

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

End-to-end neural segmental models for speech recognition

Citation for published version:
Hao, T, Lu, L, Kong, L, Gimpel, K, Livescu, K, Dyer, C, Smith, NA & Renals, S 2017, 'End-to-end neural segmental models for speech recognition', *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 8, pp. 1254-1264. https://doi.org/10.1109/JSTSP.2017.2752462

Digital Object Identifier (DOI):
10.1109/JSTSP.2017.2752462

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
IEEE Journal of Selected Topics in Signal Processing

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.



Download date: 19. Mar. 2024

END-TO-END NEURAL SEGMENTAL MODELS FOR SPEECH RECOGNITION

Hao Tang, *Student Member, IEEE*, Liang Lu, *Member, IEEE*, Lingpeng Kong, Kevin Gimpel, Karen Livescu, *Member, IEEE*, Chris Dyer, Noah A. Smith, and Steve Renals, *Fellow, IEEE*

arXiv:1708.00531v2 [cs.CL] 15 Aug 2017

***Abstract*—Segmental models are an alternative to frame-based models for sequence prediction, where hypothesized path weights are based on entire segment scores rather than a single frame at a time. Neural segmental models are segmental models that use neural network-based weight functions. Neural segmental models have achieved competitive results for speech recognition, and their end-to-end training has been explored in several studies. In this work, we review neural segmental models, which can be viewed as consisting of a neural network-based acoustic encoder and a finite-state transducer decoder. We study end-to-end segmental models with different weight functions, including ones based on frame-level neural classifiers and on segmental recurrent neural networks. We study how reducing the search space size impacts performance under different weight functions. We also compare several loss functions for end-to-end training. Finally, we explore training approaches, including multi-stage vs. end-to-end training and multitask training that combines segmental and frame-level losses.**

***Index Terms*—segmental models, connectionist temporal classification, end-to-end training, multitask training**

I. Introduction

Automatic speech recognition (ASR) has been treated as a graph search problem since its early development [1], and the graph search approach has been popularized by the use of hidden Markov models (HMM) [2], [3]. Given a sequence of acoustic feature vectors, such as log mel filter bank features or mel frequency cepstral coefficients (MFCC), recognition proceeds by computing a weight at every time point for every label, such as an HMM sub-phonetic state. The search space is the set of all sub-phonetic state sequences that corresponds to the set of all word sequences. Weights for transitioning from one word to another (the language model) are included at the states corresponding to boundaries between words. Recognizing speech becomes the task of finding the maximum-weight sequence of states, considering both the weights from the acoustic features and the weights from the word transitions. Since this approach computes a weight for every acoustic feature vector, or every frame, at every time point, it is commonly referred to as a frame-based approach. Many model types that have been proposed as alternatives to HMMs, such as conditional random fields (CRF) [4] and support vector machine (SVM)-based models [5], are still frame-based because the search space remains the same.

The inherent limitation of frame-based models is that the weights can only depend on a fixed length of input at a given time point. In order to incorporate richer linguistic information, units other than frames, such as segments [6], have been proposed. A segment is a variable-length unit, such as a phoneme [6], [7] or even a whole word [8], [9]. Models operating on segments, known as segmental models, can take into account the start time, end time, and the associated label to compute the weights of segments. The ability to incorporate arbitrary information within a segment, such as duration [10] and acoustic landmarks [11], makes segmental models appealing for speech recognition. In fact, segmental models were the state of the art for phoneme recognition on the TIMIT dataset [12] for many years [13].

However, the flexibility of segmental models comes at a price. The search space of segmental models includes all possible ways of segmenting the speech input and all possible ways of labeling the segments, forming a significantly larger search space than the one that frame-based models consider. To bypass the large search space, early development of segmental models considered restricted search spaces produced by pruning based on heuristics [6] or based on a first-pass frame-based recognizer [13], [9]. Segmental models that operate on the full search space—first-pass segmental models—have not been explored until recently [14]. Since then, there has been a variety of work exploring better segment representations for first-pass segmental models, especially ones that depend on neural networks [15], [16], [17], [18].

Better, but more computationally expensive, segment representations are of little practical use unless the efficiency of the models is improved. Therefore, much of the work on improving segment representations has been tied to specific approaches for reducing the search space. For example, segment representations based on multilayer perceptrons (MLP) are used in [15], where the search space is reduced by restricting the form of the weight function; segment representations based on convolutional neural networks and MLPs are explored in [17], where the search space is reduced by pruning; segment representations based on long short-term memory (LSTM) networks are used in [18], where the search space is reduced by reducing the time resolution. In this work, we will consider different segment representations and study how they behave under different search spaces.

Hao Tang, Liang Lu, Kevin Gimpel, and Karen Livescu are with Toyota Technological Institute at Chicago, Chicago, IL 60637 USA (email: {haotang,llu,kgimpel,klivescu}@ttic.edu).

Lingpeng Kong is with Carnegie Mellon University, Pittsburgh, PA 15213 USA (email: lingpengk@cs.cmu.edu).

Chris Dyer is with Google DeepMind, London, UK and Carnegie Mellon University, Pittsburgh, PA 15213 USA (email: cdyer@google.com).

Noah A. Smith is with University of Washington, Seattle, WA 98195 USA (email: nasmith@cs.washington.edu).

Steve Renals is with University of Edinburgh, Edinburgh, UK (email: s.renals@ed.ac.uk)

Segmental models have been proposed and rediscovered under different names based on the definitions of the weight functions and the training losses. For example, hidden semi-Markov models are defined in the generative setting [19]; semi-Markov CRFs [20] were introduced as segmental models trained with a log loss; segmental structured SVMs refer to segmental models trained with a hinge loss [21]. Segmental CRFs [9] were proposed as segmental models trained with marginal log loss. In this work, we will separate the loss functions from the definition of segmental models and consider different combinations of segmental weight functions and losses.

When segmental models are trained with marginal log loss (or another loss that marginalizes over segmentations), they can be trained end to end without the need for ground truth segmentations [18]. This property is particularly useful when obtaining ground-truth segmentations, such as alignments at the phonetic level, is expensive or time-consuming. Though training systems end to end reduces the amount of human intervention, the learned representations may not be interpretable, making it harder to diagnose errors made by end-to-end systems. In this work, we will show results comparing segmental models trained in multiple stages with intermediate supervision to ones trained end to end. In this context, we explore two weight functions, one based on frame classifiers [15], [17], [16], and one based on segmental recurrent neural networks [22], [18]. We will also compare end-to-end frame-based and segmental models in terms of their search spaces and loss functions. Finally, we will use multitask learning as a tool to constrain and to analyze the learned representations.

II. Segmental Models

We consider the problem of sequence prediction, such as speech recognition, as a graph search problem. The graph, usually represented as a finite-state transducer (FST), is a search space consisting of all of the ways of segmenting and labeling the input. A vertex in the graph corresponds to a point in time, and an edge in the graph corresponds to a segment, that is a time span in the acoustic input and a possible label. The graph is weighted, and the weight of an edge corresponds to how well the edge (segment) matches the input. To compute the weight of an edge, we first transform the input to an intermediate representation with a feature encoder. There are many choices for the type of encoder; here we mainly consider ones based on long short-term memory (LSTM) networks [23]. The intermediate representation is then used to compute weights, and we refer to the weighted graph and the graph search algorithm as the sequence decoder. Below we will formally define these components.

Let $\mathcal{X}$ be the input space, the set of all sequences of acoustic feature vectors, e.g., log mel filter bank features or mel frequency cepstral coefficients (MFCCs). Specifically, for a sequence of T vectors $x = (x_1, \ldots, x_T) \in \mathcal{X}$, each $x_t \in \mathbb{R}^d$, for $t \in \{1, \ldots, T\}$, is a d-dimensional acoustic feature vector, also referred to as a **frame**. Let $\mathcal{Y}$ be the output space, the set of all label sequences, where each label in a label sequence comes from a label set L, e.g., a phoneme set in the case of phoneme recognition. Given any T frames, a **segmentation** of length K is a sequence of time points $((0 = s_1, t_1), \ldots, (s_K, t_K = T))$, where $s_k < t_k$ and $t_k = s_{k+1}$ for $k \in \{1, \ldots, K\}$. A **segment** (typically denoted e in later sections) is a tuple (ℓ, s, t) where $\ell \in L$ is its label, s is the start time, and t is the end time.

