Unsupervised Search-based Structured Prediction

Hal Daumé III ME@HAL3.NAME
School of Computing, University of Utah, Salt Lake City, UT 84112

Abstract

We describe an adaptation and application of a search-based structured prediction algorithm "SEARN" to unsupervised learning problems. We show that it is possible to reduce unsupervised learning to supervised learning and demonstrate a high-quality unsupervised shift-reduce parsing model. We additionally show a close connection between unsupervised SEARN and expectation maximization. Finally, we demonstrate the efficacy of a semi-supervised extension. The key idea that enables this is an application of the *predict-self* idea for unsupervised learning.

1. Introduction

A prevalent and useful version of unsupervised learning arises when both the observed data and the latent variables are structured. Examples range from hidden alignment variables in speech recognition (Rabiner, 1989) and machine translation (Brown et al., 1993; Vogel et al., 1996), to latent trees in unsupervised parsing (Paskin, 2001; Klein & Manning, 2004; Smith & Eisner, 2005; Titov & Henderson, 2007), and to pose estimation in computer vision (Ramanan et al., 2005). These techniques are all based on probabilistic models. Their applicability hinges on the tractability of (approximately) computing latent variable expectations, thus enabling the use of EM (Dempster et al., 1977). In this paper we show that a recently-developed *search-based* algorithm, SEARN (Daumé III et al., 2009 to appear) (see Section 2.2), can be utilized for unsupervised structured prediction (Section 3). We show: (1) that under an appropriate construction, SEARN can imitate the expectation maximization (Section 4); (2) that unsupervised SEARN can be used to obtain competitive performance on an unsupervised dependency parsing task (Section 6); and (3) that unsupervised SEARN naturally extends to a semi-supervised setting (Section 7). The key insight that enables this work is that we can consider the prediction of the (observed) input to be, itself, a structured prediction problem.

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

2. Structured Prediction

The *supervised* structured prediction problem is the task of mapping inputs x to complex structured outputs y (e.g., sequences, trees, etc.). Formally, let $\mathcal{X}$ be an arbitrary input space and $\mathcal{Y}$ be structure output space. $\mathcal{Y}$ is typically assumed to *decompose* over some smaller substructures (e.g., labels in a sequence). $\mathcal{Y}$ comes equipped with a loss function, often assumed to take the form of a Hamming loss over the substructures. Features are defined over pairs (x, y) in such a way that they obey the substructures (e.g., one might have features over adjacent label pairs in a sequence). Under strong assumptions on the structures, the loss function and the features (essentially "locality" assumptions), a number of learning algorithms can be employed: for example, conditional random fields (Lafferty et al., 2001) or max-margin Markov networks (Taskar et al., 2005).

A key difficulty in structured prediction occurs when the output space $\mathcal{Y}$, the features, or the loss, does not decompose nicely. All of these issues can lead to intractable computations at either training or prediction time (often both). An attractive approach for dealing with this intractability is to employ a search-based algorithm. The key idea in search-based structured prediction is to first decompose the output y into a sequence of (dependent) smaller predictions $y_1, \ldots, y_T$. These may each be predicted in turn, with later predictions dependent of previous decisions.

2.1. Search-based Structured Prediction

A recently proposed algorithm for solving the structured prediction problem is SEARN (Daumé III et al., 2009 to appear). SEARN operates by considering each substructure prediction $y_1, \ldots, y_T$ as a classification problem. A classifier h is trained so that at time t,

given a feature vector, it predict the best value for y_t. The feature vector can be based on any part of the input x and any previous decision $y_1, \ldots, y_{t-1}$. This introduces a chicken-and-egg problem. h should ideally be trained so that it makes the best decision for y_t *given* that h makes all past decisions $y_1, \ldots, y_{t-1}$ and all future decisions $y_{t+1}, \ldots, y_T$. Of course, at training time we do not have access to h (we are trying to construct it). The solution is to use an iterative scheme.

2.2. Searn

The presentation we give here differs slightly from the original presentation of the SEARN algorithm. Our motivation for straying from the original formulation is because our presentation makes more clear the connection between our unsupervised variant of SEARN and more standard unsupervised learning methods (such as standard algorithms on hidden Markov models).

