Discriminative Online Algorithms for Sequence Labeling - A Comparative Study

Kevin Gimpel and Shay Cohen KGIMPEL,SCOHEN@CS.CMU.EDU
Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract

We describe a natural alternative for training sequence labeling models, based on MIRA (Margin Infused Relaxed Algorithm). In addition, we describe a novel method for performing Viterbi-like decoding. We test MIRA and contrast it with other training algorithms and contrast our decoding algorithm with the vanilla Viterbi algorithm.

1. Introduction

Most sequence labeling models in Natural Language Processing contain a log-linear model in their underlying representation. Commonly-used generative and discriminative models, including Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs), are log-linear models with particular classes of feature templates and parametrized by a vector of weights.

The first important problem with these models is *parameter estimation*, which is usually performed by optimizing an objective function which has a probabilistic interpretation, such as the log-likelihood or conditional log-likelihood of the data. It is also possible to forgo the probabilistic interpretation of the estimation and use other algorithms for learning, such as the Perceptron algorithm, which has shown to perform well in a variety of NLP tasks (Collins, 2002).

The second important problem with sequence labeling models is that of *decoding*. Given an observation sequence, decoding is the process of finding the state sequence which maximizes the dot product of the weight vector and the feature vector from the observation/state sequence pair. The decoding algorithm for a particular subfamily of log-linear models comes in tandem with the features that can be put into the model. Exact inference is NP-hard for arbitrary graphical models, so most work in sequence labeling has focused on models which can be expressed with a linear-chain topology, which allows us to use the Viterbi algorithm in its traditional form to do efficient decoding. For example, when using HMMs, we are limited to emission and transition features which can only be defined on pairs of nodes and must fit a generative story. If we want to move beyond the limitations imposed by the generative nature of HMMs and have access to an enriched set of possibly overlapping features, we can choose Conditional Markov Models (McCallum et al., 2000) and still use the Viterbi algorithm for decoding. Finally, if want to avoid the parameter estimation problem present in CMMs (Lafferty et al., 2001) and the label bias problem that stems from the set of features permitted by the model, we may choose CRFs, which provide an even richer set of features.

In this paper, we describe a natural alternative to the estimation methods of CRFs and the Perceptron algorithm for sequence labeling tasks. The approach is based on the *Margin Infused Relaxed Algorithm* (MIRA) (Crammer & Singer, 2003), a relatively new discriminative learning algorithm which has been successfully applied to tasks in NLP such as dependency parsing (McDonald et al., 2005b; McDonald et al., 2005a). MIRA does not explicitly optimize any function, so there is no probabilistic interpretation to its learning procedure. Therefore, we forgo any probabilistic interpretation of our model and simply use the model to give a *score* to an observation sequence and state sequence. MIRA attempts to find parameters that separate correct label sequences from incorrect label sequences with a particular margin with re-

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

spect to these scores. The margin is affected by a *sequence loss function* which specifies the amount of divergence between two label sequences. We show that MIRA achieves performance exceeding that of a CRF with the same features for a standard sequence labeling task and, perhaps more significantly, that MIRA takes less than one-tenth of the training time of a CRF. Furthermore, MIRA cleanly allows us to direct the learning towards task-specific evaluation criteria by allowing a choice of loss function.

In addition, we present a modification to the Viterbi algorithm which makes use of hypothesized paths constructed during the algorithm's execution. We experiment with the modified algorithm and show that we can obtain performance improvements by handling long-range dependencies in this way.

The rest of the paper is organized as follows. §2 introduces notation and describes MIRA. §3 presents the novel decoding algorithm. We present our experiments in §4 and conclude in §5.

2. Training

2.1. The Model

Let $\mathcal{X}$ be the set of sequences over a finite alphabet Σ_O (the "observations" alphabet) and let $\mathcal{Y}$ be the set of sequences over a finite alphabet Σ_S (the "states" alphabet). In the sequence labeling task, we are interested in learning a mapping $f\colon \mathcal{X} \rightarrow \mathcal{Y}$ such that for any sequence $\mathrm{x} = (x_1, ..., x_n)$, $f(\mathrm{x})$ is a sequence of the same length $\mathrm{y} \in \mathcal{Y}$, $\mathrm{y} = (y_1, ..., y_n)$. The mapping is allowed to choose a state sequence from the *support* of the observation sequence, $\mathrm{G}(\mathrm{x}) \subset \mathcal{Y}$ which will typically be the set of state sequences of the same length as x.[1]

It is common practice to have a log-linear model which defines over the sequences either a joint distribution (such as with HMMs) or a conditional distribution (probability of a state sequence y given the observation sequence x, such as with CRFs). Such log-linear models are defined using a features function $\Phi\colon \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$ and are parametrized using a weight vector $w \in \mathbb{R}^d$. For example, in the case of defining a conditional distribution, we have:

$$p(\mathrm{y} \mid \mathrm{x}) \quad = \quad \frac{e^{\mathrm{w}\cdot\Phi(\mathrm{x},\mathrm{y})}}{Z_{\mathrm{w}}(\mathrm{x})}$$

where $Z_{\mathrm{w}}(\mathrm{x})$ is a normalization constant to make p a distribution that sums the scores over the support of x.

