Encoding linear models as weighted finite-state transducers

Ke Wu°, Cyril Allauzen†, Keith Hall†, Michael Riley†, Brian Roark†

°Institute for Advanced Computer Studies, University of Maryland

†Google, Inc.

wuke@cs.umd.edu, {allauzen,kbhall,riley,roark}@google.com

Abstract

We present algorithms, implemented as an extension to the OpenFst library, that yield a class of transducers that encode linear models for structured inference tasks like segmentation and tagging. This allows the use of general finite-state operations with such models. For instance, finite-state composition can be used to apply the model to lattice input (or other more general automata) and then the result automaton can be passed to subsequent processing such as general shortest path algorithms. We demonstrate the use of the library extension on grapheme-to-phoneme conversion, encoding multiple varieties of linear models for that task, and achieve solid PER/WER gains over previous best reported results on g2p conversion of a publicly available dataset (CMU).

Index Terms: finite-state transducers, linear models, grapheme-to-phoneme conversion

10.21437/Interspeech.2014-315

1. Introduction

Finite-state sequence models are very widely used in speech and natural language processing, for annotating strings with various kinds of hidden information (e.g., segmentation, tagging or labeled bracketing), and also for language modeling. Certain kinds of models, such as hidden Markov models or conventionally smoothed n-gram models, have structures that are relatively easily representable compactly in explicit weighted finite-state automata or transducers (WFST) [1, 2, 3]. Other kinds of models may have a structure that, while finite-state, yield a state space that is difficult to represent compactly in a WFST. For example, linear models, such as Maximum Entropy Markov models [4], conditional random fields (CRF) [5], or the structured perceptron algorithm [6], combine evidence from multiple, typically overlapping, features defined over input/output string pairs (e.g., words and their POS-tags). States in a WFST must remember enough of the previous input and output strings to be able to derive all necessary features of the model, e.g., the previous k input symbols $x_{t-k} \dots x_{t-1}$ and j output symbols $y_{t-j} \dots y_{t-1}$. If some active features include $x_{t-k} \dots x_{t-1}$ and some (possibly different) features include $y_{t-j} \dots y_{t-1}$, then a state corresponding to the configuration $(x_{t-k} \dots x_{t-1}, y_{t-j} \dots y_{t-1})$ must be included in the WFST, even if no active feature in the model includes all of it. This generally results in a state space which grows much faster than the feature set, unless the features are very carefully selected, as in the n-gram modeling of [7]. Similar issues arise for other complex model topologies, e.g., factored language models [8].

Encoding such models as WFSTs, however, can yield some major benefits, due to general operations that can be applied to them, such as composition. Composition of sequence models and joint inference can often be preferable to a simple cascading of the models where the single-best output of one serves as the input to the other [9]; and also for model combination. In this paper, we present an extension to the OpenFst software library [1] that provides a C++ class that implicitly represents linear models as WFSTs and expands them on-the-fly, hence avoiding the kind of state-space explosion described above.

After introducing linear FSTs, we then present a case study in their use for grapheme-to-phoneme conversion (G2P). There are several ways to model the G2P problem, including as a joint multi-gram transducer [10] – essentially an n-gram model over input/output pairs – or by mapping groups of letters to groups of phonemes. This latter approach is achieved either by first segmenting the input letter sequence into groups of letters, then transducing those groups of letters into (potentially groups of) phonemes, as is done in the Phonetisaurus system [11], or by performing phrase-based "translation" [12]. Recently, CRF models have been explored in this space [12, 13, 14], though they have not been shown to improve upon generative joint multi-gram models. We demonstrate how to encode various models as linear FSTs and then use composition to combine segmentation with tagging and to combine system outputs, achieving the best reported results for the CMU dictionary task.

