# Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms

**Michael Collins**
AT&T Labs-Research, Florham Park, New Jersey.
mcollins@research.att.com

### Abstract

We describe new algorithms for training tagging models, as an alternative to maximum-entropy models or conditional random fields (CRFs). The algorithms rely on Viterbi decoding of training examples, combined with simple additive updates. We describe theory justifying the algorithms through a modification of the proof of convergence of the perceptron algorithm for classification problems. We give experimental results on part-of-speech tagging and base noun phrase chunking, in both cases showing improvements over results for a maximum-entropy tagger.

## 1 Introduction

Maximum-entropy (ME) models are justifiably a very popular choice for tagging problems in Natural Language Processing: for example see (Ratnaparkhi 96) for their use on part-of-speech tagging, and (McCallum et al. 2000) for their use on a FAQ segmentation task. ME models have the advantage of being quite flexible in the features that can be incorporated in the model. However, recent theoretical and experimental results in (Lafferty et al. 2001) have highlighted problems with the parameter estimation method for ME models. In response to these problems, they describe alternative parameter estimation methods based on Conditional Markov Random Fields (CRFs). (Lafferty et al. 2001) give experimental results suggesting that CRFs can perform significantly better than ME models.

In this paper we describe parameter estimation algorithms which are natural alternatives to CRFs. The algorithms are based on the perceptron algorithm (Rosenblatt 58), and the voted or averaged versions of the perceptron described in (Freund & Schapire 99). These algorithms have been shown by (Freund & Schapire 99) to be competitive with modern learning algorithms such as support vector machines; however, they have previously been applied mainly to classification tasks, and it is not entirely clear how the algorithms can be carried across to NLP tasks such as tagging or parsing.

This paper describes variants of the perceptron algorithm for tagging problems. The algorithms rely on Viterbi decoding of training examples, combined with simple additive updates. We describe theory justifying the algorithm through a modification of the proof of convergence of the perceptron algorithm for classification problems. We give experimental results on part-of-speech tagging and base noun phrase chunking, in both cases showing improvements over results for a maximum-entropy tagger (a 11.9% relative reduction in error for POS tagging, a 5.1% relative reduction in error for NP chunking). Although we concentrate on tagging problems in this paper, the theoretical framework and algorithm described in section 3 of this paper should be applicable to a wide variety of models where Viterbi-style algorithms can be used for decoding: examples are Probabilistic Context-Free Grammars, or ME models for parsing. See (Collins and Duffy 2001; Collins and Duffy 2002; Collins 2002) for other applications of the voted perceptron to NLP problems.[1]

## 2 Parameter Estimation

### 2.1 HMM Taggers

In this section, as a motivating example, we describe a special case of the algorithm in this paper: the algorithm applied to a trigram tagger. In a trigram HMM tagger, each trigram

[1] The theorems in section 3, and the proofs in section 5, apply directly to the work in these other papers.

of tags and each tag/word pair have associated parameters. We write the parameter associated with a trigram $\langle x, y, z\rangle$ as $\alpha_{x,y,z}$, and the parameter associated with a tag/word pair $(t, w)$ as $\alpha_{t,w}$. A common approach is to take the parameters to be estimates of conditional probabilities: $\alpha_{x,y,z} = \log P(z \mid x, y)$, $\alpha_{t,w} = \log P(w \mid t)$.

For convenience we will use $w_{[1:n]}$ as shorthand for a sequence of words $[w_1, w_2 \ldots w_n]$, and $t_{[1:n]}$ as shorthand for a taq sequence $[t_1, t_2 \ldots t_n]$. In a trigram tagger the score for a tagged sequence $t_{[1:n]}$ paired with a word sequence $w_{[1:n]}$ is[2] $\sum_{i=1}^{n} \alpha_{t_{i-2},t_{i-1},t_i} + \sum_{i=1}^{n} \alpha_{t_i,w_i}$. When the parameters are conditional probabilities as above this "score" is an estimate of the log of the joint probability $P(w_{[1:n]}, t_{[1:n]})$. The Viterbi algorithm can be used to find the highest scoring tagged sequence under this score.

As an alternative to maximum–likelihood parameter estimates, this paper will propose the following estimation algorithm. Say the training set consists of $n$ tagged sentences, the $i$'th sentence being of length $n_i$. We will write these examples as $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ for $i = 1 \ldots n$. Then the training algorithm is as follows:

- Choose a parameter $T$ defining the number of iterations over the training set.[3]
- Initially set all parameters $\alpha_{x,y,z}$ and $\alpha_{t,w}$ to be zero.
- For $t = 1 \ldots T, i = 1 \ldots n$: Use the Viterbi algorithm to find the best tagged sequence for sentence $w^i_{[1:n_i]}$ under the current parameter settings: we call this tagged sequence $z_{[1:n_i]}$. For every tag trigram $\langle x, y, z\rangle$ seen $c_1$ times in $t^i_{[1:n_i]}$ and $c_2$ times in $z_{[1:n_i]}$ where $c_1 \neq c_2$ set $\alpha_{x,y,z} = \alpha_{x,y,z} + c_1 - c_2$. For every tag/word pair $\langle t, w\rangle$ seen $c_1$ times in $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ and $c_2$ times in $(w^i_{[1:n_i]}, z_{[1:n_i]})$ where $c_1 \neq c_2$ set $\alpha_{t,w} = \alpha_{t,w} + c_1 - c_2$.

