Discriminative Language Modeling with Conditional Random Fields and the Perceptron Algorithm

Brian Roark **Murat Saraclar** **Michael Collins** **Mark Johnson**
AT&T Labs - Research MIT CSAIL Brown University
{roark,murat}@research.att.com mcollins@csail.mit.edu Mark_Johnson@Brown.edu

Abstract

This paper describes discriminative language modeling for a large vocabulary speech recognition task. We contrast two parameter estimation methods: the perceptron algorithm, and a method based on conditional random fields (CRFs). The models are encoded as deterministic weighted finite state automata, and are applied by intersecting the automata with word-lattices that are the output from a baseline recognizer. The perceptron algorithm has the benefit of automatically selecting a relatively small feature set in just a couple of passes over the training data. However, using the feature set output from the perceptron algorithm (initialized with their weights), CRF training provides an additional 0.5% reduction in word error rate, for a total 1.8% absolute reduction from the baseline of 39.2%.

1 Introduction

A crucial component of any speech recognizer is the *language model* (LM), which assigns scores or probabilities to candidate output strings in a speech recognizer. The language model is used in combination with an acoustic model, to give an overall score to candidate word sequences that ranks them in order of probability or plausibility.

A dominant approach in speech recognition has been to use a "source-channel", or "noisy-channel" model. In this approach, language modeling is effectively framed as density estimation: the language model's task is to define a distribution over the source – i.e., the possible strings in the language. Markov (n-gram) models are often used for this task, whose parameters are optimized to maximize the likelihood of a large amount of training text. Recognition performance is a direct measure of the effectiveness of a language model; an indirect measure which is frequently proposed within these approaches is the perplexity of the LM (i.e., the log probability it assigns to some held-out data set).

This paper explores alternative methods for language modeling, which complement the source-channel approach through discriminatively trained models. The language models we describe do not attempt to estimate a generative model $P(w)$ over strings. Instead, they are trained on acoustic sequences with their transcriptions, in an attempt to directly optimize error-rate. Our work builds on previous work on language modeling using the perceptron algorithm, described in Roark et al. (2004). In particular, we explore conditional random field methods, as an alternative training method to the perceptron. We describe how these models can be trained over lattices that are the output from a baseline recognizer. We also give a number of experiments comparing the two approaches. The perceptron method gave a 1.3% absolute improvement in recognition error on the Switchboard domain; the CRF methods we describe give a further gain, the final absolute improvement being 1.8%.

A central issue we focus on concerns feature selection. The number of distinct n-grams in our training data is close to 45 million, and we show that CRF training converges very slowly even when trained with a subset (of size 12 million) of these features. Because of this, we explore methods for picking a small subset of the available features.[1] The perceptron algorithm can be used as one method for feature selection, selecting around 1.5 million features in total. The CRF trained with this feature set, and initialized with parameters from perceptron training, converges much more quickly than other approaches, and also gives the optimal performance on the held-out set. We explore other approaches to feature selection, but find that the perceptron-based approach gives the best results in our experiments.

While we focus on n-gram models, we stress that our methods are applicable to more general language modeling features – for example, syntactic features, as explored in, e.g., Khudanpur and Wu (2000). We intend to explore methods with new features in the future. Experimental results with n-gram models on 1000-best lists show a very small drop in accuracy compared to the use of lattices. This is encouraging, in that it suggests that models with more flexible features than n-gram models, which therefore cannot be efficiently used with lattices, may not be unduly harmed by their restriction to n-best lists.

1.1 Related Work

Large vocabulary ASR has benefitted from discriminative estimation of Hidden Markov Model (HMM) parameters in the form of Maximum Mutual Information Estimation (MMIE) or Conditional Maximum Likelihood Estimation (CMLE). Woodland and Povey (2000) have shown the effectiveness of lattice-based MMIE/CMLE in challenging large scale ASR tasks such as Switchboard. In fact, state-of-the-art acoustic modeling, as seen, for example, at annual Switchboard evaluations, invariably includes some kind of discriminative training.

Discriminative estimation of language models has also been proposed in recent years. Jelinek (1995) suggested an acoustic sensitive language model whose parameters

[1] Note also that in addition to concerns about training time, a language model with fewer features is likely to be considerably more efficient when decoding new utterances.

are estimated by minimizing $H(W|A)$, the expected uncertainty of the spoken text W, given the acoustic sequence A. Stolcke and Weintraub (1998) experimented with various discriminative approaches including MMIE with mixed results. This work was followed up with some success by Stolcke et al. (2000) where an "anti-LM", estimated from weighted N-best hypotheses of a baseline ASR system, was used with a negative weight in combination with the baseline LM. Chen et al. (2000) presented a method based on changing the trigram counts discriminatively, together with changing the lexicon to add new words. Kuo et al. (2002) used the generalized probabilistic descent algorithm to train relatively small language models which attempt to minimize string error rate on the DARPA Communicator task. Banerjee et al. (2003) used a language model modification algorithm in the context of a reading tutor that listens. Their algorithm first uses a classifier to predict what effect each parameter has on the error rate, and then modifies the parameters to reduce the error rate based on this prediction.

2 Linear Models, the Perceptron Algorithm, and Conditional Random Fields

This section describes a general framework, global linear models, and two parameter estimation methods within the framework, the perceptron algorithm and a method based on conditional random fields. The linear models we describe are general enough to be applicable to a diverse range of NLP and speech tasks – this section gives a general description of the approach. In the next section of the paper we describe how global linear models can be applied to speech recognition. In particular, we focus on how the decoding and parameter estimation problems can be implemented over lattices using finite-state techniques.

