larger than for standard SVMs, since within the model here, at any iteration, the number of observations assigned to an expert can legitimately be very small. Without any regularisation, the SVM could therefore produce a parameter setting that does not generalise. Here, a prior is used. The mean of the prior is set not to zero (as is often the implicit setting), but to the parameters of an SVM trained on the whole data set. The regularisation constant, which implies the covariance of the prior, will be empirically set using the development set.

6 Source code

To support the algorithms that the work for this project requires, a new library is required. In particular, to compute the exact word error (in section 3 on page 14) and to perform fast extraction of segmental features (see van Dalen *et al.* 2013*b*), a library for finite-state automata is required. Finite-state automata are general and well-defined, making them are excellent material for a software library. The values in semirings that weighted automata carry can be implemented using type templates for speed. However,

existing libraries, including the most used one, OpenFst (Allauzen *et al.* 2007), are limited in that they require automata to be represented explicitly. It is true that OpenFst provides lazy implementations of such algorithms as composition, but it makes the fatal mistake (for computing error automata and fast segmental feature extraction) of renaming the states to integer values. To keep track of the meaning of these integers, all states must be explicitly instantiated when the automaton is completely traversed. For the computation of the exact word error, in addition the single-source shortest-distance algorithm cannot exploit a topological order to drop states that have been traversed from memory.

Therefore, the work in this project has included a new C++ library for manipulating finite-state automata. This library has been open-sourced under the Apache license, Version 2.0 (The Apache Software Foundation 2004) and placed on GitHub (van Dalen 2015). The Apache license is a permissive and well-known open-source software license. GitHub is a popular site that enables and encourages social coding. It allows publicly available source code and a modern distributed version control system (Git). Its interface makes it easy to "fork" code and to contribute back to it. It is hoped that this will foster further development, whether by the authors of this project or by other researchers.

The flexibility of the Flipsta library, compared to other libraries that deal with finite-state automata, derives in large part from its extreme view on the labels on the transitions between states of automata. Other libraries fix the number of symbol sequences to 1 (to get finite-state *acceptors*) or 2 (to get finite-state *transducers*), and zero or one weight. Through these are fine choices for many applications, many require an n-tape automaton with $n > 2$. Whole papers are written about how to do this. Indeed, the OpenFst library allows users to work around the 2-tape restriction by moving extra symbols into the weights. (They then become awkward to manipulate.)

The Flipsta library circumvents this by allowing only one *label*, which must be in a semiring, but by allowing it to be composite. Though C++11 variadic templates (and elaborate machinery underneath that manipulates the types), any combination and number of symbols and more traditional (or more outlandish) semirings can be used as labels. This makes it easy to use, say, three-tape automata, which is hoped to enable easier implementation of various algorithms.

7 Experiments

The techniques in this report are tested on various corpora: two noise-corrupted corpora, the small-vocabulary AURORA 2 and medium-vocabulary AURORA 4, the Hub4 broadcast news corpus, and a YouTube corpus from Google.

Since the start of the project, neural networks have become the new state of the art in acoustic modelling with HMMs. The experiments in this section therefore both compare against HMM systems based on neural networks, and use them to extract features, as detailed in section 2.2.2.

7.1 Setups

AURORA 2 is a small vocabulary digit string recognition task (Hirsch and Pearce 2000), with vocabulary size of 12 (one to nine, plus zero, oh and silence). 8 real-world noise

conditions have been added to the speech artificially over a variant of signal to noise ratios (SNRs). The generative models, GMM-HMMs, are trained on the clean data with 8840 utterances recorded from 55 male and 55 female adults. The feature vectors used by the front-end HMMs consisted of 12 MFCCs appended with the zeroth cepstrum coefficient and delta and delta-delta coefficients. The baseline HMM system uses VTS compensation, with the noise model for VTS compensation estimated on each utterance. The log-linear models are trained on a subset of the multi-style training data containing 4 noise conditions and 3 SNRs (20dB, 15dB and 10dB). All three test sets, A, B and C, are used in the evaluation. The sets contain 4, 4, and 2 noise conditions at 5 different SNRs (0dB to 20dB).