As an example, say the $i$'th tagged sentence $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ in training data is

the/D man/N saw/V the/D dog/N

and under the current parameter settings the highest scoring tag sequence $(w^i_{[1:n_i]}, z_{[1:n_i]})$ is

the/D man/N saw/N the/D dog/N

Then the parameter update will add 1 to the parameters $\alpha_{D,N,V}$, $\alpha_{N,V,D}$, $\alpha_{V,D,N}$, $\alpha_{V,saw}$ and subtract 1 from the parameters $\alpha_{D,N,N}$, $\alpha_{N,N,D}$, $\alpha_{N,D,N}$, $\alpha_{N,saw}$. Intuitively this has the effect of increasing the parameter values for features which were "missing" from the proposed sequence $z_{[1:n_i]}$, and downweighting parameter values for "incorrect" features in the sequence $z_{[1:n_i]}$. Note that if $z_{[1:n_i]} = t^i_{[1:n_i]}$ — i.e., the proposed tag sequence is correct — no changes are made to the parameter values.

### 2.2 Local and Global Feature Vectors

We now describe how to generalize the algorithm to more general representations of tagged sequences. In this section we describe the feature-vector representations which are commonly used in maximum-entropy models for tagging, and which are also used in this paper.

In maximum-entropy taggers (Ratnaparkhi 96; McCallum et al. 2000), the tagging problem is decomposed into sequence of decisions in tagging the problem in left-to-right fashion. At each point there is a "history" – the context in which a tagging decision is made – and the task is to predict the tag given the history. Formally, a history is a 4-tuple $\langle t_{-1}, t_{-2}, w_{[1:n]}, i\rangle$ where $t_{-1}, t_{-2}$ are the previous two tags, $w_{[1:n]}$ is an array specifying the $n$ words in the input sentence, and $i$ is the index of the word being tagged. We use $\mathcal{H}$ to denote the set of all possible histories.

Maximum-entropy models represent the tagging task through a *feature-vector representation* of history-tag pairs. A feature vector representation $\phi : \mathcal{H} \times \mathcal{T} \to \mathbb{R}^d$ is a function $\phi$ that maps a history–tag pair to a $d$-dimensional feature vector. Each component $\phi_s(h, t)$ for $s = 1 \ldots d$ could be an arbitrary function of $(h, t)$. It is common (e.g., see (Ratnaparkhi 96)) for each feature $\phi_s$ to be an indicator function. For example, one such feature might be

$$\phi_{1000}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ is } \texttt{the} \\ & \text{and } t = \texttt{DT} \\ 0 & \text{otherwise} \end{cases}$$

Similar features might be defined for every word/tag pair seen in training data. Another

[2]We take $t_{-1}$ and $t_{-2}$ to be special `NULL` tag symbols.

[3]$T$ is usually chosen by tuning on a development set.

feature type might track trigrams of tags, for example $\phi_{1001}(h, t) = 1$ *if* $\langle t_{-2}, t_{-1}, t\rangle = \langle D,\ N,\ V\rangle$ *and* 0 *otherwise.* Similar features would be defined for all trigrams of tags seen in training. A real advantage of these models comes from the freedom in defining these features: for example, (Ratnaparkhi 96; McCallum et al. 2000) both describe feature sets which would be difficult to incorporate in a generative model.

In addition to feature vector representations of history/tag pairs, we will find it convenient to define feature vectors of $(w_{[1:n]}, t_{[1:n]})$ pairs where $w_{[1:n]}$ is a sequence of $n$ words, and $t_{[1:n]}$ is an entire tag sequence. We use $\Phi$ to denote a function from $(w_{[1:n]}, t_{[1:n]})$ pairs to d-dimensional feature vectors. We will often refer to $\Phi$ as a "global" representation, in contrast to $\phi$ as a "local" representation. The particular global representations considered in this paper are simple functions of local representations:

$$\Phi_s(w_{[1:n]}, t_{[1:n]}) = \sum_{i=1}^{n} \phi_s(h_i, t_i) \qquad (1)$$

where $h_i = \langle t_{i-1}, t_{i-2}, w_{[1:n]}, i\rangle$. Each global feature $\Phi_s(w_{[1:n]}, t_{[1:n]})$ is simply the value for the local representation $\phi_s$ summed over all history/tag pairs in $(w_{[1:n]}, t_{[1:n]})$. If the local features are indicator functions, then the global features will typically be "counts". For example, with $\phi_{1000}$ defined as above, $\Phi_{1000}(w_{[1:n]}, t_{[1:n]})$ is the number of times `the` is seen tagged as `DT` in the pair of sequences $(w_{[1:n]}, t_{[1:n]})$.