Let $\mathcal{D}^{\text{SP}}$ denote a distribution over pairs (x, y) drawn from $\mathcal{X} \times \mathcal{Y}$, and let $\ell(y, \hat{y})$ be the loss associated with predicting $\hat{y}$ when the true answer is y. We assume that $y \in \mathcal{Y}$ can be decomposed into atomic predictions $y_1, \ldots, y_T$, where each y_t is drawn from a discrete set Y. A *policy*, π, is a (possibly stochastic) function that maps tuples $(x, y_1, \ldots, y_{t-1})$ to atomic predictions y_t.

The key ingredient in SEARN is to use the loss function ℓ and a "current" policy π to turn $\mathcal{D}^{\text{SP}}$ into a distribution over cost-sensitive (multiclass) classification problems (Beygelzimer et al., 2005). A cost-sensitive classification example is given by an input x and a cost vector $\boldsymbol{c} = \langle c_1, \ldots, c_K \rangle$, where c_k is the cost of predicting class k on input x. Define by SEARN$(\mathcal{D}^{\text{SP}}, \ell, \pi)$ a distribution over cost-sensitive classification problems derived as follows. To sample from this induced distribution, we first sample an example $(x, y) \sim \mathcal{D}^{\text{SP}}$. We then sample t uniformly from $[1, T]$ and run π for $t-1$ steps on (x, y). This yields a partial prediction $(\hat{y}_1, \ldots, \hat{y}_{t-1})$. The input for the cost sensitive classification problem is then the tuple $(x, \hat{y}_1, \ldots, \hat{y}_{t-1})$. The costs are derived as follows. For each possible choice k of $\hat{y}_t$, we defined c_k as the *expected* loss if π were run, beginning at $(\hat{y}_1, \ldots, \hat{y}_{t-1}, k)$ on input x. Formally:

$$c_k = \mathbb{E}_{\hat{y}_{t+1}, \ldots, \hat{y}_T \sim \pi} \ell(y, (\hat{y}_1, \ldots, \hat{y}_{t-1}, k, \hat{y}_{t+1}, \ldots, \hat{y}_T)) \quad (1)$$

SEARN assumes access to an "initial policy" π^* (sometimes called the "optimal policy"). Given an input x, a true output y and a prefix of predictions $\hat{y}_1, \ldots, \hat{y}_{t-1}$, π^* produces a best next-action, $\hat{y}_t$. It should be constructed so that the choice $\hat{y}_t$ is optimal (or close to optimal) with respect to the problem-specific loss function. For example, if the loss function is Hamming loss, the π^* will always produce $\hat{y}_t = y_t$. For more complex loss functions, computing π^* may be more involved.

Algorithm SEARN-Learn$(\mathcal{A}, \mathcal{D}^{\text{SP}}, \ell, \pi^*, \beta)$
1: Initialize $\pi = \pi^*$
2: **while** not converged **do**
3: Sample: $D \sim$SEARN$(\mathcal{D}^{\text{SP}}, \ell, \pi)$
4: Learn: $h \leftarrow \mathcal{A}(D)$
5: Update: $\pi \leftarrow (1 - \beta)\pi + \beta h$
6: **end while**
7: Return π without reference to π^*

Figure 1. The complete SEARN algorithm. It's parameters are: a cost-sensitive classification algorithm $\mathcal{A}$, a distribution over structured problems $\mathcal{D}^{\text{SP}}$, a loss function ℓ, an initial policy π^* and an interpolation parameter β.

Given these ingredients, SEARN operates according the algorithm given in Figure 1. Operationally, the sampling step is typically implemented by generating *every* example from a fixed structured prediction training set. The costs (expected losses) are computed by sampling with tied randomness (Ng & Jordan, 2000).

If $\beta = 1/T^3$, one can show (Daumé III et al., 2009 to appear) that after at most $2T^3 \ln T$ iterations, SEARN is guaranteed to find a solution π with structured prediction loss bounded as:

$$L(\pi) \leq L(\pi^*) + 2\ell_{\text{avg}} T \ln T + c(1 + \ln T)/T \quad (2)$$

where $L(\pi^*)$ is the loss of the initial policy (typically zero), T is the length of the longest example, c is the worse-case per-step loss and ℓ_{avg} is the average multiclass classification loss. This shows that the structured prediction algorithm learned by SEARN is guaranteed to be not-much-worse than that produced by the initial policy, *provided* that the created classification problems are easy (i.e., that ℓ_{avg} is small). Note that one can use *any* classification algorithm one likes.