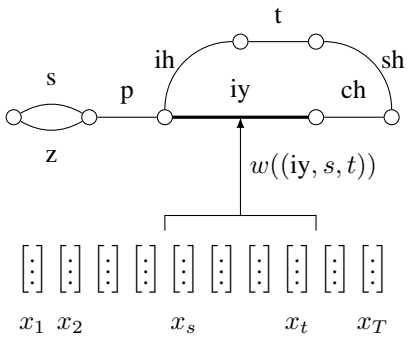


Fig. 1. An example of a segmental model. A search space is built based on the input frames. Each edge (segment) has a start time, an end time, and a label, which the weight function can make use of. Once the weights of edges are computed, decoding becomes the problem of finding the maximum-weight path.

A **segmental model** is a tuple (Θ, w) where Θ is a set of parameters, and $w : \mathcal{X} \times E \to \mathbb{R}$ is a weight function parameterized by Θ and E is the set of all segment tuples (ℓ, s, t). The set of parameters Θ includes all parameters for both the feature encoder and the sequence decoder. A sequence of segments forms a **path**. Specifically, a path of length K is a sequence of segments $(e_1, \ldots, e_K)$, where $e_k \in E$ for $k \in \{1, \ldots, K\}$. Let $\mathcal{P}$ be the set of all paths. For any path p, we overload w such that $w(x, p) = \sum_{e \in p} w(x, e)$. We will also abbreviate $w(x, e)$ and $w(x, p)$ as $w(e)$ and $w(p)$ respectively when the context is clear. An example is shown in Figure 1.

Given an input $x \in \mathcal{X}$, segmental models aim to solve sequence prediction by reducing it to finding the maximum-weight path

$$\operatorname*{argmax}_{p \in \mathcal{P}} w(x, p). \tag{1}$$

The set of paths $\mathcal{P}$, commonly referred to as the **search space**, can be compactly represented as an FST.

Here we briefly review the definition of FSTs. A multigraph (a graph that can have multiple edges between any pair of vertices) G is a tuple $(V, E, \text{tail}, \text{head})$, where V is a set of vertices, E is a set of edges, $\text{tail} : E \to V$ is a function that returns the vertex where an edge starts, and $\text{head} : E \to V$ is a function that returns the vertex where an edge ends. We deliberately overload E, because every segment has a corresponding edge in the graph. An FST is a tuple $(G, \Sigma, \Lambda, I, F, i, o, w)$, where G is a multigraph, Σ is a set of input symbols, Λ is a set of output symbols, $I \subseteq V$ is a set of initial vertices, $F \subseteq V$ is a set of final vertices, $i : E \to \Sigma$ is a function that defines the symbol an edge takes as input, $o : E \to \Lambda$ is a function that defines the symbol an edge outputs, and $w : E \to \mathbb{R}$ is a function that puts weights on

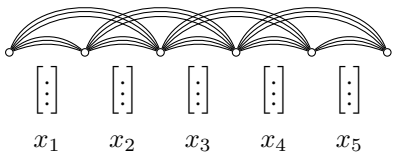


Fig. 2. An example of a search space for a five-frame utterance with a label set L of size three and maximum segment duration D of two frames, i.e., $T = 5$, $|L| = 3$, and $D = 2$. The three edges between any two nodes are associated with the three labels.

edges. We deliberately overload w as well, because the weight of a segment will be the weight of the corresponding edge.

We also associate a time function $\tau : V \to \mathbb{N}$ that maps a vertex to a time point (frame index). For convenience, we define $\text{in}(v) = \{e \in E : \text{head}(e) = v\}$ and $\text{out}(v) = \{e \in E : \text{tail}(e) = v\}$. We will also assume there is one unique start vertex and unique end vertex, i.e., $|I| = |F| = 1$, but this can be easily relaxed. More detailed discussion of FSTs and their applications for speech recognition can be found in [24].

To represent the set of paths $\mathcal{P}$ as an FST, we place a vertex at every time point and connect vertices based on the set of segments. Specifically, suppose we have T frames. The set of segments E is an exhaustive enumeration of tuples (ℓ, s, t) for all $\ell \in L$ and $0 \le s < t \le T$. In practice, a maximum duration D is typically imposed, i.e., for any segment (ℓ, s, t), $t - s \le D$, reducing the possible number of segments from $O(T^2|L|)$ to $O(TD|L|)$. We create a set of vertices $V = \{v_0, v_1, \ldots, v_T\}$ such that $\tau(v_t) = t$ for $t \in \{0, 1, \ldots, T\}$. For every segment $(\ell, s, t) \in E$, we create an edge e such that $i(e) = o(e) = \ell$, $\text{tail}(e) = v_s$, and $\text{head}(e) = v_t$. We set $\Sigma = \Lambda = L$, $I = \{v_0\}$, and $F = \{v_T\}$ to complete the construction of the FST given any T frames. An example of a search space is shown in Figure 2. One of the many benefits of representing the search space as an FST is that higher-order segmental models can be constructed by structurally composing the search space with higher-order language models [17].

Given the search space constructed above, inference, i.e., finding the maximum-weight path (1), can be done efficiently with dynamic programming. Let $\mathcal{P}(u, v)$ be the set of paths that starts at vertex u and ends at vertex v. By our previous definition, $\mathcal{P} = \mathcal{P}(v_0, v_T)$. Define

$$d(v) = \max_{p \in \mathcal{P}(v_0, v)} \sum_{e \in p} w(e). \tag{2}$$

In words, $d(v)$ is the maximum path weight for all paths between v_0 and v, and the goal is to find $d(v_T)$. By definition, we have

$$d(v) = \max_{e \in \text{in}(v)} \max_{p' \in \mathcal{P}(v_0, \text{tail}(e))} \left[w(e) + \sum_{e' \in p'} w(e') \right] \tag{3}$$

$$= \max_{e \in \text{in}(v)} \Big[w(e) + d(\text{tail}(e)) \Big]. \tag{4}$$

Algorithm 1 shows how to compute all of the entries in d based on the above recursion and how to backtrack to find the path; it is the same as the shortest path algorithm for directed acyclic graphs [25]. Since $v_0, \ldots, v_T$ follows a topological order, Algorithm 1 is guaranteed to return the maximum-weight path.

Algorithm 1 Finding the maximum-weight path

$d(v_0) = 0$
for $v = v_0, v_1, \ldots, v_T$ **do**
  $d(v) = \max_{e \in \text{in}(v)} \Big[d(\text{tail}(e)) + w(e) \Big]$
  $\delta(v) = \text{argmax}_{e \in \text{in}(v)} \Big[d(\text{tail}(e)) + w(e) \Big]$
end for
$u = v_T$, $p = \emptyset$
while $u \neq v_0$ **do**
  $p = \delta(u) \cup p$
  $u = \text{tail}(\delta(u))$
end while
return p

Since for any vertex v, $|\text{in}(v)| \le D|L|$, the runtime of Algorithm 1 is $O(TD|L|) = O(|E|)$. In fact, Algorithm 1 evaluates $w(e)$ for every edge $e \in E$ exactly once.

A segmental model can be trained by finding a set of parameters Θ that minimizes a loss function $\mathcal{L}$. The model definition is not tied to any loss function, allowing us to study the behavior of segmental models under different loss functions.

III. Weight Functions

Here we detail two types of neural weight functions based on prior work by ourselves and others. The term feature function is often used in the literature to denote the function $\phi : \mathcal{X} \times E \to \mathbb{R}^m$ for some m, where the weight function $w(x, e)$ is of the form $\theta^\top \phi(x, e)$ for some parameter vector $\theta \in \mathbb{R}^m$. When considering neural networks, the weight function need not be a dot product, but can be any differentiable real-valued function.

