UNIVERSITY OF
CAMBRIDGE
Department of Engineering

Generative Kernels and Score-Spaces for Classification of Speech: Progress Report 11

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

J. Yang
jy308@cam.ac.uk

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

S. X. Zhang
sxz20@cam.ac.uk

Technical Report CUED/F-INFENG/TR.689

January 2013

This is the second 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 work reports progress on efficient computation of generative scores, and two viarants of support vector machines for speech recognition.

Contents

1 Introduction

This is the second 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

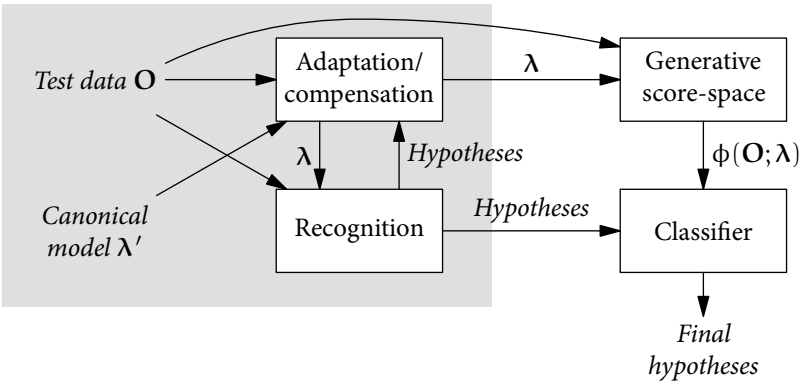

Figure 1 *Flow diagram of the project plan. The shaded region encompasses the components of a state-of-the-art speech recogniser.*

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.

This report will report on improvements in the two non-standard components in figure 1: score-space computation, and classifiers. It will also report on some work related to the project. Section 2 will discuss score-space computation. It presents a method to compute scores from generative models for all segmentations in amortised constant time (published as van Dalen *et al.* 2013). Section 3 will discuss two forms of speech recognition with large-margin classifiers. The first is from work related to the project (published as Zhang and Gales 2013), about kernelised log-linear models. The second was performed within the project (submitted as Yang *et al.* 2013), and applies the infinite SVM.

2 Features

This work uses log-linear models and SVMs for classification (see section 3 on page 10). This requires use of *joint features*: the features depend not only on the observations, but also on the labels. This work performs speech recognition. Recognising continuous speech means finding a sequence of symbols, e.g., words, of unknown length. To make inference feasible, this work aims not just to find the optimal word sequence, but to jointly optimise the word sequence and the segmentation of the audio into words. In an HMM speech recogniser, this joint optimisation by inferring sequences of HMM states that imply combinations of segmentations and word sequences. However, this relies on independence assumptions which do not model the data well. In this work, the segmentation therefore needs to be considered explicitly. The feature function will be called $\phi(\mathbf{O}, \boldsymbol{w}, \mathbf{s})$. Here, $\mathbf{O}$ is the observation sequence. $\boldsymbol{w}$ and $\mathbf{s}$ have the same length, and are the word and segment sequence, respectively. The function returns a fixed-length feature vector.

The next subsection, 2.1, will discuss the form of the feature vector used in this work, which derives from word-level hidden Markov models. Section 2.2 will then discuss how to extract these features efficiently.

2.1 Generative score-spaces

Since a class is a sequence $\boldsymbol{w}$, the number of classes grows exponentially with the length of $\boldsymbol{w}$. It is infeasible to consider all those classes as completely separate when decoding, and only a small fraction will be seen in training. Therefore, the commonality must be exploited by introducing structure into the classes. Since the feature function relates

the observation and the class, this structure must be expressed in the feature function. This work focuses on acoustic modelling, and assumes that a language model $P(\boldsymbol{w})$ is given, and produces one element of the feature vector.[1] The acoustic model part of the feature vector is additive over segments of audio:

$$\phi(\mathbf{O}, \boldsymbol{w}, \mathbf{s}) \triangleq \begin{bmatrix} \sum_{i=1}^{|\boldsymbol{w}|} \phi'(\mathbf{O}_{s_i}, w_i) \\ P(\boldsymbol{w}) \end{bmatrix}, \tag{1}$$

where $\mathbf{O}_{s_i}$ represents the audio in segment s_i. $\phi'(\mathbf{O}_{s_i}, w_i)$ is a function that extracts a feature from the combination of one segment of audio and one word. This feature vector contains zeroes except for the portion of the vector designated for the hypothesised word:

$$\phi'(\mathbf{O}_{\tau\ldots t}, w) \triangleq \begin{bmatrix} \delta(w = 1)\, \phi''(\mathbf{O}_{\tau\ldots t}, 1) \\ \vdots \\ \delta(w = V)\, \phi''(\mathbf{O}_{\tau\ldots t}, V) \end{bmatrix}. \tag{2}$$

Here, $\delta(w = v)$ is the Kronecker delta, which switches on the word-independent feature for words 1 to V, the number of words in the vocabulary. $\phi''(\mathbf{O}_{\tau\ldots t}, v)$ returns the feature vector for the word v. This report uses generative score-spaces, which means that the feature vectors are derived from the log-likelihoods of generative models. In speech recognition, the standard generative model is the hidden Markov model. Denote the likelihood for word v with $l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v)$, with parameter $\boldsymbol{\lambda}_v$.

This report will use three types of generative score-spaces for log-linear models. First, the log-likelihood score-space, which contains only the log-likelihoods of words:

$$\phi''(\mathbf{O}_{\tau\ldots t}, v) \triangleq \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v). \tag{3a}$$

The second type is the first-order derivative score-space, which appends the likelihood with respect to the parameters to the log-likelihood:

$$\phi''(\mathbf{O}_{\tau\ldots t}, v) \triangleq \begin{bmatrix} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v) \\ \nabla_{\boldsymbol{\lambda}} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v) \end{bmatrix}. \tag{3b}$$

Note that the second element, the partial derivative, is vector-valued. Higher-order derivative score-spaces are also possible. The second-order derivative score-space is defined as

$$\phi''(\mathbf{O}_{\tau\ldots t}, v) \triangleq \begin{bmatrix} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v) \\ \nabla_{\boldsymbol{\lambda}} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v) \\ \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_v) \end{bmatrix}, \tag{3c}$$

where $\nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}}$ is assumed to produce a vector.

The large-margin trained models will include for each class the features for all generative models. Log-likelihood score-spaces then use instead of (3a)

$$\phi''(\mathbf{O}_{\tau\ldots t}, v) \triangleq \begin{bmatrix} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_1) \\ \vdots \\ \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_V) \end{bmatrix}. \tag{4a}$$

[1] Last year's report (van Dalen *et al.* 2012*b*) looked into using more dimensions for language modelling but did not improve performance.