2.1 Global linear models

We follow the framework outlined in Collins (2002; 2004). The task is to learn a mapping from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. We assume the following components: (1) Training examples (x_i, y_i) for $i = 1 \ldots N$. (2) A function $\mathbf{GEN}$ which enumerates a set of candidates $\mathbf{GEN}(x)$ for an input x. (3) A **representation** Φ mapping each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ to a feature vector $\Phi(x, y) \in \mathbb{R}^d$. (4) A **parameter vector** $\bar{\alpha} \in \mathbb{R}^d$.

The components $\mathbf{GEN}, \Phi$ and $\bar{\alpha}$ define a mapping from an input x to an output $F(x)$ through

$$F(x) = \underset{y \in \mathbf{GEN}(x)}{\operatorname{argmax}} \Phi(x, y) \cdot \bar{\alpha} \quad (1)$$

where $\Phi(x, y) \cdot \bar{\alpha}$ is the inner product $\sum_s \alpha_s \Phi_s(x, y)$. The learning task is to set the parameter values $\bar{\alpha}$ using the training examples as evidence. The *decoding algorithm* is a method for searching for the y that maximizes Eq. 1.

2.2 The Perceptron algorithm

We now turn to methods for training the parameters $\bar{\alpha}$ of the model, given a set of training examples $(x_1, y_1) \ldots (x_N, y_N)$. This section describes the perceptron algorithm, which was previously applied to language modeling in Roark et al. (2004). The next section describes an alternative method, based on conditional random fields.

Inputs: Training examples (x_i, y_i)
Initialization: Set $\bar{\alpha} = 0$
Algorithm:
For $t = 1 \ldots T, i = 1 \ldots N$
Calculate $z_i = \operatorname{argmax}_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$
If($z_i \neq y_i$) then $\bar{\alpha} = \bar{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Output: Parameters $\bar{\alpha}$

Figure 1: A variant of the perceptron algorithm.

The perceptron algorithm is shown in figure 1. At each training example (x_i, y_i), the current best-scoring hypothesis z_i is found, and if it differs from the reference y_i , then the cost of each feature[2] is increased by the count of that feature in z_i and decreased by the count of that feature in y_i. The features in the model are updated, and the algorithm moves to the next utterance. After each pass over the training data, performance on a held-out data set is evaluated, and the parameterization with the best performance on the held out set is what is ultimately produced by the algorithm.

Following Collins (2002), we used the *averaged* parameters from the training algorithm in decoding held-out and test examples in our experiments. Say $\bar{\alpha}_i^t$ is the parameter vector after the i'th example is processed on the t'th pass through the data in the algorithm in figure 1. Then the averaged parameters $\bar{\alpha}_{AVG}$ are defined as $\bar{\alpha}_{AVG} = \sum_{i,t} \bar{\alpha}_i^t / NT$. Freund and Schapire (1999) originally proposed the averaged parameter method; it was shown to give substantial improvements in accuracy for tagging tasks in Collins (2002).

2.3 Conditional Random Fields

Conditional Random Fields have been applied to NLP tasks such as parsing (Ratnaparkhi et al., 1994; Johnson et al., 1999), and tagging or segmentation tasks (Lafferty et al., 2001; Sha and Pereira, 2003; McCallum and Li, 2003; Pinto et al., 2003). CRFs use the parameters $\bar{\alpha}$ to define a conditional distribution over the members of $\mathbf{GEN}(x)$ for a given input x:

$$p_{\bar{\alpha}}(y|x) = \frac{1}{Z(x, \bar{\alpha})} \exp\left(\Phi(x, y) \cdot \bar{\alpha}\right)$$

where $Z(x, \bar{\alpha}) = \sum_{y \in \mathbf{GEN}(x)} \exp\left(\Phi(x, y) \cdot \bar{\alpha}\right)$ is a normalization constant that depends on x and $\bar{\alpha}$.

Given these definitions, the log-likelihood of the training data under parameters $\bar{\alpha}$ is

$$\begin{aligned} LL(\bar{\alpha}) &= \sum_{i=1}^{N} \log p_{\bar{\alpha}}(y_i|x_i) \\ &= \sum_{i=1}^{N} \left[\Phi(x_i, y_i) \cdot \bar{\alpha} - \log Z(x_i, \bar{\alpha})\right] \end{aligned} \quad (2)$$

[2]Note that here lattice weights are interpreted as costs, which changes the sign in the algorithm presented in figure 1.

Following Johnson et al. (1999) and Lafferty et al. (2001), we use a zero-mean Gaussian prior on the parameters resulting in the regularized objective function:

$$LL_R(\bar{\alpha}) = \sum_{i=1}^{N} [\Phi(x_i, y_i) \cdot \bar{\alpha} - \log Z(x_i, \bar{\alpha})] - \frac{||\bar{\alpha}||^2}{2\sigma^2} \quad (3)$$

The value σ dictates the relative influence of the log-likelihood term vs. the prior, and is typically estimated using held-out data. The optimal parameters under this criterion are $\bar{\alpha}^* = \text{argmax}_{\bar{\alpha}} LL_R(\bar{\alpha})$.

We use a *limited memory variable metric* method (Benson and Moré, 2002) to optimize LL_R. There is a general implementation of this method in the Tao/PETSc software libraries (Balay et al., 2002; Benson et al., 2002). This technique has been shown to be very effective in a variety of NLP tasks (Malouf, 2002; Wallach, 2002). The main interface between the optimizer and the training data is a procedure which takes a parameter vector $\bar{\alpha}$ as input, and in turn returns $LL_R(\bar{\alpha})$ as well as the gradient of LL_R at $\bar{\alpha}$. The derivative of the objective function with respect to a parameter α_s at parameter values $\bar{\alpha}$ is

$$\frac{\partial LL_R}{\partial \alpha_s} = \sum_{i=1}^{N} \left[\Phi_s(x_i, y_i) - \sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i) \Phi_s(x_i, y) \right] - \frac{\alpha_s}{\sigma^2} \quad (4)$$

Note that $LL_R(\bar{\alpha})$ is a convex function, so that there is a globally optimal solution and the optimization method will find it. The use of the Gaussian prior term $||\bar{\alpha}||^2/2\sigma^2$ in the objective function has been found to be useful in several NLP settings. It effectively ensures that there is a large penalty for parameter values in the model becoming too large – as such, it tends to control over-training. The choice of LL_R as an objective function can be justified as maximum a-posteriori (MAP) training within a Bayesian approach. An alternative justification comes through a connection to support vector machines and other large margin approaches. SVM-based approaches use an optimization criterion that is closely related to LL_R – see Collins (2004) for more discussion.