### 2.3 Maximum-Entropy Taggers

In maximum-entropy taggers the feature vectors $\phi$ together with a parameter vector $\bar{\alpha} \in \mathbb{R}^d$ are used to define a conditional probability distribution over tags given a history as

$$P(t \mid h, \bar{\alpha}) = \frac{e^{\sum_s \alpha_s \phi_s(h,t)}}{Z(h, \bar{\alpha})}$$

where $Z(h, \bar{\alpha}) = \sum_{l \in \mathcal{T}} e^{\sum_s \alpha_s \phi_s(h,l)}$. The log of this probability has the form $\log p(t \mid h, \bar{\alpha}) = \sum_{s=1}^{d} \alpha_s \phi_s(h, t) - \log Z(h, \bar{\alpha})$, and hence the log probability for a $(w_{[1:n]}, t_{[1:n]})$ pair will be

$$\sum_i \sum_{s=1}^{d} \alpha_s \phi_s(h_i, t_i) - \sum_i \log Z(h_i, \bar{\alpha}) \qquad (2)$$

where $h_i = \langle t_{i-1}, t_{i-2}, w_{[1:n]}, i\rangle$. Given parameter values $\bar{\alpha}$, and an input sentence $w_{[1:n]}$, the highest probability tagged sequence under the formula in Eq. 2 can be found efficiently using the Viterbi algorithm.

The parameter vector $\bar{\alpha}$ is estimated from a training set of sentence/tagged-sequence pairs. Maximum-likelihood parameter values can be estimated using Generalized Iterative Scaling (Ratnaparkhi 96), or gradient descent methods. In some cases it may be preferable to apply a bayesian approach which includes a prior over parameter values.

### 2.4 A New Estimation Method

We now describe an alternative method for estimating parameters of the model. Given a sequence of words $w_{[1:n]}$ and a sequence of part of speech tags, $t_{[1:n]}$, we will take the "score" of a tagged sequence to be

$$\sum_{i=1}^{n} \sum_{s=1}^{d} \alpha_s \phi_s(h_i, t_i) = \sum_{s=1}^{d} \alpha_s \Phi_s(w_{[1:n]}, t_{[1:n]}) \ .$$

where $h_i$ is again $\langle t_{i-1}, t_{i-2}, w_{[1:n]}, i\rangle$. Note that this is almost identical to Eq. 2, but without the local normalization terms $\log Z(h_i, \bar{\alpha})$. Under this method for assigning scores to tagged sequences, the highest scoring sequence of tags for an input sentence can be found using the Viterbi algorithm. (We can use an almost identical decoding algorithm to that for maximum-entropy taggers, the difference being that local normalization terms do not need to be calculated.)

We then propose the training algorithm in figure 1. The algorithm takes $T$ passes over the training sample. All parameters are initially set to be zero. Each sentence in turn is decoded using the current parameter settings. If the highest scoring sequence under the current model is not correct, the parameters $\alpha_s$ are updated in a simple additive fashion.

Note that if the local features $\phi_s$ are indicator functions, then the global features $\Phi_s$ will be counts. In this case the update will add $c_s - d_s$ to each parameter $\alpha_s$, where $c_s$ is the number of times the $s$'th feature occurred in the correct tag sequence, and $d_s$ is the number of times

**Inputs:** A training set of tagged sentences, $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ for $i = 1 \ldots n$. A parameter $T$ specifying number of iterations over the training set. A "local representation" $\phi$ which is a function that maps history/tag pairs to $d$-dimensional feature vectors. The global representation $\Phi$ is defined through $\phi$ as in Eq. 1.
**Initialization:** Set parameter vector $\bar{\alpha} = 0$.
**Algorithm:**
For $t = 1 \ldots T, i = 1 \ldots n$
• Use the Viterbi algorithm to find the output of the model on the $i$'th training sentence with the current parameter settings, i.e.,

$$z_{[1:n_i]} = \arg\max_{u_{[1:n_i]} \in \mathcal{T}^{n_i}} \sum_s \alpha_s \Phi_s(w^i_{[1:n_i]}, u_{[1:n_i]})$$

where $\mathcal{T}^{n_i}$ is the set of all tag sequences of length $n_i$.
• If $z_{[1:n_i]} \neq t^i_{[1:n_i]}$ then update the parameters

$$\alpha_s = \alpha_s + \Phi_s(w^i_{[1:n_i]}, t^i_{[1:n_i]}) - \Phi_s(w^i_{[1:n_i]}, z_{[1:n_i]})$$

**Output:** Parameter vector $\bar{\alpha}$.

Figure 1: The training algorithm for tagging.

it occurs in highest scoring sequence under the current model. For example, if the features $\phi_s$ are indicator functions tracking all trigrams and word/tag pairs, then the training algorithm is identical to that given in section 2.1.