Note that this ϕ'' does not actually depend on v, because log-likelihoods for all words are used. When this ϕ'' is used in ϕ' in (2), the effect is therefore that the same vector ϕ'' is placed in a word-dependent position in ϕ'.

The first-order derivative score-space for large-margin, SVM-based, models is

$$\phi''(\mathbf{O}_{\tau\ldots t}, v) \triangleq \begin{bmatrix} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_1) \\ \nabla_{\boldsymbol{\lambda}} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_1) \\ \vdots \\ \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_V) \\ \nabla_{\boldsymbol{\lambda}} \log l(\mathbf{O}_{\tau\ldots t}; \boldsymbol{\lambda}_V) \end{bmatrix}. \tag{4b}$$

A problem with segmental features is that the segmentation, say, into words, of the adio must be considered explicitly. If all segmentations are to be considered, features for all segments must be known. The next section will therefore discuss how to extract features in generative score-spaces for all segments efficiently.

2.2 Computing scores with higher-order expectation semirings

Decoding with an explicitly segmental model requires two steps: extracting segmental features, and finding the joint optimal word sequence and segmentation. Keeping the language model fixed, the latter task can be performed in $\Theta(T^2)$ time, where T is the length of the utterance (Sarawagi and Cohen 2004; Ragni and Gales 2012). The complexity of the former task depends on the nature of the features, but computing features for the $\Theta(T^2)$ segments must take at least $\Theta(T^2)$ time. This is therefore the bottleneck for performance. Previous work (Ragni and Gales 2012) has shown how to extract generative scores that include nth-order derivatives in $\Theta(T^{2+n})$ time. This section will introduce a method to extract derivatives of any order for all segments in $\Theta(T^2)$ time, that is, amortised constant time. This method was published as van Dalen *et al.* (2013).

Since the derivatives in the generative score-space depend on all frames in a segment, a direct implementation would re-compute them completely for every hypothesised segment. This was the approach adopted in Layton (2006), using an algorithm with nested passes of forward–backward that was run separately for each hypothesised segmentation. Van Dalen *et al.* (2013) introduced a method that incrementally computes scores for all segmentations that share a common start time with only a forward pass. It augments the output and transition probabilities with their derivatives. As long as the HMMs have only few states, which for word HMMs is the case, this algorithm requires a modest amount of extra storage. When scores are required for all possible segments, they can be computed in amortised constant time.

This section will discuss how to extract the features in (3) from a word HMM and a sequence of observations. The following will first recall how the HMM likelihood can be computed for a given segment of observations. Then, it will explain how to extend this to compute the likelihood for a range of segments at the same time, with the method introduced in Ragni and Gales (2012). After that, it will discuss how first- or higher-order derivatives can be found with the same time complexity (but with a greater constant factor).

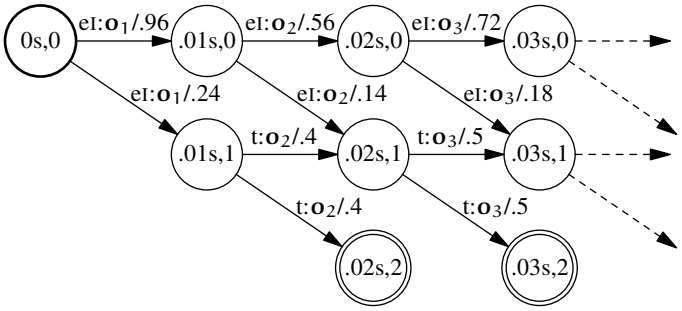

Figure 2 *Weighted finite-state transducer $\mathcal{T}$ representing a trellis.*

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

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

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

It is well-known how to compute (5) efficiently: with either the forward or the backward algorithm. Both are "single-source shortest-distance" algorithms (as they are called in the literature about finite-state automata, e.g., Mohri (2002)) that exploit the trellis structure. Both compute the sum of the paths from node s_1 to node s_2:[3]

$$\mathrm{forwd}(\mathrm{s}_1, \mathrm{s}_2, \boldsymbol{\lambda}_v) \triangleq \mathrm{backwd}(\mathrm{s}_1, \mathrm{s}_2, \boldsymbol{\lambda}_v) \triangleq \sum_{\substack{\pi:\mathrm{p}[\pi]=\mathrm{s}_1 \\ \wedge\, \mathrm{n}[\pi]=\mathrm{s}_2}} \prod_{e\in\pi} \sigma[e; \boldsymbol{\lambda}_v]\,. \tag{6}$$

The forward algorithm computes forward weights from one start node to many end nodes incrementally. Similarly, the backward algorithm computes backward weights from one end node to many start nodes. (6) can be rewritten recursively, where $\mathrm{p}[e]$

[2] Since the trellis is represented as a Mealy machine, it combines contributions from the HMM output distributions and transition matrices. This will be useful in section 2.3. This trellis automaton can be produced by composing an automaton representing the output probabilities for consecutive observations with one that produces all possible HMM state sequences $\mathbf{o}_1 \ldots \mathbf{o}_\mathrm{T}$. See van Dalen *et al.* (2012*a*) for more details, or Hoffmeister *et al.* (2012) for a more general discussion.

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

denotes the source node of an edge, and $n[e]$ the destination:

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

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

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

These can be computed efficiently with dynamic programming. The forward weights for one time in a trellis like Fig. 2 depend only on those of the previous time. The algorithm therefore progresses time instance by time instance. For a fixed HMM, the forward algorithm takes linear time in the length of the observation segment.

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

Instead, it can be noted that the forward algorithm from time τ to end time T generates likelihoods for segments starting at time τ and ending at all times $t = \tau \ldots T$ (Ragni and Gales 2012). By applying the forward algorithm for each time step $\tau = 1 \ldots T-1$, likelihoods for all $\Theta(T^2)$ segments can be found in $\Theta(T^2)$ time. Per segment, this therefore takes amortised constant time.

2.3 Computing derivatives with the expectation semiring

In addition to the log-likelihood, which the method discussed above finds efficiently, (3) contains its derivative with respect to the parameters $\boldsymbol{\lambda}_v$. This section will first explain why applying the same strategy as above speeds up the standard method for computing the derivatives merely by a constant factor. It will then introduce a different strategy, attaching derivatives to all weights, to improve the time complexity. For simplicity, this section will find the derivative of the likelihood, not of the log-likelihood. The latter is straightforward to compute by dividing the former by the likelihood.