3 Linear models for speech recognition

We now describe how the formalism and algorithms in section 2 can be applied to language modeling for speech recognition.

3.1 The basic approach

As described in the previous section, linear models require definitions of $\mathcal{X}$, $\mathcal{Y}$, x_i, y_i, $\mathbf{GEN}$, Φ and a parameter estimation method. In the language modeling setting we take $\mathcal{X}$ to be the set of all possible acoustic inputs; $\mathcal{Y}$ is the set of all possible strings, Σ^*, for some vocabulary Σ. Each x_i is an utterance (a sequence of acoustic feature-vectors), and $\mathbf{GEN}(x_i)$ is the set of possible transcriptions under a first pass recognizer. ($\mathbf{GEN}(x_i)$ is a huge set, but will be represented compactly using a lattice – we will discuss this in detail shortly). We take y_i to be the member of $\mathbf{GEN}(x_i)$ with lowest error rate with respect to the reference transcription of x_i.

All that remains is to define the feature-vector representation, $\Phi(x, y)$. In the general case, each component $\Phi_i(x, y)$ could be essentially any function of the acoustic input x and the candidate transcription y. The first feature we define is $\Phi_0(x, y)$ as *the log-probability of y given x under the lattice produced by the baseline recognizer*. Thus this feature will include contributions from the acoustic model and the original language model. The remaining features are restricted to be functions over the transcription y alone and they track all n-grams up to some length (say $n = 3$), for example:

$$\Phi_1(x, y) \quad = \quad \text{Number of times "}\textit{the the of}\text{" is seen in } y$$

At an abstract level, features of this form are introduced for *all* n-grams up to length 3 seen in some training data lattice, i.e., n-grams seen in any word sequence within the lattices. In practice, we consider methods that search for sparse parameter vectors $\bar{\alpha}$, thus assigning many n-grams 0 weight. This will lead to more efficient algorithms that avoid dealing explicitly with the entire set of n-grams seen in training data.

3.2 Implementation using WFA

We now give a brief sketch of how weighted finite-state automata (WFA) can be used to implement linear models for speech recognition. There are several papers describing the use of weighted automata and transducers for speech in detail, e.g., Mohri et al. (2002), but for clarity and completeness this section gives a brief description of the operations which we use.

For our purpose, a WFA $A = (\Sigma, Q, q_s, F, E, \rho)$, where Σ is the vocabulary, Q is a (finite) set of states, $q_s \in Q$ is a unique start state, $F \subseteq Q$ is a set of final states, E is a (finite) set of transitions, and $\rho : F \rightarrow \mathbb{R}$ is a function from final states to final weights. Each transition $e \in E$ is a tuple $e = (l[e], p[e], n[e], w[e])$, where $l[e] \in \Sigma$ is a label (in our case, words), $p[e] \in Q$ is the origin state of e, $n[e] \in Q$ is the destination state of e, and $w[e] \in \mathbb{R}$ is the weight of the transition. A successful path $\pi = e_1 \ldots e_j$ is a sequence of transitions, such that $p[e_1] = q_s$, $n[e_j] \in F$, and for $1 < k \leq j$, $n[e_{k-1}] = p[e_k]$. Let Π_A be the set of successful paths π in a WFA A. For any $\pi = e_1 \ldots e_j$, $l[\pi] = l[e_1] \ldots l[e_j]$.

The weights of the WFA in our case are always in the log semiring, which means that the weight of a path $\pi = e_1 \ldots e_j \in \Pi_A$ is defined as:

$$w_A[\pi] \quad = \quad \left(\sum_{k=1}^{j} w[e_k] \right) + \rho(e_j) \quad (5)$$

By convention, we use negative log probabilities as weights, so lower weights are better. All WFA that we will discuss in this paper are deterministic, i.e. there are no ϵ transitions, and for any two transitions $e, e' \in E$, if $p[e] = p[e']$, then $l[e] \neq l[e']$. Thus, for any string $\mathbf{w} = w_1 \ldots w_j$, there is at most one successful path $\pi \in \Pi_A$, such that $\pi = e_1 \ldots e_j$ and for $1 \leq k \leq j$, $l[e_k] = w_k$, i.e. $l[\pi] = \mathbf{w}$. The set of strings $\mathbf{w}$ such that there exists a $\pi \in \Pi_A$ with $l[\pi] = \mathbf{w}$ define a regular language $L_A \subseteq \Sigma$.

We can now define some operations that will be used in this paper.

• λA. For a set of transitions E and $\lambda \in \mathbb{R}$, define $\lambda E = \{(l[e],\ p[e],\ n[e],\ \lambda w[e]) : e \in E\}$. Then, for any WFA $A = (\Sigma, Q, q_s, F, E, \rho)$, define λA for $\lambda \in \mathbb{R}$ as follows: $\lambda A = (\Sigma, Q, q_s, F, \lambda E, \lambda\rho)$.

• $A \circ A'$. The intersection of two deterministic WFAs $A \circ A'$ in the log semiring is a deterministic WFA such that $L_{A \circ A'} = L_A \bigcap L_{A'}$. For any $\pi \in \Pi_{A \circ A'}$, $w_{A \circ A'}[\pi] = w_A[\pi_1] + w_{A'}[\pi_2]$, where $l[\pi] = l[\pi_1] = l[\pi_2]$.