The first type of weight function is similar to those of [15], [17], [16], consisting of outputs of frame-level neural network classifiers "summarized" in various ways over the span of a segment; the specific formulation we use is that of [26]. The second type of weight function is a segmental recurrent neural network, as in [18]. We study segmental models in the context of these particular weight functions due to their prior success. In both cases the acoustic encoder is based on long short-term memory networks (LSTMs) [23].

Recall that the weight function w takes a sequence of acoustic features $x = (x_1, \ldots, x_T)$ and a segment (ℓ, s, t) as input. The weight function first passes the acoustic features through multiple layers of LSTMs. Let $h_1, \ldots, h_{\tilde{T}}$ be the sequence of output vectors[1] of the final LSTM. The output of each layer can be subsampled before feeding to the next layer to reduce the time resolution. For example, if we subsample

[1]Specifically, $h_t = W_f h_t^f + W_b h_t^b$, where h_t^f and h_t^b are the output vectors of the forward and backward LSTMs for some weight matrices W_f and W_b. The output vector for the forward LSTM is defined as $h_t^f = \tanh(c_t^f) \odot o_t^f$, where c_t^f is the cell, and o_t^f is the output gate at time t. The output vector h_t^b is defined similarly for the backward LSTM.

at layers two and three for a 3-layer LSTM, then $\tilde{T} = T/4$. Otherwise, $\tilde{T} = T$. We will use Θ_{enc} to denote the parameters of the LSTMs, and let Θ_{dec} be the remaining parameters in the weight function. Note that $\Theta = \Theta_{\text{enc}} \cup \Theta_{\text{dec}}$.

A. FC weight function

The first type of weight function, termed the frame classifier (FC) weight, is similar to weight functions used in a variety of prior work [15], [17], [16]. A frame classifier takes in the LSTM output $h_1, \ldots, h_{\tilde{T}}$ and produces a sequence of log probability vectors over the labels

$$z_i = \text{logsoftmax}(W h_i + b) \tag{5}$$

where $z_i \in \mathbb{R}^{|L|}$ and W and b are the parameters, for $i \in \{1, \ldots \tilde{T}\}$. Based on these posterior vectors, we define several functions that summarize the posteriors over a segment:

a) frame average: The average of transformed log probabilities

$$w_{\text{avg}}((\ell, s, t)) = \frac{1}{t-s} \sum_{i=s}^{t-1} u_{i,\ell}, \tag{6}$$

where $u_i = W_{\text{avg}} z_i$ for $i \in \{1, \ldots, \tilde{T}\}$.

b) frame samples: A sample of transformed log probabilities

$$w_{\text{spl-}j}((\ell, s, t)) = u_{j,\ell} \tag{7}$$

at time $j \in \{(t-s)/6, (t-s)/2, 5(t-s)/6\}$, where $u_i = W_{\text{spl}} z_i$ for $i \in \{1, \ldots, \tilde{T}\}$.

c) boundary: The samples of transformed log probabilities around the left boundary (start) and right boundary (end) of the segment:

$$w_{\text{left-}k}((\ell, s, t)) = u_{k, i-k, \ell} \tag{8}$$

$$w_{\text{right-}k}((\ell, s, t)) = u'_{k, i+k, \ell} \tag{9}$$

where $u_{k,i} = W_{\text{left-}k} z_i$ and $u'_{k,i} = W_{\text{right-}k} z_i$ for $k = 1, 2, 3$ and $i \in \{1, \ldots, \tilde{T}\}$.

d) duration: The label-dependent duration weight

$$w_{\text{dur}}((\ell, s, t)) = d_{\ell, t-s}. \tag{10}$$

e) bias: A label-dependent bias

$$w_{\text{bias}}((\ell, s, t)) = b'_\ell. \tag{11}$$

The final FC weight function is the sum of all of the above weight functions. When the FC weight function is used, Θ_{dec} is $\{W, b, W_{\text{avg}}, W_{\text{spl}}, W_{\text{left}}, W_{\text{right}}, d, b'\}$.

B. SRNN weight function

The second type of weight function is based on segmental recurrent neural networks (SRNNs) [22], [18]. Suppose the LSTM output vectors are $h_1, \ldots, h_{\tilde{T}}$. To compute $w((\ell, s, t))$, two hidden layers

$$z^{(1)}_{\ell,s,t} = \text{ReLU}(W_1 [h_s; h_t; c_\ell; d_k] + b_1)$$
$$z^{(2)}_{\ell,s,t} = \tanh(W_2 z^{(1)}_{\ell,s,t} + b_2)$$

are computed directly from the LSTM outputs before computing the final weight, where c_ℓ is a label embedding vector for the label ℓ, d_k is a duration embedding vector for the duration k in log scale, and $\text{ReLU}(x) = \max(x, 0)$. The final weight for the segment is defined as

$$w((\ell, s, t)) = \theta^\top z^{(2)}_{\ell,s,t}.$$

Note that instead of encoding the LSTM output vectors $h_1, \ldots, h_{\tilde{T}}$ with an additional LSTM per segment as in [22], for efficiency we use the left and right output vectors h_s and h_t and use a simple feed-forward network to compute the weight $w((\ell, s, t))$. When the SRNN weight function is used, Θ_{dec} is $\{W_1, b_1, W_2, b_2, \theta\}$. Although the SRNN weight function is conceptually simple, it is more expensive to compute than the FC weight function.

IV. Losses

Recall that a path $p = ((\ell_1, s_1, t_1), \ldots, (\ell_K, s_K, t_K))$ consists of a label sequence $y = (\ell_1, \ldots, \ell_K)$ and a segmentation $z = ((s_1, t_1), \ldots, (s_K, t_K))$. We will use (y, z) and p interchangeably. We will denote the space of all segmentations $\mathcal{Z}$.

Training aims to find a set of parameters Θ that minimizes the expected task loss, in our case, the expected edit distance

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[\text{edit}(y, h_\Theta(x))] \tag{12}$$

where h is the inference algorithm Algorithm 1 parameterized with Θ, edit computes the edit distance of two sequences, and the expectation is taken over samples $(x, y) \in \mathcal{X} \times \mathcal{Y}$ drawn from a distribution $\mathcal{D}$. The edit distance is discrete and therefore difficult to optimize; instead we minimize the expected loss

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(\Theta; x, y)], \tag{13}$$

where $\mathcal{L}$ is a surrogate loss function. Some surrogate losses refer to a particular choice of segmentation z; in that case we wish to minimize

$$\mathbb{E}_{(x,y,z)\sim\mathcal{D}'}[\mathcal{L}(\Theta; x, y, z)], \tag{14}$$

where $\mathcal{D}'$ is a distribution over $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. We will use $\mathcal{D}(y|x)$ and $\mathcal{D}'(y, z|x)$ to denote the conditional distribution of y and y, z, respectively, given the input. Since the distribution $\mathcal{D}$ is unknown, we use a training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$[2] of size n to approximate the expectation and instead minimize

$$\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\Theta; x_i, y_i). \tag{15}$$

The connection between the surrogate loss $\mathcal{L}$ and the edit distance depends on the choice of loss. Below we list the loss functions we consider, along with reasons for using them and their (sub)gradients with respect to the weight $w(e)$ for some edge e. The (sub)gradients are used in the first-order methods, such as stochastic gradient descent, that we use for optimization. We assume that the weight function w is differentiable and the (sub)gradients with respect to the parameters can be obtained with backpropagation. Other losses for training segmental models, such as ramp loss and empirical Bayes risk, are not included here but are treated in [27].

[2]A training set $S = \{(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\}$ of size n is needed if we optimize (14) with the approximation being $\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\Theta; x_i, y_i, z_i)$.

There are interesting connections between loss functions and discriminative training criteria in speech recognition (see, for example, [28], [29]).

A. Hinge loss

Given an utterance x and a ground-truth path $p = (y, z)$, the hinge loss is defined as

$$\mathcal{L}(\Theta; x, p) = \max_{p' \in \mathcal{P}} [\text{cost}(p', p) - w(p) + w(p')] \tag{16}$$

where cost is a user-defined, non-negative cost function. The connection between the hinge loss and the task loss is through the cost function. Suppose $\hat{p} = \text{argmax}_{p \in \mathcal{P}} w(p)$ is the best-scoring path found by Algorithm 1. The cost of the inferred path $\hat{p}$ can be upper-bounded by the hinge loss:

$$\text{cost}(\hat{p}, p) \leq \text{cost}(\hat{p}, p) - w(p) + w(\hat{p}) \leq \mathcal{L}(\Theta; x, p). \tag{17}$$

When the cost function is the edit distance, minimizing the hinge loss minimizes an upper bound on the edit distance.