### 2.5 Averaging Parameters

There is a simple refinement to the algorithm in figure 1, called the "averaged parameters" method. Define $\alpha^{t,i}_s$ to be the value for the $s$'th parameter after the $i$'th training example has been processed in pass $t$ over the training data. Then the "averaged parameters" are defined as $\gamma_s = \sum_{t=1\ldots T, i=1\ldots n} \alpha^{t,i}_s / nT$ for all $s = 1 \ldots d$. It is simple to modify the algorithm to store this additional set of parameters. Experiments in section 4 show that the averaged parameters perform significantly better than the final parameters $\alpha^{T,n}_s$. The theory in the next section gives justification for the averaging method.

## 3 Theory Justifying the Algorithm

In this section we give a general algorithm for problems such as tagging and parsing, and give theorems justifying the algorithm. We also show how the tagging algorithm in figure 1 is a special case of this algorithm. Convergence theorems for the perceptron applied to classification problems appear in (Freund & Schapire 99) – the results in this section, and the proofs in section 5, show how the classification results can be carried over to problems such as tagging.

**Inputs:** Training examples $(x_i, y_i)$
**Initialization:** Set $\bar{\alpha} = 0$
**Algorithm:**
For $t = 1 \ldots T$, $i = 1 \ldots n$
Calculate $z_i = \arg\max_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$
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 2: A variant of the perceptron algorithm.

The task is to learn a mapping from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. For example, $\mathcal{X}$ might be a set of sentences, with $\mathcal{Y}$ being a set of possible tag sequences. We assume:

- Training examples $(x_i, y_i)$ for $i = 1 \ldots n$.
- A function **GEN** which enumerates a set of candidates $\mathbf{GEN}(x)$ for an input $x$.
- A **representation** $\Phi$ mapping each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ to a feature vector $\Phi(x, y) \in \mathbb{R}^d$.
- A **parameter vector** $\bar{\alpha} \in \mathbb{R}^d$.

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

$$F(x) = \arg \max_{y \in \mathbf{GEN}(x)} \Phi(x, y) \cdot \bar{\alpha}$$

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 tagging problem in section 2 can be mapped to this setting as follows:

- The training examples are sentence/tagged-sequence pairs: $x_i = w^i_{[1:n_i]}$ and $y_i = t^i_{[1:n_i]}$ for $i = 1 \ldots n$.
- Given a set of possible tags $\mathcal{T}$, we define $\mathbf{GEN}(w_{[1:n]}) = \mathcal{T}^n$, i.e., the function **GEN** maps an input sentence $w_{[1:n]}$ to the set of all tag sequences of length $n$.
- The representation $\Phi(x, y) = \Phi(w_{[1:n]}, t_{[1:n]})$ is defined through *local* feature vectors $\phi(h, t)$ where $(h, t)$ is a history/tag pair. (See Eq. 1.)

Figure 2 shows an algorithm for setting the weights $\bar{\alpha}$. It can be verified that the training algorithm for taggers in figure 1 is a special case of this algorithm, if we define $(x_i, y_i)$, **GEN** and $\Phi$ as just described.

We will now give a first theorem regarding the convergence of this algorithm. This theorem therefore also describes conditions under which the algorithm in figure 1 converges. First, we need the following definition:

**Definition 1** *Let* $\overline{\mathbf{GEN}}(x_i) = \mathbf{GEN}(x_i) - \{y_i\}$. *In other words* $\overline{\mathbf{GEN}}(x_i)$ *is the set of* incorrect *candidates for an example* $x_i$. *We will say that a training sequence* $(x_i, y_i)$ *for* $i = 1 \ldots n$ *is* **separable with margin** $\delta > 0$ *if there exists some vector* $\mathbf{U}$ *with* $||\mathbf{U}|| = 1$ *such that*

$$\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i), \quad \mathbf{U} \cdot \Phi(x_i, y_i) - \mathbf{U} \cdot \Phi(x_i, z) \geq \delta \quad (3)$$

*(*$||\mathbf{U}||$ *is the 2-norm of* $\mathbf{U}$*, i.e.,* $||\mathbf{U}|| = \sqrt{\sum_s \mathbf{U}_s^2}$*.)*

We can then state the following theorem (see section 5 for a proof):

**Theorem 1** *For any training sequence* $(x_i, y_i)$ *which is separable with margin* $\delta$*, then for the perceptron algorithm in figure 2*

$$\textit{Number of mistakes} \leq \frac{R^2}{\delta^2}$$

*where* $R$ *is a constant such that* $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i)$ $||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R$.

This theorem implies that if there is a parameter vector $\mathbf{U}$ which makes zero errors on the training set, then after a finite number of iterations the training algorithm will have converged to parameter values with zero training error. A crucial point is that the number of mistakes is independent of the number of candidates for each example (i.e. the size of $\mathbf{GEN}(x_i)$ for each $i$), depending only on the separation of the training data, where separation is defined above. This is important because in many NLP problems $\mathbf{GEN}(x)$ can be exponential in the size of the inputs. All of the convergence and generalization results in this paper depend on notions of separability rather than the size of **GEN**.