• $\mathbf{BestPath}(A)$. This operation takes a WFA A, and returns the best scoring path $\hat{\pi} = \text{argmin}_{\pi \in \Pi_A} w_A[\pi]$.

• $\mathbf{MinErr}(A, y)$. Given a WFA A, a string y, and an error-function $E(y, \mathbf{w})$, this operation returns $\hat{\pi} = \text{argmin}_{\pi \in \Pi_A} E(y, l[\pi])$. This operation will generally be used with y as the reference transcription for a particular training example, and $E(y, \mathbf{w})$ as some measure of the number of errors in $\mathbf{w}$ when compared to y. In this case, the $\mathbf{MinErr}$ operation returns the path $\pi \in \Pi_A$ such $l[\pi]$ has the smallest number of errors when compared to y.

• $\mathbf{Norm}(A)$. Given a WFA A, this operation yields a WFA A' such that $L_A = L_{A'}$ and for every $\pi \in \Pi_A$ there is a $\pi' \in \Pi_{A'}$ such that $l[\pi] = l[\pi']$ and

$$w_{A'}[\pi'] \quad = \quad w_A[\pi] + \log\left(\sum_{\bar{\pi} \in \Pi_A} \exp(-w_A[\bar{\pi}])\right) \quad (6)$$

Note that

$$\sum_{\pi \in \mathbf{Norm}(A)} \exp(-w_{\mathbf{Norm}(A)}[\pi]) \quad = \quad 1 \qquad (7)$$

In other words the weights define a probability distribution over the paths.

• $\mathbf{ExpCount}(A, \mathbf{w})$. Given a WFA A and an n-gram $\mathbf{w}$, we define the expected count of $\mathbf{w}$ in A as

$$\mathbf{ExpCount}(A, \mathbf{w}) = \sum_{\pi \in \Pi_A} w_{\mathbf{Norm}(A)}[\pi] C(\mathbf{w}, l[\pi])$$

where $C(\mathbf{w}, l[\pi])$ is defined to be the number of times the n-gram $\mathbf{w}$ appears in a string $l[\pi]$.

Given an acoustic input x, let $\mathcal{L}_x$ be a deterministic word-lattice produced by the baseline recognizer. The lattice $\mathcal{L}_x$ is an acyclic WFA, representing a weighted set of possible transcriptions of x under the baseline recognizer. The weights represent the combination of acoustic and language model scores in the original recognizer.

The new, discriminative language model constructed during training consists of a deterministic WFA which we will denote $\mathcal{D}$, together with a single parameter α_0. The parameter α_0 is the weight for the log probability feature Φ_0 given by the baseline recognizer. The WFA $\mathcal{D}$ is constructed so that $L_{\mathcal{D}} = \Sigma^*$ and for all $\pi \in \Pi_{\mathcal{D}}$

$$w_{\mathcal{D}}[\pi] = \sum_{j=1}^{d} \Phi_j(x, l[\pi]) \alpha_j$$

Recall that $\Phi_j(x, \mathbf{w})$ for $j > 0$ is the count of the j'th n-gram in $\mathbf{w}$, and α_j is the parameter associated with that n-gram. Then, by definition, $\alpha_0 \mathcal{L} \circ \mathcal{D}$ accepts the same set of strings as $\mathcal{L}$, but

$$w_{\alpha_0 \mathcal{L} \circ \mathcal{D}}[\pi] = \sum_{j=0}^{d} \Phi_j(x, l[\pi]) \alpha_j$$

and $\text{argmin}_{\pi \in \mathcal{L}} \Phi(x, l[\pi]) \cdot \bar{\alpha} = \mathbf{BestPath}(\alpha_0 \mathcal{L} \circ \mathcal{D})$.

Thus decoding under our new model involves first producing a lattice $\mathcal{L}$ from the baseline recognizer; second, scaling $\mathcal{L}$ with α_0 and intersecting it with the discriminative language model $\mathcal{D}$; third, finding the best scoring path in the new WFA.

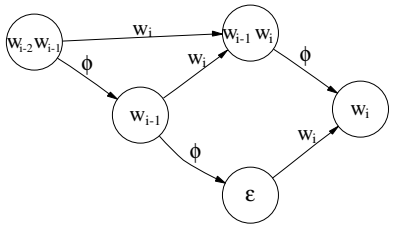

Figure 2: Representation of a trigram model with failure transitions.

We now turn to training a model, or more explicitly, deriving a discriminative language model $(\mathcal{D}, \alpha_0)$ from a set of training examples. Given a training set (x_i, r_i) for $i = 1 \ldots N$, where x_i is an acoustic sequence, and r_i is a reference transcription, we can construct lattices $\mathcal{L}_i$ for $i = 1 \ldots N$ using the baseline recognizer. We can also derive target transcriptions $y_i = \mathbf{MinErr}(\mathcal{L}_i, r_i)$. The training algorithm is then a mapping from $(\mathcal{L}_i, y_i)$ for $i = 1 \ldots N$ to a pair $(\mathcal{D}, \alpha_0)$. Note that the construction of the language model requires two choices. The first concerns the choice of the set of n-gram *features* Φ_i for $i = 1 \ldots d$ implemented by $\mathcal{D}$. The second concerns the choice of *parameters* α_i for $i = 0 \ldots d$ which assign weights to the n-gram features as well as the baseline feature Φ_0.

Before describing methods for training a discriminative language model using perceptron and CRF algorithms, we give a little more detail about the structure of $\mathcal{D}$, focusing on how n-gram language models can be implemented with finite-state techniques.