3. Unsupervised Searn

In unsupervised structured prediction, we no longer receive an pair (x, y) but instead observes only an input x. Our job is to construct a classifier that produces y, even though we have never observed it.

3.1. Reduction for Unsupervised to Supervised

The key idea—one that underlies much work in unsupervised learning—is that a good y is one that enables us to easily recover x. This is precisely the intuition we build in to our model. The observation that makes this practical is that there is nothing in the theory or application of SEARN that says that π^* cannot be

stochastic. Moreover, there is not requirement that the loss function depend on *all* components of the prediction. Our model will essentially first predict y and then predict x based on y. Importantly, the loss function is agnostic to y (since we do not have true outputs).

The general construction is as follows. Let $\mathcal{D}^{\text{unsup}}$ be a distribution over inputs $x \in \mathcal{X}$ and let $\mathcal{Y}$ be the space of desired latent structures (e.g., trees). We define a distribution $\mathcal{D}^{\text{sup}}$ over $\mathcal{X} \times (\mathcal{Y} \times \mathcal{X})$ by defining a sampling procedure. To sample from $\mathcal{D}^{\text{sup}}$, we first sample $x \sim \mathcal{D}^{\text{unsup}}$. We then sample uniformly from the set of all $\mathcal{Y}$ that are valid structures for x. Finally, we return the pair $(x, (y, x))$. We define a loss function L by $L((y, x), (\hat{y}, \hat{x})) = L^{\text{input}}(x, \hat{x})$ where L^{input} is any loss function on the input space (e.g., Hamming loss). We apply SEARN to the supervised structured prediction problem $\mathcal{D}^{\text{sup}}$, and implicitly learn latent structures.

3.2. Sequence Labeling Example

To gain insight into the operation of SEARN in the unsupervised setting, it is useful to consider a sequence labeling example. That is, our input x is a sequence of length T and we desire a label sequence y of length T drawn from a label space of size K. We convert this into a supervised learning problem by considering the "true" structured output to be a label sequence of length $2T$, with the first T components drawn from the label space of size K and the second T components drawn from the input vocabulary. The loss function can then be anything that depends only on the last T components. For simplicity, we can consider it to be Hamming loss. The construction of the optimal policy in this case is straightforward. For the first T components, π^* may behave arbitrarily (e.g., it may produce a uniform distribution over the K labels). For the second T components, π^* always predicts the true label (which is known, because it is part of the input).

An important aspect of the model is the construction of the feature vectors. It is most useful to consider this construction as having two parts. The first part has to do with predicting the hidden structure (the first T components). The second part has to do with predicting the observed structure (the second T components). For the first part, we are free to use whatever features we desire, so long as they can be computed based on the input x and a partial output. For instance, in the HMM case, we could use the two most recent label predictions and windowed features from x.

The construction of the features for the second part is, however, also crucial. For instance, if the feature vector corresponding to "predict the tth component of x" contains the t component of x, then this learning problem is trivial—but also renders the latent structure useless. The goal of the designer of the feature space is to construct features for predicting x_t that crucially depend on getting the latent structure y correct. That is, the ideal feature set is one for which you can predict x_t accurately *if an only if* we have found the correct latent structure (more on this in Section 5). For instance, in the HMM case, we may predict x_t based only on the corresponding label y_t, or maybe on the basis of y_{t-1}, y_t, y_{t+1}. (Note that we are not limited to the Markov assumption, as in the case of HMMs.)

In the first iteration of SEARN, all costs for the prediction of the latent structure are computed with respect to the initial policy. Recalling that the initial policy behaves randomly when predicting the latent labels and correctly when predicting the words, we can see that these costs are all *zero*. Thus, for the latent structure actions, SEARN will not induce any classification examples (because the cost of all actions is equal). However, it will create example for predicting the x component. For predicting the xs, the cost will be zero for the correct word and one for any incorrect word. These examples will have associated features: we will predict word x_t based *exclusively* on y_t. Remember: y_t was generated randomly by the initial policy.

In the *second* iteration, the behavior is different. SEARN returns to creating examples for the latent structure components. However, in this iteration, since the current policy is not longer optimal, the future cost estimates may be non-zero. Consider generating an example corresponding to a (latent) state y_t. For some small percentage (as dictated by β) of the "generate x" decisions, the previously learned classifier will fire. If this learned classifier does well, then the associated cost will be low. However, if the learned classifier does poorly, the the associated cost will be high. Intuitively, the learned classifier will do well if and only if the action that labels y_t is "good" (i.e., consistent with what was learned previously). This, in the second pass through the data, SEARN *does* create classification examples specific to the latent decisions.