AURORA 4 is a noise-corrupted medium to large vocabulary database based on the Wall Street Journal (WSJ) data. Here the HMM used to generate features is trained on clean data (SI-84 WSJ0 part, 14 hours). The HMMs are state-clustered tri-phones (3140 states) with 16 Gaussian components per state. Four iterations of VTS compensation [1] are performed for the test data. The log-linear models are trained on the multi-style data with 7033 utterances. Evaluation is performed using the standard 5000-word WSJ0 bi-gram model on 4 noise-corrupted test sets based on NIST Nov92 WSJ0 test set. Set A is clean, set B has 6 types of noise added, set C has channel distortion, and set D has both additive noise and channel distortion.

A third training set is the "Hub4" data, which contains English broadcast news. The 144 hours used here have good manual transcriptions. The data and setup are similar to that in Gales *et al.* (2006).

As part of the project, Google has provided audio data from YouTube videos that displays differences in speaking and adverse environments of various kinds. This is a hard task for a speech recogniser. Most of the videos that the data is from is still on YouTube, so that metadata can be found. This is not currently used.

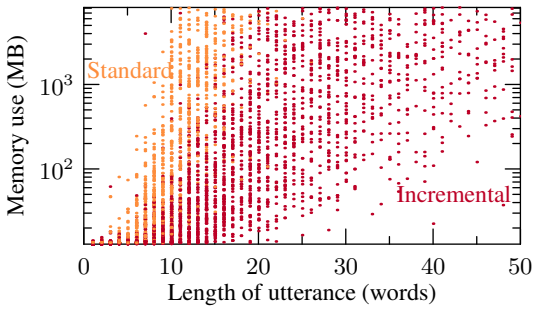
The YTElect data (see also Alberti *et al.* 2009) contains audio from videos about the 2008 American presidential election, such as candidates' speeches. It is 9 hours of audio. The YTGeneral data is a random sample from YouTube data, and contains a "dev" set of 9 hours of audio and an "eval" set of 8½ hours.

Since especially the YTGeneral data contains a variety of audio that should not be recognised (such as music, background noise, and speech in other languages), automatic segmentation was performed. The YTGeneral corpus also has been segmented manually. The total fraction of audio present in the manual segmentation that is missing in the automatic segmentation is 5.3 %.

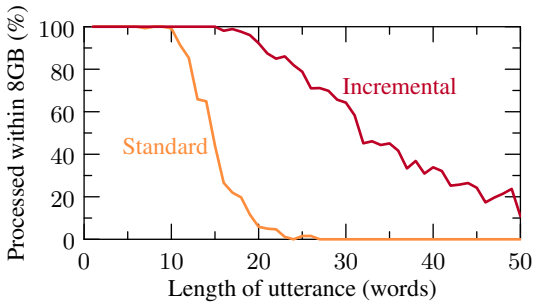
7.2 Results

Section 3 (and van Dalen and Gales 2015) have described a new method for annotating lattices with the exact error. This is a hard problem. The traditional approach to minimum Bayes risk training ues an approximation that relies on overlap between fixed segmentations in the reference and hypothesis. This may affect performance of traditional HMM systems; but it is definitely a problem for training segmental models, where trained with fixed segmentations has been shown to be suboptimal (Ragni and Gales 2012).

To test the new algorithm for error-marking lattices, a realistic training set was chosen with long utterances. The training set was 34 hours of randomly selected shows



(a) Memory use of error-marking lattices, up to 8 GB



(b) Percentage of lattices that can be error-marked within 8 GB of memory

Figure 11 *Standard and incremental error-marking algorithms.*

from the '96 release of the Hub 4 broadcast news. The hypothesis lattices (sometimes called "denominator lattices" in a reference to the CML training criterion) are produced for minimum phone error training in the standard setup at the Department of Engineering at Cambridge, which is similar to Gales *et al.* (2006).

