Learning as Search Optimization: Approximate Large Margin Methods for Structured Prediction

Hal Daumé III HDAUME@ISI.EDU
Daniel Marcu MARCU@ISI.EDU
Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292 USA

Abstract

Mappings to structured output spaces (strings, trees, partitions, etc.) are typically learned using extensions of classification algorithms to simple graphical structures (eg., linear chains) in which search and parameter estimation can be performed exactly. Unfortunately, in many complex problems, it is rare that exact search or parameter estimation is tractable. Instead of learning exact models and searching via heuristic means, we embrace this difficulty and treat the structured output problem in terms of approximate search. We present a framework for learning as search optimization, and two parameter updates with convergence theorems and bounds. Empirical evidence shows that our integrated approach to learning and decoding can outperform exact models at smaller computational cost.

1. Introduction

Many general techniques for learning and decoding with structured outputs are computationally demanding, are ill-suited for dealing with large data sets, and employ parameter optimization for an intractable search (decoding) problem. In some instances, such as syntactic parsing, efficient task-specific decoding algorithms have been developed, but, unfortunately, these are rarely applicable outside of one specific task.

Rather than separating the learning problem from the decoding problem, we propose to consider these two aspects in an integrated manner. By doing so, we are able to learn model parameters appropriate for the search procedure, avoiding the need to heuristically combine an *a priori* unrelated learning technique and search algorithm. After phrasing the learning problem in terms of search, we present two online parameter update methods: a simple perceptron-style update and an approximate large margin update. We apply our model to two tasks: a simple syntactic chunking task for which exact search *is* possible (to allow for comparison to exact learning and decoding methods) and a joint tagging/chunking task for which exact search is intractable.

Appearing in *Proceedings of the 22^{st} International Conference on Machine Learning*, Bonn, Germany, 2005.

2. Previous Work

Most previous work on the structured outputs problem extends standard classifiers to linear chains. Among these are maximum entropy Markov models and conditional random fields (McCallum et al., 2000; Lafferty et al., 2001); case-factor diagrams (McAllester et al., 2004); sequential Gaussian process models (Altun et al., 2004); support vector machines for structured outputs (Tsochantaridis et al., 2004) and max-margin Markov models (Taskar et al., 2003); and kernel dependency estimation models (Weston et al., 2002). These models learn distributions or weights on simple graphs (typically linear chains). Probabilistic models are optimized by gradient descent on the log likelihood, which requires computable expectations of features across the structure. Margin-based techniques are optimized by solving a quadratic program (QP) whose constraints specify that the best structure must be weighted higher than all other structures. Linear chain assumptions can reduce the exponentially-many constraints to a polynomial, but training remains computationally expensive.

Recent effort to reduce this computational demand considers employing constraints that the correct output only outweigh the k-best model hypotheses (Bartlett et al., 2004). Alternatively an online algorithm for which only very small QPs are solved is also possible (McDonald et al., 2004).

At the heart of all these algorithms, batch or online, likelihood- or margin-based, is the computation:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} f(x, y; w) \tag{1}$$

This seemingly innocuous statement is necessary in *all* models, and "simply" computes the structure $\hat{y}$ from the set of all possible structures $\mathcal{Y}$ that maximizes some function f on an input x, parametrized by a weight vector w. This computation is typically left unspecified, since it is "problem specific."

Unfortunately, this $\arg\max$ computation is, in real problems with complex graphical structure, often intractable. Compounding this issue is that this best guess $\hat{y}$ is only one ingredient to the learning algorithms: likelihood-based models require feature expectations and the margin-based methods require either a k-best list of best y or a marginal distribution across the graphical structure. One alternative that alleviates some of these issues is to use a perceptron algorithm, where *only* the $\arg\max$ is required (Collins, 2002), but performance can be adversely affected by the fact that even the $\arg\max$ cannot be computed exactly; see (McCallum & Wellner, 2004) for example.

3. Search Optimization

We present the *Learning as Search Optimization (LaSO)* framework for predicting structured outputs. The idea is to delve into Eq (1) to first reduce the requirement that an algorithm need to compute an $\arg\max$, and also to produce generic algorithms that can be applied to problems that are significantly more complex that the standard sequence labeling tasks that the majority of prior work has focused on.

3.1. Search

The generic *search* problem is covered in great depth in any introductory AI book. Its importance stems from the intractability of computing the "best" solution to many problems; instead, one must search for a "good" solution. Most AI texts contain a definition of the search problem and a general search algorithm; we work here with that from Russell and Norvig (1995). A *search problem* is a structure containing four fields: STATES (the world of exploration), OPERATORS (transitions in the world), GOAL TEST (a subset of states) and PATH COST (computes the cost of a path).