The weight $\sigma[e; \boldsymbol{\lambda}_v]$ on each arc in Fig. 2 is a product of a transition weight and an HMM output probability. Computing its derivative $\nabla_\lambda \sigma[e; \boldsymbol{\lambda}_v]$ is therefore straightforward. The standard way of computing the derivative of the likelihood of an HMM is by using unnormalised arc posteriors $\gamma_{\tau:t}(e)$:

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

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

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

It is clear from (8b) that $\gamma_{\tau:t}(e)$ depends on the weights on edges before and after e, so that the derivative, in (8a), relaxes the Markov property (see Layton 2006, for more detail). (8b) is normally computed with the forward–backward algorithm. This combines

the weight from the start node (τ, start) up to the source of e, $p[e]$, and the backward weight from the end node (t, end) to the destination of e, $n[e]$:

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

The computation of all required forward weights can be done efficiently as described above, as can the computation of all backward weights. However, for each segment $\tau : t$, $\gamma_{\tau:t}(e)$ derives from different forward and backward passes and must be recomputed. The average time to compute (8a) for one segment is therefore $\Theta(T)$. Finding the derivatives for all segments in this way then takes $\Theta(T^3)$ time. Extending this strategy to second-order derivatives, posteriors for pairs of arcs must be computed, and the algorithm will take $\Theta(T^4)$ time, et cetera (Ragni and Gales 2012). Taking $\Theta(T^3)$ time or more to find features becomes prohibitive for speech recognition.

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

$$\sigma \triangleq \langle l, \nabla_{\boldsymbol{\lambda}} l \rangle . \tag{9}$$

This starts from the arcs weights in Fig. 2. Original weights $l[e; \boldsymbol{\lambda}]$ on arc e, dependent on parameters $\boldsymbol{\lambda}$, are replaced with

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

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

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

Then, finding the first-order score in (3) with the derivative of the log-likelihood the values in (11) is straightforward.

This can be done by generalising the operations on weights so that they maintain the invariant that the second element is the derivative of the first element, as in (9). This is called "automatic differentiation" using dual numbers (Pearlmutter and Siskind 2007). Given values for weights l_1 and l_2, the derivatives of their sum and product can be found with

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

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

The trellis in Fig. 2 is a weighted finite-state automaton. Requirements on the types of weights that can be used in weights automata are well-established (Mohri 2009). For many algorithms, including the forward algorithm, the requirement is that the weights are in a semiring. Semirings define operations $\oplus$ and $\otimes$, which generalise $+$ and $\times$ on scalars, and identities $\bar{0}$ and $\bar{1}$, which generalise 0 and 1. An important property for a, b, c to be in a semiring is that multiplication must distribute over addition, that is, $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$. This makes it possible to rewrite (6) to (7).

It turns out that the tuple in (9) is in the *expectation semiring* (Eisner 2002), so that it can be used in weighted finite-state automata. Denoting the weights with $\langle l, \nabla_\lambda l \rangle$, it follows from (12) that the semiring operations must be defined as

$$\langle l_1, \nabla_\lambda l_1 \rangle \oplus \langle l_2, \nabla_\lambda l_2 \rangle \triangleq \langle l_1 + l_2, \nabla_\lambda l_1 + \nabla_\lambda l_2 \rangle ; \tag{13a}$$

$$\langle l_1, \nabla_\lambda l_1 \rangle \otimes \langle l_2, \nabla_\lambda l_2 \rangle \triangleq \langle l_1 \cdot l_2, l_1 \cdot \nabla_\lambda l_2 + l_2 \cdot \nabla_\lambda l_1 \rangle , \tag{13b}$$

and the additive and multiplicative identities are defined as

$$\overline{0} \triangleq \langle 0, 0 \rangle ; \qquad \overline{1} \triangleq \langle 1, 0 \rangle . \tag{13c}$$

This definition ensures that both operations $\oplus$ and $\otimes$ propagate tuples with as the second element the derivative of the first element. It is also possible to extend this to higher-order derivatives to find higher-order generative scores (Li and Eisner 2009).

The expectation semiring can be used to accumulate any statistic that is additive along the edges of a path and is weighted by the original weight along the path (Eisner 2001; Heigold *et al.* 2008). It was proposed for the expectation step of expectation–maximisation in probabilistic transducers. In theory, training a speech recogniser would be possible with just a forward pass and the expectation semiring. However, this would entail keeping statistics for all speech recogniser parameters relevant to the utterance for all states in the HMM, which requires much memory, and the operations in (13) become expensive. By using the forward–backward algorithm, the order of the semiring can be one lower than using just the forward algorithm (Eisner 2001; Heigold *et al.* 2008). Since in normal speech recogniser training the start and end times are known, the sensible trade-off is to use the forward–backward algorithm.

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

2.4 Implementation

The likelihoods, and their derivatives, have a large dynamic range. It is usual for likelihoods in sequential probabilistic models to be represented by their logarithms. This is possible because the likelihoods are always non-negative. The additional statistics, on the other hand, can be positive or negative. It is possible to separately store the logarithm of the absolute value and the sign (Li and Eisner 2009).

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

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

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

of σ_1 and σ_2 as

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

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

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

In the log-domain, a slightly more numerically stable way to compute $l_1/(l_1 + l_2)$ is $1/(1 + l_2/l_1)$. The result of this can be converted to the normal domain and be used to scale $\nabla_\lambda l_1/l_1$, and similar for $\nabla_\lambda l_2/l_2$. Then, (15b) can be computed. The resulting weights are tuples of the log-likelihood (not the likelihood) and its derivative, so they can be used directly as the scores in (3).

Decoding (but not parameter estimation) can be sped up by a constant factor. The trick is to apply the dot product of the score with the parameters in (19) within the semiring. Denote the part of the parameter vector that applies to the derivative (in general, the highest-order derivative) with $\boldsymbol{\alpha}_\nabla$. Instead of the tuple as in (14), with the normalised derivative, the tuple becomes

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

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

3 Classifiers

Most current speech recognition systems are based on hidden Markov models (HMMs). These are generative models, which can be used for classifying data using Bayes' rule. The restriction on the form of classifier this yields can be lifted by extracting features from the audio with a generative model, and applying any of a number of known classifiers.

Section 2 has shown how to generate generative score-spaces from HMM-like models to model the acoustics. This section will discuss how to use these in two large-margin classifiers that allow non-linear decision boundaries in this space. Section 3.2 will report on speech recognition with kernelised structured SVMs. Section 3.3 will discuss a different strategy to produce non-linear decision boundaries, using a Bayesian non-parametric mixture of SVMs.