The hinge loss is difficult to optimize when the cost function is the edit distance. In practice, the cost function is assumed to be decomposable to allow efficient dynamic programming:

$$\text{cost}(p', p) = \sum_{e' \in p'} \text{cost}(e', p). \tag{18}$$

When the cost is decomposable, the hinge loss can be written as

$$\begin{aligned}\mathcal{L}(\Theta; x, p) &= \max_{p' \in \mathcal{P}} \left[\sum_{e' \in p'} \text{cost}(e', p) - \sum_{e \in p} w(e) + \sum_{e' \in p'} w(e') \right] \\ &= \max_{p' \in \mathcal{P}} \sum_{e' \in p'} [\text{cost}(e', p) + w(e')] - \sum_{e \in p} w(e),\end{aligned}$$

and the max operator in the first term can be solved with Algorithm 1 by adding the costs to the weights for all segments.

A subgradient of the hinge loss with respect to $w(e)$ is

$$\frac{\partial \mathcal{L}(\Theta; x, p)}{\partial w(e)} = -\mathbb{1}_{e \in p} + \mathbb{1}_{e \in \tilde{p}} \tag{19}$$

where

$$\tilde{p} = \underset{p' \in \mathcal{P}}{\text{argmax}} [\text{cost}(p', p) + w(p')], \tag{20}$$

which is the path that maximizes the first term in the hinge loss, and can be obtained with Algorithm 1 with cost added.

Linear models trained with hinge loss are referred to as support vector machines (SVM), or as structured SVMs when applied to structured prediction problems, e.g., sequence prediction in our case. Segmental models trained with the hinge loss have been studied by [21], [27], [17].

B. Log loss

Segmental models can be treated as probabilistic models by defining probability distributions on the set of all paths. Specifically, the probability of a path $p = (y, z)$ is defined as

$$P(y, z|x) = P(p|x) = \frac{1}{Z(x)} \exp(w(x, p)) \tag{21}$$

where

$$Z(x) = \sum_{p' \in \mathcal{P}} \exp(w(x, p')) \tag{22}$$

is the partition function. Given an input x and a ground-truth path p, the log loss is defined as

$$\mathcal{L}(\Theta; x, p) = -\log P(p|x). \tag{23}$$

Minimizing the log loss is equivalent to maximizing the conditional likelihood. In addition, the conditional likelihood can be written as

$$\begin{aligned}P(y, z|x) &= \mathbb{E}_{(y', z') \sim P(y', z'|x)} [\mathbb{1}_{(y', z') = (y, z)}] \\ &= 1 - \mathbb{E}_{(y', z') \sim P(y', z'|x)} [\mathbb{1}_{(y', z') \neq (y, z)}].\end{aligned}$$

Therefore, maximizing the conditional likelihood is equivalent to minimizing the expected zero-one loss

$$\mathbb{E}_{(y', z') \sim P(y', z'|x)} [\mathbb{1}_{(y', z') \neq (y, z)}], \tag{24}$$

where $P(y, z|x)$ is used to approximate $\mathcal{D}'(y, z|x)$. The use of the log loss can be justified by viewing the expectation above as an approximation of (14). Segmental models trained with log loss have been referred to as semi-Markov CRFs [20].

Since the weight for the ground-truth path p can be efficiently computed, we are left with the problem of computing the partition function $Z(x)$. The partition function can also be computed efficiently with the following dynamic programming algorithm. Recall that $\mathcal{P}(u, v)$ is the set of paths that start at vertex u and end at vertex v. For any vertex v, define the forward marginal as

$$\alpha(v) = \log \sum_{p' \in \mathcal{P}(v_0, v)} \exp(w(p')). \tag{25}$$

By expanding the edges ending at v, we have

$$\begin{aligned}\alpha(v) &= \log \sum_{p' \in \mathcal{P}(v_0, v)} \exp \left(\sum_{e \in p'} w(e) \right) \\ &= \log \sum_{e \in \text{in}(v)} \sum_{p' \in \mathcal{P}(v_0, \text{tail}(e))} \exp \left(w(e) + \sum_{e' \in p'} w(e') \right) \\ &= \log \sum_{e \in \text{in}(v)} \exp \left(w(e) + \alpha(\text{tail}(e)) \right)\end{aligned}$$

Similarly, the backward marginal at v is defined as

$$\beta(v) = \log \sum_{p' \in \mathcal{P}(v, v_T)} \exp(w(p')), \tag{26}$$

and has a similar recursive structure. The complete algorithm is shown in Algorithm 2. Once all entries in α and β are computed, the log partition function is

$$\log Z(x) = \alpha(v_T) = \beta(v_0). \tag{27}$$

We store all of the entries in log space for numerical stability.

The gradient of the log loss with respect to $w(e)$ is

$$\begin{aligned}\frac{\partial \mathcal{L}(\Theta; x, p)}{\partial w(e)} &= -\mathbb{1}_{e \in p} + \frac{1}{Z(x)} \sum_{p' \ni e} \exp(w(p')) \\ &= -\mathbb{1}_{e \in p} + \exp \Big[\alpha(\text{tail}(e)) + w(e) \\ &\qquad + \beta(\text{head}(e)) - \log Z(x) \Big],\end{aligned}$$

which can also be efficiently computed once the marginals are computed.

Algorithm 2 Computing forward and backward marginals

$\alpha(v_0) = 0$
$\beta(v_T) = 0$
$\text{logadd}(a, b) = \log(\exp(a) + \exp(b))$
for $v = v_0, v_1, \ldots, v_T$ **do**
  $\alpha(v) = \text{logadd}_{e \in \text{in}(v)} \Big[\alpha(\text{tail}(e)) + w(e)\Big]$
end for
for $v = v_T, v_{T-1}, \ldots, v_0$ **do**
  $\beta(v) = \text{logadd}_{e \in \text{out}(v)} \Big[\beta(\text{head}(e)) + w(e)\Big]$
end for

C. Marginal log loss

Given an input x and a label sequence y, the marginal log loss is defined as

$$\mathcal{L}(\Theta; x, y) = -\log P(y|x) = -\log \sum_{z \in \mathcal{Z}} P(y, z|x) \tag{28}$$

where the segmentation is marginalized compared to log loss. Following the same argument as for log loss, the marginal distribution can be written as

$$P(y|x) = 1 - \mathbb{E}_{y' \sim P(y'|x)}[\mathbb{1}_{y \neq y'}], \tag{29}$$

and maximizing the marginal distribution is equivalent to minimizing the expected zero-one loss

$$\mathbb{E}_{y' \sim P(y'|x)}[\mathbb{1}_{y \neq y'}], \tag{30}$$

where $P(y|x)$ is used to approximate $\mathcal{D}(y|x)$. Note that the zero-one loss $\mathbb{1}_{y \neq y'}$ only depends on the label sequence. While the log loss has a connection to (14), the marginal log loss directly approximates (13) with the above expected zero-one loss.

Note that both the hinge and log loss depend on the ground-truth segmentation. The marginal log loss does not require the ground-truth segmentation, making it attractive for tasks such as speech recognition, because collecting ground-truth segmentations for phonemes or words is time-consuming and/or expensive. In addition, the boundaries of phonemes and words tend to be ambiguous, so it can be preferable to leave the decision to the model. Segmental models trained with the marginal log loss have been referred to as segmental CRFs [9].