Two questions come to mind. First, are there guarantees for the algorithm if the training data is not separable? Second, performance on a training sample is all very well, but what does this guarantee about how well the algorithm generalizes to newly drawn test examples? (Freund & Schapire 99) discuss how the theory can be extended to deal with both of these questions. The next sections describe how these results can be applied to the algorithms in this paper.

### 3.1 Theory for inseparable data

In this section we give bounds which apply when the data is not separable. First, we need the following definition:

**Definition 2** *Given a sequence* $(x_i, y_i)$*, for a* $\mathbf{U}$*,* $\delta$ *pair define* $m_i = \mathbf{U} \cdot \Phi(x_i, y_i) - \max_{z \in \overline{\mathbf{GEN}}(x_i)} \mathbf{U} \cdot \Phi(x_i, z)$ *and* $\epsilon_i = \max\{0, \delta - m_i\}$. *Finally, define* $D_{\mathbf{U},\delta} = \sqrt{\sum_{i=1}^{n} \epsilon_i^2}$.

The value $D_{\mathbf{U},\delta}$ is a measure of how close $\mathbf{U}$ is to separating the training data with margin $\delta$. $D_{\mathbf{U},\delta}$ is 0 if the vector $\mathbf{U}$ separates the data with at least margin $\delta$. If $\mathbf{U}$ separates almost all of the examples with margin $\delta$, but a few examples are incorrectly tagged or have margin less than $\delta$, then $D_{\mathbf{U},\delta}$ will take a relatively small value.

The following theorem then applies (see section 5 for a proof):

**Theorem 2** *For any training sequence* $(x_i, y_i)$*, for the first pass over the training set of the perceptron algorithm in figure 2,*

$$\textit{Number of mistakes} \leq \min_{\mathbf{U},\delta} \frac{(R + D_{\mathbf{U},\delta})^2}{\delta^2}$$

*where* $R$ *is a constant such that* $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i)$ $||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R$*, and the min is taken over* $\delta > 0$*,* $||\mathbf{U}|| = 1$.

This theorem implies that if the training data is "close" to being separable with margin $\delta$ – i.e., there exists some $\mathbf{U}$ such that $D_{\mathbf{U},\delta}$ is relatively small – then the algorithm will again make a small number of mistakes. Thus theorem 2 shows that the perceptron algorithm can be robust to some training data examples being difficult or impossible to tag correctly.

### 3.2 Generalization results

Theorems 1 and 2 give results bounding the number of errors on training samples, but the question we are really interested in concerns guarantees of how well the method generalizes to new test examples. Fortunately, there are several theoretical results suggesting that if the perceptron algorithm makes a relatively small number of mistakes on a training sample then it is likely to generalize well to new examples. This section describes some of these results, which originally appeared in (Freund & Schapire 99), and are derived directly from results in (Helmbold and Warmuth 95).

First we define a modification of the perceptron algorithm, the voted perceptron. We can consider the first pass of the perceptron algorithm to build a sequence of parameter settings $\bar{\alpha}^{1,i}$ for $i = 1 \ldots n$. For a given test example $x$, each of these will define an output $v_i = \arg\max_{z \in \mathbf{GEN}(x)} \bar{\alpha}^{1,i} \cdot \Phi(x, z)$. The voted perceptron takes the most frequently occurring output in the set $\{v_1 \ldots v_n\}$. Thus the voted perceptron is a method where each of the parameter settings $\bar{\alpha}^{1,i}$ for $i = 1 \ldots n$ get a single vote for the output, and the majority wins. The averaged algorithm in section 2.5 can be considered to be an approximation of the voted method, with the advantage that a single decoding with the averaged parameters can be performed, rather than $n$ decodings with each of the $n$ parameter settings.

In analyzing the voted perceptron the one assumption we will make is that there is some unknown distribution $P(x, y)$ over the set $\mathcal{X} \times \mathcal{Y}$, and that both training and test examples are drawn independently, identically distributed (i.i.d.) from this distribution. Corollary 1 of (Freund & Schapire 99) then states:

**Theorem 3** *(Freund & Schapire 99) Assume all examples are generated i.i.d. at random. Let $\langle(\mathbf{x}_1, y_1)\rangle \ldots (\mathbf{x}_n, y_n)\rangle$ be a sequence of training examples and let $(\mathbf{x}_{n+1}, y_{n+1})$ be a test example. Then the probability (over the choice of all $n+1$ examples) that the voted-perceptron algorithm does not predict $y_{n+1}$ on input $\mathbf{x}_{n+1}$ is at most*

$$\frac{2}{n+1} E_{n+1}\left[\min_{\mathbf{U},\delta} \frac{(R + D_{\mathbf{U},\delta})^2}{\delta^2}\right]$$