Two algorithms are compared, both exact. Both find the phone error, which is usually used for training speech recognisers, for all paths in the hypothesis lattice. The cost metric uses monophone identities, which is realistic but in preliminary experiments increased time and space required compared to using the triphones the lattices contain. The standard algorithm is implemented in C++ using the OpenFst toolkit (Allauzen *et al.* 2007). It uses lazy composition to create the automaton in figure 6, but uses standard determinisation, which instantiates all states anyway. The implementation of the incremental algorithm uses the author's Flipsta C++ library for finite-state automata (soon to be open-sourced), which is not as optimised as OpenFst but allows more lazy operations. It lazily constructs an automaton like in figure 6, and transforms it on the fly to one like in figure 7. It provides a single-source shortest-distance algorithm that exploits the acyclicity by releasing memory for computed distances as soon as they are not needed any more.

Both algorithms were run constrained to 8 gigabytes of memory, and fail if they try to use more. A separate process was run for each utterance, measuring memory use with *GNU time*. Figure 11a shows peak memory use for the two algorithms; each dot represents an utterance (drawn alternating between the two algorithms). The hori-

Training Data (hours)		BN		YTB		
		dev03	eval03	Elect	Gdev	Geval
h4e96	74	13.4	11.8	30.6	57.4	60.7
bne	144	12.1	—	29.4	54.5	57.7
bne+tdt4	375	11.0	—	28.8	52.9	56.4

Table 1 *Performance in WER (%) of Tandem-SAT system and training set.*

zontal axis has the number of words in the utterance. The number of states in the edit distance automaton increases roughly quadratically with this, but the output roughly linearly. It is clear from the graph that on utterances around 10 words the standard algorithm starts displaying the exponential complexity that the theory predicts. The incremental algorithm shows a similar pattern, but only around 20 words.

Figure 11b illustrates the range of practical use of the two algorithms, by displaying the percentage of utterances that can be error-marked within 8 GB (the story would be similar for any other amount of memory available in real computers). The standard algorithm error-marks 99 % of utterances of up to 11 words and then quickly becomes unable to finish the computation. The incremental algorithm, however, can deal with much longer sentences: it error-marks 99 % of utterances of up to 21 words.

This improvement makes it much more feasible to error-mark lattices. For some corpora, a 20 word limit may be sufficient, or automatic segmentation can be used. It must be noted that here no pruning is used, and no admissible heuristic can be found. Pruning based on the error is hazardous because it may strip away legitimate high-error paths, which should be retained for training. It may, however, be possible to prune conservatively based on lattice arc timings.

7.3 Tandem and hybrid systems

Recently, acoustic models with neural networks have had a resurgence and have become the state of the art. This section therefore examines the two types of neural network-based acoustic models: tandem and hybrid systems.

In *hybrid* systems, the output distributions of the HMM are replaced with values derived from applying a neural network to, say, filterbank features. A tandem system, on the other had, uses a traditional GMM-HMM, but on features that contain, say, traditional PLP features augmented with neural-network features derived from them. The neural network here takes 9 frames of PLP features, and then has 4 layers of 1000 nodes each, and a *bottleneck layer* of 26 nodes. The network is initialised discriminatively, layer by layer, by appending a 6000-node output layer and training the network to context-dependent targets.

Table 1 shows word error rates (from confusion network decoding) for a tandem system trained on the broadcast news data. On matching data, dev03 and eval03, performance is in the low 10's. However, the more noisy YouTube data causes more problems. The YTElect test set, with its election campaign videos, is closer to broadcast news than the other two test sets, so performance is better. YTGeneral-dev and -eval have a greater mismatch and therefore see worse performance.