3.1 Log-linear models

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:

$$P(\boldsymbol{w}, \mathbf{s}|\mathbf{O}; \boldsymbol{\alpha}) \triangleq \frac{1}{Z(\mathbf{O}, \boldsymbol{\alpha})} \exp\Big(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})\Big). \tag{17}$$

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

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

$$\boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s}) \triangleq \sum_i \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i), \tag{18}$$

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 (17) will be found:

$$\begin{aligned} \arg\max_{\boldsymbol{w}, \mathbf{s}} P(\boldsymbol{w}, \mathbf{s}|\mathbf{O}; \boldsymbol{\alpha}) &= \arg\max_{\boldsymbol{w}, \mathbf{s}} \frac{1}{Z(\mathbf{O}, \boldsymbol{\alpha})} \exp\Big(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})\Big) \\ &= \arg\max_{\boldsymbol{w}, \mathbf{s}} \Big(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})\Big) = \arg\max_{\boldsymbol{w}, \mathbf{s}} \sum_i \boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i). \end{aligned} \tag{19}$$

This maximisation can be performed exactly (as in section 2.2), or by constraining the hypotheses to those found in a lattice, which will be done in the rest of this work. From here on, to simplify notation, the segmentation $\mathbf{s}$ will therefore be dropped.

3.1.1 Training criteria

There are a number of ways in which a log-linear model can be trained. The training criterion used on standard HMMs, the likelihood of observations and transcription, is unavailable since the probability of the observations is not modelled. However, it is possible to optimise the likelihood of the correct word sequence $\boldsymbol{w}_{\text{ref}}^{(r)}$ given the observations, the "conditional likelihood". This is possible for HMMs as well, when it is often called "maximum mutual information". The criterion can be written, summing over all utterances r,

$$\begin{aligned} \boldsymbol{\alpha}^* &= \arg\max_{\boldsymbol{\alpha}} \sum_r \log P\Big(\boldsymbol{w}_{\text{ref}}^{(r)} \Big| \mathbf{O}^{(r)}; \boldsymbol{\alpha}\Big) \\ &= \arg\max_{\boldsymbol{\alpha}} \sum_r \Big[\log\Big(\exp\Big(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)})\Big)\Big) - \log\Big(Z(\mathbf{O}^{(r)}, \boldsymbol{\alpha})\Big)\Big] \\ &= \arg\max_{\boldsymbol{\alpha}} \sum_r \left[\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)}) - \log\left(\sum_{\boldsymbol{w}} \exp\Big(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}^{(r)}, \boldsymbol{w})\Big)\right)\right]. \end{aligned} \tag{20}$$

This criterion can be maximised with a form of expectation–maximisation.

Another criterion that is frequently used for speech recognition is minimum Bayes' risk (MBR). This uses a loss function $\mathcal{L}(\boldsymbol{w}_{\text{ref}}, \boldsymbol{w})$ between the reference word sequence

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \sum_{r} \sum_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_{\text{ref}}, \boldsymbol{w}) \, P\big(\boldsymbol{w} \big| \mathbf{O}^{(r)}; \boldsymbol{\alpha}\big). \tag{21}$$

This will be used for the experiments in section 4.1.

A third training criterion, which will be used in the next two sections, is the maximum margin criterion. This aims to improve the margin between the reference transcription and the most competing sequence $\boldsymbol{w}$:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \sum_{r} \left[\max_{\boldsymbol{w} \neq \boldsymbol{w}_{\text{ref}}^{(r)}} \mathcal{L}\Big(\boldsymbol{w}_{\text{ref}}^{(r)}, \boldsymbol{w}\Big) - \log\left(\frac{P\big(\boldsymbol{w}_{\text{ref}}^{(r)} \big| \mathbf{O}^{(r)}; \boldsymbol{\alpha}\big)}{P\big(\boldsymbol{w} \big| \mathbf{O}^{(r)}; \boldsymbol{\alpha}\big)} \right) \right]_+ . \tag{22}$$

Here, $[\cdot]_+$ is the hinge-loss function, and the margin is defined with a loss function and the log-posterior ratio (Zhang *et al.* 2010).

This criterion is the same as for structured SVMs, as is decoding as per (19). Known algorithms for structured SVMs can therefore be applied. A Gaussian prior, $\log p(\boldsymbol{\alpha}) = \log \mathcal{N}(\boldsymbol{\mu}_\alpha, C\mathbf{I}) \propto \frac{1}{2C} \|\boldsymbol{\alpha} - \boldsymbol{\mu}_\alpha\|_2^2$, is usually introduced into the training criterion (Zhang *et al.* 2010). Substituting (17) into (22) and cancelling out the normalisation term yields the following convex optimisation:

$$\begin{aligned} &\arg\min_{\boldsymbol{\alpha}, \xi} \frac{1}{2} \|\boldsymbol{\alpha} - \boldsymbol{\mu}_\alpha\|_2^2 + \frac{C}{R} \xi \\ &\text{s.t. } \forall \text{ competing hypotheses } \left\{ \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(R)} \right\} : \\ &\boldsymbol{\alpha}^{\mathsf{T}} \sum_{r} \Big[\phi(\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)}) - \phi(\mathbf{O}^{(r)}, \boldsymbol{w}^{(r)}) \Big] \geq \sum_{r} \mathcal{L}\Big(\boldsymbol{w}_{\text{ref}}^{(r)}, \boldsymbol{w}^{(r)}\Big) - \xi, \end{aligned} \tag{23}$$

where $\xi \geq 0$ is the "slack variable", introduced to replace the hinge loss. (23) can be solved using the cutting-place algorithm (Joachims *et al.* 2009).

3.2 Kernelised log-linear models

So far, large-margin training of log-linear models has assumed the feature, or primal, space. However, it is normal to use SVMs on a *kernelised* (or "dual") space, which allows non-linear boundaries in the original space. This involves using training data points explicitly in classification. This is not trivial to do with joint feature spaces, because the inclusion of an infinite number of different classes makes the number of data points essentially infinite. This section will summarise the approach used in work related to the project (Zhang and Gales 2013) to kernelise large-margin log-linear models for speech recognition.

A good way of training the parameters $\boldsymbol{\alpha}$ in feature space of a structured SVM for speech recognition is to alternate between optimising the parameters and finding the new competing hypotheses (Zhang and Gales 2012). The competing hypotheses form the constraints for a quadratic programming problem that can be solved with the cutting-plane algorithm. At every iteration, new constraints are added to the constraint set. There are two ways of doing this (Joachims *et al.* 2009). The obvious method is to add the current most competing hypothesis for each word at each iteration (the

n-slack algorithm). The alternative is to add a constraint which is a sum over all utterances (the 1-slack algorithm). Both algorithms converge to a minimum of the same function. Though the n-slack algorithm would take fewer iterations, it generates so many constraints that even for moderate-sized tasks, it exhausts a modern computer's memory (Zhang and Gales 2012). The 1-slack algorithm is therefore used.