Such a log-linear model induces a natural mapping $f(\mathrm{x})$, which we desire to use to label new sequences from $\mathcal{X}$. This $f(\mathrm{x})$ will select y such that $p(\mathrm{y} \mid \mathrm{x})$ is maximized. Since log is a monotone function, we can maximize $\log p(\mathrm{y} \mid \mathrm{x})$. In fact, we can also ignore the normalization factor $Z_{\mathrm{w}}(\mathrm{x})$ in the maximization process, since it does not depend on the y being maximized. In that case, our f will have the following form:

$$f_{\mathrm{w}}(\mathrm{x}) \quad = \quad \operatorname*{argmax}_{\mathrm{y}\in\mathrm{G}(\mathrm{x})} \mathrm{w} \cdot \Phi(\mathrm{x}, \mathrm{y})$$

2.2. Probabilistic Estimation

Once we have defined the log-linear family of models we work with, the fundamental question to be answered is how to pick the weights for that model according to a set of training data, consisting of observations sequences paired with states sequences. With log-linear models, we usually define an objective function that has a probabilistic interpretation and we find the weight vector that maximizes this objective function. For example, with HMMs, the objective function is the likelihood of the data, yielding a frequency count estimation of the weights in which we count and normalize the occurrences of the transition and emission features. Alternatively, we can use the *conditional* likelihood as an objective function, which is known to have an advantage over regular likelihood in performance (Klein & Manning, 2002). Using conditional likelihood is in fact necessary when the model family, such as CRFs, defines a conditional distribution as opposed to a joint distribution.

2.3. Estimation Using an Online Algorithm

Since in essence we are interested in finding a mapping $f(\mathrm{x})$ that performs well on unseen data, we might as well forgo the underlying probabilistic interpretation, and ensure directly for the training instances $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{n}$ that

[1]The support could be further shrunk to an even smaller subset by using domain knowledge or other kind of constraints that are known in advance.

```
   Input: $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{n}$
1. $\mathrm{w}^{(0)} = 0; \mathrm{v} = 0; i = 0$
2. for $n : 1..N$
3.    for $t : 1..T$
4.       $\mathrm{w}^{(i+1)} =$ update $\mathrm{w}^{(i)}$ according to instance $(\mathrm{x}_t, \mathrm{y}_t)$
5.       $v = v + \mathrm{w}^{(i+1)}$
6.       $i = i + 1$
7. $\mathrm{w} = \frac{\mathrm{v}}{(N*T)}$
```

Figure 1. Generic online learning algorithm.

$$\mathrm{y}_i \quad = \quad \operatorname*{argmax}_{\mathrm{y} \in \mathrm{G}(\mathrm{x})} \mathrm{w} \cdot \Phi(\mathrm{x}_i, \mathrm{y})$$

This is the basic idea in *large-margin* methods (Taskar et al., 2003; Tsochantaridis et al., 2005), in which we seek a weight vector that separates the correct labeling y_i from the rest of the labellings by a certain margin. A common way to obtain such a weight vector is by an online learning algorithm, which goes iteratively through the training instances, each time performing some update to the weight vector according to the current training instance it looks at. The generic online algorithm is given in Fig. 1. One thing to note is that the algorithm returns an *averaged* weight vector, which helps in reducing overfitting (Collins, 2002).[2]

The Perceptron algorithm, which was extended for structured classification by Collins (2002), is an example of such an online algorithm. In the update step, the Perceptron algorithm labels the training instance according to the current weight vector, and adds to the weight vector the difference between the feature vector of the true label sequence and the feature vector of the decoded label sequence.

2.4. Using MIRA for Estimation

In the large-margin framework, we usually have a *loss function*, $L(\mathrm{y}, \mathrm{y}')$, which denotes the cost of labeling a sequence as y' when in fact the real sequence is y. Then, following Crammer and Singer (2001; 2003) we can look for a weight vector according to the following formulation:

$$\begin{array}{ll} \min & ||\mathrm{w}|| \\ \text{s.t.} & \mathrm{w} \cdot (\Phi(\mathrm{x}, \mathrm{y}) - \Phi(\mathrm{x}, \mathrm{y}')) \geq L(\mathrm{y}, \mathrm{y}') \\ & \forall (\mathrm{x}, \mathrm{y}) \in \mathcal{T}, \mathrm{y}' \in \mathrm{G}(\mathrm{x}) \end{array}$$

Intuitively, this formulation looks for a weight vector that creates a margin at least the size of the loss between the real state sequence and each sequence in the support of the observation sequence. The weight vector is constrained to have minimal norm, in order to avoid a blow-up in its size.