One defines a general search algorithm given a search problem, an initial state and a "queuing function." The search algorithm will either fail (if it cannot find a goal state) or will return a path. Such an algorithm (Figure 1) operates by cycling through a queue, taking the first element off, testing it as a goal and expanding it according to operators if otherwise. Each node stores the path taken to get there and the cost of this path. The *enqueue* function places the expanded nodes, *next*, onto the queue according to some variable ordering that can yield depth-first, breadth-first, greedy, beam, hill-climbing, and A* search (among others). Since most search techniques can be described in this framework, we will treat it as fixed.

```
Algo Search(problem, initial, enqueue)
nodes ← MakeQueue(MakeNode(problem,initial))
while nodes is not empty do
  node ← RemoveFront(nodes)
  if GoalTest(node) then return node
  next ← Operators(node)
  nodes ← enqueue(problem, nodes, next)
end while
return failure
```

Figure 1. The generic search algorithm.

3.2. Search Parameterization

Given the search framework described, for a given task the search problem will be fixed, the initial state will be fixed and the generic search algorithm will be fixed. The only place left, therefore, for parameterization is in the *enqueue* function, whose job it is to essentially rank hypotheses on a queue. The goal of learning, therefore, is to produce an *enqueue* function that places good hypotheses high on the queue and bad hypotheses low on the queue. In the case of optimal search, this means that we will find the optimal solution quickly; in the case of approximate search (with which we are most interested), this is the difference between finding a good solution or not.

In our model, we will assume that the enqueue function is based on two components: a path component g and a heuristic component h, and that the score of a node will be given by $g + h$. This formulation includes A* search when h is an admissible heuristic, heuristic search when h is inadmissible, best-first search when h is identically zero, and any variety of beam search when a queue is cut off at a particular point at each iteration. We will assume h is given and that g is a linear function of features of the input x and the path to and including the current node, n: $g = \boldsymbol{w}^\top \boldsymbol{\Phi}(x, n)$, where $\boldsymbol{\Phi}(\cdot, \cdot)$ is the vector of features.

3.3. Learning the Search Parameters

The supervised learning problem in this search-based framework is to take a search problem, a heuristic function, and training data with the goal of producing a good weight vector $\boldsymbol{w}$ for the path function g. As in standard structured output learning, we will assume that our training data consists of N-many pairs $(x^{(n)}, y^{(n)}) \in \mathcal{X} \times \mathcal{Y}$ that tell us for a given input $x^{(n)}$ what is the correct structured output $y^{(n)}$. We will

```
Algo Learn(problem, initial, enqueue, w, x, y)
nodes ← MakeQueue(MakeNode(problem,initial))
while nodes is not empty do
  node ← RemoveFront(nodes)
  if none of nodes ∪ {node} is y-good or
     GoalTest(node) and node is not y-good then
    sibs ← siblings(node, y)
    w ← update(w, x, sibs, node ∪ nodes)
    nodes ← MakeQueue(sibs)
  else
    if GoalTest(node) then return w
    next ← Operators(node)
    nodes ← enqueue(problem, nodes, next, w)
  end if
end while
```

Figure 2. The generic search/learning algorithm.

make one more important *monotonicity assumption*: for any given node $s \in \mathcal{S}$ and an output $y \in \mathcal{Y}$, we can tell whether s can or cannot lead to y. In the case that s can lead to y, we refer to s as "y-good."[1]

The learning problem can thus be formulated as follows: we wish to find a weight vector $\boldsymbol{w}$ such that: (1) the first goal state dequeued is y-good and (2) the queue always contains at least one y-good state. In this framework, we explore an online learning scenario, where learning is tightly entwined with the search procedure. From a pragmatic perspective, this makes sense: it is useless to the model to learn parameters for cases that it will never actually encounter. We propose a learning algorithm of the form shown in Figure 2. In this algorithm, we write *siblings*(*node*, y) to denote the set of y-good siblings of this node. This can be calculated recursively by back-tracing to the first y-good ancestor and then tracing forward through only y-good nodes to the same search depth as n (in tasks where there is a unique y-good search path – which is common – the sibling of a node is simply the appropriate initial segment of this path).

There are two changes to the search algorithm to facilitate learning (comparing Figure 1 and Figure 2). The first change is that whenever we make an error (a non y-good goal node is dequeued or none of the queue is y-good), we update the weight vector $\boldsymbol{w}$. Secondly, when an error is made, instead of continuing along this bad search path, we instead clear the queue and insert all the *correct* moves we could have made.[2]

Note that this algorithm cannot fail (in the sense that it will always find a goal state). Aiming at a contradiction, suppose it were to fail; this would mean that *nodes* would have become empty. Since "Operators" will never return an empty set, this means that *sibs* must have been empty. But since a node that is inserted into the queue is either itself good or has an ancestor that is good, so could never have become empty. (There may be a complication with cyclic search spaces – in this case, both algorithms need to be augmented with some memory to avoid such loops, as is standard.)