The kernel trick rewrites (23) so that the feature function $\phi(\cdot)$ is not evaluated except through the dot product $\phi(\cdot)^{\mathsf{T}}\phi(\cdot)$ of the feature functions applied to two observations. The dot product can then be replaced by a kernel function $k(\cdot,\cdot)$ that relates two observations. Since the feature function here is a joint feature function, taking as parameters not only the observation but also the assigned class, the kernel function must take classes also. The straightforward linear kernel can then be defined as

$$k\big((\mathbf{O}^{(r)}, \boldsymbol{w}^{(r)}), (\mathbf{O}^{(r')}, \boldsymbol{w}^{(r')})\big) \triangleq \phi(\mathbf{O}^{(r)}, \boldsymbol{w}^{(r)})^{\mathsf{T}}\phi(\mathbf{O}^{(r')}, \boldsymbol{w}^{(r')}). \tag{24}$$

Normally, the results of the kernel function applied to each pair of training data points are used as entries of the *Gram matrix* $\mathbf{G}$. However, since the number of possible classes is infinite, the Gram matrix would have infinite size. Instead, the 1-slack algorithm is used, and at each iteration the Gram matrix is extended by one row and column, representing the sum over the competing hypotheses for each utterance. The criterion, formulated in terms of a vector of parameters $\boldsymbol{\alpha}^{\text{dual}}$ instead of $\boldsymbol{\alpha}$, becomes

$$\max_{\alpha_m^{\text{dual}} \geq 0} \left[\sum_{m=1}^{M} \alpha_m^{\text{dual}} \mathcal{L}_m - \frac{1}{2} \sum_{m=1}^{M} \sum_{m'=1}^{M} \boldsymbol{\alpha}^{\text{dual}\mathsf{T}} \mathbf{G} \boldsymbol{\alpha}^{\text{dual}} \right], \tag{25a}$$

$$\text{s.t.} \sum_{m=1}^{M} \alpha_m^{\text{dual}} = C, \tag{25b}$$

where $\mathcal{L}_m$ is the average loss at iteration m

$$\mathcal{L}_m \triangleq \frac{1}{R} \sum_{r=1}^{R} \mathcal{L}(\boldsymbol{w}_{\text{ref}}^{(r)}, \boldsymbol{w}_m^{(r)}), \tag{25c}$$

where $\boldsymbol{w}_m^{(r)}$ is the competing word sequence for the mth iteration. $\mathbf{G}$ is the Gram matrix, with elements

$$g_{mm'} \triangleq \frac{1}{R^2} \left[\sum_{r=1}^{R} \Big(\phi(\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)}) - \phi(\mathbf{O}^{(r)}, \boldsymbol{w}_m^{(r)}) \Big) \right]^{\mathsf{T}} \left[\sum_{r=1}^{R} \Big(\phi(\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)}) - \phi(\mathbf{O}^{(r)}, \boldsymbol{w}_{m'}^{(r)}) \Big) \right]. \tag{25d}$$

$g_{mm'}$ can be entirely expressed in terms of the kernel function:

$$g_{mm'} = \frac{1}{R^2} \sum_{r=1}^{R} \sum_{r'=1}^{R} \Big[k\big((\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)}), (\mathbf{O}^{(r')}, \boldsymbol{w}_{\text{ref}}^{(r')})\big) - k\big((\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)}), (\mathbf{O}^{(r')}, \boldsymbol{w}_{m'}^{(r')})\big) - k\big((\mathbf{O}^{(r)}, \boldsymbol{w}_m^{(r)}), (\mathbf{O}^{(r')}, \boldsymbol{w}_{\text{ref}}^{(r')})\big) + k\big((\mathbf{O}^{(r)}, \boldsymbol{w}_m^{(r)}), (\mathbf{O}^{(r')}, \boldsymbol{w}_{m'}^{(r')})\big) \Big]. \tag{25e}$$

This relates every training utterance to every training utterance, which, depending on the form of k, can be slow. Section 3.2.1 will therefore discuss a sensible form that makes training faster.

Training follows the 1-slack algorithm, in three steps. First, solve the dual quadratic program based on the current Gram matrix $\mathbf{G}$. At iteration m, this will return a m-dimensional parameter vector $\boldsymbol{\alpha}^{\text{dual}}$. Second, use this $\boldsymbol{\alpha}^{\text{dual}}$ to find the closest competing hypothesis $\boldsymbol{w}_{m+1}^{(r)}$ for each utterance r in parallel. These hypotheses will be used to compute the losses and the evaluate the kernel function. Third, compute the kernel values and update the Gram matrix.

Decoding an utterance $\mathbf{o}$ with a kernelised log-linear model uses the dual version of (19) which sums over all training utterances and all competing hypotheses:

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \sum_{m,r} \alpha_m^{\text{dual}} \left[k\big((\mathbf{O}, \boldsymbol{w}), (\mathbf{O}^{(r)}, \boldsymbol{w}_{\text{ref}}^{(r)})\big) - k\big((\mathbf{O}, \boldsymbol{w}), (\mathbf{O}^{(r)}, \boldsymbol{w}_m^{(r)})\big) \right]. \tag{26}$$

This can be seen as appending an additional row and column to the Gram matrix and optimising for $\boldsymbol{w}^*$ (unlike (25a), which optimises for α_m^{dual}). Without any further requirements on the form of the kernels, this maximisation would involve explicitly evaluating $k(\cdot,\cdot)$ for all possible $\boldsymbol{w}$, which is infeasible. It is therefore important to constrain the form of the kernel, which will be discussed in the next section.