*where $E_{n+1}[]$ is an expected value taken over $n+1$ examples, $R$ and $D_{\mathbf{U},\delta}$ are as defined above, and the min is taken over $\delta > 0$, $||\mathbf{U}|| = 1$.*

# 4 Experiments

## 4.1 Data Sets

We ran experiments on two data sets: part-of-speech tagging on the Penn Wall Street Journal treebank (Marcus et al. 93), and base noun-phrase recognition on the data sets originally introduced by (Ramshaw and Marcus 95). In each case we had a training, development and test set. For part-of-speech tagging the training set was sections 0–18 of the treebank, the development set was sections 19–21 and the final test set was sections 22-24. In NP chunking the training set was taken from section 15–18, the development set was section 21, and the test set was section 20. For POS tagging we report the percentage of correct tags on a test set. For chunking we report F-measure in recovering bracketings corresponding to base NP chunks.

| | | |
|---|---|---|
| Current word | $w_i$ | & $t_i$ |
| Previous word | $w_{i-1}$ | & $t_i$ |
| Word two back | $w_{i-2}$ | & $t_i$ |
| Next word | $w_{i+1}$ | & $t_i$ |
| Word two ahead | $w_{i+2}$ | & $t_i$ |
| Bigram features | $w_{i-2}, w_{i-1}$ | & $t_i$ |
| | $w_{i-1}, w_i$ | & $t_i$ |
| | $w_i, w_{i+1}$ | & $t_i$ |
| | $w_{i+1}, w_{i+2}$ | & $t_i$ |
| Current tag | $p_i$ | & $t_i$ |
| Previous tag | $p_{i-1}$ | & $t_i$ |
| Tag two back | $p_{i-2}$ | & $t_i$ |
| Next tag | $p_{i+1}$ | & $t_i$ |
| Tag two ahead | $p_{i+2}$ | & $t_i$ |
| Bigram tag features | $p_{i-2}, p_{i-1}$ | & $t_i$ |
| | $p_{i-1}, p_i$ | & $t_i$ |
| | $p_i, p_{i+1}$ | & $t_i$ |
| | $p_{i+1}, p_{i+2}$ | & $t_i$ |
| Trigram tag features | $p_{i-2}, p_{i-1}, p_i$ | & $t_i$ |
| | $p_{i-1}, p_i, p_{i+1}$ | & $t_i$ |
| | $p_i, p_{i+1}, p_{i+2}$ | & $t_i$ |

Figure 3: Feature templates used in the NP chunking experiments. $w_i$ is the current word, and $w_1 \ldots w_n$ is the entire sentence. $p_i$ is POS tag for the current word, and $p_1 \ldots p_n$ is the POS sequence for the sentence. $t_i$ is the chunking tag assigned to the $i$'th word.

## 4.2 Features

For POS tagging we used identical features to those of (Ratnaparkhi 96), the only difference being that we did not make the rare word distinction in table 1 of (Ratnaparkhi 96) (i.e., spelling features were included for all words in training data, and the word itself was used as a feature regardless of whether the word was rare). The feature set takes into account the previous tag and previous pairs of tags in the history, as well as the word being tagged, spelling features of the words being tagged, and various features of the words surrounding the word being tagged.

In the chunking experiments the input "sentences" included words as well as parts-of-speech for those words from the tagger in (Brill 95). Table 3 shows the features used in the experiments. The chunking problem is represented as a three-tag task, where the tags are `B`, `I`, `O` for words beginning a chunk, continuing a chunk, and being outside a chunk respectively. All chunks begin with a `B` symbol, regardless of whether the previous word is tagged `O` or `I`.

**NP Chunking Results**

| Method | F-Measure | Numits |
|---|---|---|
| Perc, avg, cc=0 | 93.53 | 13 |
| Perc, noavg, cc=0 | 93.04 | 35 |
| Perc, avg, cc=5 | 93.33 | 9 |
| Perc, noavg, cc=5 | 91.88 | 39 |
| ME, cc=0 | 92.34 | 900 |
| ME, cc=5 | 92.65 | 200 |

**POS Tagging Results**

| Method | Error rate/% | Numits |
|---|---|---|
| Perc, avg, cc=0 | 2.93 | 10 |
| Perc, noavg, cc=0 | 3.68 | 20 |
| Perc, avg, cc=5 | 3.03 | 6 |
| Perc, noavg, cc=5 | 4.04 | 17 |
| ME, cc=0 | 3.4 | 100 |
| ME, cc=5 | 3.28 | 200 |

Figure 4: Results for various methods on the part-of-speech tagging and chunking tasks on development data. All scores are error percentages. Numits is the number of training iterations at which the best score is achieved. Perc is the perceptron algorithm, ME is the maximum entropy method. Avg/noavg is the perceptron with or without averaged parameter vectors. cc=5 means only features occurring 5 times or more in training are included, cc=0 means all features in training are included.

