Search-Based Structured Prediction as Classification

Hal Daumé III[1], **John Langford**[2] and **Daniel Marcu**[3]
[1,3]Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, 90292
[2]Toyota Technological Institute at Chicago, 1427 East 60th Street, Chicago, IL, 60637
hdaume@isi.edu, jl@tti-c.org, marcu@isi.edu

Abstract

Solutions to computationally hard problems often require that *search* be used. Integrating search into the learning phase has been previously proposed in an ad-hoc manner [2]. In this paper, we show that structured prediction can be mapped into a search setting using language from reinforcement learning, and known techniques for reinforcement learning [7] can give formal performance bounds on the structured prediction task.

1 Introduction

Structured prediction (SP) is the task of learning a function that maps $x \in \mathcal{X}$ to $y \in \mathcal{Y}$, where elements y have structure represented in the *features* and loss function. Typical approaches assume that $\mathcal{Y}$ has a well-behaved structure (eg., linear chain or tree) and the features and loss decompose over substructures (the Markov and context-free assumptions) [6, 8]. However, in many real-world problems, this assumption is invalid, and dynamic programming techniques are not applicable. In such problems, the final prediction must be performed using search; for example, in machine translation [3] or speech recognition [4]. In such cases, we believe it to be appropriate to consider search during learning, since it is worthless to build a model that can score a good $y \in \mathcal{Y}$ high, if such a y cannot be found.

In this paper, we present a reduction from SP to reinforcement learning (RL) that can be composed with known results for RL [7] to give bounds on the performance of a search-based SP method. Moreover, this reduction suggests novel training methods for SP models, yielding efficient learning algorithms that appear to perform well in practice.

2 Definitions

Definition 1. *A* structured prediction *problem is a cost-sensitive classification problem with structure in the prediction space $\mathcal{Y}$; elements $y \in \mathcal{Y}$ decompose into variable-length vectors $(y_1, y_2, \ldots, y_k)$. The problem $\mathcal{D}_{SP}$ is a distribution over inputs $x \in \mathcal{X}$ and cost vectors $\mathbf{c}$, where the length of $\mathbf{c}$ is a variable in 2^k.*

Treating y as a vector is simply a useful encoding. The goal is a function h that maps x to y and minimizes the corresponding cost: $L(\mathcal{D}_{SP}, h) = \mathcal{E}_{(x,\mathbf{c})\sim\mathcal{D}_{SP}}\left\{c_{h(x)}\right\}$. Similarly, the RL problem can be stated generally as in [7].

Definition 2. *A* reinforcement learning *problem $\mathcal{D}_{RL}$ is a conditional probability table $\mathcal{D}_{RL}(o', r \mid (o, a, r)^*)$ on an observation set O and rewards $r \in [0, \infty)$ given any (possibly empty) history $(o, a, r)^*$ of past observations, actions (from an action set A), and rewards.*

The goal of the RL problem is as follows. Given some horizon T, find a policy $\pi : (o, a, r)^* \rightarrow a$, optimizing the expected sum of rewards: $\eta(\mathcal{D}_{RL}, \pi) = \mathcal{E}_{(o,a,r)^T\sim\mathcal{D}_{RL},\pi}\{\sum_{t=1}^{T} r_t\}$. Here, r_t is the tth observed reward, and the expectation is over the process which generates a history using $\mathcal{D}_{RL}$ and choosing actions from π.

3 Reducing Structured Prediction to Reinforcement Learning

Our mapping from SP to RL is as follows. The RL action set A is the space of indexed predictions, so $A^k = \mathcal{Y}$, and $A = \mathcal{Y}_i$. The observation o' is x initially, and the empty set otherwise. The reward r is zero, except at the final iteration when it is the negative loss for the corresponding structured output. Putting this together, we can define a RL problem $\mathcal{D}_{RL}(\mathcal{D}_{SP})$ according to the following rules. When the history is empty, $o' = x$ and $r = 0$, where x is drawn from the marginal $\mathcal{D}_{SP}(x)$. For all non-empty histories, $o' = \emptyset$. The reward r is zero, except when $t = k$, in which case $r = -c_{\boldsymbol{a}}$, where $\boldsymbol{c}$ is drawn from the conditional $\mathcal{D}_{SP}(\boldsymbol{c} \mid x)$, and $c_{\boldsymbol{a}}$ is the $\boldsymbol{a}$th value of $\boldsymbol{c}$, thinking of $\boldsymbol{a}$ as an index. Solving the search-based SP problem is equivalent to solving the induced RL problem, as stated in the following theorem (the proof is a straightforward application of the definitions):

Theorem 1. *Let $\mathcal{D}_{SP}$ be an SP problem and let $\mathcal{D}_{RL}(\mathcal{D}_{SP})$ be the induced RL problem. Let π be a policy for $\mathcal{D}_{RL}(\mathcal{D}_{SP})$. Then $\eta(\mathcal{D}_{RL}(\mathcal{D}_{SP}), \pi) = L(\mathcal{D}_{SP}, \textit{search}(\pi))$.*

Notice that under this mapping, RL is more general than SP. Moreover, note that whereas in RL the concept of *time* is important, it serves no purpose in SP.