Table 2 explores the relative advantages of tandem and hybrid systems. The top row has performance of an HMM system without neural networks. The three neural-

System	BN		YTB		
	dev03	eval03	Elect	Gdev	Geval
P1 - PLP-SI	15.0	—	35.9	65.6	67.6
T1 - Tandem-SI	12.6	11.7	30.6	56.4	59.3
T2 - Tandem-SAT	12.1	—	29.4	54.5	57.7
H1 - Hybrid-SI	10.8	9.5	29.0	53.0	57.2
T1 $\otimes$ H1	10.9	9.7	28.1	52.4	55.8
T2 $\otimes$ H1	10.5	—	27.6	51.7	55.3

Table 2 *Performance in WER (%) of phonetic PLP-SI, Tandem-SI, Tandem-SAT and Hybrid-SI system trained on 144 (bne) hours of data.*

System	BN		YTB		
	dev03	eval03	Elect	Gdev	Geval
P1 - Phonetic	13.4	11.8	30.6	57.4	60.7
G1 - Graphemic	14.5	12.6	30.3	56.7	60.0
P1 $\otimes$ G1	13.0	11.4	29.9	56.4	59.8

Table 3 *Performance in WER (%) of phonetic and graphemic Tandem-SAT system trained on 74 hours (h4e96) of data.*

network systems in the next block perform much better. The difference between the two tandem system illustrates an advantage of this approach, based on GMM-HMMs: traditional methods for speaker and environment adaptation can be used. The SAT system uses speaker-adaptive training, where for each speaker a transformation is estimated that alleviates the mismatch between training and test data. This consistently results in a 3–4 % relative reduction in word error rate, which especially for the more challenging data is impressive. The hybrid system, on the next row, performs slightly better. The bottom two rows, then, combine the hybrid system with one or the other tandem system. There are two interesting things to note here. First, the combination of the two systems improves performance, which means that they are complementary. Second, the effect making the tandem system robust to noise using adaptation carries through to the combined performance.

An interesting contrast can be seen in an initial experiment with results in table 3. Two systems are compared that are both tandem systems trained on 74 hours of data, but the one is phonetic, i.e. it has a pronunciation dictionary that maps words to phones, whereas the other one is graphemic: the words are mapped to their constituent letters, *graphemes*. It is expected that for a language with such a severely non-phonetic spelling as English, removing the information about pronunciation from the system makes performance worse. Indeed, it does, on the broadcast news data. However, on the YouTube General data, performance actually increases. At low signal-to-noise ratios, the graphemic representation apparently lends more robustness to the models. The combination of the two systems, in the bottom row, again outperforms both of the single systems.

Features	HMM criterion	Log-linear model criterion	Test set A	B	C	D	Avg.
Tandem	ML	—	6.78	11.73	15.39	26.21	17.85
		CML	6.87	11.60	15.02	25.98	17.67
		large-margin	6.67	11.52	14.85	25.80	17.53
	MPE	—	7.15	11.06	14.37	24.54	16.79
		CML	6.95	11.01	14.31	24.37	16.68
		large-margin	7.06	10.93	14.10	24.26	16.59
Hybrid	cross-entropy	—	4.09	8.17	8.07	19.41	12.69
		CML	4.02	8.16	7.94	18.64	12.34
		large-margin	3.92	8.06	8.03	18.57	12.26
	MPE	—	3.96	7.64	7.79	18.51	12.05
		CML	3.81	7.61	7.45	18.25	11.89
		large-margin	3.68	7.59	7.44	17.96	11.74

Table 4 *AURORA 4: performance with log-linear models on a log-likelihood score-space from a tandem and a hybrid HMM.*

7.4 Structured log-linear models

The previous section reported performance of HMM systems that use neural networks. In this section, the HMMs are used to derive log-likelihood score-spaces. These can be used in a segmental log-linear model; the HMMs and neural networks in a hybrid system produce something analogous to a log-likelihood. Scaling for the features from the different systems was approximately tuned to the test sets.