To compute the marginal log loss, we can rewrite it as

$$\begin{aligned}
\mathcal{L}(\Theta; x, y) &= -\log \sum_{z \in \mathcal{Z}} P(y, z|x) && (31)\\
&= -\log \sum_{z \in \mathcal{Z}} \exp(w(x, (y, z))) + \log Z(x) && (32)\\
&= -\log \underbrace{\sum_{p': \Gamma(p') = y} \exp(w(x, p'))}_{\log Z(x, y)} + \log Z(x) && (33)
\end{aligned}$$

where Γ extracts the label sequence from a path, i.e., for $p' = (y', z')$, $\Gamma(p') = y'$. Since the partition function can be efficiently computed from Algorithm 2, we only need to compute $\log Z(x, y)$. Since the term $\log Z(x, y)$ is identical to $\log Z(x)$ except that it involves a constrained search space considering all paths with the same label sequence y, the strategy is to construct the constrained search space with an FST and run Algorithm 2 on the FST. Let F be a chain FST that represents y, with edges $\{e_1, \ldots, e_{|y|}\}$, where $i(e_k) = o(e_k) = y_k$ for all $k \in \{1, \ldots, |y|\}$. Let G be the search space consisting of all paths in $\mathcal{P}$. The term $\log Z(x, y)$ can be efficiently computed by running Algorithm 2 on the intersection of G and F, i.e., $G \cap F$. Let the forward and backward marginals computed on $G \cap F$ be α' and β'. We have $\log Z(x, y) = \alpha'(v_T) = \beta'(v_0)$.

The gradient of the marginal log loss is

$$\begin{aligned}
&\frac{\partial \mathcal{L}(\Theta; x, y)}{\partial w(e)}\\
&= -\frac{1}{Z(x, y)} \sum_{\substack{p' \ni e\\ \Gamma(p') = y}} \exp(w(p')) + \frac{1}{Z(x)} \sum_{p' \ni e} \exp(w(p'))\\
&= -\exp\Big[\alpha'(\text{tail}(e)) + w(e) + \beta'(\text{head}(e)) - \log Z(x, y)\Big]\\
&\quad + \exp\Big[\alpha(\text{tail}(e)) + w(e) + \beta(\text{head}(e)) - \log Z(x)\Big].
\end{aligned}$$

and can be efficiently computed once the marginals are computed.

V. Multi-stage Training and Multitask Training

Following the conventional ASR pipeline, we can first build a frame classifier and use it to build the rest of the segmental models. Such an approach, using the weight functions defined above, has been successful for training segmental models, either for multi-stage training or as an initialization for end-to-end training [26], [30]. We will review these training approaches in detail and present a unified view for both.

Recall that our parameters can be divided into two parts: Θ_{enc} for the acoustic feature encoder and Θ_{dec} for the sequence decoder. The acoustic feature encoder can be trained jointly with the sequence decoder, or separately with other loss functions, such as the frame-wise cross entropy or the connectionist temporal classification (CTC) loss [31]. We refer to the case where the encoder and decoder are trained jointly as **end-to-end training**, and the case where the training is separated into multiple stages (end-to-end or not) as **multi-stage training**.

Consider the end-to-end training approach. We can write the objective

$$\min_{\Theta_{\text{enc}}, \Theta_{\text{dec}}} \mathcal{L}(\Theta_{\text{enc}}, \Theta_{\text{dec}}), \tag{34}$$

in terms of both Θ_{enc} and Θ_{dec} where $\mathcal{L}$ is a loss function that takes both the encoder and the decoder into account, such as the hinge loss, log loss, or marginal log loss. For multi-stage training, we use a loss function to train the encoder in the first stage by solving

$$\hat{\Theta}_{\text{enc}} = \operatorname*{argmin}_{\Theta_{\text{enc}}} \mathcal{L}_{\text{enc}}(\Theta_{\text{enc}}), \tag{35}$$

where $\mathcal{L}_{\text{enc}}$ can be the frame-wise cross entropy or the CTC loss. In the second stage, we use the obtained $\hat{\Theta}_{\text{enc}}$ to solve

$$\hat{\Theta}_{\text{dec}} = \operatorname*{argmin}_{\Theta_{\text{dec}}} \mathcal{L}(\hat{\Theta}_{\text{enc}}, \Theta_{\text{dec}}) \tag{36}$$

while holding the first argument in the loss fixed. In the third stage, we can then use $\hat{\Theta}_{\text{enc}}$ and $\hat{\Theta}_{\text{dec}}$ as initialization and solve (34).

In addition, we can also consider a convex combination of multiple loss functions

$$\min_{\Theta_{\text{enc}},\Theta_{\text{dec}}} \lambda\mathcal{L}(\Theta_{\text{enc}},\Theta_{\text{dec}}) + (1-\lambda)\mathcal{L}_{\text{enc}}(\Theta_{\text{enc}}) \qquad (37)$$

where λ is the interpolation factor. End-to-end training can be seen as optimizing (37) with $\lambda = 1$, while multi-stage training can be seen as optimizing the second term in (37) followed by optimizing the first term.

While there are many benefits for training systems end to end, such as the potential to find a better optimum and without requiring supervision at the intermediate level, end-to-end training might be challenging due to optimization difficulties and might require more samples. On the other hand, while multi-stage training requires supervision at the intermediate level, it might make the optimization easier (sometimes making it convex), might require fewer samples, and might produce models that are more interpretable.

VI. Experiments

We apply segmental models to phonetic recognition on TIMIT, a dataset consisting of a training set of 3696 utterances and a test set, of which a subset of 192 utterances is called the core test set. Following standard protocol [32], we use 400 utterances from the complete test set (disjoint from the core test set) as the validation set, and report the final results on the core test set. In addition, we reserve 376 utterances from the training set for tuning hyperparameters, such as optimizers, step sizes, and dropout rates, and use the remaining 3320 utterances for training. The development set is used solely for early stopping. As is often done for TIMIT experiments, we collapse the 61 phones in the phone set to 48 for training, and further collapse them to 39 for evaluation [33]. TIMIT is phonetically transcribed, so we have the option of training the feature encoder with frame-wise cross entropy based on the ground-truth frame labels. The acoustic input to the feature encoder consists of 40-dimensional log filter bank features (without energy) and their first and second derivatives. The resulting 120-dimensional acoustic features are speaker-normalized by subtracting the per-speaker mean and dividing by the per-speaker standard deviation of every dimension.

The feature encoder is a 3-layer bidirectional LSTM with 250 hidden units in each direction. Previous work has shown that subsampling either the frames or the LSTM outputs can reduce the decoding time while maintaining accuracy [34], [35]. We consider subsampling the output of the LSTMs by a factor of two after the second and third layers. The subsampled encoder is referred to as a pyramid encoder [18]. Dropout [36] is added to the input and output of the LSTMs at a rate of 0.2.

For the segmental models, we enforce a maximum segment duration of 30 frames when a regular feature encoder is used, and a maximum duration of 8 when a pyramid feature encoder is used. The maximum duration is applied to all labels, including silences. For the SRNN weight function, following [18], the duration embedding is of size 5, the label embedding is of size 32, and the two subsequence hidden layers are both of size 64. All parameters of the weight functions are initialized based on [37].

All loss functions are optimized with stochastic gradient descent (SGD) with a minibatch size of 1 utterance. The gradient norm is clipped to 5. The default optimizer is vanilla SGD unless otherwise stated. We run the optimizer for 20 epochs with step size 0.1; starting from the best model among the first 20 epochs, we run for another 20 epochs with step size decayed by 0.75 after each epoch (i.e., exponential decay). We choose the epoch that has the best performance on the development set (early stopping).

A. Multi-stage training

We first compare different segmental models trained in multiple stages. The first stage trains the feature encoder Θ_{enc} either with the frame-wise cross entropy or with the CTC loss, and the second stage trains Θ_{dec} with hinge loss, log loss, or marginal log loss. Finally, after the second stage, we fine-tune both Θ_{enc} and Θ_{dec} with each of the three losses.

To construct a frame classifier, the 250-dimensional output vectors of the encoder are projected down to 48 dimensions followed by a softmax layer. Depending on whether we use a pyramid encoder, we subsample the frame labels accordingly during training. The resulting frame classifier achieves frame error rates of 18.3% for the regular encoder and 29.1% for the pyramid encoder (where outputs are upsampled to evaluate performance) on the development set.