The Margin Infused Relaxed Algorithm (Crammer & Singer, 2003) is an online learning algorithm, as described in §2.3, that employs this optimization directly. The update rule it uses at each step is:

$$\begin{array}{ll} \min & ||\mathrm{w}^{(i+1)} - \mathrm{w}^{(i)}|| \\ \text{s.t.} & \mathrm{w}^{(i+1)} \cdot (\Phi(\mathrm{x}_t, \mathrm{y}_t) - \Phi(\mathrm{x}_t, \mathrm{y}')) \geq L(\mathrm{y}_t, \mathrm{y}') \\ & \forall \mathrm{y}' \in \mathrm{G}(\mathrm{x}_t) \end{array}$$

Unfortunately, this formulation is computationally prohibitive because the number of sequences in the set $\mathrm{G}(\mathrm{x}_t)$ is exponential. Following McDonald et al. (2005b), we use *k-Best MIRA*, which assumes that the constraints that matter for the update rule are the constraints with the highest scoring sequences. In this case, the update rule becomes:

$$\begin{array}{ll} \min & ||\mathrm{w}^{(i+1)} - \mathrm{w}^{(i)}|| \\ \text{s.t.} & \mathrm{w}^{(i+1)} \cdot (\Phi(\mathrm{x}_t, \mathrm{y}_t) - \Phi(\mathrm{x}_t, \mathrm{y}')) \geq L(\mathrm{y}_t, \mathrm{y}') \\ & \forall \mathrm{y}' \in best_k(\mathrm{x}_t; \mathrm{w}^{(i)}) \end{array}$$

In Section §4.2.4 we compare the results for different values of k and show that this assumption is met empirically.

A key difference between MIRA and the Perceptron is MIRA's use of a loss function during training. The implicit objective of the Perceptron algorithm is to label the *entire* sequence correctly, while MIRA can make more informed, piecewise decisions based on the loss function. In §4.2.2 we investigate the use of different loss functions and show how the choice of a loss function can affect performance.

[2]When we discuss the Perceptron and MIRA in this paper, we will generally be referring to the versions of these algorithms that return an averaged weight vector at the end of training. In §4.2.1, we show how performance drops when we do not average the weight vector.

3. Decoding Algorithms

3.1. The Viterbi Algorithm

The actual algorithm for decoding (finding the sequence y that maximizes the score $\mathrm{w} \cdot \Phi(\mathrm{x}, \mathrm{y})$) is dependent on our choice of the features function. In the general case, decoding for arbitrary features functions is intractable. To circumvent this computational burden, we usually use features which are local with respect to the state sequence. With HMMs or linear-chain CRFs, for example, this constraint on the features function is apparent in the *transition features* which are dependent only on two states in adjacent positions. This means that our feature function $\Phi(x, y)$ will consist of elements of the form $\Phi_i(x, y_{j-1}, y_j, j)$ where j denotes the position in the sequence. This way, we can employ a dynamic programming algorithm, such as Viterbi, which keeps track of partial sequences with highest scores until it reaches the final state in the sequence. More formally, for a given sequence x, the Viterbi algorithm maintains a two dimensional chart $V_i(s)$ where $s \in \Sigma_S$ and i is a position in the sequence, and computes it dynamically using the following recursion:

$$V_i(s) \quad = \quad \max_{s'} V_{i-1}(s') + \mathrm{w} \cdot \Phi(\mathrm{x}, s', s, i)$$

To compose the output sequence, we select the states which achieved the maximal value at each position, starting from the last position in the sequence.

3.2. Decoding with Hypothesized Partial Sequence

The complexity of the Viterbi algorithm grows exponentially with the number of states we allow to look back at, and is already quadratic (in the size of Σ_S) in the case of a linear-chain. Specifically, in order to consider p states backwards (have features of the form $\Phi(\mathrm{x}, y_{j-p}, ..., y_j, j)$) we would need to maintain a chart of the form $V_i(s_1, ..., s_p)$ where s_p is the furthest state back that we consider. We are interested in using features which look back for more than one position and still preserve the feasible computational complexity of the Viterbi algorithm. To do this, we use features defined on the *hypothesized partial sequence* constructed during execution of the Viterbi algorithm. The idea of using features from the *hypothesized* path also appears in Malouf (2002). The author, who worked on Named Entity Recognition, used a binary feature which fires if a word is part of a personal name in the partial path constructed by the Viterbi algorithm. We generalize the use of these kind of features in a more principled manner. We divided the vector $(s_1, ...s_p)$ in the chart into two parts: $(s_1, ..., s_m)$ for which we do the usual Viterbi maximization and $(s_{m+1}, ..., s_p)$ which are the hypothesized states according to the maximization performed for $(s_1, ..., s_m)$ (these states can be obtained by backtracking). The complexity of decoding in this way grows exponentially as we enlarge m, but grows only linearly as we enlarge p and keep m constant[3].