3.4. Parameter Updates

We propose two methods for updating the model parameters. To facilitate discussion, we will refer to a problem as *linearly separable* if there exists a weight vector $\boldsymbol{w}$ with $||\boldsymbol{w}||_2 \leq 1$ such that the search algorithm parameterized by $\boldsymbol{w}$ (a) will not fail and (b) will return an optimal solution. Note that with this definition, linear separability is a joint property of the problem *and* the search algorithm: what is separable with exact search may not be separable with a heuristic search. In the case of linearly separable data, we define the *margin* as the maximal γ such that the data remain separable when all y-good states are down-weighted by γ. In other words, γ is the minimum over all decisions of $\max_{g,b} |\boldsymbol{w}^\top \boldsymbol{\Phi}(x, g) - \boldsymbol{w}^\top \boldsymbol{\Phi}(x, b)|$, where g is a y-good node and b is a y-bad node.

Perceptron Updates. A simple perceptron-style update rule (Rosenblatt, 1958), given $(\boldsymbol{w}, x, sibs, nodes)$ is $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\Delta}$, where:

$$\boldsymbol{\Delta} = \sum_{n \in sibs} \frac{\boldsymbol{\Phi}(x, n)}{|sibs|} - \sum_{n \in nodes} \frac{\boldsymbol{\Phi}(x, n)}{|nodes|} \tag{2}$$

When an update is made, the feature vector for the incorrect decisions are subtracted off, and the feature vectors for all possible correct decisions are added. Whenever $|sibs| = |nodes| = 1$, this looks exactly like the standard perceptron update. When there is only one sibling but many nodes, this resembles the gradi-

[1] We assume that the *loss* we optimize is monotonic on a path in $\mathcal{S}$; in this paper, we only use 0/1 loss.

[2] Performing parameter optimization within search resembles reinforcement learning without the confounding factor of "exploration." Early research in reinforcement learning focused on arbitrary input/output mappings (Farley & Clark, 1954), though this was not framed as search. Later, *associative RL* was introduced, where a *context input* (akin to our input x) was given to a RL algorithm (Barto et al., 1981; Barto & Anandan, 1985). Similar approaches attempt to predict value functions for generalization using techniques such as temporal difference (TD) or Q-learning (Bellman et al., 1963; Boyan & Moore, 1996; Sutton, 1996). More recently, Zhang and Dietterich (1997) applied RL techniques to solving combinatorial scheduling problems, but again focus on the standard TD(λ) framework. These frameworks, however, are not explicitly tailored for supervised learning and without the aid of our monotonicity assumption it is difficult to establish convergence and generalization proofs. Despite these differences, our search optimization framework clearly lies on the border between supervised learning and reinforcement learning, and further investigation may reveal interesting connections.

ent of the log likelihood for conditional models after approximating the "log sum exp" with Jensen's inequality to turn it into a simple sum. When there is more than one correct next hypothesis, this update rule resembles that used in multi-label or ranking variants of the perceptron (Crammer & Singer, 2003). In that work, different "weighting" schemes are proposed, including, for instance, one that weights the nodes in the sums proportional to the loss suffered; such schemes are also possible in our framework, but space prohibits a discussion of them here. Based on this update, we can prove the following theorem:

Theorem 1 *For any training sequence that is separable with by a margin of size γ, using the perceptron update rule the number of errors made during training is bounded above by R^2/γ^2, where, R is a constant such that for all training instances (x, y), for all nodes n in the path to y and all successors m of n (good or otherwise), $||\mathbf{\Phi}(x, n) - \mathbf{\Phi}(x, m)||_2 \le R$.*

In the case of inseparable data, we follow Freund and Shapire (1999) and define $D_{\boldsymbol{w},\gamma}$ as the least obtainable error with weights $\boldsymbol{w}$ and margin γ over the training data: $D_{\boldsymbol{w},\gamma} = [\sum_s (\max\{0, \gamma - p_s\})^2]^{1/2}$, where the sum is over all states leading to a solution and p_s is the empirical margin between the correct state and the hypothesized state s. Using this notation, we obtain two corollaries (proofs are direct adaptations of Freund and Shapire (1999) and Collins (2002)):

Corollary 2 *For* any *training sequence, the number of mistakes made by the training algorithm is bounded above by $\min_{\boldsymbol{w},\gamma} (R + D_{\boldsymbol{w},\gamma})^2/\gamma^2$, where R is as before.*