3.3 Representation of n-gram language models

An n-gram model can be efficiently represented in a deterministic WFA, through the use of failure transitions (Allauzen et al., 2003). Every string accepted by such an automaton has a single path through the automaton, and the weight of the string is the sum of the weights of the transitions in that path. In such a representation, every state in the automaton represents an n-gram history h, e.g. $w_{i-2} w_{i-1}$, and there are transitions leaving the state for every word w_i such that the feature hw_i has a weight. There is also a failure transition leaving the state, labeled with some reserved symbol ϕ, which can only be traversed if the next symbol in the input does not match any transition leaving the state. This failure transition points to the backoff state h', i.e. the n-gram history h minus its initial word. Figure 2 shows how a trigram model can be represented in such an automaton. See Allauzen et al. (2003) for more details.

Note that in such a deterministic representation, the entire weight of all features associated with the word w_i following history h must be assigned to the transition labeled with w_i leaving the state h in the automaton. For example, if $h = w_{i-2}w_{i-1}$, then the trigram $w_{i-2}w_{i-1}w_i$ is a feature, as is the bigram $w_{i-1}w_i$ and the unigram w_i. In this case, the weight on the transition w_i leaving state h must be the sum of the trigram, bigram and unigram feature weights. If only the trigram feature weight were assigned to the transition, neither the unigram nor the bigram feature contribution would be included in the path weight. In order to ensure that the correct weights are assigned to each string, every transition encoding an order k n-gram must carry the sum of the weights for all n-gram features of orders $\leq k$. To ensure that every string in Σ^* receives the correct weight, for any n-gram hw represented explicitly in the automaton, $h'w$ must also be represented explicitly in the automaton, even if its weight is 0.

3.4 The perceptron algorithm

The perceptron algorithm is incremental, meaning that the language model $\mathcal{D}$ is built one training example at a time, during several passes over the training set. Initially, we build $\mathcal{D}$ to accept all strings in Σ^* with weight 0. For the perceptron experiments, we chose the parameter α_0 to be a fixed constant, chosen by optimization on the held-out set. The loop in the algorithm in figure 1 is implemented as:
For $t = 1 \ldots T, i = 1 \ldots N$:

- Calculate $z_i = \text{argmax}_{y \in \mathbf{GEN}(x)} \Phi(x, y) \cdot \bar{\alpha}$
  $= \mathbf{BestPath}(\alpha_0 \mathcal{L}_i \circ \mathcal{D})$
- If $z_i \neq \mathbf{MinErr}(\mathcal{L}_i, r_i)$, then update the feature weights as in figure 1 (modulo the sign, because of the use of costs), and modify $\mathcal{D}$ so as to assign the correct weight to all strings.

In addition, averaged parameters need to be stored (see section 2.2). These parameters will replace the unaveraged parameters in $\mathcal{D}$ once training is completed.

Note that the only n-gram features to be included in $\mathcal{D}$ at the end of the training process are those that occur in either a best scoring path z_i or a minimum error path y_i at some point during training. Thus the perceptron algorithm is in effect doing feature selection as a by-product of training. Given N training examples, and T passes over the training set, $O(NT)$ n-grams will have non-zero weight after training. Experiments in Roark et al. (2004) suggest that the perceptron reaches optimal performance after a small number of training iterations, for example $T = 1$ or $T = 2$. Thus $O(NT)$ can be very small compared to the full number of n-grams seen in all training lattices. In our experiments, the perceptron method chose around 1.4 million n-grams with non-zero weight. This compares to 43.65 million possible n-grams seen in the training data.

This is a key contrast with conditional random fields, which optimize the parameters of a fixed feature set. Feature selection can be critical in our domain, as training and applying a discriminative language model over *all* n-grams seen in the training data (in either correct or incorrect transcriptions) may be computationally very demanding. One training scenario that we will consider will be using the output of the perceptron algorithm (the averaged parameters) to provide the feature set and the initial feature weights for use in the CRF algorithm. This leads to a model which is reasonably sparse, but has the benefit of CRF training, which as we will see gives gains in performance.

3.5 Conditional Random Fields

The CRF methods that we use assume a fixed definition of the n-gram features Φ_i for $i = 1 \ldots d$ in the model. In the experimental section we will describe a number of ways of defining the feature set. The optimization methods we use begin at some initial setting for $\bar{\alpha}$, and then search for the parameters $\bar{\alpha}^*$ which maximize $LL_R(\bar{\alpha})$ as defined in Eq. 3.

The optimization method requires calculation of $LL_R(\bar{\alpha})$ and the gradient of $LL_R(\bar{\alpha})$ for a series of values for $\bar{\alpha}$. The first step in calculating these quantities is to take the parameter values $\bar{\alpha}$, and to construct an acceptor $\mathcal{D}$ which accepts all strings in Σ^*, such that

$$w_{\mathcal{D}}[\pi] = \sum_{j=1}^{d} \Phi_j(x, l[\pi])\alpha_j$$

For each training lattice $\mathcal{L}_i$, we then construct a new lattice $\mathcal{L}'_i = \mathbf{Norm}(\alpha_0 \mathcal{L}_i \circ \mathcal{D})$. The lattice $\mathcal{L}'_i$ represents (in the log domain) the distribution $p_{\bar{\alpha}}(y|x_i)$ over strings $y \in \mathbf{GEN}(x_i)$. The value of $\log p_{\bar{\alpha}}(y_i|x_i)$ for any i can be computed by simply taking the path weight of π such that $l[\pi] = y_i$ in the new lattice $\mathcal{L}'_i$. Hence computation of $LL_R(\bar{\alpha})$ in Eq. 3 is straightforward.