At first glance, this decoding algorithm is reminiscent of *beam search*, which was used by Ratnaparkhi (1996) for performing part-of-speech (POS) tagging; the value of p can be seen a parameter that controls the beam size, and instead of keeping in the chart p states, we keep only m states and use a heuristic for the rest of the states. However, there is a key difference. Our algorithm considers all states for a certain position in the chart, while with beam search, we may consider the same state several times for a certain position because we prioritize *whole paths*. This means that an unsuccessful tag that happened to get a high probability in an earlier stage will end up being chosen for the final sequence. Our algorithm maintains a more varied selection of tags in each position, avoiding this issue.

3.3. k-Best Decoding

In each update step, MIRA makes use of a k-Best decoder designed for the Viterbi algorithm. A k-Best decoder for the Viterbi algorithm can be naively implemented by keeping at each position in the chart $V_i(s)$ a k-Best list of partial hypotheses. In order to construct the k-Best list for a certain position, all we have to do is to combine the k-Best list from the previous position (which was constructed recursively) with all the states in the current position, and then keep the k-Best list from these combinations for the current position.

Such a naive implementation would slow the com-

[3]Malouf (2002) can be seen as a specific case of our algorithm with $p = \infty$.

plexity of the Viterbi decoder by a factor $O(k^2 \log k)$. Huang and Chiang (2005) present a method, based on search in hypergraphs, to keep the factor at $O(k \log k)$. Still, it is impractical to use large k values. In §4.2.4 we compare the performance of MIRA and its speed as we change k.

4. Experiments

4.1. Experiments with Part-Of-Speech Tagging

While part-of-speech tagging is now a fairly well-worn road, we decided to use it for preliminary experiments to compare MIRA and the Perceptron.

For our experiments, we used the Wall Street Journal data from Penn Treebank III (Marcus et al., 1994). We split the data into training, development and test sets exactly as described in Toutanova et al. (2003). As commonly done, we used sections 0-18 for training, 19-21 for development and 22-24 for testing.

The features we used are given in Table 1. For handling out-of-vocabulary (OOV) symbols, we marked all words in the training set that appeared less than three times as an OOV symbol for which the *isOOV* feature fires. Then, at test time, each symbol that did not appear in the training set had this same *isOOV* feature fire.

Feature name	Feature template
Tag	(t_{i-1}, t_i)
Word	(t_i, w_i)
isOOV	$(t_i, isOOV(w_i))$
isCapitalized	$(t_i, isCapitalized(w_i))$
isFirstLetterCapital	$(t_i, isFirstLetterCapital(w_i))$
isNumber	$(t_i, isNumber(w_i))$
endsInING	$(t_i, endsInING(w_i))$
endsInED	$(t_i, endsInED(w_i))$
endsInS	$(t_i, endsInS(w_i))$
Tag2	(t_{i-2}, t_i)

Table 1. Features used with the POS dataset. w_i denotes the current word, t_i denotes the current tag and t_{i-1} denotes the previous tag. The last feature (Tag2) is used only in the case of $p > 1$.

We run MIRA and the Perceptron both for 30 iterations. MIRA was run with $k = 5$ and the natural accuracy loss function that counts the number of wrong labels in a sequence. As can be seen from Table 2, MIRA performs better than the Perceptron algorithm on both the development set and the test set. MIRA did slightly better on both the development set and the test set when using hypothesized tags as described in §3.2 ($n = 1, p = 2$), leading us to a more thorough investigation of using $p > 1$ in §4.2.3. Fig. 2 shows the performance of MIRA and the Perceptron on the development set as the number of iterations increases. MIRA's performance curve is consistently higher than the Perceptron's, even though it seems to converge to its final result slower.

Split	MIRA		Perceptron
	$m=1, p=1$	$m=1, p=2$	
Development	96.31	96.35	95.33
Test	96.38	96.4	95.04

Table 2. Results on the POS dataset when using MIRA and the Perceptron algorithm on the development set and the test set. m and p are the values when decoding using the hypothesized path (see §3.2). $m = 1, p = 1$ means regular Viterbi decoding.

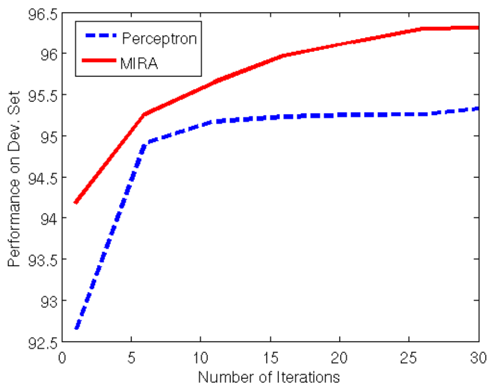

Figure 2. Performance of MIRA and the Perceptron on the development set as the number of iterations changes.