As SEARN iterates, more and more of the latent prediction decisions are made according to the learned classifiers and not with respect to the random policy.

4. Comparison to EM

In this section, we show an equivalence between expectation maximization in directed probabilistic structures and unsupervised SEARN. We use mixture of multinomials as a motivating example (primarily for simplicity), but the results easily extend to more com-

plicated models (e.g., HMMs: see Section 4.3).

4.1. EM for Mixture of Multinomials

In the mixture of multinomials problem, we are given N documents $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_N$, where $\boldsymbol{d}_n$ is a vector of word counts over a vocabulary of size V; that is, $d_{n,v}$ is the number of times word v appeared in document n. The mixture of multinomials is a probabilistic clustering model, where we assume an underlying set of K clusters (multinomials) that generated the documents. Denote by θ_k the multinomial parameter associated with cluster k, ρ_k the prior probability of choosing cluster k, and let $\boldsymbol{z}_n$ be an indicator vector associating document n with the unique cluster k such that $z_{n,k} = 1$. The probabilistic model has the form:

$$p(\boldsymbol{d} \mid \boldsymbol{\theta}, \boldsymbol{\rho}) = \prod_n \frac{(\sum_v d_{n,v})!}{\prod_v d_{n,v}!} \sum_{\boldsymbol{z}_n} \prod_k \left[\rho_k \prod_v \theta_{k,v}^{d_{n,v}} \right]^{z_{n,k}} \tag{3}$$

Expectation maximization in this model involves first computing expectations over the $\boldsymbol{z}$ vectors and then updating the model parameters $\boldsymbol{\theta}$:

$$\text{E-step:} \quad z_{n,k} \propto \rho_k \prod_v \theta_{k,v}^{d_{n,v}} \tag{4}$$

$$\text{M-step:} \quad \theta_{k,v} \propto \sum_n z_{n,k} d_{n,v} \quad ; \quad \rho_k \propto \sum_n z_{n,k} \tag{5}$$

In both cases, the constant of proportionality is chosen so that the variables sum to one over the last component. These updates are repeated until convergence of the incomplete data likelihood, Eq (3).

4.2. An Equivalent Model in Searn

Now, we show how to construct an instance of unsupervised Searn that effectively mimics the behavior of EM on the mixture of multinomials problem. The ingredients are as follows:

- The input space $\mathcal{X}$ is the space of documents, represented as word count vectors.
- The (latent) output space $\mathcal{Y}$ is a single discrete variable in the range $[1, K]$ that specifies the cluster.
- The feature set for predicting y (document counts).
- The feature set for predicting x is the label y and the total number of words in the document. The predictions for a document are estimated word probabilities, not the words themselves.
- The loss function ignores the prediction y and returns the log loss of the true document x under the word probabilities predicted.
- The cost-sensitive learning algorithm is different depending on whether the latent structure y is being predicted or if the document x is being predicted:
  - Structure: The base classifier is a multinomial naïve Bayes classifier, parameterized by (say) h^m
  - Document: The base classifier is a collection of independent maximum likelihood multinomial estimators for each cluster.

Consider the behavior of this setup. In particular, consider the distribution $\textsc{Searn}(\mathcal{D}^{\text{SP}}, \ell, \pi)$. There are two "types" of examples drawn from this distribution: (1) latent structure examples and (2) document examples. The claim is that *both* classifiers learned are identical to the mixture of multinomials model from Section 4.1.

Consider the generation of a latent structure example. First, a document n is sampled uniformly from the training set. Then, for each possible label k of this document, a cost $\mathbb{E}_{\hat{\boldsymbol{d}} \sim \pi} l((y, \boldsymbol{d}_n), (k, \hat{\boldsymbol{d}}))$ is computed. By definition, the $\hat{\boldsymbol{d}}$ that is computed is exactly the prediction according to the current multinomial estimator, h^m. Interpreting the multinomial estimator in terms of the EM parameters, the costs are *precisely* the $z_{n,k}$s from EM (see Eq (4)). These latent structure examples are fed in to the multinomial naïve Bayes classifier, which re-estimates a model exactly as per the M-step in EM (Eq (5)).