Calculating the n-gram feature gradients for the CRF optimization is also relatively simple, once $\mathcal{L}'_i$ has been constructed. From the derivative in Eq. 4, for each $i = 1 \ldots N, j = 1 \ldots d$ the quantity

$$\Phi_j(x_i, y_i) - \sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i)\Phi_j(x_i, y) \quad (8)$$

must be computed. The first term is simply the number of times the j'th n-gram feature is seen in y_i. The second term is the expected number of times that the j'th n-gram is seen in the acceptor $\mathcal{L}'_i$. If the j'th n-gram is $w_1 \ldots w_n$, then this can be computed as $\mathbf{ExpCount}(\mathcal{L}'_i, w_1 \ldots w_n)$. The GRM library, which was presented in Allauzen et al. (2003), has a direct implementation of the function $\mathbf{ExpCount}$, which simultaneously calculates the expected value of all n-grams of order less than or equal to a given n in a lattice $\mathcal{L}$.

The one non-ngram feature weight that is being estimated is the weight α_0 given to the baseline ASR negative log probability. Calculation of the gradient of LL_R with respect to this parameter again requires calculation of the term in Eq. 8 for $j = 0$ and $i = 1 \ldots N$. Computation of $\sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i)\Phi_0(x_i, y)$ turns out to be not as straightforward as calculating n-gram expectations. To do so, we rely upon the fact that $\Phi_0(x_i, y)$, the negative log probability of the path, decomposes to

the sum of negative log probabilities of each transition in the path. We index each transition in the lattice $\mathcal{L}_i$, and store its negative log probability under the baseline model. We can then calculate the required gradient from $\mathcal{L}'_i$, by calculating the expected value in $\mathcal{L}'_i$ of each indexed transition in $\mathcal{L}_i$.

We found that an approximation to the gradient of α_0, however, performed nearly identically to this exact gradient, while requiring substantially less computation. Let w_1^n be a string of n words, labeling a path in word-lattice $\mathcal{L}'_i$. For brevity, let $\mathrm{P}_i(w_1^n) = p_{\bar{\alpha}}(w_1^n|x_i)$ be the conditional probability under the current model, and let $\mathrm{Q}_i(w_1^n)$ be the probability of w_1^n in the normalized baseline ASR lattice $\mathbf{Norm}(\mathcal{L}_i)$. Let L_i be the set of strings in the language defined by $\mathcal{L}_i$. Then we wish to compute E_i for $i = 1 \ldots N$, where

$$\begin{aligned} E_i &= \sum_{w_1^n \in L_i} \mathrm{P}_i(w_1^n) \log \mathrm{Q}_i(w_1^n) \\ &= \sum_{w_1^n \in L_i} \sum_{k=1\ldots n} \mathrm{P}_i(w_1^n) \log \mathrm{Q}_i(w_k|w_1^{k-1}) \end{aligned} \quad (9)$$

The approximation is to make the following Markov assumption:

$$\begin{aligned} E_i &\approx \sum_{w_1^n \in L_i} \sum_{k=1\ldots n} \mathrm{P}_i(w_1^n) \log \mathrm{Q}_i(w_k|w_{k-2}^{k-1}) \\ &= \sum_{xyz \in S_i} \mathbf{ExpCount}(\mathcal{L}'_i, xyz) \log \mathrm{Q}_i(z|xy) \end{aligned} \quad (10)$$

where S_i is the set of all trigrams seen in L_i. The term $\log \mathrm{Q}_i(z|xy)$ can be calculated once before training for every lattice in the training set; the $\mathbf{ExpCount}$ term is calculated as before using the GRM library. We have found this approximation to be effective in practice, and it was used for the trials reported below.

When the gradients and conditional likelihoods are collected from all of the utterances in the training set, the contributions from the regularizer are combined to give an overall gradient and objective function value. These values are provided to the parameter estimation routine, which then returns the parameters for use in the next iteration. The accumulation of gradients for the feature set is the most time consuming part of the approach, but this is parallelizable, so that the computation can be divided among many processors.

4 Empirical Results

We present empirical results on the Rich Transcription 2002 evaluation test set (rt02), which we used as our development set, as well as on the Rich Transcription 2003 Spring evaluation CTS test set (rt03). The rt02 set consists of 6081 sentences (63804 words) and has three subsets: Switchboard 1, Switchboard 2, Switchboard Cellular. The rt03 set consists of 9050 sentences (76083 words) and has two subsets: Switchboard and Fisher.

We used the same training set as that used in Roark et al. (2004). The training set consists of 276726 transcribed utterances (3047805 words), with an additional 20854 utterances (249774 words) as held out data. For each utterance, a weighted word-lattice was produced, representing alternative transcriptions, from the ASR system. From each word-lattice, the oracle best path was extracted, which gives the best word-error rate from among all of the hypotheses in the lattice. The oracle word-error rate for the training set lattices was 12.2%. We also performed trials with 1000-best lists for the same training set, rather than lattices. The oracle score for the 1000-best lists was 16.7%.

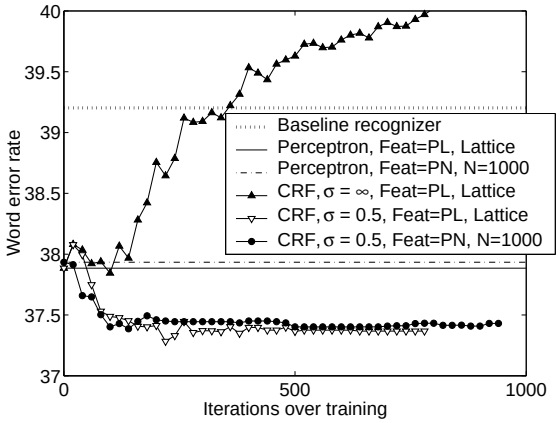

Figure 3: Word error rate on the rt02 eval set versus training iterations for CRF trials, contrasted with baseline recognizer performance and perceptron performance. Points are at every 20 iterations. Each point (x,y) is the WER at the iteration with the best objective function value in the interval (x-20,x].