For comparison, the best results known to date on this dataset are given by Toutanova et al. (2003). The authors report an accuracy of 97.24% on the test set using their model. It is important to note that our feature templates are less rich than the feature templates used in Toutanova et al. (2003), because of computational limitations. Our hypothesis is that this is the main reason for our accuracy being lower.

We were unable to compare these results with a CRF using the same set of features because of the training time required. While using MIRA to train on this dataset took approximately 116 hours for 30 iterations, the CRF toolkit[4] did not even complete a single

[4]Available from http://crf.sourceforge.net/. The package uses LBFGS for training.

iteration of its training optimization within 24 hours.[5]

4.2. Experiments with Named-Entity Recognition

For our named-entity recognition (NER) experiments, we used the English dataset from the CoNLL 2003 NER shared task.[6] Table 3 shows statistics of the training set, development set, and test set.

Split	Articles	Sentences	Tokens
Training	946	14,987	203,621
Development	216	3,466	51,362
Test	231	3,684	46,435

Table 3. Statistics of the CoNLL 2003 NER English dataset.

The data consists of Reuters newswire articles which are tokenized and each token is tagged as being either not an entity or one of the following entity types: organization, person, location, or miscellaneous. The standard IOB labeling scheme is used, with begin (B) and inside (I) labels being relatively rare. Each token is also labeled with a hypothesized POS tag and syntactic chunk label produced by the MBT tagger (Daelemans et al., 2002). Performance of the task is evaluated using entity-level precision and recall, which are then combined into an F_1 score. Since precision and recall are computed at the entity level, an incorrect boundary lowers both precision and recall.

In our experiments, we considered variations along the following four dimensions:

- Learning Algorithms - Fixing the feature set, we compare MIRA, the Perceptron, and a CRF. We also try using the unaveraged versions of MIRA and the Perceptron and show how performance of the online algorithms varies as a function of the number of training iterations.
- Loss Functions - In §4.2.2 we consider four different loss functions for use with MIRA. Our goal was to determine whether using a loss function which is more closely related to the evaluation metric could improve performance.
- Values of m and p - When decoding, we were interested to see if increasing the value of p (see §3.2), without necessarily increasing the value of m, improves performance.
- Value of k - When training with MIRA, we tried different values of k to see if the assumption inherent in using k-Best MIRA described in §2.4 is valid for this dataset.

[5]The CRF package limits its optimization to 100 iterations by default, so training would take several weeks on the entire POS dataset. These experiments were run on a 2.2 GHz Sun Opteron with 8GB RAM.

[6]Available from http://cnts.uia.ac.be/conll2003/ner/.

For all experiments, we used the base set of features shown in the upper part of Table 4. The features are fairly standard for NER tasks, and while we made use of the given POS tags, we found that the syntactic chunk labels did not help, which was also reported by Wu et al. (2003). In the decoding experiments in which we increase p, we use the feature template in the second section of Table 4 which adds features for each label in the hypothesized feature history. We also consider features which consider hypothesized labels from the current document, as shown in the final section of the table and similar to those used by Malouf (2002). These features allow repeated entities in a document to have access to earlier instances that were labeled. Our decoding algorithm generalizes this case with $p = \infty$. In the experiments below, we will refer to the use of this latter set of features as "document-level" features.

Feature Templates for $p = 1$ (all include current label)
Label of previous word
{Last, current, next} word
{Last, current, next} POS
Current POS + current word
Last POS + current word
Last POS + current POS + current word
Shape of {last, current, next} word
Shape of last word + current word
{Last, current, next} word starts with a capital letter
{Last, current, next} word is {all caps, all lowercase}
{Last, current} word is a digit
Word is first word in sentence
Word ends in "s"
Word is longer than 4 characters
Word is a {month, day}
3-char prefix/suffix of {last, current, next} word
{Last, current, next} word is a title
Current word is surrounded by {quotation marks, parentheses}
Current word is surrounded by words starting with capitals
Feature Templates for $p > 1$
Label of word w_{i-p} + label of word w_i
Document-level Feature Templates ($p = \infty$)
$\forall p$, if (w_i == w_{i-p}), label of word w_{i-p} + label of word w_i

Table 4. Features used with the NER dataset. i is the position of the current word.

learn alg.	dev. set Precision	dev. set Recall	dev. set F_1	test set Precision	test set Recall	test set F_1
MIRA (acc. loss)	89.46	**88.27**	88.86	82.84	**81.71**	**82.27**
Unavg. MIRA (acc. loss)	89.46	84.26	86.78	**83.29**	77.30	80.18
Perceptron	**90.00**	87.98	**88.98**	83.19	80.97	82.06
Unavg. Perceptron	88.32	85.64	86.96	81.42	77.27	79.29
CRF	88.57	87.38	87.97	82.18	81.16	81.67

Table 5. Algorithm comparison for the NER dataset. Bold denotes best result for that measure according to the algorithm used.