Next, consider the generation of the document examples. These examples are generated by π first choosing a cluster according to the structure classifier. This cluster id is then used as the (only) feature to the "generate document" multinomial. As we saw before, the probability that π will select label k for document n is precisely $z_{n,k}$ from Eq (4). Thus, the multinomial estimator will effectively receive weighted examples, weighted by these $z_{n,k}$s, thus making the maximum likelihood estimate exactly the same as the M-step from EM (Eq (5)).

4.3. Synthetic experiments

To demonstrate the advantages of the generality of Searn, we report here the result of some experiments on synthetic data. We generate synthetic data according to two different HMMs. The first HMM is a first-order model. The initial state probabilities, the transition probabilities, and the observation probabilities are all drawn uniformly. The second HMM is a second-order model, also will all probabilities drawn uniformly. The lengths of observations are given by a Poisson with a fixed mean.

In our experiments, we consider the following learning algorithms: EM, Searn with HMM features and a naïve Bayes classifier, and Searn with a logistic regression classifier (and an enhanced feature space: predicting y_t depends on $x_{t-1:t+1}$. The first Searn

Table 1. Error rates on first- and second-order Markov data with 2, 5 or 10 latent states. Models are the true data generating distribution (approximated by a first-order Markov model in the case of HMM2), a model learned by EM, one learned by SEARN with a naïve Bayes base classifier, and one learned by SEARN with a logistic regression base classifier. Standard deviations are given in small text. The best results by row are bolded; the results within the standard deviation of the best results are italicized.

Model	States	Truth	EM	SEARN -NB	SEARN -LR
1st order HMM	K = 2	0.227 ±0.107	*0.275* ±0.128	**0.287** ±0.138	*0.276* ±0.095
1st order HMM	K = 5	0.687 ±0.043	*0.678* ±0.026	*0.688* ±0.025	**0.672** ±0.022
1st order HMM	K = 10	0.806 ±0.035	*0.762* ±0.021	*0.771* ±0.019	**0.755** ±0.019
2nd order HMM	K = 2	0.294 ±0.072	0.396 ±0.057	0.408 ±0.056	**0.271** ±0.057
2nd order HMM	K = 5	0.651 ±0.068	0.695 ±0.027	0.710 ±0.016	**0.633** ±0.018
2nd order HMM	K = 10	0.815 ±0.032	0.764 ±0.021	0.771 ±0.015	**0.705** ±0.019

should mimic EM, but by using sampling rather than exact expectation computations. The models are all first-order, regardless of the underlying process.

We run the following experiment. For a given number of states (which we will vary), we generate 10 random data sets according to each model. Each data set consists of 5 examples with mean example length of 40 observations. The vocabulary size of the observed data is always 10. We compute error rates by matching each predicted label to the best-matching true label and the compute Hamming loss. Forward-backward is initialized randomly. We run experiments with the number of latent states equal to 2, 5 and 10.[1]

The results of the experiments are shown in Table 1. The observations show two things. When the true model matches the model we attempt to learn (HMM1), there is essentially no statistically significant difference between any of the algorithms. Where once sees a difference is when the true model does not match the learned model (HMM2). In this case, we see that SEARN-LR obtains a significant advantage over both EM and SEARN-NB, due to its ability to employ a richer set of features. These results hold over all values of K. This is encouraging, since in the real world our model is rarely (if ever) right. The (not statistically significant) difference in error rates between EM and SEARN-NB are due to a sampling versus exact computation of expectations. Many of the models outperform "truth" because likelihood and accuracy do not necessarily correlate (Liang & Klein, 2008).

5. Analysis

There are two keys to success in unsupervised-SEARN. The first key is that the features on the $\mathcal{Y}$-component of the output space be descriptive enough that it be learnable. One way of thinking of this constraint is that if we had labeled data, then we would be able to learn well. The second key is that the features on the $\mathcal{X}$-component of the output space be intrinsically tied to the hidden component. Ideally, these features will be such that $\mathcal{X}$ can be predicted with high accuracy if and only if $\mathcal{Y}$ is predicted accurately.

The general–though very trivial–result is that if we can guarantee that the loss on $\mathcal{Y}$ is bounded by some function f of the loss on $\mathcal{X}$, then the loss on $\mathcal{Y}$ is guaranteed after learning to be bounded by $f(L(\pi^*) + 2\ell_{\text{avg}} T_{\max} ln T_{\max} + c(1 + ln T_{\max})/T_{\max})$, where all the constants now depend on the induced structured prediction problem; see Eq 2.