Corollary 3 *For any i.i.d. training sequence of length n and any test example $(\tilde{x}, \tilde{y})$, the probability of error on the test example is bounded above by $(2/(n+1))\mathcal{E}\{\min_{\boldsymbol{w},\gamma}(R + D_{\boldsymbol{w},\gamma})^2/\gamma^2\}$, where the expectation is taken over all $n + 1$ data points.*

Approximate Large Margin Updates. One major disadvantage to the perceptron algorithm is that it only updates the weights when errors are made. This can lead to a brittle estimate of the parameters, in which the "good" states are weighted only minimally better than the "bad" states. We would like to enforce a large margin between the good states and bad states, but would like to do so without adding significant computational complexity to the problem. In the case of binary classification, Gentile (2001) has presented an online, approximate large margin algorithm that trains similarly to a perceptron called ALMA. The primary difference (aside from a step size on the weight updates) is that updates are made if either (a) the algorithm makes a mistake or (b) *the algorithm is close to making a mistake.* Here, we adapt this algorithm to structured outputs in our framework.

Our algorithm, like ALMA, has four parameters: α, B, C, p. α determines the degree of approximation required: for $\alpha = 1$, the algorithm seeks the true maximal margin solution, for $\alpha < 1$, it seeks one within α of the maximal. B and C can be seen as tuning parameters, but a default setting of $B = 1/\alpha$ and $C = \sqrt{2}$ is reasonable (see Theorem 4 below). We measure the instance vectors with norm p and the weight vector with its dual value q (where $1/p + 1/q = 1$). We use $p = q = 2$, but large p produces sparser solutions, since the weight norm will approach 1. The update is:

$$\boldsymbol{w} \leftarrow \wp(\boldsymbol{w} + Ck^{-1/2}\ \wp(\Delta)) \tag{3}$$

Here, k is the "generation" number of the weight vector (initially 1 and incremented at every update) and $\wp(\boldsymbol{u})$ is the projection of $\boldsymbol{u}$ into the l_2 unit sphere: $\boldsymbol{u}/\max\{1, ||\boldsymbol{u}||_2\}$. One final change to the algorithm is to down-weight the score of all y-good nodes by $(1-\alpha)Bk^{-1/2}$. Thus, a good node will only survive if it is good by a large margin. This setup gives rise to a bound on the number of updates made (proof sketched in Appendix A) and two corollaries (proofs are nearly identical to Theorem 4 and (Gentile, 2001)):

Theorem 4 *For any training sequence that is separable with by a margin of size γ using the approximate large margin update rule with parameters $\alpha, B = \sqrt{8}/\alpha, C = \sqrt{2}$, the number of errors made during training is bounded above by $\frac{2}{\gamma^2}\left(\frac{2}{\alpha} - 1\right)^2 + \frac{8}{\alpha} - 4$.*

Corollary 5 *Suppose for a given α, B and C are such that $C^2 + 2(1-\alpha)BC = 1$; letting $\rho = (C\gamma)^{-2}$, the number of corrections made is bounded above by:*

$$\min_{\boldsymbol{w},\gamma} \frac{1}{\gamma} D_{\boldsymbol{w},\gamma} + \frac{\rho^2}{2} + \rho \left[\frac{\rho^2}{4} + \frac{1}{\gamma} D_{\boldsymbol{w},\gamma} + 1\right]^{1/2} \tag{4}$$

Corollary 6 *For any i.i.d. training sequence of length n and any test example $(\tilde{x}, \tilde{y})$, the probability of error on the test example is bounded above by $(2/(n+1))\mathcal{E}\{\cdot\}$, where $(\cdot)$ is given in Eq (4) and the expectation is taken over all $n + 1$ data points.*

4. Experiments

4.1. Syntactic Chunking

The syntactic chunking problem is a sequence segmentation and labeling problem; for example:

[Great American]$_{\text{NP}}$ [said]$_{\text{VP}}$ [it]$_{\text{NP}}$ [increased]$_{\text{VP}}$ [its loan-loss reserves]$_{\text{NP}}$ [by]$_{\text{PP}}$ [$ 93 million]$_{\text{NP}}$ [after]$_{\text{PP}}$ [reviewing]$_{\text{VP}}$ [its loan portfolio]$_{\text{NP}}$, [raising]$_{\text{VP}}$ [its total loan and real estate reserves]$_{\text{NP}}$ [to]$_{\text{PP}}$ [$ 217 million]$_{\text{NP}}$.

Typical approaches to this problem recast it as a sequence labeling task and then solve it using any of the standard sequence labeling models; see (Sha & Pereira, 2002) for a prototypical example using CRFs. The reduction to sequence labeling is typically done through the "BIO" encoding, where the beginning of an X phrase is tagged B-X, the non-beginning (inside) of an X phrase is tagged I-X and any word not in a phrase is tagged O (outside). More recently, Sarawagi and Cohen (2004) have described a straightforward extension to the CRF (called a Semi-CRF) in which the segmentation and labeling is done directly.