4.2.1. Algorithm Comparison

We were interested in seeing how MIRA compares to CRFs and the Perceptron using the same set of features. We run MIRA and the Perceptron for 40 iterations with $m = p = 1$. MIRA is run with the accuracy loss function and $k = 4$. Table 5 describes the results of this comparison. MIRA and the Perceptron have similar performance, and both are slightly better than the CRF.

Our hypothesis is that CRFs are not as robust as MIRA and the Perceptron to a large number of features unless one employs regularization and feature selection. Specifically, there were a total of 2.3 million unique features instantiated for the NER training set and MIRA and the Perceptron both find relatively sparse solutions in which approximately two million of these features have zero weight. However, none of the features has zero weight after performing CRF training without regularization.

We were also interested to see if the unaveraged versions of MIRA and the Perceptron behave differently than the averaged ones which are supposed to help against overfitting (Collins, 2002). Fig. 3 shows how the algorithms perform when the number of training iterations is varied. The unaveraged Perceptron oscillates throughout training, while the averaged version appears more stable and exceeds the performance of the unaveraged version by a substantial margin. Unaveraged MIRA is more stable than the unaveraged Perceptron, but there is still a wide margin between the unaveraged and averaged versions of MIRA. MIRA finds a higher-scoring weight vector in fewer iterations, but the Perceptron catches up when run for enough iterations.

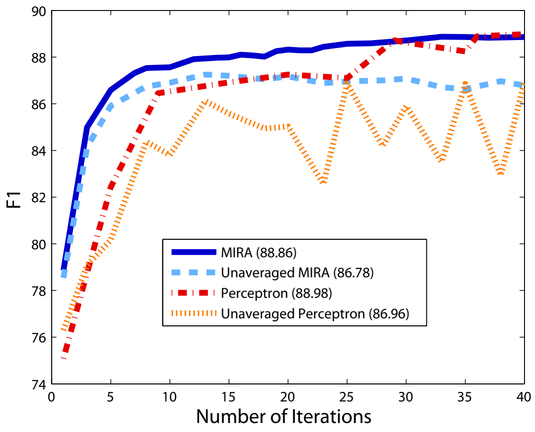

Figure 3. Convergence rates of online algorithms. Results are for dev data in the NER task. The unaveraged versions are not as stable as the averaged versions and do not achieve the same levels of performance. The numbers in parentheses are the F_1 scores after 40 iterations of training.

4.2.2. Loss Functions

In the second setting of our experiments, we considered four label sequence loss functions[7]:

- *Accuracy Loss* - Counts the number of wrong labels in the sequence.
- F_1 *Loss* - Computes the entity-level F_1 score of the labeled sentence, which is the same metric used for evaluation, and returns $\frac{1}{F_1}$ as the loss.
- *P/R Loss* - The F_1 loss indeed makes use of the evaluation metric used to directly optimize performance. However, the F_1 loss is computed *locally* for each sentence ("micro-averaging") while the evaluation is computed for the whole set of sentences ("macro-averaging"). To avoid

[7]For all of our NER experiments, we label one sentence at a time, so each of these loss functions is computed for a pair of labeled sentences.

	loss function	dev. set Precision	dev. set Recall	dev. set F_1	test set Precision	test set Recall	test set F_1
MIRA							
	Accuracy loss	89.45	**88.29**	88.86	**82.84**	81.71	**82.27**
	F_1 loss	88.97	86.77	87.86	82.17	79.90	81.02
	P/R loss ($\alpha = 0.25$)	88.73	88.24	88.48	82.14	**82.19**	82.17
	P/R-weighted acc. loss ($\beta = 1.0$)	89.45	**88.29**	88.86	**82.84**	81.71	**82.27**
MIRA + document-level							
	Accuracy loss	**90.58**	89.28	89.92	**84.03**	83.20	**83.61**
	F_1 loss	89.93	87.93	88.92	83.09	81.36	82.22
	P/R loss ($\alpha = 0.5$)	89.79	88.05	88.91	83.18	81.53	82.35
	P/R-weighted acc. loss ($\beta = 1.4$)	90.27	**89.78**	**90.03**	83.38	**83.75**	83.56

Table 6. Comparison of using different sequence loss functions with MIRA with two feature sets: (1) the base set of features in the upper part of Table 4, and (2) the base set along with the "document-level" features from the lower part of Table 4, where $p = \infty$. Boldface denotes best result for that column. The α and β parameters for the P/R loss and P/R-weighted accuracy loss, respectively, were optimized on the development set.

the myopic decisions made while training, we used a measure which combines precision and recall without normalization (so that a sentence is penalized according to its length). Let G be the entities in the real sequence and let T be the entities which are hypothesized during decoding. We define the "recall complement" to be $|G \setminus (T \cap G)|$ and the "precision complement" to be $|T \setminus (T \cap G)|$. Given a value α between 0 and 1, the P/R loss is the weighted average of the recall complement and the precision complement: $\alpha|T \setminus (T \cap G)| + (1 - \alpha)|G \setminus (T \cap G)|$.