Table 4 has results for this on the AURORA 4 data set. The first row of each block of these is the HMM performance. The top half uses tandem features in a GMM-HMM, the bottom half a neural-network-HMM hybrid. The "ML" and "cross-entropy"-trained models have not seen sequence-discriminative training, whereas the MPE-trained models have. The hybrid MPE-trained HMMs[2] has extremely competitive performance.

It could be hypothesised that applying a log-line model trained with a sequence-discriminative criterion would result in much less gain on HMMs that are trained discriminatively. Another effect that could be expected is that the score-space from the hybrid system, which has "faked" log-likelihoods, might be less useful. However, neither of these effects really turn out visible. The log-linear model consistently improves performance over the HMM system. Training the log-linear model with conditional maximum likelihood results in a 0.1–0.4 % absolute improvement. Using a large-margin criterion instead consistently improves performance furth, to 0.2–0.4 % absolute from the HMM system. Most notably, on the sequence-trained hybrid system the log-linear model improves from 12.05 to 11.74 %.

7.5 Infinite models

This section looks into performance of infinite log-linear models, discussed in section 5. These models use an infinite mixture of experts, where the experts are log-linear models. This makes training harder.

[2] Thanks to Chao Zhang for these

Expert			Test set			
Number	Type	Criterion	A	B	C	Avg.
1	HMM	ML	9.83	9.11	9.53	9.48
	log-linear	CML	8.21	7.74	8.36	8.05
		large-margin	7.97	7.54	8.31	7.86
∞	log-linear	Bayesian	8.19	7.71	8.36	8.03
		CML	8.22	7.71	8.35	8.04
		large-margin	7.69	7.39	7.98	7.63

Table 5 *AURORA 2: performance of infinite mixtures of experts, trained with different methods.*

Expert			Test set				
Number	Type	Criterion	A	B	C	D	Avg.
1	HMM	ML	7.10	15.30	12.20	23.10	17.80
	log-linear	CML	7.16	14.86	11.39	22.78	17.46
		large-margin	7.55	14.22	11.31	21.89	16.83
∞	log-linear	Bayesian	7.20	14.85	11.42	22.72	17.43
		CML	7.16	14.85	11.40	22.79	17.46
		large-margin	7.55	14.17	11.39	21.81	16.77

Table 6 *AURORA 4: performance of infinite mixtures of experts, trained with different methods.*

Table 5 shows results on the AURORA 2 corpus. In the top block are the baseline numbers: the HMM, and the single log-linear model using the log-likelihood score-space from that HMM, trained probabilistically or with a large-margin criterion. The bottom block has infinite models. The row labelled "Bayesian" is trained probabilistically, with a criterion similar to CML but with a prior over the parameters. As discussed in section 4.3, in practice this requires Gibbs sampling, here applied for the whole model: the gating function and the experts. See Yang *et al.* (2015) for details and the precise expressions. This does not yield a significant improvement over CML on one expert: the additional nonlinearity does not help performance.

The next two rows use approximations to the overall criterion. When expert is trained with CML, this replaces the sampling from the expert parameters by finding the maximum posterior estimate given the other model parameters. Though this is not exact, it is easier to implement. The results are similar.

Training the whole model with a large-margin criterion, as described in section 4.3, is hard. Again, see Yang *et al.* (2015) for details and the precise expressions. The bottom row therefore shows the result of replacing Gibbs sampling with a large-margin optimisation for the expert parameters. This type of system has previously been called an infinite structured SVM (Yang *et al.* 2014). Training the mixture of experts this way compared to training a single expert yields a 0.23 % absolute improvement in word error rate.