3.2.1 Form of kernel

The general form of the kernel in (24) relates two utterances and corresponding transcriptions. This makes training and decoding slow. In standard HMM systems, the independence assumptions between the acoustics of words in the sequence make training and decoding feasible. In log-linear models using the feature space, a similar effect (at least for decoding) can be achieved by constraining the acoustic feature, in (1), to be a sum over segments. For the dual form, to make words independent, and since kernels must be symmetric, the kernel must be additive over the words in both sequences. Assume that word sequence $\boldsymbol{w}^{(r)}$ for observation $\mathbf{O}^{(r)}$ can be associated deterministically with segmentation $\mathbf{s}^{(r)}$, the kernel function must be of the form

$$\begin{aligned} k\big((\mathbf{O}^{(r)}, \boldsymbol{w}^{(r)}), (\mathbf{O}^{(r')}, \boldsymbol{w}^{(r')})\big) = \\ \sum_{i=1}^{|\boldsymbol{w}^{(r)}|} \sum_{i'=1}^{|\boldsymbol{w}^{(r')}|} \delta\big(w_i^{(r)} = w_{i'}^{(r')}\big) k_{w_i^{(r)}}\left(\mathbf{O}_{s_i^{(r)}}^{(r)}, \mathbf{O}_{s_{i'}^{(r')}}^{(r')}\right), \end{aligned} \tag{27}$$

where k_w is a sequence kernel, potentially specific to the word, relating the two audio segments. This restriction means that for any hypothesised word in an utterance, only the kernel between it and all occurrences of the same word in the training data have to be considered. This is feasible (if slower than decoding with a standard HMM) and detailed in Zhang and Gales (2012).

Different word kernels k_w can be chosen, which is where the strength of kernelising log-linear models lies. Extracting two feature vectors using $\phi_w''(\cdot)$ as in (4), the equivalent of the model in primal space results from a linear kernel,

$$k_{\text{linear},w}(\mathbf{O}^{(1)}, \mathbf{O}^{(2)}) \triangleq \phi_w''(\mathbf{O}^{(1)})^{\mathsf{T}} \phi_w''(\mathbf{O}^{(2)}). \tag{28a}$$

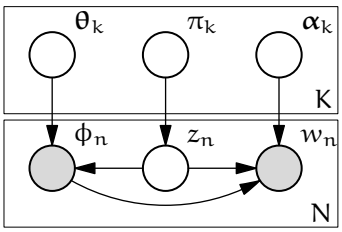

Figure 3 *The graphical model for a mixture of experts.*

A common generalisation of this is the dth-order polynomial kernel:

$$k_{\text{poly},w}(\mathbf{O}^{(1)}, \mathbf{O}^{(2)}) \triangleq \left(\phi''_w(\mathbf{O}^{(1)})^{\mathsf{T}} \phi''_w(\mathbf{O}^{(2)}) + b\right)^d. \tag{28b}$$

Another well-known kernel is the radial basis function kernel with covariance σ^2:

$$k_{\text{rbf},w}(\mathbf{O}^{(1)}, \mathbf{O}^{(2)}) \triangleq \exp\left(-\frac{1}{2\sigma^2}\left\|\phi''_w(\mathbf{O}^{(1)}) - \phi''_w(\mathbf{O}^{(2)})\right\|_2^2\right). \tag{28c}$$

3.3 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.

3.3.1 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 3 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 3.3.2 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 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.

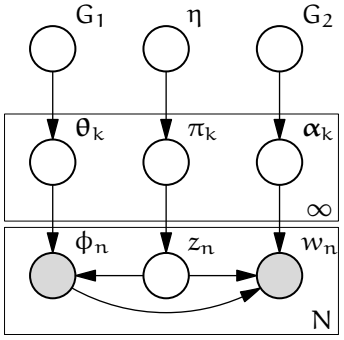

Figure 4 *The graphical model for an infinite mixture of experts.*

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

$$P(w|\phi, \theta, \boldsymbol{\alpha}) = \sum_k P(z = k|\phi, \theta)\, P(w|\phi, \boldsymbol{\alpha}, z = k)\,. \tag{29}$$

3.3.2 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 *et al.* 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 3.

The graphical model is given in figure 4. 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

$$\boldsymbol{\pi} \sim \text{Dirichlet}(\eta)\,; \tag{30}$$

$$z_n \sim \text{Categorical}[\boldsymbol{\pi}]\,. \tag{31}$$

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:

$$P\big(z_n = k \big| \boldsymbol{z}_{\setminus n}, \eta\big) \propto \begin{cases} N_k, & \text{if } k \text{ is an existing expert;} \\ \eta, & \text{if } k \text{ is a new expert.} \end{cases} \tag{32}$$

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 (32), 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 $\boldsymbol{\Theta} = \{\theta, \boldsymbol{\alpha}\}$ 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. $\theta, \boldsymbol{\alpha}$ remain the same. Because of the exchangeability of $\boldsymbol{z} = \{z_1, \ldots, z_N\}$, the posterior distribution of z_n given the value of its neighbours, according to which it is sampled, is

$$P\big(z_n \big| \boldsymbol{z}_{\setminus n}, \phi_n, \theta_k, w_n, \boldsymbol{\alpha}_k\big) \propto P\big(z_n \big| \boldsymbol{z}_{\setminus n}\big)\, p(\phi_n | \theta_{z_n})\, P(w_n | \phi_n, \boldsymbol{\alpha}_{z_n}), \tag{33}$$

where the first term $P\big(z_n \big| \boldsymbol{z}_{\setminus n}\big)$ is given according to (32). When k is a represented component, z_n can be directly sampled from equation (33). When k is an unrepresented component, however, the term $p(\phi_n | \theta_{z_n})$ is given by $\int p(\phi_n|\theta)\, G_1(\theta) d\theta$; and $P(w_n | \phi_n, \boldsymbol{\alpha}_{z_n})$, is given by $\int P(w_n|\phi_n, \boldsymbol{\alpha})\, , G_2(\boldsymbol{\alpha}) d\boldsymbol{\alpha}$.

In order to sample all z_n in parallel, an approximation is made here. All unrepresented components are treated as one, so if an observation ϕ_n is allocated to the unrepresented component, then always $z_n = K + 1$, where K is the current number of representative components.

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:

$$P(w|\phi, \mathcal{D}) = \int P(w|\phi, \boldsymbol{\Theta})\, p(\boldsymbol{\Theta}|\mathcal{D}) d\boldsymbol{\Theta}, \tag{34a}$$

where $\boldsymbol{\Theta} = \{\theta_k, \boldsymbol{\alpha}_k\}_{k=1\ldots K}$ are all the parameters of the mixture of experts. However, since the number of components is infinite, (34a) cannot be computed. Instead, the samples acquired with Gibbs sampling are used. Denote with $\boldsymbol{\Theta}^{(l)}$ 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:

$$\boldsymbol{\Theta}^{(l)} \sim p\big(\boldsymbol{\Theta}|\mathcal{D}\big). \tag{34b}$$

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

$$\begin{aligned} P(w|\phi, \mathcal{D}) &\simeq \frac{1}{L} \sum_{l=1}^{L} P\left(w\middle|\phi, \Theta^{(l)}\right) \\ &= \frac{1}{L} \sum_{l=1}^{L} \sum_{k=1}^{K^{(l)}} P\left(z = k\middle|\phi, \theta^{(l)}\right) P\left(w\middle|\phi, \alpha_k^{(l)}\right). \end{aligned} \tag{34c}$$

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

3.3.3 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, discussed in section 3.1.1, where the classes have no structure. Unlike standard SVMs, however, there are more than two classes, so that the large-margin criterion in (23) is required.