2. Linear FSTs

Consider a linear sequence model of the form:

$$\mu(\mathbf{x}_t, \mathbf{y}_t) = \sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}_t, \mathbf{y}_t) \tag{1}$$

where input $\mathbf{x}_t = x_{t-l} \dots x_{t-1} x_t x_{t+1} \dots x_{t+m}$ for $x_i \in \Sigma$; output $\mathbf{y}_t = y_{t-n} \dots y_{t-1} y_t$ for $y_j \in \Delta$; and f_k is dimension k in a K-dimensional feature vector Φ, while λ_k is dimension k in a K-dimensional parameter vector Λ. At position t, the model assigns weight $\mu(\mathbf{x}_t, \mathbf{y}_t)$ and to do so it can look back at most l symbols and forward m symbols in the input and back n symbols in the output. We refer to l as the (input) *lookback* or *history* size, m as the (input) *lookahead* or *future* size, n as the (output) order, the triple (l, m, n) as the *context window* size of the model, and f_k as a *feature*. The weight assigned to a full input/output string pair $(\mathbf{x}, \mathbf{y})$ is

$$\mathcal{M}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{|\mathbf{x}|} \mu(\mathbf{x}_t, \mathbf{y}_t) \tag{2}$$

To simplify the presentation, we assume that the alphabets Σ and Δ contain a padding symbol $\diamond$. That symbol is used to pad the end of the shorter of the two strings $\mathbf{x}$ and $\mathbf{y}$, allowing us to assume that $|\mathbf{x}| = |\mathbf{y}|$ in the rest of this paper.

2.1. Naive Construction

We can represent a model of this class as a weighted finite-state transducer $T_\Phi = (\Sigma, \Delta, Q, I, F, E, \mathbb{K})$ with states $Q = \Sigma^{l+m} \times \Delta^n$ and transitions $(q, x_l, y_n, w, q') \in E \subseteq Q \times \Sigma \times \Delta \times \mathbb{K} \times Q$, where $q = (x_0 \dots x_{l+m-1}, y_0 \dots y_{n-1})$, $q' = (x_1 \dots x_{l+m}, y_1 \dots y_n)$ for $x_i \in \Sigma$ and $y_j \in \Delta$ and weight $w = \mu(x_0 \dots x_{l+m}, y_0 \dots y_n) \in \mathbb{K}$. The initial state is $(\diamond^{l+m}, \diamond^n)$, all states are final ($F = Q$), and the weights are interpreted in the *log semiring*, $\mathbb{K} = (\mathbb{R}_+ \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$ [15].[1]

[1] For a and b in $\mathbb{R}_+$, $a \oplus_{\log} b \equiv -\log(\exp(-a) + \exp(-b))$.

In other words, from state q, x_{l+m} can be read from the input, y_n is output with weight w and we transition to state q'. We used x_{l+m} rather than x_l as the transition input label to reduce non-determinism at the cost of the output being delayed relative to the input when $m > 0$. In this case, m can also be referred to as the *delay* of the transducer.

There are $|\Sigma|^{l+m}|\Delta|^n$ states and $|\Sigma|^{l+m+1}|\Delta|^{n+1}$ transitions in above construction. Thus, the size of T_Φ is exponential in l, m and n but independent of K, the number of features. It is impractical to explicitly build this transducer unless the context window (l, m, n) is small.

2.2. Compact Representation

To handle larger context windows, we can build T_Φ *on-the-fly*: as the transducer is explored, states and transitions are constructed only as needed. OpenFst natively supports on-the-fly WFST representations as C++ classes derived from an abstract base. To do so, one provides class methods defining the initial state and, for a given state, whether it is final and what the transitions are leaving that state [1]. Given a source state q and a transition labeled (x_{l+m}, y_n), the destination state q' and weight w are easily determined from the above definitions since the source state stores the necessary history $(x_0 \dots x_{l+m-1}, y_0 \dots y_{n-1})$.[2] Once defined, this new WFST can be used with algorithms such as composition just like any other WFST.

In this approach, the fixed memory used is on the order of the number of features K, while the variable memory used is on the order of number of states and transitions explored by a given input. With lattice input, a large number of states of T_Φ may be constructed and there could still be a problem in time and space. In the rest of this section, we outline methods to reduce the state space of T_Φ.

2.2.1. Minimization

We can conveniently associate T_Φ with the corresponding weighted finite-state acceptor A_Φ over the alphabet $\Sigma \times \Delta$ where a transition labeled with (x, y) is treated as a single label. It is easy to see that A_Φ is deterministic. As such it can, in theory, be transformed into the minimal, deterministic weighted automaton equivalent to A_Φ, reducing the state space as much as possible (while retaining the determinism for efficiency) [16].