We explore similar models in the context of syntactic chunking, where entire chunks are hypothesized, and no reduction to word-based labels is made. We use the same set of features across all models, separated into "base features" and "meta features." The base features apply to words individually, while meta features apply to entire chunks. The base features we use are: the chunk length, the word (original, lower cased, stemmed, and original-stem), the case pattern of the word, the first and last 1, 2 and 3 characters, and the part of speech and its first character. We additionally consider membership features for lists of names, locations, abbreviations, stop words, etc. The meta features we use are, for any base feature b, b at position i (for any sub-position of the chunk), b before/after the chunk, the entire b-sequence in the chunk, and any 2- or 3-gram tuple of bs in the chunk. We use a first order Markov assumption (chunk label only depends on the most recent previous label) and all features are placed on labels, not on transitions. In this task, the $\arg\max$ computation from Eq (1) *is* tractable; moreover, through a minor adaptation of the standard HMM forward and backward algorithms, we can compute feature expectations, which enable us to do training in a likelihood-based fashion.

Our search space is structured so that each state is the segmentation and labeling of an initial segment of the input string, and an operation extends a state by an entire labeled chunk (of any number of words). For instance, on the example shown at the beginning of this section, the initial hypothesis would be empty; the first correct child would be to hypothesize a chunk of length 2 with the tag NP. The next correct hypothesis would be a chunk of length 1 with tag VP. This process would continue until the end of the sentence is reached. For beam search, we execute search as described, but after every expansion we only retain the b best hypotheses to continue on to the next round.

Our models for this problem are denoted LaSOp_b and LaSOa_b, where b is the size of the beam we use in search, which we vary over $\{1, 5, 25, \infty\}$, where ∞ denotes full, exact Viterbi search and forward-backward updates similar to those used in the semi-CRF. This points out an important issue in our framework: if the graphical structure of the problem *is* amenable to exact search and exact updates, then the framework can accommodate this. In this case, for example, when using exact search, updates are only made at the end of decoding when the highest ranking output is incorrect (after adjusting the weights down for LaSOa), but other than this exception, the sum over the bad nodes in the updates is computed over the entire search lattice and strongly resemble almost identical to those used in the conditional likelihood models for the gradient of the log normalization constant. We always use averaged weights.

We report results on the CoNLL 2000 data set, which includes 8936 training sentences ($212k$ words) and 2012 test sentences ($47k$ words). We compare our proposed models against several baselines. The first baseline is denoted ZDJ02 and is the best system on this task to date (Zhang et al., 2002). The second baseline is the likelihood-trained model, denoted SemiCRF. We use 10% of the training data to tune model parameters. The third baseline is the standard structured perceptron algorithm, denoted Perceptron. For the SemiCRF, this is the prior variance; for the online algorithms, this is the number of iterations to run (for ALMA, $\alpha = 0.9$; changing α in the range $[0.5, 1]$ does not affect the score by more than ± 0.1 in all cases).

The results, in terms of training time, test decoding time, precision, recall and f-score are shown in Table 1. As we can see, the SemiCRF is by far the most computationally expensive algorithm, more than twice as slow to train than even the LaSOp_∞ algorithm. The Perceptron has roughly comparable training time to the exactly trained LaSO algorithms (slightly faster since it only updates for the best solution), but its performance falls short. Moreover, decoding time for the SemiCRF takes a half hour for the two thousand test sentences, while the greedy decoding takes only 52 seconds. It is interesting to note that at the larger beam sizes, the large margin algorithm is actually faster than the perceptron algorithm.

In terms of the quality of the output, the SemiCRF falls short of the previous reported results (92.2 versus 94.1 f-score). Our simplest model, LaSOp_1 already outperforms the SemiCRF with an f-score of 92.4; the large margin variant achieves 93.0. Increasing the beam past 5 does not seem to help with large margin updates, where performance only increases from 94.3 to 94.4 going from a beam of 5 to an infinite beam (at

Table 1. Results on syntactic chunking task; columns are training and testing time (h:m), and precision/recall/f-score on test data.