To use the infinite SVM, however, SVMs must be interpreted as probabilistic models. This is necessary both for classification (in (34c)), because the experts' distributions must be summed, but also for Gibbs sampling. The interpretation as log-linear models was given in section 3.1.1, and this 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

$$P(w|\phi, z, \alpha) \propto \exp\left(\alpha_z^T \mathbf{\Phi}(\phi, w)\right), \tag{35}$$

where α_z is the parameter vector for expert z, and $\mathbf{\Phi}(\phi, w)$, gives a vector like the one in (2) 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 (as introduced in section 3.1.1) 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.

4 Experiments

The methods in this report were tested in a log-linear model on a small, noise-corrupted corpus: AURORA 2. This makes it possible to test the interaction with noise compensation methods. The task uses a small vocabulary and no language model, which makes experiments without such optimisations as pruning possible. AURORA 2 (Hirsch and

SNR	Test set											
	A			B			C			Average		
(dB)	HMM	l	l, ∇l	HMM	l	l, ∇l	HMM	l	l, ∇l	HMM	l	l, ∇l
20	1.69	1.43	1.01	1.46	1.20	0.80	1.57	1.42	0.96	1.57	1.46	1.03
15	2.36	1.95	1.28	2.37	1.82	1.34	2.47	2.18	1.72	2.38	1.99	1.32
10	4.39	3.62	2.62	4.12	3.22	2.53	4.49	3.82	2.80	4.30	3.63	2.65
05	11.20	8.94	7.48	10.05	7.89	6.74	10.69	8.76	7.86	10.64	9.01	7.52
00	29.54	23.25	21.57	27.54	22.18	20.84	28.41	23.96	23.31	28.51	23.81	22.14
00–20	9.84	7.84	6.79	9.11	7.26	6.45	9.53	8.03	7.33	9.48	7.98	6.93

Table 1 *WERs for decoding with the expectation semiring.*

Pearce 2000) is a standard digit string recognition task. The generative model has whole-word HMMs with 16 states and 3 components per state. The number of HMM parameters is 46 732. The HMMs are compensated with unsupervised vector Taylor series (VTS) compensation as in Gales and Flego (2010). The HMM parameters to derive features for the discriminative model are trained on clean data.

With features consisting of just word log-likelihoods, the discriminative model has 13 parameters, corresponding to the log-likelihoods of the 13 words (11 digits plus "sil" and "sp"). In first-order score-spaces the derivatives of the log-likelihood are computed as in section 2.3 and appended (like in Gales and Flego (2010) the data are whitened separately for each Gaussian before computing the derivative).

The discriminative models are trained on multi-style training data.

4.1 Optimising segmentations

The feature extraction process in section 2.2 was used to decode with a discriminatively-trained log-linear model, finding the optimal segmentation. The discriminative models were initialised to use just the likelihoods from the generative model. They were then trained with a minimum Bayes risk criterion as in Ragni and Gales (2011*b*). This used a large lattice with many, but not all, segmentations for the numerator and denominator. Test set A was used as the validation set to stop training.

Only derivatives of the compensated means are used, since including variances led to rapid over-fitting. The number of parameters was 21 554. Second-order score-spaces resulted in generalisation problems because of the small training set, and initial experiments did not yield improvements over first-order score-spaces.

Word error rates for the experiments are in table 1. Apart from numerical differences, they are the same as in Ragni and Gales (2012). However, there the derivatives were recomputed for every segment, whereas in this paper they are found with the expectation semiring as in section 2.3, which even with the unoptimised implementation is much faster.

With just likelihood features ("l"), the log-linear model is closely related to the HMM. The difference is merely that within words, all paths are taken into account, and that there are word-specific discriminatively-trained parameters, effectively scaling factors. This yields consistent improvements of 10–15 % on test sets A and B, with greater improvements at lower signal-to-noise ratios. Adding derivative features ("l, ∇l") introduces longer-range dependencies that break the Markov assumption. This im-

Model	Criterion	Kernel	Test set A	B	C	Average
HMM-VTS	ML	—	9.8	9.1	9.5	9.5
Multi-class SVM	large-margin	linear	8.3	8.1	8.6	8.3
Log-linear model	conditional ML	linear	8.1	7.7	8.3	8.1
	large-margin	linear	7.9	7.3	8.0	7.7
	large-margin	polynomial	7.6	7.1	7.9	7.5

Table 2 *Word error rates for VTS-compensated HMMs, multi-class SVMs, and log-linear models trained using conditional maximum likelihood and large-margin criteria, with linear and second-order polynoial kernels.*

proves recognition consistently. Where discrimination relies most on modelling the speech accurately, at higher signal-to-noise ratios, this helps most, with 18–33 % relative improvement at 10–20 dB.

Using the optimal segmentation instead of the HMM segmentation accounts for around 5 % of the improvement. Features for large-vocabulary systems will be extracted per phone, like in Ragni and Gales (2011*a*), so that the segmentation is likely to have greater impact on performance.

4.2 Kernelised structured SVM

Section 3.2 discussed kernelised log-linear models trained with a large-margin criterion. This section reports on experiments with that.

Test set A was used as the development set for tuning parameters for all systems, such as C in (25a). To evaluate the performance of log-linear models, large margin training and kernels, several configurations were compared. The baseline system was an HMM with VTS compensation. These compensated HMMs were also used to derive: the noise-robust log-likelihood features; the word-level segmentation for the multi-class SVMs; and the training and decoding lattices for the log-linear models. The performance of VTS-compensated HMMs, multi-class SVMs, and log-linear models with different training criteria and kernels are shown in table 2.

Examining the results in table 2 shows that the large-margin log-linear model with a second-order polynomial kernel achieved the best results among all the systems. For multi-class SVMs, the observation sequence is first segmented into words based on HMMs and individual words classified independently. The difference in performance between the log-linear model and multi-class SVM systems shows the impact of sentence-level modelling. The overall gain from using kernelised log-=linear models over the VTS-compensated HMM system was over 22 %. The gain from using polynomial kernels over linear kernels was 3 %. Note that without kernelisation, it is impractical to apply large-margin log-linear models with a polynomial kernel, since it requires computing and keeping all the high-dimensional joint features explicitly. However, in the dual space only the Gram matrix is required.