There are, however, two issues that must be resolved. First, applying the weighted minimization algorithm explicitly to such a large input would be difficult. Second, since the result is still likely to be very large if explicitly represented, we would want to keep an on-the-fly representation. This means, given a source state and a transition label (x, y), we still need an efficient way to compute the destination state and weight, but a state is now not necessarily the simple history tuple as before, which had made these computations straightforward.

2.2.2. N-Gram Pair Machines

If we signficantly restrict the form of the features, we can immediately deal with these two aforementioned issues; in later sections we will relax these restrictions, building up more complex features based on the results here. These restrictions are:

1. (No-lookahead) There is no lookahead (i.e. $m = 0$);
2. All feature functions $f_k(\mathbf{x}, \mathbf{y})$ are predicates whether $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}_k, \mathbf{y}_k)$ for some $(\mathbf{x}_k, \mathbf{y}_k)$ associated with f_k, where $|\mathbf{x}_k| = l + 1$ and $|\mathbf{y}_k| = n + 1$.

Under these restrictions, a feature function f_k can be conveniently represented by the pair of an input $(l + 1)$-gram and an output $(n + 1)$-gram $(\mathbf{x}_k, \mathbf{y}_k)$.

If the underlying linear model is sparse, the number of states needed in this case to represent A_Φ will not be $|\Sigma|^l|\Delta|^n$ but only on the order of the number of features, K, in the model. To obtain this, we need to efficiently identify and merge states that have *equivalent futures*, i.e., states from which the same set of string pairs can be read with the same weights, the basis for minimization [16]. To further explain this, we first need some definitions. For any two string pairs $(\mathbf{x}, \mathbf{y}), (\mathbf{z}, \mathbf{w}) \in \Sigma^* \times \Delta^*$,

Definition 1. $(\mathbf{x}, \mathbf{y})$ is a suffix of $(\mathbf{z}, \mathbf{w})$ if and only if $\mathbf{x}$ is a suffix of $\mathbf{z}$ and $\mathbf{y}$ is a suffix of $\mathbf{w}$. Refer to the set of all suffixes of $(\mathbf{z}, \mathbf{w})$ as $\text{suff}(\mathbf{z}, \mathbf{w})$.

Definition 2. Given a set of string pairs S, $(\mathbf{x}, \mathbf{y}) \in S$ is a maximal string pair from S if and only if any other $(\mathbf{x}', \mathbf{y}') \in S$, $(\mathbf{x}, \mathbf{y})$ is not a suffix of $(\mathbf{x}', \mathbf{y}')$.

Definition 3. $(\mathbf{x}, \mathbf{y})$ is a prefix of $(\mathbf{z}, \mathbf{w})$ if and only if there exists two strings $\mathbf{u} \in \Sigma^*, \mathbf{v} \in \Delta^*$ such that $\mathbf{xu} = \mathbf{z}$ and $\mathbf{yv} = \mathbf{w}$, and one of the following is true: (1) $|\mathbf{u}| = |\mathbf{v}|$; (2) $\mathbf{x} = \epsilon$ and $|\mathbf{u}| \leq |\mathbf{v}|$; or (3) $\mathbf{y} = \epsilon$ and $|\mathbf{u}| \geq |\mathbf{v}|$. Refer to the set of all proper prefixes of $(\mathbf{x}, \mathbf{y})$ as $\text{pref}(\mathbf{x}, \mathbf{y})$.

Definition 4. Given a set of features $\Phi = \{f_1, \dots, f_K\}$ and a string pair $(\mathbf{x}, \mathbf{y}) \in \Sigma^* \times \Delta^*$, let $S = \{s \mid s \in \text{suff}(\mathbf{x}, \mathbf{y}), \exists f_k \in \Phi, s \in \text{pref}(f_k)$, i.e., all suffixes of $(\mathbf{x}, \mathbf{y})$ that are a proper prefix of some f_k. $\beta_\Phi(\mathbf{x}, \mathbf{y})$ is defined as the maximal string pair from S.