4 Bounds for Structured Prediction Performance

In [7], it is shown that when the RL problem is solved using classification (i.e., using a classifier to predict actions), one can lower-bound the RL reward by the loss on the induced classification problems. Since SP can be seen as a degenerate RL problem, a similar result holds here. The main difference is that in the RL analysis, it was assumed that the horizon T (the number of steps) was fixed; in SP, it is variable. For completeness, and to account for this change, we will sketch the reduction here and state the relevant theorem.

The reduction from RL to classification involves running search to solve the RL problem, and using the search path to define a sequence of cost-sensitive classification problems. Let π be a policy; for a given input x, π defines a search path (sequence of observations, actions and rewards) that results in a structured output, $(p_1, p_2, \ldots, p_J)$. For each history $p_j = (o, a, r)_{1:j}$, we define one cost-sensitive classification problem. The classification problem is: of all possible next actions, which should we choose. Each possible action has a corresponding cost. To compute this cost, we first compute the *optimal completion* of the current history: given the history p_j, find the sequence of actions $a_{j+1}, \ldots, a_J$ that maximize the cumulative reward. Denote this reward r_j. Next, for each possible subsequent action, we compute the optimal completion, *given that we first perform that action*. The cost associated with an action is then J (the length of the search path) times the difference between r_j and the reward of the optimal completion given that we take the chosen action.

This reduction gives the same bounds as the reduction in [7], but without a constant horizon (the proof is omitted, but is an adaptation of Lem 2.3 from [7]).

Theorem 2. *Denote by $F(\mathcal{D}, \pi_h)$ the above reduction. For every RL problem $\mathcal{D}_{RL}$ and every policy π given by a classifier h, we have $L(\mathcal{D}_{RL}, \pi_h) = L(F(\mathcal{D}_{RL}, \pi_h), h)$.*

For solving the resulting cost-sensitive multiclass problem, we use the weighted all-pairs (WAP) reduction [1]. In SP, it is often the case that the total number of actions is large, but only a few are possible at any given step. WAP has the advantage over more complex techniques (like trees or error coding methods) because only the possible actions need to be considered for a single prediction. When composed with the Costing reduction [9], the test error of the weighted problem is upper-bounded by two times the error of the resulting binary classification problem, times the expected sum of costs for a single problem. Putting this all together, we obtain the following bounds for the SP problem:

$$L(\mathcal{D}_{SP}, search(h)) \leq 2\ Z\ L(\text{ Costing}(\text{ WAP}(\ F(\ \mathcal{D}_{RL}(\mathcal{D}_{SP}), \pi_h\ )\ )\ ), h) \qquad (1)$$

where Z is the expected cost for the SP problem (this term accounts for "scaling" the loss to the range expected by the SP loss).

5 Training

The results in the previous section tell us that *if* we have a classifier h, *then* we have guarantees about the performance on the SP problem. They do *not* tell us how to get such a classifier. For this, we appeal to a RL technique: Conservative Policy Iteration (CPI) [5]. CPI solves the RL problem using prior knowledge in the form of a "restart distribution" μ. Space precludes in-depth discussion of CPI, but the basic algorithm is as follows: (0) initialize a policy π; (1) compute a policy π' based on π and μ that is good; (2) estimate how much better π' is than π; (3) if it is not better, return π; (4) otherwise, update π to be a linear combination of π and π' and go to (1). This will converge (with high probability) after $\mathcal{O}(72R^2/\epsilon^2)$ iterations to a solution within 2ϵ of being optimal, where R is the maximum reward possible (see [5, Thm 4.4]). Also, the difference between the reward of the optimal policy and the found policy will be bounded by ϵ times a constant times a factor that represents the mismatch between μ and the true state distribution.

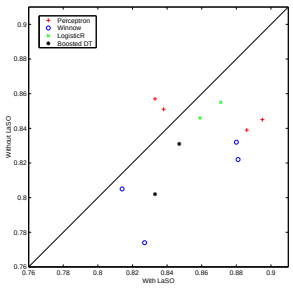
The μ restart distribution, central to the RL technique, is useful primarily to account for "exploration." In SP, however, exploration is not of significant importance; in fact, we often know (or can compute) the optimal search path. By defining μ to be this path, the divergence in the optimality bound is reduced optimally, leading to a tight result. Under this definition of μ, step (1) of CPI requires that we use the current policy π together with the optimal path to learn a new policy. In the language of SP, this means that we first draw a state from a distribution μ over search states, then use the current classifier (policy) to trace a search path from that initial state. We train a new classifier that performs well on this set of sequences (one for each training instance).