One can see the unsupervised SEARN analysis as justifying a small variant on "Viterbi training"–the process of performing EM where the E-step is approximated with a delta function centered at the maximum. One significant issue with Viterbi training is that it is not guaranteed to converge. However, Viterbi training is recovered as a special case of unsupervised SEARN where the interpolation parameter is fixed at 1. While the SEARN theorem no longer applies in this degenerate case, any algorithm that uses Viterbi training could easily be retrofitted to simply make some decisions randomly. In doing so, one would obtain an algorithm that does have theoretical guarantees.

6. Unsupervised Dependency Parsing

The dependency formalism is a practical and linguistically interesting model of syntactic structure. One can think of a dependency structure for a sentence of length T as a directed tree over a graph over $T + 1$ nodes: one node for each word plus a unique root node. Edges point from heads to dependents. An example dependency structure for a $T = 7$ word sentence is shown in Figure 2 . To date, unsupervised dependency parsing has only been viewed in the context of

[1] We ran experiments varying the number of samples SEARN uses in $\{1, 2, 5\}$; there was no statistically significant difference. The results we report are based on 2 samples.

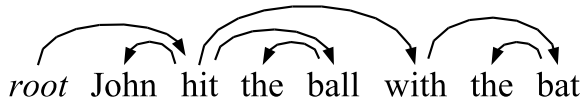

Figure 2. Dependency parse of a $T = 7$ word sentence.

global probabilistic models specified over dependency pairs (Paskin, 2001) or spanning trees (Klein & Manning, 2004; Smith & Eisner, 2005). However, there is an alternative, popular method for producing dependency trees in a supervised setting: shift-reduce parsing (Nivre, 2003; Sagae & Lavie, 2005).

6.1. Shift-reduce dependency parsing

Shift-reduce dependency parsing (Nivre, 2003) is a left-to-right parsing algorithm that operates by maintaining three state variables: a stack S, a current position i and a set of arcs A. The algorithm begins with $\langle S, i, A\rangle = \langle \emptyset, 1, \emptyset\rangle$: the stack and arcset are empty and the current index is 1 (the first word). The algorithm then proceeds through a series of actions until a final state is reached. A final state is one in which $i = T$, at which point the set A contains all dependency edges for the parse. Denote by $i|I$ a stack with i at the head and stack I at the tail. There are four actions:

LeftArc: $\langle t|S, i, A\rangle \longrightarrow \langle S, i, (i, t)|A\rangle$, so long as there does not exist an arc $(\cdot, t) \in A$. (Adds a left dependency to the arc set between the word t at the top of the stack and the word i at the current index.)

RightArc: $\langle t|S, i, A\rangle \longrightarrow \langle i|t|s, i + 1, (t, i)|A\rangle$, so long as there is no arc $(\cdot, i) \in A$. (Adds a right dependency between the top of the stack and the next input.)

Reduce: $\langle t|S, i, A\rangle \longrightarrow \langle S, i, A\rangle$, so long as there does exist an arc $(\cdot, t) \in A$. (Removes a word from the stack.)

Shift: $\langle S, i, A\rangle \longrightarrow \langle n|S, i + 1, A\rangle$. (Place item on stack.)

This algorithm is guaranteed to terminate in at most $2T$ steps with a valid dependency tree (Nivre, 2003), unlike standard probabilistic algorithms that have a time-complexity that is cubic in T (McDonald & Satta, 2007). The advantage of the shift-reduce framework is that it fits nicely into SEARN. However, until now, it has been an open question how to train a shift-reduce model in an unsupervised fashion. The techniques described in this paper give a solution to this problem.

6.2. Experimental setup

We follow the same experimental setup as (Smith & Eisner, 2005), using data from the WSJ10 corpus (sentences of length at most ten from the Penn Treebank (Marcus et al., 1993)). The data is stripped of punctuation and parsing depends on the part-of-speech tags, not the words. We use the same train/dev/test split as Smith and Eisner: 5301 sentences of training data, 531 sentences of development data and 530 sentences of blind test data. All algorithm development and tuning was done on the development data.

Table 2. Accuracy on training and test data, plus number of iterations for a variety of dependency parsing algorithms (all unsupervised except for the last two rows).