Table 6 shows results for the same set of experiments on the AURORA 4 corpus. The results are similar. There is no discernible difference between methods that train a probabilistic criterion, whether CML or its Bayesian variant. However, the large-margin criterion shows a good improvement, and optimising a large-margin criterion inside

an infinite mixture of experts yields a small additional improvement.

8 Conclusion

This third and last progress report has described the last part of the EPSRC Project EP/I006583/1, Generative Kernels and Score Spaces for Classification of Speech. The project's aim has been to improv speech recognition by building discriminative classifiers using features from generative models. The project has delivered work in three areas: adaptation/compensation, score-space generation and classifiers.

In the first are, a novel variational method has been proposed for compensating HMM speech recognisers for noise, which does not assume that the resulting distribution is Gaussian (van Dalen and Gales 2011).

The second area is score-space generation. A method has been proposed to compute scores from generative models for all segmentations in amortised constant time (van Dalen *et al.* 2012*a*; 2013*a*). It has been shown what property makes it so efficient by analysing the feature extraction process as a computation on a *monoid* (van Dalen and Gales 2013). This is of particular interest since the state of the art in speech recognition has shifted to using neural networks. The insight can be exploited to extract features in score-spaces derived from these.

The third and most important area is the classifiers. Since speech recognition is sequence-to-sequence classification, there are many difficulties to overcome.

A contribution that is necessary to train with all segmentations of the audio, but should also be useful within standard training of speech recognisers, is a method to compute the exact error. Discriminative training of speech recognisers usually involves an approximation of the expected word error over a lattice. New implementations of operations on finite-state automata have been proposed that allow this to be possible on much larger lattices (van Dalen and Gales 2015).

Much work was done on infinite mixtures of experts for speech recognition. A plain support vector machine or log-linear model has linear decision boundaries, which is often limiting. SVMs are usually applied with a kernel to produce nonlinear boundaries. For speech recognition, this is possible but awkward because of the sequential nature of speech. Instead, methods for using infinite structured SVMs and infinite log-linear models have therefore been proposed. They use multiple experts for different regions of space, which also leads to nonlinear decision boundaries. These two infinite models are essentially the same, but differ in the criteria used to train them. To apply these criteria to structured classification, it is necessary to generalise Bayesian methods for training models. In particular, Gibbs sampling can be used. Various approximations have been explored to make this possible.

Bibliography

Christopher Alberti, Michiel Bacchiani, Ari Bezman, Ciprian Chelba, Anastassia Drofa, Hank Liao, Pedro Moreno, Ted Power, Arnaud Sahuguet, Maria Shugrina, and Olivier Siohan (2009). "An audio indexing system for election video material." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. pp. 4873–4876.

David J. Aldous (1985). "Exchangeability and related topics." In P.L. Hennequin (ed.), *École d'Été de Probabilités de Saint-Flour XIII*, Springer, Berlin/Heidelberg, *Lecture Notes in Mathematics*, vol. 1117, pp. 1–198.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri (2007). "OpenFst: A General and Efficient Weighted Finite-State Transducer Library." In *Proceedings of the International Conference on Implementation and Application of Automata, (CIAA 2007)*. Springer, *Lecture Notes in Computer Science*, vol. 4783, pp. 11–23. URL: ⟨http://openfst.org/⟩.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2009). *Introduction to Algorithms*. MIT Press, 3rd edn.

Jason Eisner (2003). "Simpler and More General Minimization for Weighted Finite-State Automata." In *Proceedings of the Joint Meeting of the Human Language Technology Conference and the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*. Edmonton, pp. 64–71.

M. J. F. Gales and F. Flego (2010). "Discriminative classifiers with adaptive kernels for noise robust speech recognition." *Computer Speech and Language* 24 (4), pp. 648–662.

M. J. F. Gales, D. Y. Kim, P. C. Woodland, H. Y. Chan, D. Mrva, R. Sinha, and S. E. Tranter (2006). "Progress in the CU-HTK Broadcast News Transcription System." *IEEE Transactions on Audio, Speech, and Language Processing* 14 (5), pp. 1513–1525.