	Train	Test	Pre	Rec	F
ZDJ02	-	-	94.3	94.0	94.1
SemiCRF	53:56	:31	92.3	92.1	92.2
Perceptron	18:05	:22	93.4	93.5	93.4
LaSOp$_1$	:55	:01	92.5	92.3	92.4
LaSOp$_5$	1:49	:04	93.7	92.6	93.1
LaSOp$_{25}$	6:32	:11	94.2	94.1	94.1
LaSOp$_\infty$	21:43	:24	94.3	94.1	94.2
LaSOa$_1$	:51	:01	93.6	92.5	93.0
LaSOa$_5$	2:04	:04	93.8	94.4	94.3
LaSOa$_{25}$	5:59	:10	93.9	94.6	94.4
LaSOa$_\infty$	20:12	:25	94.0	94.8	94.4

the cost of an extra 18 hours of training time).

4.2. Joint Tagging and Chunking

In Section 4.1, we described an approach to chunking based on search without reduction. This assumed that part of speech tagging had been performed as a preprocessing step. In this section, we discuss models in which part of speech tagging and chunking are performed jointly. This task has previously been used as a benchmark for factorized CRFs (Sutton et al., 2004). In that work, the authors discuss many approximate inference methods to deal with the fact that inference in such joint models is intractable.

For this task, we *do* use the BIO encoding of the chunks so that a more direct comparison to the factorized CRFs would be possible. We use the same features as the last section, together with the regular expressions given by (Sutton et al., 2004) (so that our feature set and their feature set are nearly identical). We do, however, omit their final feature, which is active whenever the part of speech at position i matches the most common part of speech assigned by Brill's tagger to the word at position i in a very large corpus of tagged data. This feature is somewhat unrealistic: the CoNLL data set is a small subset of the Penn Treebank, but the Brill tagger is trained on all of the Treebank. By using this feature, we are, in effect, able to leverage the rest of the Treebank for part of speech tagging. Using just their features without the Brill feature, our performance is quite poor, so we added the lists described in the previous section.

In this problem, states in our search space are again initial taggings of sentences (both part of speech tags and chunk tags), but the operators simply hypothesize the part of speech and chunk tag for the single next word, with the obvious constraint that an I-X tag cannot follow anything but a B-X or I-X tag.

The results are shown in Table 2. The models are

Table 2. Results on joint tagging/chunking task; columns are time to train (h:m), tag accuracy, chunk accuracy, joint accuracy and chunk f-score.

	Train	Test	Tag	Chn	Jnt	F
Sutton	-	-	98.9	97.4	96.5	93.9
LaSOp$_1$	1:29	:01	98.9	95.5	94.7	93.1
LaSOp$_5$	3:24	:04	98.9	95.8	95.1	93.5
LaSOp$_{10}$	4:47	:09	98.9	95.9	95.1	93.5
LaSOp$_{25}$	4:59	:16	98.9	95.9	95.1	93.7
LaSOp$_{50}$	5:53	:30	98.9	95.8	94.9	93.4
LaSOa$_1$	:41	:01	99.0	96.5	95.8	93.9
LaSOa$_5$	1:43	:03	99.0	96.8	96.1	94.2
LaSOa$_{10}$	2:21	:07	99.1	97.3	96.4	94.4
LaSOa$_{25}$	3:38	:20	99.1	97.4	96.6	94.3
LaSOa$_{50}$	3:15	:23	99.1	97.4	96.6	94.4

compared against Sutton, the factorized CRF with tree reparameterization. We do not report on infinite beams, since such a calculation is intractable. We report training time[3], testing time, tag accuracy, chunk accuracy and joint accuracy and f-score for chunking. In this table, we can see that the large margin algorithms are much faster to train than the perceptron (they require fewer iterations to converge – typically two or three compared to seven or eight). In terms of chunking f-score, none of the perceptron-style algorithms is able to out-perform the Sutton model, but our LaSOa algorithms easily outperform it. With a beam of only 1, we achieve the same f-score (93.9) and with a beam of 10 we get an f-score of 94.4. Comparing Table 1 and Table 2, we see that, in general, we can do a better job chunking with the large margin algorithm when we do part of speech tagging simultaneously.

To verity Theorem 4 experimentally, we have run the same experiments using a 1000 sentence ($25k$ word) subset of the training data (so that a positive margin could be found) with a beam of 5. On this data, LaSOa made 19033 corrections, of which 15932 corresponded to true errors (the remaining $3k$ were cases where the correct hypothesis would have been correct, but the down-weighting factor caused it to "lose"). The empirical margin at convergence was $1.299e - 2$; according to Theorem 4, the number of updates should have been ≤ 17724, which is borne out experimentally.

4.3. Effect of Beam Size

Clearly, from the results presented in the preceding sections, the beam size plays an important role in the modeling. In many problems, particularly with generative models, training is done exactly, but decoding is done using an inexact search. In this paper, we

[3] Sutton et al. report a training time of 13.6 hours on 5% of the data (400 sentences); it is unclear from their description how this scales. The scores reported from their model are, however, based on training on the full data set.