### 4.3 Results

We applied both maximum-entropy models and the perceptron algorithm to the two tagging problems. We tested several variants for each algorithm on the development set, to gain some understanding of how the algorithms' performance varied with various parameter settings, and to allow optimization of free parameters so that the comparison on the final test set is a fair one. For both methods, we tried the algorithms with feature count cut-offs set at 0 and 5 (i.e., we ran experiments with all features in training data included, or with all features occurring 5 times or more included – (Ratnaparkhi 96) uses a count cut-off of 5). In the perceptron algorithm, the number of iterations $T$ over the training set was varied, and the method was tested with both averaged and unaveraged parameter vectors (i.e., with $\alpha_s^{T,n}$ and $\gamma_s^{T,n}$, as defined in section 2.5, for a variety of values for $T$). In the maximum entropy model the number of iterations of training using Generalized Iterative Scaling was varied.

Figure 4 shows results on development data on the two tasks. The trends are fairly clear: averaging improves results significantly for the perceptron method, as does including all features rather than imposing a count cut-off of 5. In contrast, the ME models' performance suffers when all features are included. The best perceptron configuration gives improvements over the maximum-entropy models in both cases: an improvement in F-measure from 92.65% to 93.53% in chunking, and a reduction from 3.28% to 2.93% error rate in POS tagging. In looking at the results for different numbers of iterations on development data we found that averaging not only improves the best result, but also gives much greater stability of the tagger (the non-averaged variant has much greater variance in its scores).

As a final test, the perceptron and ME taggers were applied to the test sets, with the optimal parameter settings on development data. On POS tagging the perceptron algorithm gave 2.89% error compared to 3.28% error for the maximum-entropy model (a 11.9% relative reduction in error). In NP chunking the perceptron algorithm achieves an F-measure of 93.63%, in contrast to an F-measure of 93.29% for the ME model (a 5.1% relative reduction in error).

## 5 Proofs of the Theorems

This section gives proofs of theorems 1 and 2. The proofs are adapted from proofs for the classification case in (Freund & Schapire 99).

**Proof of Theorem 1:** Let $\bar{\alpha}^k$ be the weights before the $k$'th mistake is made. It follows that $\bar{\alpha}^1 = 0$. Suppose the $k$'th mistake is made at the $i$'th example. Take $z$ to the output proposed at this example, $z = \mathrm{argmax}_{y \in \mathbf{GEN}(x_i)} \Phi(x_i, y) \cdot \bar{\alpha}^k$. It follows from the algorithm updates that $\bar{\alpha}^{k+1} = \bar{\alpha}^k + \Phi(x_i, y_i) - \Phi(x_i, z)$. We take inner products of both sides with the vector $\mathbf{U}$:

$$\begin{aligned} \mathbf{U} \cdot \bar{\alpha}^{k+1} &= \mathbf{U} \cdot \bar{\alpha}^k + \mathbf{U} \cdot \Phi(x_i, y_i) - \mathbf{U} \cdot \Phi(x_i, z) \\ &\geq \mathbf{U} \cdot \bar{\alpha}^k + \delta \end{aligned}$$

where the inequality follows because of the property of $\mathbf{U}$ assumed in Eq. 3. Because $\bar{\alpha}^1 = 0$, and therefore $\mathbf{U} \cdot \bar{\alpha}^1 = 0$, it follows by induction on $k$ that for all $k$, $\mathbf{U} \cdot \bar{\alpha}^{k+1} \geq k\delta$. Because $\mathbf{U} \cdot \bar{\alpha}^{k+1} \leq ||\mathbf{U}||\ ||\bar{\alpha}^{k+1}||$, it follows that $||\bar{\alpha}^{k+1}|| \geq k\delta$.

We also derive an upper bound for $||\bar{\alpha}^{k+1}||^2$:

$$\begin{aligned} ||\bar{\alpha}^{k+1}||^2 &= ||\bar{\alpha}^k||^2 + ||\Phi(x_i, y_i) - \Phi(x_i, z)||^2 \\ &\quad +2\bar{\alpha}^k \cdot (\Phi(x_i, y_i) - \Phi(x_i, z)) \\ &\leq ||\bar{\alpha}^k||^2 + R^2 \end{aligned}$$

where the inequality follows because $||\Phi(x_i, y_i) - \Phi(x_i, z)||^2 \leq R^2$ by assumption, and $\bar{\alpha}^k \cdot (\Phi(x_i, y_i) - \Phi(x_i, z)) \leq 0$ because $z$ is the highest scoring candidate for $x_i$ under the parameters $\bar{\alpha}^k$. It follows by induction that $||\bar{\alpha}^{k+1}||^2 \leq kR^2$.