M. J. F. Gales, S. Watanabe, and E. Fosler-Lussier (2012). "Structured Discriminative Models For Speech Recognition: An Overview." *IEEE Signal Processing Magazine* 29 (6), pp. 70–81.

G. Heigold, W. Macherey, R. Schlüter, and H. Ney (2005). "Minimum exact word error training." In *Proceedings of the Automatic Speech Recognition and Understanding Workshop*. pp. 186–190.

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

Björn Hoffmeister, Georg Heigold, Ralf Schlüter, and Hermann Ney (2012). "WFST Enabled Solutions to ASR Problems: Beyond HMM Decoding." *IEEE Transactions on Audio, Speech, and Language Processing* 20 (2), pp. 551–564.

Julia A. Lasserre, Christopher M. Bishop, and Thomas P. Minka (2006). "Principled Hybrids of Generative and Discriminative Models." In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Martin Layton (2006). *Augmented Statistical Models for Classifying Sequence Data*. Ph.D. thesis, Cambridge University.

Tom Minka (2005). "Discriminative models, not discriminative training." Tech. Rep. MSR-TR-2005-144, Microsoft Research. URL: ⟨http://research.microsoft.com/pubs/70229/tr-2005-144.pdf⟩.

Mehryar Mohri (2000). "Minimization Algorithms for Sequential Transducers." *Theoretical Computer Science* 234, pp. 177–201.

Mehryar Mohri (2002). "Semiring frameworks and algorithms for shortest-distance problems." *Journal of Automata, Languages and Combinatorics* 7 (3), pp. 321–350.

Mehryar Mohri (2003). "Edit-distance of Weighted Automata: General Definitions and Algorithms." *International Journal of Foundations of Computer Science* 14 (06), pp. 957–982.

Mehryar Mohri (2009). "Weighted automata algorithms." In Manfred Droste, Werner Kuich, and Heiko Vogler (eds.), *Handbook of Weighted Automata*, Springer, pp. 213–254.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley (2008). "Speech recognition with weighted finite-state transducers." In Larry Rabiner and Fred Juang (eds.), *Handbook on Speech Processing and Speech Communication, Part E: Speech recognition*, Springer-Verlag, Heidelberg, Germany.

Jim Pitman (2002). "Combinatorial Stochastic Processes." Tech. Rep. 621, Department of Statistics, University of California at Berkeley.

Daniel Povey (2003). *Discriminative Training for Large Vocabulary Speech Recognition*. Ph.D. thesis, Cambridge University.

Daniel Povey, Mirko Hannemann, Gilles Boulianne, Lukas Burget, Arnab Ghoshal, Milos Janda, Martin Karafiat, Stefan Kombrink, Petr Motlicek, Yanmin Qian, Korbinian Riedhammer, Karel Vesely, and Ngoc Thang Vu (2012). "Generating Exact Lattices in the WFST Framework." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

A. Ragni and M. J. F. Gales (2011*a*). "Derivative Kernels for Noise Robust ASR." In *Proceedings of the Automatic Speech Recognition and Understanding Workshop*.

A. Ragni and M. J. F. Gales (2011*b*). "Structured Discriminative Models for Noise Robust Continuous Speech Recognition." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

A. Ragni and M. J. F. Gales (2012). "Inference Algorithms for Generative Score-Spaces." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. pp. 4149–4152.

Carl Edward Rasmussen (1999). "The infinite Gaussian mixture model." In *Proceedings of the Conference on Neural Information Processing Systems*. MIT Press, pp. 554–560.

Carl Edward Rasmussen and Zoubin Ghahramani (2001). "Infinite mixtures of Gaussian process experts." In *Proceedings of the Conference on Neural Information Processing Systems*.

Dominique Revuz (1992). "Minimization of acyclic deterministic automata in linear time." *Theoretical Computer Science* 92 (1), pp. 181–189.