Table 3. Effect of beam size on performance; columns are for constant decoding beam; rows are for constant training beam. Numbers are chunk f-score on the joint task.

	1	**5**	**10**	**25**	**50**
1	*93.9*	92.8	91.9	91.3	90.9
5	90.5	*94.3*	*94.4*	94.1	94.1
10	89.6	*94.3*	*94.4*	94.2	94.2
25	88.7	94.2	*94.5*	*94.3*	94.3
50	88.4	94.2	*94.4*	94.2	*94.4*

have suggested that learning and decoding should be done in the *same* search framework, and in this section we briefly support this suggestion with empirical evidence. For our experiments, we use the joint tagging/chunking model from Section 4.2 and experiment by independently varying the beam size for *training* and the beam size for *decoding*. We show these results in Table 3, where the training beam size runs vertically and the decoding beam size runs horizontally; the numbers we report are the chunk f-score.

In these results, we can see that the diagonal (same training beam size as testing beam size) is heavy, indicating that training and testing with the same beam size is useful. This difference is particularly strong when one of the sizes is 1 (i.e., pure greedy search is used). When training with a beam of one, decoding with a beam of 5 drops f-score from 93.9 (which is respectable) to 90.5 (which is poor). Similarly, when a beam of one is used for decoding, training with a beam of 5 drops performance from 93.9 to 92.8. The differences are less pronounced with beams ≥ 10, but the trend is still evident. We believe (without proof) that when the beam size is large enough that the loss incurred due to search errors is at most the loss incurred due to modeling errors, then using a different beam for training and testing is acceptable. However, when some amount of the loss is due to search errors, then a large part of the learning procedure is aimed at learning how to *avoid* search errors, not necessarily modeling the data. It is in these cases that it is important that the beam sizes match.

5. Summary and Discussion

In this paper, we have suggested that one views the learning with structured outputs problem as a search optimization problem and that the same search technique should be applied during both learning and decoding. We have presented two parameter update schemes in the LaSO framework, one perceptron-style and the other based on an approximate large margin scheme, both of which can be modified to work in kernel space or with alternative norms (but not both).

Our framework most closely resembles that used by the incremental parser of Collins and Roark (2004). There are, however, several differences between the two methodologies. Their model builds on standard perceptron-style updates (Collins, 2002) in which a full pass of decoding is done before any updates are made, and thus does not fit into the search optimization framework we have outlined. Collins and Roark found experimentally that stopping the parsing early whenever the correct solution falls out of the beam results in drastically improved performance. However, theyhad little theoretical justification for doing so. These "early updates," however, do strongly resemble our update strategy, with the difference that when Collins and Roark make an error, they stop decoding the current input and move on to the next; on the other hand, when our model makes an error, it continues from the correct solution(s). This choice is justified both theoretically and experimentally. On the tasks reported in this paper, we observe the same phenomenon: early updates are better than no early updates, and the search optimization framework is better than early updates. For instance, in the joint tagging/chunking task from Section 4.2, using a beam of 10, we achieved an f-score of 94.4 in our framework; using only early updates, this drops to 93.1 and using standard perceptron updates, it drops to 92.5.

Our work also bears a resemblance to training *local classifiers* and combining them together with global inference (Punyakanok & Roth, 2001). The primary difference is that when learning local classifiers, one must assume to have access to all possible decisions and must rank them according to some loss function. Alternatively, in our model, one only needs to consider alternatives that are in the queue at any given time, which gives us direct access to those aspects of the search problem that are easily confused. This, in turn, resembles the online large margin algorithms proposed by McDonald et al. (2004), which suffer from the problem that the arg max must be computed exactly. Finally, one can also consider our framework in the context of game theory, where it resembles the iterated gradient ascent technique described by Kearns et al. (2000) and the closely related marginal best response framework (Zinkevich et al., 2005).

We believe that LaSO provides a powerful framework to learn to predict structured outputs. It enables one to build highly effective models of complex tasks efficiently, without worrying about how to normalize a probability distribution, compute expectations, or estimate marginals. It necessarily suffers against probabilistic models in that the output of the classifier will not be a probability; however, in problems with exponential search spaces, normalizing a distribution is quite impractical. In this sense, it compares favorably with the energy-based models proposed by, for example, LeCun and Huang (2005), which also avoid probabilistic normalization, but still require the exact computation of the arg max. We have applied the model to two comparatively trivial tasks: chunking and joint tagging/chunking. Since LaSO is not limited to problems with clean graphical structures, we believe that this framework will be appropriate for many other complex structured learning problems.

Acknowledgments

We thank Michael Collins, Andrew McCallum, Charles Sutton, Thomas Dietterich, Fernando Pereira and Ryan McDonald for enlightening discussions, as well as the anonymous reviewers who gave very helpful comments. This work was supported by DARPA-ITO grant NN66001-00-1-9814 and NSF grant IIS-0326276.