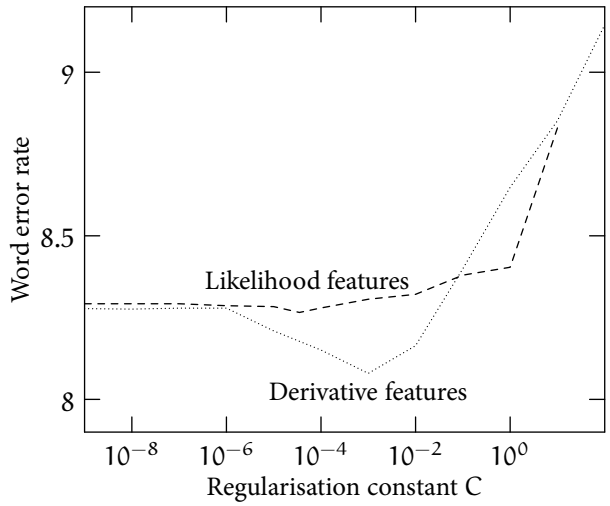

Figure 5 *Word error rates for different settings of the regularisation constant.*

			Test set			
Model	Features	Dimensions	A	B	C	Average
HMM	—	—	9.84	9.11	9.53	9.48
SVM	log-likelihood	12	8.29	7.90	8.61	8.20
iSVM			8.25	7.87	8.53	8.15
SVM	derivatives	558	8.28	7.85	8.63	8.18
iSVM			8.05	7.81	8.44	8.04

Table 3 *Word error rates for the infinite SVM on AURORA 2.*

4.3 Infinite SVMs

Section 3.3 has discussed how infinite support vector machines can be used for speech recognition. This section discusses the experimental results.

The segmentation according to the best HMM hypothesis is used, and every segment is re-classified using an infinite SVM. This is sometimes called acoustic code-breaking. In the experiments here, log-likelihood feature vectors and derivative feature vectors are used. To keep training with derivative feature vectors feasible, only the first element of the derivative with respect to each mean is used in this paper. The number of samples (in the form of a mixture of experts) from the infinite mixture model is 10.

In our experiments, all the experts (SVMs) of the iSVM share the same C, which is turned on test set A. Figure 5 illustrates the word error rate of the iSVM on different feature spaces with various values of C. Since the parameter of each expert is given a Gaussian prior with mean μ_α, which is obtained from the multi-class SVM, the iSVM only achieves the baseline performance of the multi=class SVM when C is small. By introducing the non-zero mean μ_α, the iSVM can at least achieve the performance of the multi-class SVM, and the optimised configuration can be obtained by gradually increasing C. Without the prior on the parameters, the iSVM would have poor performance, since not all experts are assigned sufficient data.

The experimental results are listed in table 3 on the preceding page. All the discriminative models outperform the VTS-compensated HMM baseline system. On the log-likelihood feature space and derivative feature space, the ISVM achieves better performance than the multi-class SVM. This gain is obtained by the fact that the ISVM explores the distribution of the training data and infers the number of experts, then applies different experts focussed on difference regions of the feature space to make an ensemble decision, rather than applying a single classifier on the whole feature space.

5 Conclusion

This report has documented progress in two key areas within the EPSRC Project EP/I006583/1, Generative Kernels and Score Spaces for Classification of Speech. The areas are score-space generation and classifiers.

Section 2 proposed a method to computing generative scores for many segmentations at once. This is vital for making speech recognition with generative score-spaces efficient (the standard Viterbi algorithm cannot be used). The method uses the forward algorithm for an generative model for one word. However, it replaces the forward probabilities with values in the *expectation semiring*, which allows derivatives of the probabilities to be propagated.

Section 3 discussed two methods for using large-margin classifiers to give nonlinear decision boundaries in this feature space. The first was by kernelising structured support vector machines; the second by using mixtures of SVMs, the ISVM.

For the coming year there are two main strands of work that will be performed. The first is to consider all segmentations for training log-linear models, by estending the minimum Bayes' risk training algorithm. The second is to combine mixtures of experts with structured classifiers, i.e. extending the ISVM to word sequences.

Bibliography

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.

Jason Eisner (2001). "Expectation Semirings: Flexible EM for Finite-State Transducers." In *Proceedings of the ESSLLI Workshop on Finite-State Methods in Natural Language Processing (FSMNLP)*.

Jason Eisner (2002). "Parameter Estimation for Probabilistic Finite-State Transducers." In *Proceedings of the Annual Meeting on Association for Computational Linguistics*. pp. 1–8.

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

Georg Heigold, Thomas Deselaers, Ralf Schlüter, and Hermann Ney (2008). "Modified MMI/MPE: A Direct Evaluation of the Margin in Speech Recognition." In *International Conference on Machine Learning*. Helsinki, Finland, pp. 384–391.

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.

T. Joachims, T. Finley, and Chun-Nam Yu (2009). "Cutting-Plane Training of Structural SVMs." *Machine Learning* 77 (1), pp. 27–59.

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

Zhifei Li and Jason Eisner (2009). "First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests." In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

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

Barak A. Pearlmutter and Jeffrey Mark Siskind (2007). "Lazy Multivariate Higher-Order Forward-Mode AD." In *Proceedings of the annual symposium on Principles of programming languages*. pp. 155–160.

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

A. Ragni and M. J. F. Gales (2011*a*). "Derivative Kernels for Noise Robust ASR." In *Proceedings of* ASRU.

A. Ragni and M. J. F. Gales (2011*b*). "Structured Discriminative Models for Noise Robust Continuous Speech Recognition." In *Proceedings of* ICASSP.

A. Ragni and M. J. F. Gales (2012). "Inference Algorithms for Generative Score-Spaces." In *Proceedings of* ICASSP. pp. 4149–4152.

Carl Edward Rasmussen, , and Zoubin Ghahramani (2001). "Infinite mixtures of Gaussian process experts." In *Proceedings of* NIPS.

Carl Edward Rasmussen (1999). "The infinite Gaussian mixture model." In *Proceedings of* NIPS. MIT Press, pp. 554–560.

Sunita Sarawagi and William W. Cohen (2004). "Semi-Markov Conditional Random Fields for Information Extraction." In *Proceedings of* NIPS.

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). "Efficient Decoding with Generative Score-Spaces Using the Expectation Semiring." In *Proceedings of* ICASSP.

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.

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 (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* ICASSP. 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* ICML. pp. 617–624.