For the CTC loss, we project the 250-dimensional output vectors of the feature encoder down to 49 dimensions (48 phones + 1 blank) and pass them through a softmax layer. The encoder is fixed to a pyramid, and frame labels are not required during training. The encoder trained with the CTC loss achieves a phoneme error rate (PER) of 17.2% on the development set with best-path decoding (followed by removing duplicates and blanks).

In the first set of experiments, we only use the frame classifiers as encoders (pyramid or not), and compare the two weight functions. Since we have the pretrained feature encoders, we freeze the encoder parameters Θ_{enc} and train the decoder parameters Θ_{dec}. The default SGD optimizer (20 epochs without decay plus 20 epochs with exponential decay) is used for the SRNN weight function because it works well with the two-layer networks in the weight function. For the FC weight function, note that hinge loss and log loss are convex in Θ_{dec}. In particular, when the encoder Θ_{enc} is frozen, optimizing hinge loss and log loss for the FC weight function are both convex problems. RMSprop [38] is favored over vanilla SGD for the FC weight function with step size 10^{-4} and decay 0.9 for 20 epochs. After the two-stage training, we can further optimize both the encoder and the decoder. Here vanilla SGD is used with the step size starting from 0.1 and decayed by 0.75 after each epoch, because the training loss is already low after two-stage training.

The multi-stage training results are shown in Table I. The results are consistent with those reported in [17]. For the

TABLE I
PHONE ERROR RATES (PER, %) ON THE DEVELOPMENT SET FOR SEGMENTAL MODELS WITH DIFFERENT WEIGHT FUNCTIONS AND DIFFERENT FEATURE ENCODERS. THE ENCODERS (PYRAMID OR NOT) ARE TRAINED WITH THE FRAME-WISE CROSS ENTROPY. TWO-STAGE TRAINING IS DENOTED 2s, AND TWO-STAGE TRAINING FOLLOWED BY FINE-TUNING IS DENOTED 2s+ft. (* TOO SLOW TO COMPLETE)

	pyramid		hinge	log loss	marginal log loss
FC		2s	19.9	20.7	19.9
		2s+ft	19.3	18.2	17.9
	✓	2s	31.3	32.0	24.4
	✓	2s+ft	23.3	22.3	17.9
SRNN		2s	22.2*	22.2*	20.5*
		2s+ft	-	-	-
	✓	2s	27.4	24.4	21.3
	✓	2s+ft	24.4	22.7	18.1

TABLE II
PERS (%) FOR SEGMENTAL MODELS TRAINED IN MULTIPLE STAGES WITH FEATURE ENCODERS PRETRAINED WITH THE FRAME-WISE CROSS ENTROPY OR THE CTC LOSS. TWO-STAGE TRAINING IS DENOTED 2s, AND TWO-STAGE TRAINING FOLLOWED BY FINE-TUNING IS DENOTED 2s+ft.

	frame loss			CTC		
	2s dev	2s+ft dev	2s+ft test	2s dev	2s+ft dev	2s+ft test
hinge	27.4	24.4		26.7	24.1	
log loss	25.9	22.7		25.7	22.7	
marginal log loss	21.3	18.1	20.9	18.7	17.8	20.2

FC weight function with a regular encoder, the three losses perform equally well, with marginal log loss having a slight edge over the other two. Using the pyramid encoder hurts the performance of hinge loss and log loss, but has less impact on marginal log loss. Hinge loss and log loss might be more sensitive to the reduced time resolution because they are tied to a specific segmentation, while marginal log loss is more forgiving due to the marginalization. Fine-tuning improves over two-stage training across all cases. The conclusion stays the same for the SRNN weight function, except that training the SRNN weight function without the pyramid is very time-consuming, and we only manage to complete a few epochs in the two-stage setting. Although the best results after fine tuning are roughly the same for both weight functions, to shorten the experimental cycle, we favor the better performer, the SRNN weight function, with a pyramid encoder in the two-stage setting.

After fixing the weight function to the SRNN, we compare encoders pretrained with the frame-wise cross entropy and with the CTC loss. Results are shown in Table II. It is clear that for all losses, using the encoder pretrained with the CTC loss leads to better performance.

B. End-to-end training from random initialization

In this section, we compare losses for end-to-end training of segmental models with the pyramid encoder and the SRNN weight function. Unlike in multi-stage training, all of the experiments here are trained from random initialization. Results are shown in Table III. While the log loss and marginal log loss achieve reasonable performance, the hinge loss completely fails. We find that hinge loss values on the training set are higher compared to the multi-stage models, suggesting that there is an optimization issue. Since log loss can be minimized reasonably well, we suspect that hinge loss is difficult to minimize because of its non-smoothness. The result with the marginal log loss is consistent with reported numbers in previous work [18]. The performance of CTC is on par with the segmental model trained with marginal log loss.

TABLE III
SEGMENTAL MODELS TRAINED END TO END WITH DIFFERENT LOSSES COMPARED WITH THE CTC MODEL.

	dev	test
hinge	74.7	
log loss	22.2	
marginal log loss	17.5	19.5
CTC	17.2	19.5

TABLE IV
PERS (%) FOR SEGMENTAL MODELS TRAINED END TO END WITH MULTIPLE TASKS, I.E., THE MARGINAL LOG LOSS PLUS EITHER THE FRAME-WISE CROSS ENTROPY OR THE CTC LOSS.

	frame loss		CTC	
λ	dev	test	dev	test
0.16	18.6		17.5	
0.33	18.0		17.0	
0.5	17.2		17.0	
0.67	16.9		16.7	18.5
0.84	16.8	19.3	17.0	
1.00	17.5	19.5	17.2	19.5

C. End-to-end multitask training

Instead of optimizing different losses in different stages as in the previous section, we next optimize multiple losses jointly from random initialization. Here we only focus on marginal log loss paired with either the frame-wise cross entropy or the CTC loss, because the marginal log loss is the best performer in the previous experiments. We use early stopping based on the PERs of the segmental model on the development set. Results are shown in Table IV. We see that end-to-end training with multiple tasks further improves over end-to-end training with a single task. The best test-set result (and the best dev-set result) is obtained by multitask training with marginal log loss + CTC loss, and improves over the CTC error rate by 1% absolute (19.5% $\longrightarrow$ 18.5% on the test set).

The success of multitask learning in Table IV indicates that there exists an encoder that can generate representations suitable for both tasks. We further investigate the loss values for the case of jointly optimizing the marginal log loss and the CTC loss. The learning curve is shown in Figure 3. In the multitask case, both the marginal log loss and the CTC loss achieve lower values on the training set compared to the single-task case, suggesting that multitask learning might help optimization. However, both loss values on the development set end up higher when multiple losses are used. The fact that models with higher losses on the development set end up having lower PERs is unsatisfying and needs further investigation.

The time to compute gradients for different losses is shown in Table V. All numbers are measured on a single quad-core 3GHz CPU, averaged over the entire training set. As a reference, the average real-time factor for computing the

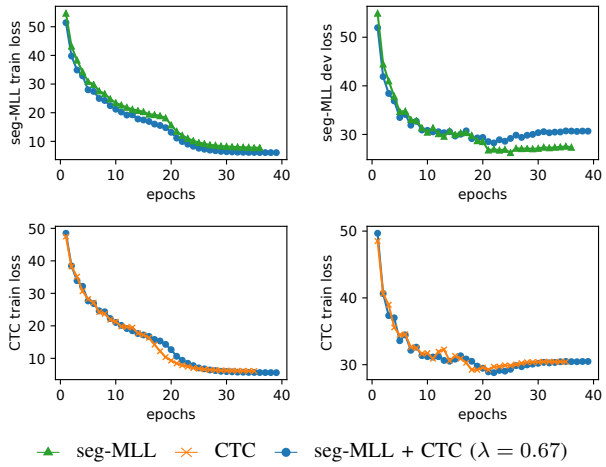


Fig. 3. Loss values for jointly optimizing the marginal log loss (seg-MLL) and the CTC loss compared with optimizing them individually.