It is not difficult to show the following theorem:

Theorem 1. *The maximal string pair, $\beta_\Phi(\mathbf{x}, \mathbf{y})$, is unique and thus well-defined. Further, for any two states $q, q' \in Q$, $\beta_\Phi(q) = \beta_\phi(q')$ if and only if q and q' have equivalent futures in A_Φ.*

This theorem is the basis for constructing A'_Φ, the minimal automaton equivalent to A_Φ. Let A'_Φ have the state set Q' of all proper prefixes of features in Φ, all of which are final states, and the transitions$(q, (x_i, y_j), w, q') \in Q' \times (\Sigma \times \Delta) \times \mathbb{R} \times Q'$ where $q = (x_1 \dots x_{i-1}, y_1 \dots y_{j-1})$, $q' = \beta_\Phi(x_1 \dots x_i, y_1 \dots y_j)$ for $x_i \in \Sigma$ and $y_j \in \Delta$ and weight $w = W_\Phi(x_1 \dots x_i, y_1 \dots y_j)$ which is defined as follows:

$$W_\Phi(\mathbf{x}, \mathbf{y}) = \begin{cases} \lambda_k & \exists f_k \in \Phi, (\mathbf{x}, \mathbf{y}) = (\mathbf{x}_k, \mathbf{y}_k) \\ 0 & \text{otherwise} \end{cases}$$

This automaton is equivalent to A_Φ and minimal[3] since, by construction, we are merging precisely those states with equivalent futures according to the theorem. In the worst case, there are $K(\max(l, n))$ proper prefixes of features in Φ, therefore $|Q'| = O(K \max(l, n))$, which is linear in K, l and n.

We can compute β_Φ efficiently using a trie with suffix links similar to what is used in [17]. We represent the set $P = \bigcup_{k=1}^{K} \text{pref}(f_k)$ in a trie labeled with symbol pairs. Each node in the trie corresponds to string pair $(\mathbf{u}, \mathbf{v})$ in P and the suffix link of that node points to the maximal suffix of $(\mathbf{u}, \mathbf{v})$ that belongs to P. When $l = n$, this corresponds exactly to the Aho-Corasick construction over sequences of pairs of symbols. β_Φ can then be computed by interpreting the suffix links as failure transitions: if there is a transition labeled (a, b) out of the node corresponding to $(\mathbf{u}, \mathbf{v})$ then $\beta_\Phi(\mathbf{u}a, \mathbf{v}b) = (\mathbf{u}a, \mathbf{v}b)$. Otherwise we follow the suffix links until we reach a node with

[2] Because OpenFst requires states to be referred to by integer IDs for uniformity, the class maintains a mapping between these state tuples and assigned IDs, internal to the WFST class.

[3] Specifically, it is minimal when considered as an unweighted automaton where a transition label and weight are treated as a single label. Weighted minimization would require the weights to also be pushed [16].

an (a, b) outgoing transition. The destination of that transition then corresponds to $\beta_\Phi(\mathbf{u}a, \mathbf{v}b)$.

When $l > n$ (resp. $l < n$), the trie will contain transitions of the form (a, ϵ) (resp. (ϵ, b)) that need to be interpreted as a set of transitions $\{(a, b) \mid b \in \Delta\}$ (resp. $\{(a, b) \mid a \in \Sigma\}$) when computing β_Φ.

2.2.3. More Complex Features

Requiring the input features to be n-grams of input words is very restrictive. One simple variant is to instead allow n-grams of *word classes*. For example, $\Gamma = \{\texttt{lowercase}, \texttt{capitalized}, \texttt{uppercase}\}$ could be one possible class set. This can be implemented by first creating a T_Φ as in Section 2.2.2 but using Γ as the input alphabet and then using $F_\Gamma \circ T_\Phi$ as the final transducer where F_Γ is a single state transducer that maps from Σ to Γ. Note that this composition can be performed on-the-fly, available in OpenFst, to preserve the dynamic expansion of the T_Φ.