To produce the word-lattices, each training utterance was processed by the baseline ASR system. However, these same utterances are what the acoustic and language models are built from, which leads to better performance on the training utterances than can be expected when the ASR system processes unseen utterances. To somewhat control for this, the training set was partitioned into 28 sets, and baseline Katz backoff trigram models were built for each set by including only transcripts from the other 27 sets. Since language models are generally far more prone to overtrain than standard acoustic models, this goes a long way toward making the training conditions similar to testing conditions.

There are three baselines against which we are comparing. The first is the ASR baseline, with no reweighting from a discriminatively trained n-gram model. The other two baselines are with perceptron-trained n-gram model re-weighting, and were reported in Roark et al. (2004). The first of these is for a pruned-lattice trained trigram model, which showed a reduction in word error rate (WER) of 1.3%, from 39.2% to 37.9% on rt02. The second is for a 1000-best list trained trigram model, which performed only marginally worse than the lattice-trained perceptron, at 38.0% on rt02.

4.1 Perceptron feature set

We use the perceptron-trained models as the starting point for our CRF algorithm: the feature set given to the CRF algorithm is the feature set selected by the perceptron algorithm; the feature weights are initialized to those of the averaged perceptron. Figure 3 shows the performance of our three baselines versus three trials of

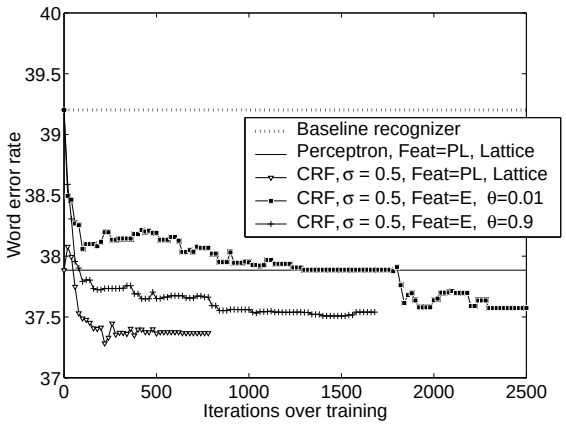

Figure 4: Word error rate on the rt02 eval set versus training iterations for CRF trials, contrasted with baseline recognizer performance and perceptron performance. Points are at every 20 iterations. Each point (x,y) is the WER at the iteration with the best objective function value in the interval (x-20,x].

the CRF algorithm. In the first two trials, the training set consists of the pruned lattices, and the feature set is from the perceptron algorithm trained on pruned lattices. There were 1.4 million features in this feature set. The first trial set the regularizer constant $\sigma = \infty$, so that the algorithm was optimizing raw conditional likelihood. The second trial is with the regularizer constant $\sigma = 0.5$, which we found empirically to be a good parameterization on the held-out set. As can be seen from these results, regularization is critical.

The third trial in this set uses the feature set from the perceptron algorithm trained on 1000-best lists, and uses CRF optimization on these on these same 1000-best lists. There were 0.9 million features in this feature set. For this trial, we also used $\sigma = 0.5$. As with the perceptron baselines, the n-best trial performs nearly identically with the pruned lattices, here also resulting in 37.4% WER. This may be useful for techniques that would be more expensive to extend to lattices versus n-best lists (e.g. models with unbounded dependencies).

These trials demonstrate that the CRF algorithm can do a better job of estimating feature weights than the perceptron algorithm for the same feature set. As mentioned in the earlier section, feature selection is a by-product of the perceptron algorithm, but the CRF algorithm is given a set of features. The next two trials looked at selecting feature sets other than those provided by the perceptron algorithm.

4.2 Other feature sets

In order for the feature weights to be non-zero in this approach, they must be observed in the training set. The number of unigram, bigram and trigram features with non-zero observations in the training set lattices is 43.65 million, or roughly 30 times the size of the perceptron feature set. Many of these features occur only rarely with very low conditional probabilities, and hence cannot meaningfully impact system performance. We pruned this feature set to include all unigrams and bigrams, but only those trigrams with an expected count of greater than 0.01 in the training set. That is, to be included, a trigram must occur in a set of paths, the sum of the conditional probabilities of which must be greater than our threshold $\theta = 0.01$. This threshold resulted in a feature set of roughly 12 million features, nearly 10 times the size of the perceptron feature set. For better comparability with that feature set, we set our thresholds higher, so that trigrams were pruned if their expected count fell below $\theta = 0.9$, and bigrams were pruned if their expected count fell below $\theta = 0.1$. We were concerned that this may leave out some of the features on the oracle paths, so we added back in all bigram and trigram features that occurred on oracle paths, giving a feature set of 1.5 million features, roughly the same size as the perceptron feature set.

Trial	Iter	rt02	rt03
ASR Baseline	-	39.2	38.2
Perceptron, Lattice	-	37.9	36.9
Perceptron, N-best	-	38.0	37.2
CRF, Lattice, Percep Feats (1.4M)	769	37.4	36.5
CRF, N-best, Percep Feats (0.9M)	946	37.4	36.6
CRF, Lattice, $\theta = 0.01$ (12M)	2714	37.6	36.5
CRF, Lattice, $\theta = 0.9$ (1.5M)	1679	37.5	36.6

Table 1: Word-error rate results at convergence iteration for various trials, on both Switchboard 2002 test set (rt02), which was used as the dev set, and Switchboard 2003 test set (rt03).

Figure 4 shows the results for three CRF trials versus our ASR baseline and the perceptron algorithm baseline trained on lattices. First, the result using the perceptron feature set provides us with a WER of 37.4%, as previously shown. The WER at convergence for the big feature set (12 million features) is 37.6%; the WER at convergence for the smaller feature set (1.5 million features) is 37.5%. While both of these other feature sets converge to performance close to that using the perceptron features, the number of iterations over the training data that are required to reach that level of performance are many more than for the perceptron-initialized feature set.