Hang Su, Gang Li, Dong Yu, and Frank Seide (2013). "Error Back Propagation For Sequence Training Of Context-Dependent Deep Networks For Conversational Speech Transcription." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

The Apache Software Foundation (2004). "Apache License, Version 2.0." ⟨http://www.apache.org/licenses/LICENSE-2.0⟩.

R. C. van Dalen and M. J. F. Gales (2011). "A Variational Perspective on Noise-Robust Speech Recognition." In *Proceedings of the Automatic Speech Recognition and Understanding Workshop*.

R. C. van Dalen and M. J. F. Gales (2013). "Monoids: efficient segmental features for speech recognition." Tech. Rep. CUED/F-INFENG/TR.687, Cambridge University Engineering Department.

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

R. C. van Dalen, A. Ragni, and M. J. F. Gales (2013*a*). "Efficient Decoding with Generative Score-Spaces Using the Expectation Semiring." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

R. C. van Dalen, J. Yang, M. J. F. Gales, A. Ragni, and S. X. Zhang (2012*b*). "Generative Kernels and Score-Spaces for Classification of Speech: Progress Report." Tech. Rep. CUED/F-INFENG/TR.676, Cambridge University Engineering Department.

R. C. van Dalen, J. Yang, M. J. F. Gales, and S. X. Zhang (2013*b*). "Generative Kernels and Score-Spaces for Classification of Speech: Progress Report II." Tech. Rep. CUED/F-INFENG/TR.689, Cambridge University Engineering Department.

Rogier C. van Dalen (2015). "Flipsta finite-state automaton library." ⟨https://github.com/rogiervd/flipsta⟩.

Rogier C. van Dalen and Mark J. F. Gales (2015). "Annotating large lattices with the exact word error." In *submitted to Interspeech*.

J. Yang, R. C. van Dalen, and M. J. F. Gales (2015). "Infinite structured log-linear models for speech recognition." Tech. Rep. CUED/F-INFENG/TR.697, Cambridge University Engineering Department.

J. Yang, R. C. van Dalen, S.-X. Zhang, and M. J. F. Gales (2014). "Infinite Structured Support Vector Machines for Speech Recognition." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

Jingzhou Yang, Rogier C. van Dalen, and Mark J. F. Gales (2013). "Infinite Support Vector Machines in Speech Recognition." In *Proceedings of Interspeech*.

S.-X. Zhang and M. J. F. Gales (2011*a*). "Extending Noise Robust Structured Support Vector Machines to Larger Vocabulary Tasks." In *Proceedings of the Automatic Speech Recognition and Understanding Workshop*.

S.-X. Zhang and M. J. F. Gales (2011*b*). "Structured Support Vector Machines for Noise Robust Continuous Speech Recognition." In *Proceedings of Interspeech*.

S.-X. Zhang and M. J. F. Gales (2012). "Structured SVMs for Automatic Speech Recognition." *IEEE Transactions on Audio, Speech, and Language Processing* 21 (3), pp. 544–55.

S.-X. Zhang and M. J. F. Gales (2013). "Kernelized Log Linear Models For Continuous Speech Recognition." In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. pp. 6950–6954.

S.-X. Zhang, Anton Ragni, and M. J. F. Gales (2010). "Structured Log Linear Models for Noise Robust Speech Recognition." *IEEE Signal Processing Letters* 17, pp. 945–948.

Jun Zhu, Ning Chen, and Eric Xing (2011). "Infinite SVM: a Dirichlet Process Mixture of Large-margin Kernel Machines." In *Proceedings of the International Conference on Machine Learning*. pp. 617–624.

Geoffrey Zweig and Patrick Nguyen (2009). "A Segmental CRF Approach to Large Vocabulary Continuous Speech Recognition." In *Proceedings of the Automatic Speech Recognition and Understanding Workshop*.