TABLE V
AVERAGE REAL-TIME FACTOR PER SAMPLE TO COMPUTE GRADIENTS FOR DIFFERENT LOSSES.

	pyramid	hinge	log loss	marginal log loss
FC		0.257	0.811	0.954
	✓	0.015	0.042	0.073
SRNN		3.253	5.669	6.110
	✓	0.165	0.286	0.384

gradients from the outputs of a regular 3-layer LSTM is 0.301, and reduces to 0.181 for pyramid 3-layer LSTM. Computing the gradient of the CTC loss is also 0.181. Though the exact number depends on the implementation, the general trend is clear. It is faster to compute the FC weight function than the SRNN weight function. Using the pyramid encoder significantly reduces the runtime. Computing the gradient of hinge loss is fastest, and computing the gradient of marginal log loss is the slowest. The real-time factor for decoding is 0.259 including the pyramid LSTM. Decoding in multiple passes can be an option if speedup is needed [39].

VII. RELATED WORK

A. First-pass segmental models

Many models, such as semi-Markov CRFs [20], segmental CRFs [9], and inverted HMMs [40], are special cases of segmental models with different weight functions and trained with different losses. In Table VI, we provide a set of highlights of results in the development of segmental models on the TIMIT data set. Zweig [14] was the first to explore discriminative segmental models that search over sequences and segmentations exhaustively, and did not use neural networks. He & Fosler-Lussier [15] first used (shallow) neural network-based frame classifiers to define weight functions, and later extended the idea to deep neural networks in [41]. Abdel-Hamid *et al.* [16] were the first to use deep convolutional neural networks for the weight functions, and were the first to train segmental models end to end. Tang *et al.* first compared different losses and training strategies for segmental models in a rescoring framework [27] and then in first-pass segmental models [26].

TABLE VI
TIMIT PERS (%) FOR VARIOUS SEGMENTAL MODELS COMPARED WITH HMMS AND THE STATE OF THE ART. THE ACOUSTIC FEATURES CAN BE SPEAKER INDEPENDENT (spk indep) OR SPEAKER ADAPTED WITH MEAN AND VARIANCE NORMALIZATION (mvn) OR MAXIMUM LIKELIHOOD LINEAR REGRESSION (fMLLR) [32]. SOME RESULTS WERE OBTAINED WITH MFCCS AND SOME WITH LOG FILTER BANK FEATURES.

	spk indep	+mvn	+fMLLR
HMM-DNN [32]	21.4		18.3
HMM-CNN [42]	16.5		
SUMMIT (1998) [43], [13]	24.4		
segmental CRF (SCRF) (2012) [14]	33.1		
SCRF + shallow NN (2012) [15]	26.5		
SCRF + DNN (2015) [41]			19.1
deep segmental NN (2013) [16]	21.9		
segmental cascades (2015) [17]	19.9		
segmental RNN (SRNN) (2016) [18]		18.9	17.3
end-to-end + two-stage training (2016) [26]	19.7		
SRNN + multitask (2017) [30]		18.7	17.5
SRNN + multitask (2017) (this work)		18.5	

They also introduced segment-level classifiers and segmental cascades for incorporating them (and other expensive features) into segmental weight functions [17]. Lu *et al.* [18] introduced an LSTM-based weight function for every segment, and were also the first to use pyramid LSTMs to speed up inference for segmental models.

B. End-to-end models

Most mainstream end-to-end speech recognition models can be broadly categorized as either frame-based models or encoder-decoder models. CTC, HMMs, and some newer approaches like the auto-segmentation criterion (ASG) [44] fall under the first category, because these models emit one symbol for every frame. Falling under the second category, encoder-decoder models proposed by [45], [46], [47] generate labels one at a time while conditioning on the input and the labels generated in the past, without an explicit alignment between labels and frames. Since frame-based models follow the same graph search framework as segmental models, we will focus on discussing the connection between these and segmental models.

Recall that training segmental models with marginal log loss requires a search space G, a constraint FST F to limit the search space to ground-truth labels, and the loss itself. To compute marginal log loss, we first compute the marginals on G for computing the partition function $Z(x)$, and then compute the marginals on the intersection $G \cap F$ for computing $Z(x, y)$. CTC, HMMs, and ASG can all be seen as special cases of this framework.

The search space of CTC has an edge for every label in the label set (including the blank label) at every time step. Specifically, the search space G includes the edges $\{e_{\ell,t} : \ell \in L, t \in \{1, \ldots, T\}\}$ with $v_{t-1} = \text{tail}(e_{\ell,t})$ and $v_t = \text{head}(e_{\ell,t})$. An example is shown in Figure 4. The weight of an edge $e_{\ell,t}$ is the log probability of label ℓ at time t. By construction, the decision made at every time point is independent of the decision at other time points. In addition, since the probabilities at every time point sum to one, the partition function $Z(x)$ of the search space is always

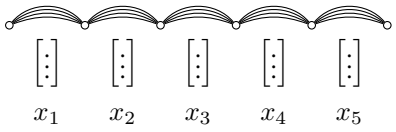


Fig. 4. An example of the CTC search space for a five-frame utterance with a label set of size three (plus one blank).

1. The constraint FST F representing the ground-truth labels consists of the sequences of one or more labels with zero or more blanks in between labels. For example, for the label sequence "k ae t," the constraint FST is the regular expression $\varnothing^*\mathrm{k}^+\varnothing^*\mathrm{ae}^+\varnothing^*\mathrm{t}^+\varnothing^*$. With the above construction, marginal log loss becomes exactly the objective of CTC.

Comparing CTC to HMMs, the search space is different depending on the HMM topology. For example, two-state HMMs are used in [48]. Since the transition probabilities and posterior probabilities are all locally normalized, the partition function $Z(x)$ is always 1. The constraint FST representing the ground-truth labels consists simply of sequences of repeating labels. For example, for the label sequence "k ae t," the constraint FST is the regular expression $\mathrm{k}^+\mathrm{ae}^+\mathrm{t}^+$. With the above construction, marginal log loss applied to HMMs is equivalent to lattice-free MMI [48].

For ASG, the search space is equivalent to that of one-state HMMs. Instead of assuming conditional independence as in CTC, ASG includes transition probabilities between states. The constraint FST is identical to that of HMMs, with repeated ground-truth labels. However, in ASG the weights on the edges are not locally normalized, so the partition function $Z(x)$ is not always 1 and has to be computed. With the above search space construction, marginal log loss becomes ASG.

Another approach similar to CTC proposed in [49] is called RNN transducers. The search space of an RNN transducer is the set of alignments from the speech signal to all possible label sequences, so the search space grows exponentially in the number of labels. The weight function of a path in this approach relies on an RNN, and is not decomposable as a sum of weights of the edges. RNN transducers are trained with marginal log loss. By the independence assumption imposed in [49], the partition function $Z(x)$ is still 1, so we do not need to marginalize over the exponentially large space. During decoding, however, we still have to search over the exponentially large space with, for example, beam search.

In view of this framework, even when using the same loss function, i.e., marginal log loss, segmental models and frame-based models differ in their search space, weight functions, and how the search space is constrained by the ground truth labels during training.

C. Word recognition

First-pass segmental models have previously been successfully applied to word recognition [13], [50]. This previous work treats first-pass segmental models as a drop-in replacement for HMM phoneme recognizers, because both models serve as functions that map acoustic features to phoneme strings. The phoneme recognizers are then composed with a lexicon and a language model to form a word recognizer.

Recent work has explored models that directly predict characters, avoiding the need for a lexicon [51], [52], [53] but still allowing for improved performance when constraining the search space with a lexicon (through FST composition) [53]. Segmental models can also be used to predict characters simply by changing the label set.

Instead of using intermediate discrete representations, such as phonemes or characters, recent advances in computing power have made it feasible to directly predict words [54], [55], [56], [57]. In this case, rather than using a pronunciation dictionary, only a list of words is needed for decoding. Segmental models can also be used to directly predict words by using the list of words as the label set. This approach is worth exploring further, although efficiency issues make it nontrivial to train such models [57].