This analysis gives the following iterative technique for learning the binary classifier for search-based SP. We initialize a classifier h_0 arbitrarily, then repeat the following steps until convergence. (A) Create a training set S_i by drawing examples (x, y) from the SP training set and drawing states n_0 from the optimal search path, then by following the path taken by h_{i-1} on n_0. Each step along this path induces a cost-sensitive multiclass classification problem among all possible actions. The cost associated with the step $n \to m$ is $l(x, y, y_m) - l(x, y, y_n)$, where y_m (y_n) is the minimum loss $y \in \mathcal{Y}$ that is reachable from node n (node m). (B) Train h_i on S_i. (C) Evaluate the advantage of h_i over h_{i-1} as in [5]. (D) If the advantage is small or time has run out, return h_i; otherwise, update $h_i = \alpha h_i + (1 - \alpha)h_{i-1}$, with α as given in [5] and go to (A).

This procedure is nearly identical to the ad-hoc method proposed recently called *Learning as Search Optimization (*LASO*)* [2]. The two major differences are that LASO considered only 0/1 loss (here we extend it to arbitrary losses using a weaker assumption than the monotonicity used by LASO) and the LASO procedure *always* resets to the optimal search path, whereas we consider both the optimal search paths as well as those given by the previous iteration's classifier. This latter difference enables us to obtain stronger theoretical guarantees than before. Despite the differences, the resemblance is sufficiently strong that we refer to the reduction described here as the LASO reduction. In practice, using α as defined above may be *too* conservative; we advocate using a 1D line search on α at each iteration. Alternatively, setting $\alpha = 1$ may perform well, but is not guaranteed to converge.

6 Experiments

To demonstrate this approach on a trivial SP problem, we consider a simple syntactic chunking task for identifying noun-, verb- and prepositional-phrases. We use a subset of the CoNLL'00 data set: 1000 sentences for training, 500 for development and 1000 for test (63k words, total). We use word identity, stem, part of speech, prefix, suffix and case as features. As structured features, we use a history of the previous two tags. The loss function used is weighted F-score, where 1 point is given for correct span and tag, and $\frac{1}{2}$ point is given for correct span and incorrect tag. In all experiments, we set α by line search; other experiments setting $\alpha = 1$ performed slightly worse and are not reported here.



	LASO		Classifier	
	+Mar	-Mar	+Mar	-Mar
Perceptron	0.895	0.886	0.845	0.839
Winnow	0.881	0.880	0.822	0.832
LogisticR	0.859	0.871	0.846	0.855
Boosted DT	0.847	0.833	0.831	0.802
Perceptron-Cost	0.833	0.838	0.857	0.851
Winnow-Cost	0.827	0.814	0.774	0.805
Perceptron-Full	0.932	0.921	0.886	0.876
Winnow-Full	0.911	0.910	0.869	0.870

Original LASO: 0.835 Structured Perceptron: 0.816

Figure 1: (Left) Plot of all results, points below the line indicate LASO performed better and points above indicate that a plain classifier performed better (the two outliers are non-Costed Perceptrons). (Right) Numerical results for LASO versus classifier, with and without Markov features.

In our experiments, we compare four base learners: Perceptron, Logistic Regression, Winnow and boosted Decision trees. We compare with and without using Markov features (the previous two predicted tags). We also compare using LASO versus using a plain classifier applied sequentially (that will potentially exhibit label-bias). The results are shown in Figure 1. In the left panel, we compare LASO(below line) versus classifiers (above); as we can see, with two exceptions, LASO performs better. In the right panel, we present the numerical results from the experiments; we additionally present results for the Perceptron and Winnow without Costing. As we can see, except for Perceptron without Costing, LASO always outperforms the Classifier. Strangely, for logistic regression, Markov features seem to hurt; they also appear to hurt in several of the non-Costed examples. We also report baseline results for "Original LASO" (0.835) and the structured voted Perceptron (0.816). Finally, we report results using the full CoNLL'00 dataset for Perceptron and Winnow (the datasets were too large for our current LR and DT implementations); these are nearly at the state of the art level (in the 94-95 range), but importantly use purely greedy search rather than Viterbi. We believe extensions to non-greedy search will easily close the gap.

7 Conclusion and Discussion

We have presented an approach to SP based on greedy search, by reduction to classification using RL as a stepping-stone. Our technique is applicable to any SP problem with any loss function; specifically, we do not require feature locality of well-behaved loss functions (i.e., ones that decompose over the output structure). We have given bounds for test set performance based on binary classification performance, and a training algorithm that is guaranteed to converge in a polynomial number of steps to a formally good solution. Our current work involves extending this technique to non-greedy search algorithms, (such as beam-search) and alternatives to WAP that do not create quadratically many examples, which is unreasonable in problems with high branching factor (like machine translation).

References

[1] A. Beygelzimer, V. Dani, T. Hayes, J. Langford, and B. Zadrozny. Error limiting reductions between classification tasks. In *ICML*, 2005.

[2] H. Daumé III and D. Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

[3] U. Germann, M. Jahr, K. Knight, D. Marcu, and K. Yamada. Fast decoding and optimal decoding for machine translation. *Artificial Intelligence*, 154(1-2):127–143, 2003.

[4] X. Huang, A. Acero, and H-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall, 2001.

[5] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[6] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[7] J. Langford and B. Zadrozny. Relating reinforcement learning performance to classification performance. In *ICML*, 2005.

[8] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *NIPS* 2003.

[9] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *ICMD*, 2003.