Another way to extend the features allowed is to combine two linear automata A_Φ and A_Ψ on $\Sigma \times \Delta$. The intersection automaton $A_\Phi \cap A_\Psi$ linearly combines the weights assigned by each component to a string pair. In this way pair n-grams of different context windows or different feature classes can be easily combined. The intersection operation can be performed on-the-fly preserving the component dynamic expansions. If the components are minimal and complete, their intersection is likely to be near minimal as well.

2.2.4. Lookahead

So far we have assumed there is no lookahead, $m = 0$. Applying a model with a context window (l, m, n) on a string pair $(\mathbf{x}, \mathbf{y})$ is equivalent to applying to the string pair $(\mathbf{x}\diamond^m, \diamond^m\mathbf{y})$ the model with context window $(l + m, 0, n)$ defined by the same feature functions. This corresponds to delaying the output by m as described in section 2.1. Moreover, this allows us to build the delayed transducer T_Φ as presented in section 2.2.2 and to combine models with same lookahead as described in section 2.2.3.

This leaves the issue of combining models with different lookaheads. Given A_Φ with delay m_1 and A_Ψ with delay $m_2 > m_1$, we need to additionally delay A_Φ by $m_2 - m_1$ in order to be able to intersect with A_Ψ: $\text{Shift}_{m_2-m_1}(A_\Phi) \cup A_\Psi$ using the Shift_m operation defined below.

Given a WFST $T = (\Sigma, \Delta, Q, I, F, E, \mathbb{K})$ and a delay $m > 0$, the transducer $\text{Shift}_m(T) = (\Sigma, \Delta, Q', I', F', E', \mathbb{K})$ such that $T(\mathbf{x}, \mathbf{y}) = \text{Shift}_m(T)(\diamond^m\mathbf{x}, \mathbf{y}\diamond^m)$ is defined by $Q' = \Sigma^m \times Q$, $I' = \{\diamond^m\} \times I$, $F' = \{\diamond^m\} \times F$, and E' contains all transitions of the form:

- $((\diamond^i\mathbf{x}, q), x, \diamond, \overline{1}, (\diamond^{i-1}\mathbf{x}x, q))$ for $0 < i \leq m$, $q \in I$ and $\mathbf{x}x \in (\Sigma \setminus \{\diamond\})^{m-i+1}$,
- $((x\mathbf{x}, q), x', y, w, (\mathbf{x}x', q'))$ for $(q, x, y, w, q') \in E$, $\mathbf{x} \in \Sigma^{m-1}$ and $x' \in \Sigma$.

This construction is implemented on-the-fly.

2.3. Normalized Model Application

Linear models are often used to represent the unnormalized conditional or joint probabilities of the input/output pairs in the negative log domain. This means normalization needs to be applied to obtain the actual conditional probability of $\mathbf{y}$ given $\mathbf{x}$:

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{\exp(-\mathcal{M}(\mathbf{x}, \mathbf{y}))}{\sum_{y' \in \Delta^*} \exp(-\mathcal{M}(\mathbf{x}, \mathbf{y}'))}.$$

This normalization is important when combining globally normalized models because the normalization factor can vary widely depending on $\mathbf{x}$. It can be performed using standard finite-state operations.

Let X be the finite-state acceptor representing the set of input strings. The acyclic WFST $M = X \circ T_\Phi$ represents the set of possible outputs for the inputs in X such that: $M(\mathbf{x}, \mathbf{y}) = \mathcal{M}(\mathbf{x}, \mathbf{y})$ for $\mathbf{x} \in X$.

We compute the WFSA $N = \pi_1(M)$, the projection of M on its input,

$$N(\mathbf{x}) \equiv \bigoplus_{y' \in \Delta^*} M(\mathbf{x}, \mathbf{y}') = -\log \sum_{y' \in \Delta^*} \exp(-\mathcal{M}(\mathbf{x}, \mathbf{y}')).$$

Normalization can then be performed by composing $-N$ (obtained by negating every weight in N) and M:

$$-N \circ M(\mathbf{x}, \mathbf{y}) = -N(\mathbf{x}) + M(\mathbf{x}, \mathbf{y}) = -\log P(\mathbf{y} \mid \mathbf{x}).$$

The algorithm is optimized by determinizing N.