Combining the bounds $||\bar{\alpha}^{k+1}|| \geq k\delta$ and $||\bar{\alpha}^{k+1}||^2 \leq kR^2$ gives the result for all $k$ that

$$k^2\delta^2 \leq ||\bar{\alpha}^{k+1}||^2 \leq kR^2 \quad \Rightarrow \quad k \leq R^2/\delta^2 \quad \blacksquare$$

**Proof of Theorem 2:** We transform the representation $\Phi(x, y) \in \mathbb{R}^d$ to a new representation $\bar{\Phi}(x, y) \in \mathbb{R}^{d+n}$ as follows. For $i = 1 \ldots d$ define $\bar{\Phi}_i(x, y) = \Phi_i(x, y)$. For $j = 1 \ldots n$ define $\bar{\Phi}_{d+j}(x, y) = \Delta$ if $(x, y) = (x_j, y_j)$, 0 otherwise, where $\Delta$ is a parameter which is greater than 0. Similary, say we are given a $\mathbf{U}, \delta$ pair, and corresponding values for $\epsilon_i$ as defined above. We define a modified parameter vector $\bar{\mathbf{U}} \in \mathbb{R}^{d+n}$ with $\bar{\mathbf{U}}_i = \mathbf{U}_i$ for $i = 1 \ldots d$ and $\bar{\mathbf{U}}_{d+j} = \epsilon_j/\Delta$ for $j = 1 \ldots n$. Under these definitions it can be verified that

$$\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i),\ \bar{\mathbf{U}} \cdot \bar{\Phi}(x_i, y_i) - \bar{\mathbf{U}} \cdot \bar{\Phi}(x_i, z) \geq \delta$$
$$\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i),\ ||\bar{\Phi}(x_i, y_i) - \bar{\Phi}(x_i, z)||^2 \leq R^2 + \Delta^2$$
$$||\bar{\mathbf{U}}||^2 = ||\mathbf{U}||^2 + \sum_i \epsilon_i^2/\Delta^2 = 1 + D_{\mathbf{U},\delta}^2/\Delta^2$$

It can be seen that the vector $\bar{\mathbf{U}}/||\bar{\mathbf{U}}||$ separates the data with margin $\delta/\sqrt{1 + D_{\mathbf{U},\delta}^2/\Delta^2}$. By theorem 1, this means that the first pass of the perceptron algorithm with representation $\bar{\Phi}$ makes at most $k_{max}(\Delta) = \frac{1}{\delta^2}(R^2 + \Delta^2)(1 + \frac{D_{\mathbf{U},\delta}^2}{\Delta^2})$ mistakes. But the first pass of the original algorithm with representation $\Phi$ is *identical* to the first pass of the algorithm with representation $\bar{\Phi}$, because the parameter weights for the additional features $\bar{\Phi}_{d+j}$ for $j = 1 \ldots n$ each affect a single example of training data, and do not affect the classification of test data examples. Thus the original perceptron algorithm also makes at most $k_{max}(\Delta)$ mistakes on its first pass over the training data. Finally, we can minimize $k_{max}(\Delta)$ with respect to $\Delta$, giving $\Delta = \sqrt{RD_{\mathbf{U},\delta}}$, and $k_{max}(\sqrt{RD_{\mathbf{U},\delta}}) = (R^2 + D_{\mathbf{U},\delta}^2)/\delta^2$, implying the bound in the theorem. $\blacksquare$

## 6 Conclusions

We have described new algorithms for tagging, whose performance guarantees depend on a notion of "separability" of training data examples. The generic algorithm in figure 2, and the theorems describing its convergence properties, could be applied to several other models in the NLP literature. For example, a weighted context-free grammar can also be conceptualized as a way of defining **GEN**, $\Phi$ and $\bar{\alpha}$, so the weights for generative models such as PCFGs could be trained using this method.


## Acknowledgements

Thanks to Nigel Duffy, Rob Schapire and Yoram Singer for many useful discussions regarding the algorithms in this paper, and to Fernando Pereira for pointers to the NP chunking data set, and for suggestions regarding the features used in the experiments.


## References


Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics.*

Collins, M., and Duffy, N. (2001). Convolution Kernels for Natural Language. In *Proceedings of Neural Information Processing Systems (NIPS 14).*

Collins, M., and Duffy, N. (2002). New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of ACL 2002.*

Collins, M. (2002). Ranking Algorithms for Named–Entity Extraction: Boosting and the Voted Perceptron. In *Proceedings of ACL 2002.*

Freund, Y. & Schapire, R. (1999). Large Margin Classification using the Perceptron Algorithm. In *Machine Learning*, 37(3):277–296.

Helmbold, D., and Warmuth, M. On weak learning. *Journal of Computer and System Sciences*, 50(3):551-573, June 1995.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML 2001.*

McCallum, A., Freitag, D., and Pereira, F. (2000) Maximum entropy markov models for information extraction and segmentation. In *Proceedings of ICML 2000.*

Marcus, M., Santorini, B., & Marcinkiewicz, M. (1993). Building a large annotated corpus of english: The Penn treebank. *Computational Linguistics,* 19.

Ramshaw, L., and Marcus, M. P. (1995). Text Chunking Using Transformation-Based Learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, Association for Computational Linguistics, 1995.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *Proceedings of the empirical methods in natural language processing conference.*

Rosenblatt, F. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65, 386–408. (Reprinted in *Neurocomputing* (MIT Press, 1998).)