This loss function is more fine-grained than the F_1 loss function defined above, and also allows us to trade off between precision and recall according to the value of α, as we will show below.

- *P/R-Weighted Accuracy Loss* - The previous two loss functions take precision and recall into account at the entity level, which is also how the CoNLL task is evaluated. However, this means that they do not give any information about partially-correct entity labellings. The accuracy loss function, on the other hand, is more fine-grained. The final loss function we consider is similar to the accuracy loss function except that it allows us to assign different weights to the different types of label errors and allows us to effectively trade off precision and recall in a different way than the P/R loss function. There are three types of labeling errors that can be made: (1) labeling a non-entity as part of an entity (*precision errors*), (2) labeling part of a true entity as a non-entity (*recall errors*), and (3) labeling an entity as the wrong type. Where we denote these three types of errors as A, B, and C, respectively, the loss function is $\alpha A + \beta B + \gamma C$. The accuracy loss function is simply a special case of this loss function which simply assigns $\alpha = \beta = \gamma = 1$. In our experiments, we keep α and γ fixed at 1 and vary β to boost recall at the expense of precision.

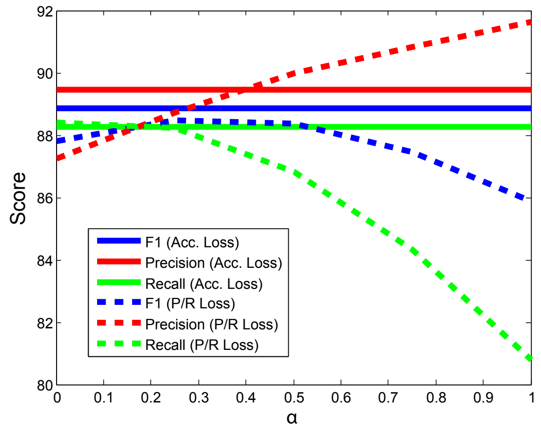

Figure 4. Trading off precision and recall via the P/R loss function. Results are for dev data in the NER task and α is the relative weight of the precision term.

We ran MIRA with the base set of features and also with the document-level features from Table 4 with each of these loss functions and show the results in Table 6. All results were obtained by running MIRA for 40 iterations with $k = 4$, $m = p = 1$. For the parametrized loss functions, namely the P/R loss and the P/R-weighted accuracy loss, we experimented with a range of values on the development data and used the highest-scoring value to test on the test data. Despite our attempts to find a loss function appropriate for the F_1 evaluation metric that would result in a higher F_1 score, none of the loss functions we tried performed better than the simple accuracy loss on test data.[8] We believe that loss functions that consider recall and precision at the entity level (such as the F_1 loss and the P/R loss) do not perform as well because they give less information about how well the data is labeled. The more fine-grained information in the two accuracy-based loss functions seems to be more beneficial for MIRA. The differences in performance with the *accuracy* loss functions are minor. This complements the conclusions in Altun et al. (2003), who compared several different objective functions with different loss functions for training CRFs. Their conclusion is that the influence of loss functions on performance is not as significant as the influence of the set of features used in the model.

Despite the relative lack of success of complex loss functions in improving performance, they can be useful for trading off between precision and recall. In Fig. 4, we show how precision, recall, and F_1 vary as we change α for the P/R loss function. The best performing α is smaller than 0.5 because we tend to obtain higher precision than recall on this task.

4.2.3. Experiments with m and p

In the third setting of our experiments, we checked the decoding algorithm described in 3.2. The results for varying m and p in the Viterbi algorithm are shown in Table 7. We see that performance does not increase when we simply increase p and add in all features that consider labels in the hypothesized sequence. This is not surprising, because even when we use $m = 2, p = 2$ with exact decoding, we do not get an improvement, suggesting that the features used with larger m (or p) are not helpful. However, when we use the features that fire for labels of repeated words within a document ($p = \infty$), we see a performance improvement for MIRA. This shows that we can achieve gains in performance by choosing the right features for $p > 1$. For named-entity recognition tasks, this simple set of features which consider repeated entities in documents were the only features found to be helpful, but for other tasks a variety of these long-range features may be useful, with very little increase in computation time.

We can see that for MIRA, as we increase p beyond 2, the performance improves, and reaches its peak for $p = \infty$. On the other hand, the Perceptron's performance is not as stable and increasing p seems to have an adverse effect. We believe that this happens because of the dynamic nature of the features as p increases. The features may change their values at training time for each iteration, and this may make the Perceptron algorithm less stable, while MIRA is more robust to these variable features.