3. Grapheme-to-Phoneme Conversion

Grapheme-to-phoneme conversion (or letter-to-sound mapping) is the transduction from an input stream of letters to an output stream of phonemes. For example, the word "ABLE" has the pronunciation "EY B AH L" (using the ARPAbet representation for phonemes) in the CMU pronouncing dictionary [18]. It is clear that the letters A, B and L correspond to the phonemes EY, B and L respectively; but there is an inserted phoneme (AH) between the B and L in this sequence; and the final letter (E) is deleted, i.e., it is unpronounced (though influencing the pronunciation of the word). One can write this transduction as a sequence of input:output pairs, where the input symbols are letters (or the empty string ϵ) and the output symbols are phonemes (or ϵ), giving, for our current example, the sequence:

A:EY B:B ϵ:AH L:L E:ϵ

This task is of high importance for very large (e.g., 1M word) vocabulary automatic speech recognition, where a core subset may have manually validated pronunciations, but many are generated automatically; or in scenarios where personalized information such as proper names are incorporated into recognition. It is also of high importance for text-to-speech synthesis, which may be required to pronounce out-of-vocabulary words.

A popular approach to this task, one shown to provide state-of-the-art results relative to other approaches [10, 19], is joint multi-gram modeling, also known as joint or pair n-gram modeling. For this approach, the training input (letter) sequences and output (phoneme) sequences are aligned, each input:output pair in the alignments is taken as a single token, and a smoothed n-gram language model is estimated. Let Σ be the set of letters and Δ the set of phonemes, and $A = (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\})$ the set of possible letter to phoneme transduction pairs $i{:}o$, including insertions and deletions. For a given alignment $\mathbf{a} = a_1 \dots a_m \in A^m$, let $i(\mathbf{a}) = \mathbf{x} \in \Sigma^*$, the input string of the alignment and $o(\mathbf{a}) = \mathbf{y} \in \Delta^*$, the output string of the alignment. Let $\mathcal{A}(\mathbf{x}, \mathbf{y})$ be the set of alignments $\mathbf{a} \in A^*$ such that $i(\mathbf{a}) = \mathbf{x}$ and $o(\mathbf{a}) = \mathbf{y}$. Then the joint probability of the string pair $(\mathbf{x}, \mathbf{y})$ is calculated as[4]

$$\text{P}(\mathbf{x}, \mathbf{y}) = \max_{a_1 \dots a_m \in \mathcal{A}(\mathbf{x}, \mathbf{y})} \prod_{j=1}^{m} \text{P}(a_j \mid a_{j-n} \dots a_{j-1}) \qquad (3)$$

This is estimated with a smoothed n-gram language model (including <s> and </s> symbols), then converted to a weighted finite-state transducer with letters on the input side and phonemes on the output side. As noted in Novak et al. [11], this presents an issue with exact encoding of backoff transitions, though this is not a major problem in practice.

[4] We follow common practice and use the most likely alignment rather than summing over all alignments.

4. Experiments and Discussion

For this paper, we look at predicting a subset of the CMU pronouncing dictionary with models trained on another, larger subset, a task that has been reported on many times over the past decade, thus allowing us to compare with the state-of-the-art on a publicly available resource. Given an input word in graphemes, the task is to predict the phoneme sequence representing the word's pronunciation. We evaluate with the conventional measures for this task: phoneme error rate (PER), which is the number of substitutions, deletions or insertions divided by the number of true phonemes; and word error rate (WER), the number of words with an error divided by the number of words.

We made use of the Phonetisaurus distribution[5] that provides the data and evaluation from Novak et al. [11], and successfully replicated the result from that paper. The training procedure for that algorithm produces an alignment between letters and phones, and we used those alignments as the starting point for our systems. Note that these alignments do not make use of epsilons for deletions and insertions but rather includes automatically learned letter pairs and phoneme pairs, e.g., `E|R` or `AH|L`. We split these so as to only include a single symbol (or ϵ) on the input or output, e.g., `L:AH|L` becomes ϵ`:AH` and `L:L`.

From the original training set of 106,837 words (with just over 113k pronunciations), we selected 2,670 words as a development set, which we used for determining stopping criteria for learning and setting meta-parameters. The final evaluation set contains 12,000 words and their nearly 13k pronunciations. This is the same evaluation setup as found in [20, 21, 10, 11].