References

Altun, Y., Hofmann, T., & Smola, A. (2004). Gaussian process classification for segmenting and annotating sequences. *ICML*.

Bartlett, P. L., Collins, M., Taskar, B., & McAllester, D. (2004). Exponentiated gradient algorithms for large-margin structured classification. *NIPS*.

Barto, A., & Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Trans. Systems, Man and Cybernetics*.

Barto, A., Sutton, R., & Watkins, C. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*.

Bellman, R., Kalaba, R., & Kotkin, B. (1963). Polynomial approximation – a new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation*

Boyan, J., & Moore, A. W. (1996). Learning evaluation functions for large acyclic domains. *ICML*.

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

Collins, M., & Roark, B. (2004). Incremental parsing with the perceptron algorithm. *ACL*.

Crammer, K., & Singer, Y. (2003). A family of additive online algorithms for category ranking. *JMLR*, *3*.

Farley, B., & Clark, W. (1954). Simulation of self-organizing systems by digital computer. *I.R.E Trans. Information Theory*.

Freund, Y., & Shapire, R. (1999). Large margin classification using the perceptron algorithm. *ML*, *37*.

Gentile, C. (2001). A new approximate maximal margin classification algorithm. *JMLR*, *2*.

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

Kearns, M., Mansour, Y., & Singh, S. (2000). Nash convergence of gradient dynamics in General-Sum games. *UAI*.

LeCun, Y., & Huang, F.J. (2005). Loss functions and discriminitive training of energy-based models. *AI-Stats*.

McAllester, D., Collins, M., & Pereira, F. (2004). Case-factor digrams for structured probabilistic modeling. *UAI*.

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

McCallum, A., & Wellner, B. (2004). Conditional models of identity uncertainty with application to noun coreference. *NIPS*.

McDonald, R., Crammer, K., & Pereira, F. (2004). Large margin online learning algorithms for scalable structured classification. *NIPS Workshop on Structured Outputs*.

Punyakanok, V., & Roth, D. (2001). The use of classifiers in sequential inference. *NIPS*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. New Jersey: Prentice Hall.

Sarawagi, S., & Cohen, W. (2004). Semi-Markov conditional random fields for information extraction. *NIPS*.

Sha, F., & Pereira, F. (2002). Shallow parsing with conditional random fields. *NAACL/HLT*.

Sutton, C., Rohanimanesh, K., & McCallum, A. (2004). Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *ICML*.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *NIPS*.

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

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. *ICML*.

Weston, J., Chapelle, O., Elisseeff, A., Schoelkopf, B., & Vapnik, V. (2002). Kernel dependency estimation. *NIPS*.

Zhang, T., Damerau, F., & Johnson, D. (2002). Text chunking based on a generalization of winnow. *JMLR 2*.

Zhang, W., & Dietterich, T. G. (1997). *Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resouce-constrained scheduling* (Technical Report). Oregon State University.

Zinkevich, M., Riley, P., Bowling, M., & Blum, A. (2005). *Marginal Best Response, Nash Equilibria, and Iterated Gradient Ascent*. In preparation.

Appendix A. Proof of Theorem 4

We follow Gentile (2001), Thm 3, modifying the bound of the normalization factor when projecting $\boldsymbol{w}$; suppose $\boldsymbol{w}$ is the optimal separating hyperplane. Denoting the normalization factor N_k on update k, we find: $N_{k+1}^2 \leq ||\boldsymbol{w}_k + \eta_k \boldsymbol{\Delta}||^2 \leq ||\boldsymbol{w}_k||^2 + \eta_k^2 + 2\eta_k \boldsymbol{w}_k{}^\top \boldsymbol{\Delta} \leq 1 + \eta_k^2 + 2(1-\alpha)\eta_k\gamma$ (γ is the margin) by observing $\boldsymbol{\Delta}$ is bounded above by γ since $\boldsymbol{w}^\top \left[\sum_{n \in sibs} \boldsymbol{\Phi}(x,n)/|sibs| - \sum_{n \in nodes} \boldsymbol{\Phi}(x,n)/|nodes|\right] \leq \boldsymbol{w}^\top \left[\max_{n \in sibs} \boldsymbol{\Phi}(x,n) - \min_{n \in nodes} \boldsymbol{\Phi}(x,n)\right] \leq \gamma$, due to the definition of the margin. N_k is bounded by $1 + (8/\alpha - 6)/k$ to bound number of updates m by $\gamma m \leq (4/\alpha - 2)\sqrt{4/\alpha - 3 + m/2}$. Algebra completes the proof.