4.2.4. Experiments with k

Table 8 shows F_1 scores on dev data and training time required for k-best MIRA with different values of k, and compares with the Perceptron and CRF using the same features. MIRA and the Perceptron were each run for 40 iterations. As can be seen, there is not a great deal of improvement in F_1 score beyond $k = 2$, suggesting that the score function is peaked around the best label sequence and quickly trails off around it. Performance continues to increase until $k = 10$, but then stabilizes and even decreases for larger values of k. We believe that when using large k, MIRA may be overfitting by ensuring that even unlikely sequences are separated from the correct sequence according to their loss, whereas in actuality there may be little to no information to get from the kth-best label sequence as k grows large.

Table 8 also contains a comparison among training times for the different algorithms. The training time for the CRF was almost twenty times that of the Perceptron, and thirteen times that of MIRA with $k = 4$. The shorter training time of online learning algorithms can be extremely helpful for certain circumstances. Feature engineering and many feature selection algorithms require continual training and testing of the model, and a reduction in training time of an order of magnitude or more can allow more iterations of fea-

[8]We did, however, achieve an improvement on the development data by using the P/R-weighted accuracy loss and tuning the β parameter to weight recall more highly.

learn alg.	$m=1$ $p=1$	$m=1$ $p=2$	$m=1$ $p=3$	$m=1$ $p=4$	$m=1$ $p=5$	$m=1$ $p=\infty$	$m=1, p=\infty$ doc-level feats only	$m=2$ $p=2$
MIRA w/ acc. loss (dev)	88.86	88.44	88.61	88.64	88.71	89.50	89.92	88.79
Perceptron (dev)	88.98	88.74	88.33	88.26	88.44	88.07	88.46	88.98

Table 7. F_1 scores obtained when varying m and p (on development set).

	MIRA								Perceptron	CRF
	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=10$	$k=15$	$k=20$		
F_1 Score (dev)	88.48	88.85	88.84	88.86	88.86	88.96	88.90	88.95	88.98	87.97
Training Time (minutes)	176	190	209	227	229	350	431	529	166	3050

Table 8. Results on the development data for different values of k with the accuracy loss function. Times are given in minutes.

ture set refinement and, therefore, better results. Also, for certain large datasets, the training of a CRF can be prohibitively expensive, as we saw with POS-tagging.

5. Conclusion

We have shown that discriminative online learning algorithms such as MIRA and the Perceptron can achieve performance better than a CRF for sequence labeling tasks in a small fraction of the training time. These algorithms can be extremely useful for feature engineering and rapid prototyping of systems, in addition to being essential for learning on large datasets.

We have also described a novel Viterbi-like decoding algorithm which makes use of hypothesized states when decoding. We have shown that we can improve performance with long-range features using this algorithm but that naively adding features can have mixed results. We believe this to be the case because these features are dynamic, meaning that they change according to the model parameters, while regular features are static and do not depend on the model itself. Nonetheless, with the right set of features, this approach to incorporating long-range dependencies can be useful for many sequence labeling tasks because of its efficiency.

References

Altun, Y., Johnson, M., & Hofmann, T. (2003). Loss functions and optimization methods for discriminative learning of label sequences. *Proc. of EMNLP*.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. *In Proceedings of EMNLP*.

Crammer, K., & Singer, Y. (2001). On the algorithmic implementation of multiclass kernel based vector machines. *Journal of Machine Learning Research*.

Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*.

Daelemans, W., Zavrel, J., van der Sloot, K., & van den Bosch, A. (2002). *Mbt: Memory-based tagger, version 1.0, reference guide* (Technical Report ILK-0209). University of Tilburg.

Huang, L., & Chiang, D. (2005). Better k-best parsing. *International Workshop on Parsing Technologies*.

Klein, D., & Manning, C. D. (2002). Conditional structure vs. conditional estimation in NLP models. *Proc. of EMNLP*.

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

Malouf, R. (2002). Markov models for language-independent named entity recognition. *In Proc. CoNLL*.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1994). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, *19*, 313–330.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proc. of ICML*.

McDonald, R., Crammer, K., & Pereira, F. (2005a). Online large-margin training of dependency parsers. *Proc. of ACL*.

McDonald, R., Pereira, F., Ribarov, K., & Hajič, J. (2005b). Non-projective dependency parsing using spanning tree algorithms. *Proc. of HLT-EMNLP*.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (pp. 133–142). Somerset, New Jersey: Association for Computational Linguistics.

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-margin markov networks. *In Proc. NIPS*.

Toutanova, K., Klein, D., & Manning, C. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. *In Proceedings of HLT-NAACL'03.*.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *JMLR*, *6*, 1453–1484.

Wu, D., Ngai, G., & Carpuat, M. (2003). A stacked, voted, stacked model for named entity recognition. *Proceedings of CoNLL-2003* (pp. 200–203). Edmonton, Canada.