Joint multi-gram. We used the OpenGrm n-gram library [2] to build joint multi-gram language models encoded as OpenFst [1] format WFSA. We convert these to transducers by splitting the input and output part of the paired symbols. We tried a number of smoothing methods and n-gram orders on the development set, and settled on Kneser-Ney smoothing [22] for an 8-gram model, which matches the order used in [11]. This model has approximately 2 million parameters. Table 1 presents the PER and WER on the dev set for this model.

CRF trained model. We then trained a linear tagging model using the Wapiti toolkit [23]. Note that, in order to train this as a tagging task, the input symbols (including ϵ where insertions occur) must be given at time of inference. To do this, we train an n-gram model (also an 8-gram Kneser-Ney model) over the input-side of aligned sequences, which predicts, based solely on letter context, where the insertions can occur. Using this model, we produce a weighted lattice of input sequences corresponding to the original letter string. This lattice is then simply composed with the linear model encoded as a linear FST, which predicts the most likely outputs given the inputs.

The CRF features include n-gram input sequences with either the current label or the previous and current label. The n-gram sequences are all n-grams up to 8-grams ending in the current word; up to 5-grams ending in the next word (look-ahead 1); and up to 5-grams ending in the next word after that (look-ahead 2). Given Wapiti's relatively high memory usage, this feature set is too large to train with L-BFGS directly, so we trained the model in two stages: (1) trained a model using stochastic gradient descent (SGD) and L1 regularization; then (2) fixed the features to only those in the SGD trained model and re-trained with L-BFGS using a joint L1 and L2 regularization. Based on dev set results, we set the L1 regularization meta-parameter to 0.1 for both the first and second stage, and the L2 regularization meta-parameter to 1.0 for stage two. Together with the input-side n-gram model, the total number of parameters in this approach is approximately 3 million.

[5] https://code.google.com/p/phonetisaurus/

Model	PER %	WER %
Joint multi-gram (8-gram Kneser-Ney)	6.7	26.5
CRF trained model (8-gram features)	7.0	29.6
Segmentation/Tagging	6.7	28.0
Joint multi-gram + Seg/Tag	6.1	24.5
Joint multi-gram + CRF	6.1	24.9

Table 1: CMU development set results.

Segmentation/Tagging. An alternative approach is to use the output of the Phonetisaurus alignments of the training set as if they were supervised training examples for a segmenter and tagger. We use the alignment segmentation to label the graphemes as either beginning (B), inside (I), or a singleton (S) segmentation component. Given an input segmentation, we train a tagger to tag input segments with phonetic tags, where the tags are groups of phonemes or epsilon (no pronunciation). We train a CRF [5] for each of these models using local grapheme n-grams for the segmentation model and segment n-grams for the tagging model, similar to the features described above. These models together have approximately 2 million parameters.

The advantage of encoding these models as linear FST models is that we can process lattice input, allowing us to perform joint decoding by way of FST composition. We compose each input grapheme sequence (a word) with the segmentation linear FST, transform the output lattice to a segment-based lattice, and then compose with the tagging linear FST. Results for this approach are shown in the third row of Table 1.

System combination. Finally, we combined the joint multi-gram model and both the CRF and Segmentation/Tagging models by simply intersecting the two output lattices and finding the shortest path through the result. Although we have the ability to scale the weights of the different system outputs, we found little to no improvement over leaving the weights unscaled, so we report that here. As we can see from the development set results in Table 1, the CRF and Segmentation/Tagging models do substantially worse on WER than the baseline, despite their similarity in terms of PER, indicating that they are better at tagging longer words that are less likely to be completely correctly labeled. Indeed, the complementary nature of the models is demonstrated in the combined result, which yields strong PER and WER reductions over the individual methods.

Table 2 presents results of our systems on the evaluation set, alongside prior reported results for this train/test setup. We find a similar pattern for the eval set as we find for the dev set, in that the CRF and Segmentation/Tagging models underperform relative to the joint multi-gram (particularly on WER) but the combination yields strong gains in both PER and WER. As far as we know, these are the best results reported for this task.