VIII. Conclusion

We have presented the formal framework of segmental models and several potential losses for training such models. Segmental models are now able to run efficiently enough for end-to-end training and obtain competitive error rates. We have explored segmental models with two types of weight functions and various training losses on the task of phonetic recognition. We have found that the best results obtained with the two types of weight functions (frame classifier-based and segmental recurrent neural networks) are quite similar, and are typically best with marginal log loss.

We also consider the relationship between segmental models and frame-based models trained with CTC. Both models, while having different search spaces and different weight functions, are optimizing the same loss, the marginal log loss. Empirically, with the same feature encoder and the same optimizer, there is no significant difference between the two in terms of final performance. However, each type of model benefits from training jointly with the other in a multitask training approach. We hope that drawing the connection between these models will spawn more research in exploring different search spaces and loss functions. In future work, we plan to extend this study of segmental models to word recognition by exploiting other efficiency and performance trade-offs.

References

[1] F. Jelinek, "Continuous speech recognition by statistical methods," *Proceedings of the IEEE*, vol. 64, no. 4, pp. 532–556, 1976.

[2] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[3] L. Bahl, F. Jelinek, and R. Mercer, "A maximum likelihood approach to speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, pp. 179–190, 1983.

[4] E. Fosler-Lussier, Y. He, P. Jyothi, and R. Prabhavalkar, "Conditional random fields in speech, audio, and language processing," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1054–1075, 2013.

[5] N. Smith and M. Gales, "Speech recognition using SVMs," in *Advances in neural information processing systems (NIPS)*, 2001.

[6] V. Zue, J. Glass, M. Phillips, and S. Seneff, "Acoustic segmentation and phonetic classification in the SUMMIT system," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1989.

[7] M. De Wachter, M. Matton, K. Demuynck, P. Wambacq, R. Cools, and D. Van Compernolle, "Template-based continuous speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 4, pp. 1377–1390, 2007.

[8] M. A. Bush and G. E. Kopec, "Network-based connected digit recognition using vector quantization," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1985.

[9] G. Zweig and P. Nguyen, "A segmental CRF approach to large vocabulary continuous speech recognition," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2009.

[10] G. Chung and S. Seneff, "Hierarchical duration modelling for speech recognition using the ANGIE framework." in *Eurospeech*, 1997.

[11] M. Hasegawa-Johnson, J. Baker, S. Borys, K. Chen, E. Coogan, S. Greenberg, A. Juneja, K. Kirchhoff, K. Livescu, S. Mohan *et al.*, "Landmark-based speech recognition: Report of the 2004 Johns Hopkins summer workshop," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2005.

[12] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1," *NASA STI/Recon technical report*, vol. 93, 1993.

[13] J. R. Glass, "A probabilistic framework for segment-based speech recognition," *Computer Speech & Language*, vol. 17, no. 2, pp. 137–152, 2003.

[14] G. Zweig, "Classification and recognition with direct segment models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[15] Y. He and E. Fosler-Lussier, "Efficient segmental conditional random fields for phone recognition," in *INTERSPEECH*, 2012.

[16] O. Abdel-Hamid, L. Deng, D. Yu, and H. Jiang, "Deep segmental neural networks for speech recognition," in *INTERSPEECH*, 2013.

[17] H. Tang, W. Wang, K. Gimpel, and K. Livescu, "Discriminative segmental cascades for feature-rich phone recognition," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2015.

[18] L. Lu, L. Kong, C. Dyer, N. A. Smith, and S. Renals, "Segmental recurrent neural networks for end-to-end speech recognition," in *INTERSPEECH*, 2016.

[19] M. Ostendorf, V. Digalakis, and O. Kimball, "From HMM's to segment models: A unified view of stochastic modeling for speech recognition," *IEEE Transactions on Speech and Audio Processing*, pp. 360–378, 1996.

[20] S. Sarawagi and W. W. Cohen, "Semi-Markov conditional random fields for information extraction." in *Advances in Neural Information Processing Systems (NIPS)*, vol. 17, 2004.

[21] S.-X. Zhang and M. J. Gales, "Structured SVMs for automatic speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 3, pp. 544–555, 2013.

[22] L. Kong, C. Dyer, and N. A. Smith, "Segmental recurrent neural networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] M. Mohri, "Finite-state transducers in language and speech processing," *Computational linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

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

[26] H. Tang, W. Wang, K. Gimpel, and K. Livescu, "End-to-end training approaches for discriminative segmental models," in *IEEE Workshop on Spoken Language Technology (SLT)*, 2016.

[27] H. Tang, K. Gimpel, and K. Livescu, "A comparison of training approaches for discriminative segmental models," in *INTERSPEECH*, 2014.

[28] G. Heigold, T. Deselaers, R. Schlüter, and H. Ney, "Modified MMI/MPE: A direct evaluation of the margin in speech recognition," in *International Conference on Machine learning (ICML)*, 2008.

[29] E. McDermott, S. Watanabe, and A. Nakamura, "Discriminative training based on an integrated view of MPE and MMI in margin and error space," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, 2010.

[30] L. Lu, L. Kong, C. Dyer, and N. A. Smith, "Multi-task learning with CTC and segmental CRF for speech recognition," *CoRR*, vol. abs/1702.06378, 2017.

[31] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2006.

[32] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The Kaldi speech recognition toolkit," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

[33] K.-F. Lee, "On large-vocabulary speaker-independent continuous speech recognition," *Speech communication*, vol. 7, no. 4, pp. 375–379, 1988.

[34] V. Vanhoucke, M. Devin, and G. Heigold, "Multiframe deep neural networks for acoustic modeling," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[35] Y. Miao, J. Li, Y. Wang, S. Zhang, and Y. Gong, "Simplifying long short-term memory acoustic models for fast training and decoding," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[36] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014.

[37] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[38] T. Tieleman and G. Hinton, "Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, 2012.

[39] H. Tang, W. Wang, K. Gimpel, and K. Livescu, "Efficient segmental cascades for speech recognition," in *INTERSPEECH*, 2016.

[40] P. Doetsch, S. Hegselmann, R. Schlüter, and H. Ney, "Inverted HMM – a proof of concept," in *NIPS 2016 End-to-end Learning for Speech and Audio Processing Workshop*, 2016.

[41] Y. He, "Segmental models with an exploration of acoustic and lexical grouping in automatic speech recognition," Ph.D. dissertation, The Ohio State University, 2015.

[42] L. Tóth, "Phone recognition with hierarchical convolutional deep maxout networks," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2015, no. 1, p. 25, 2015.

[43] A. Halberstadt and J. Glass, "Heterogeneous measurements and multiple classifiers for speech recognition," in *International Conference on Spoken Language Processing*, 1998.

[44] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *CoRR*, vol. abs/1609.03193, 2016.

[45] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[46] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[47] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[48] D. Povey, V. Peddinti, D. Galvez, P. Ghahrmani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely sequence-trained neural networks for ASR based on lattice-free MMI," in *INTERSPEECH*, 2016.

[49] A. Graves, "Sequence transduction with recurrent neural networks," *CoRR*, vol. abs/1211.3711, 2012.

[50] Y. He and E. Fosler-Lussier, "Segmental conditional random fields with deep neural networks as acoustic models for first-pass word recognition." in *INTERSPEECH*, 2015.

[51] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2014.

[52] A. L. Maas, Z. Xie, D. Jurafsky, and A. Y. Ng, "Lexicon-free conversational speech recognition with neural networks," in *Human Language Technologies: The Annual Conference of the North American Chapter of the ACL*, 2015.

[53] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2015.

[54] A. L. Maas, S. D. Miller, T. M. O'neil, A. Y. Ng, and P. Nguyen, "Word-level acoustic modeling with convolutional vector regression," in *ICML Workshop on Representation Learning*, 2012.

[55] S. Bengio and G. Heigold, "Word embeddings for speech recognition," in *INTERSPEECH*, 2014.

[56] H. Soltau, H. Liao, and H. Sak, "Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition," *CoRR*, vol. abs/1610.09975, 2016.

[57] K. Audhkhasi, B. Ramabhadran, G. Saon, M. Picheny, and D. Nahamoo, "Direct acoustics-to-word models for english conversational speech recognition," *CoRR*, vol. abs/1703.07754, 2017.