Table 1 shows the word-error rate at the convergence iteration for the various trials, on both rt02 and rt03. All of the CRF trials are significantly better than the perceptron performance, using the Matched Pair Sentence Segment test for WER included with SCTK (NIST, 2000). On rt02, the N-best and perceptron initialized CRF trials were were significantly better than the lattice perceptron at $p < 0.001$; the other two CRF trials were significantly better than the lattice perceptron at $p < 0.01$. On rt03, the N-best CRF trial was significantly better than the lattice perceptron at $p < 0.002$; the other three CRF trials were significantly better than the lattice perceptron at $p < 0.001$.

Finally, we measured the time of a single iteration over the training data on a single machine for the perceptron algorithm, the CRF algorithm using the approximation to the gradient of α_0, and the CRF algorithm using an exact gradient of α_0. Table 2 shows these times in hours. Because of the frequent update of the weights in the model, the perceptron algorithm is more expensive than the CRF algorithm for a single iteration. Further, the CRF algorithm is parallelizable, so that most of the work of an

Features	Percep	CRF approx	CRF exact
Lattice, Percep Feats (1.4M)	7.10	1.69	3.61
N-best, Percep Feats (0.9M)	3.40	0.96	1.40
Lattice, $\theta = 0.01$ (12M)	-	2.24	4.75

Table 2: Time (in hours) for one iteration on a single Intel Xeon 2.4Ghz processor with 4GB RAM.

iteration can be shared among multiple processors. Our most common training setup for the CRF algorithm was parallelized between 20 processors, using the approximation to the gradient. In that setup, using the 1.4M feature set, one iteration of the perceptron algorithm took the same amount of real time as approximately 80 iterations of CRF.

5 Conclusion

We have contrasted two approaches to discriminative language model estimation on a difficult large vocabulary task, showing that they can indeed scale effectively to handle this size of a problem. Both algorithms have their benefits. The perceptron algorithm selects a relatively small subset of the total feature set, and requires just a couple of passes over the training data. The CRF algorithm does a better job of parameter estimation for the same feature set, and is parallelizable, so that each pass over the training set can require just a fraction of the real time of the perceptron algorithm.

The best scenario from among those that we investigated was a combination of both approaches, with the output of the perceptron algorithm taken as the starting point for CRF estimation.

As a final point, note that the methods we describe do not replace an existing language model, but rather complement it. The existing language model has the benefit that it can be trained on a large amount of text that does not have speech transcriptions. It has the disadvantage of not being a discriminative model. The new language model is trained on the speech transcriptions, meaning that it has less training data, but that it has the advantage of discriminative training – and in particular, the advantage of being able to learn negative evidence in the form of negative weights on n-grams which are rarely or never seen in natural language text (e.g., "the of"), but are produced too frequently by the recognizer. The methods we describe combines the two language models, allowing them to complement each other.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47.

Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. 2002. Petsc users manual. Technical Report ANL-95/11-Revision 2.1.2, Argonne National Laboratory.

Satanjeev Banerjee, Jack Mostow, Joseph Beck, and Wilson Tam. 2003. Improving language models by learning from speech recognition errors in a reading tutor that listens. In *Proceedings of the Second International Conference on Applied Artificial Intelligence*, Fort Panhala, Kolhapur, India.

Steven J. Benson and Jorge J. Moré. 2002. A limited memory variable metric method for bound constrained minimization. Preprint ANL/ACSP909-0901, Argonne National Laboratory.

Steven J. Benson, Lois Curfman McInnes, Jorge J. Moré, and Jason Sarich. 2002. Tao users manual. Technical Report ANL/MCS-TM-242-Revision 1.4, Argonne National Laboratory.

Zheng Chen, Kai-Fu Lee, and Ming Jing Li. 2000. Discriminative training on language model. In *Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP)*, Beijing, China.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

Michael Collins. 2004. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*. Kluwer.

Yoav Freund and Robert Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 3(37):277–296.

Frederick Jelinek. 1995. Acoustic sensitive language modeling. Technical report, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD.

Mark Johnson, Stuart Geman, Steven Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 535–541.

Sanjeev Khudanpur and Jun Wu. 2000. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. *Computer Speech and Language*, 14(4):355–372.

Hong-Kwang Jeff Kuo, Eric Fosler-Lussier, Hui Jiang, and Chin-Hui Lee. 2002. Discriminative training of language models for speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Orlando, Florida.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, pages 282–289, Williams College, Williamstown, MA, USA.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proc. CoNLL*, pages 49–55.

Andrew McCallum and Wei Li. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proc. CoNLL*.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88.

NIST. 2000. Speech recognition scoring toolkit (sctk) version 1.2c. Available at `http://www.nist.gov/speech/tools`.

David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. 2003. Table extraction using conditional random fields. In *Proc. ACM SIGIR*.

Adwait Ratnaparkhi, Salim Roukos, and R. Todd Ward. 1994. A maximum entropy model for parsing. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, pages 803–806.

Brian Roark, Murat Saraclar, and Michael Collins. 2004. Corrective language modeling for large vocabulary ASR with the perceptron algorithm. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 749–752.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. HLT-NAACL*, Edmonton, Canada.

A. Stolcke and M. Weintraub. 1998. Discriminitive language modeling. In *Proceedings of the 9th Hub-5 Conversational Speech Recognition Workshop*.

A. Stolcke, H. Bratt, J. Butzberger, H. Franco, V. R. Rao Gadde, M. Plauche, C. Richey, E. Shriberg, K. Sonmez, F. Weng, and J. Zheng. 2000. The SRI March 2000 Hub-5 conversational speech transcription system. In *Proceedings of the NIST Speech Transcription Workshop*.

Hanna Wallach. 2002. Efficient training of conditional random fields. Master's thesis, University of Edinburgh.

P.C. Woodland and D. Povey. 2000. Large scale discriminative training for speech recognition. In *Proc. ISCA ITRW ASR2000*, pages 7–16.