In future work, we would like to further extend the form of the features allowed, e.g., n-grams of *heterogenous* features, which is straightforward for coarse-to-fine feature hierarchies.

Table 2: Evaluation set results, development set included in training.

Model	PER %	WER %
Galescu and Allen [20]	7.0	28.5
Chen [21]	5.9	24.7
Bisani and Ney [10]	5.9	24.5
Novak et al. [11]	5.8	24.4
Joint multi-gram (8-gram Kneser-Ney)	6.0	24.7
CRF trained model (8-gram features)	6.4	28.9
Segmentation/Tagging	6.4	28.3
Joint multi-gram + Seg/Tag	5.6	24.0
Joint multi-gram + CRF	5.5	23.4

5. References

[1] C. Allauzen, M. Mohri, and B. Roark, "The design principles and algorithms of a weighted grammar library," *International Journal of Foundations of Computer Science*, vol. 16, no. 3, pp. 403–421, 2005.

[2] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai, "The OpenGrm open-source finite-state grammar software libraries," in *Proceedings of the ACL 2012 System Demonstrations*, 2012, pp. 61–66.

[3] R. Sproat, M. Yarmohammadi, I. Shafran, and B. Roark, "Applications of lexicographic semirings to problems in speech and language processing," *Computational Linguistics*, 2014, *forthcoming*.

[4] A. McCallum, D. Freitag, and F. C. Pereira, "Maximum entropy markov models for information extraction and segmentation." in *Proceedings of ICML*, 2000, pp. 591–598.

[5] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.

[6] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *Proceedings of EMNLP*, 2002, pp. 1–8.

[7] B. Roark, M. Saraclar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech & Language*, vol. 21, no. 2, pp. 373–392, 2007.

[8] J. A. Bilmes and K. Kirchhoff, "Factored language models and generalized parallel backoff," in *Proceedings of HLT-NAACL–short papers-Volume 2*, 2003, pp. 4–6.

[9] C. Sutton and A. McCallum, "Composition of conditional random fields for transfer learning," in *Proceedings of HLT-EMNLP*, 2005, pp. 748–754.

[10] M. Bisani and H. Ney, "Joint-sequence models for grapheme-to-phoneme conversion," *Speech Communication*, vol. 50, no. 5, pp. 434–451, 2008.

[11] J. R. Novak, N. Minematu, and K. Hirose, "Failure transitions for joint n-gram models and g2p conversion," in *Proceedings of Interspeech*, 2013.

[12] S. Jiampojamarn, C. Cherry, and G. Kondrak, "Joint processing and discriminative training for letter-to-phoneme conversion." in *Proceedings of ACL*, 2008, pp. 905–913.

[13] D. Wang and S. King, "Letter-to-sound pronunciation prediction using conditional random fields," *IEEE Signal Processing Letters*, vol. 18, no. 2, pp. 122–125, 2011.

[14] P. Lehnen, A. Allauzen, T. Lavergne, F. Yvon, S. Hahn, and H. Ney, "Structure learning in hidden conditional random fields for grapheme-to-phoneme conversion," in *Proceedings of Interspeech*, 2013.

[15] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[16] M. Mohri, "Minimization algorithms for sequential transducers," *Theoretical Computer Science*, vol. 234, no. 1, pp. 177–201, 2000.

[17] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[18] R. Weide, "The CMU pronunciation dictionary, release 0.6," 1998.

[19] S. Hahn, P. Vozila, and M. Bisani, "Comparison of grapheme-to-phoneme methods on large pronunciation dictionaries and LVCSR tasks." in *Proceedings of Interspeech*, 2012.

[20] L. Galescu and J. F. Allen, "Pronunciation of proper names with a joint n-gram model for bi-directional grapheme-to-phoneme conversion," in *Proceedings of Interspeech*, 2002.

[21] S. F. Chen, "Conditional and joint models for grapheme-to-phoneme conversion," in *Proceedings of Interspeech*, 2003.

[22] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *Proceedings of ICASSP*, 1995, pp. 181–184.

[23] T. Lavergne, O. Cappé, and F. Yvon, "Practical very large scale CRFs," in *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 2010, pp. 504–513.