Algorithm	Acc-Tr	Acc-Tst	# Iter
Rand-Gen	23.5 $\pm$0.9	23.5 $\pm$1.3	
Rand-SEARN	21.3 $\pm$0.2	21.0 $\pm$0.6	
K+M:Rand-Init	23.6 $\pm$3.8	23.6 $\pm$4.3	63.3
K+M:Smart-Init	35.2 $\pm$6.6	35.2 $\pm$6.0	64.1
S+E:Length	33.8 $\pm$3.6	33.7 $\pm$5.9	173.1
S+E:DelOrTrans1	47.3 $\pm$6.0	47.1 $\pm$5.9	132.2
S+E:Trans1	48.8 $\pm$0.9	49.0 $\pm$1.5	173.4
SEARN: Unsup	45.8 $\pm$1.6	45.4 $\pm$2.2	27.6
S+E: Sup	79.9 $\pm$0.2	78.6 $\pm$0.8	350.5
SEARN: Sup	81.0 $\pm$0.3	81.6 $\pm$0.4	24.4

We use a slight modification to SearnShell to facilitate the development of our algorithm together with a multilabel logistic regression classifier, MegaM.[2] Our algorithm uses the following features for the tree-based decisions (inspired by (Hall et al., 2006)), where t is the top of the stack and i is the next token: the parts-of-speech within a window of 2 around t and i; the pair of tokens at t and i; the distance (discretized) between t and i; and the part-of-speech at the head (resp. tail) of any existing arc pointing to (resp. from) t or i. For producing word i, we use the part of speech of i's parent, grandparent, daughters and aunts.

We use SEARN with a fixed $\beta = 0.1$. One sample is used to approximate expected losses. The development set is used to tune the scale of the prior variances for the logistic regression (different variances are allowed for the "produce tree" and "produce words" features). The initial policy makes uniformly random decisions. Accuracy is directed arc accuracy.

6.3. Experimental results

The baseline systems are: two random baselines (one generative, one given by the SEARN initial policy), Klein and Manning's model (Klein & Manning, 2004) EM-based model (with and without clever initialization), and three variants of Smith and Eisner's model (Smith & Eisner, 2005) (with random initialization, which seems to be better for most of their mod-

[2]SearnShell and MegaM are available at `http://searn.hal3.name` and `http://hal3.name/megam`, respectively.

els). We also report an "upper bound" performance based on supervised training, for both the probabilistic (Smith+Eisner model) as well as supervised SEARN.

The results are reported in Table 2: accuracy on the training data, accuracy on the test data and the number of iterations required. These are all averaged over 10 runs; standard deviations are shown in small print. Many of the results (the non-SEARN results) are copied from (Smith & Eisner, 2005). The stopping criteria for the EM-based models is that the log likelihood changes by less than $10e-5$. For the SEARN-based methods, the stopping criteria is that the development accuracy ceases to increase (on the individual classification tasks, not on the structured prediction task).

All learned algorithms outperform the random algorithms (except Klein+Manning with random inits). K+M with smart initialization does slightly better than the worst of the S+E models, though the difference is not statistically significant. It does so needing only about a third of the number of iterations (moreover, a single S+E iteration is slower than a single K+M iteration). The other two S+E models do roughly comparably in terms of performance (strictly dominating the previous methods). One of them ("DelOrTrans1") requires about twice as many iterations as K+M; the other ("Trans1") requires about three times (but has much high performance variance). Unsupervised SEARN performs halfway between the best K+M model and the best S+E model (it is within the error bars for "DelOrTrans1" but not "Trans1").

Nicely, it takes significantly fewer iterations to converge (roughly 15%). Moreover, each iteration is quite fast in comparison to the EM-based methods (a complete run took roughly 3 hours on a 3.8GHz Opteron using SearnShell). Finally, we present results for the supervised case. Here, we see that the SEARN-based method converges much more quickly to a better solution than the S+E model. Note that this comparison is unfair since the SEARN-based model uses additional features (though it is a nice property of the SEARN-based model that it *can* make use of additional features). Nevertheless we provide it so as to give a sense of a reasonable upper-bound. We imagine that including more features would shift the upper-bound and the unsupervised algorithm performance up.

7. A Semi-Supervised Version

The unsupervised learning algorithm described above naturally extends to the case where some labeled data is available. In fact, the only modification to the algorithm is to change the loss function. In the unsupervised case, the loss function completely ignores the latent structure, and returns a loss dependent only on the "predict self" task. In the semi-supervised version, one plugs in a natural loss function for the "latent" structure prediction for the labeled subset of the data.

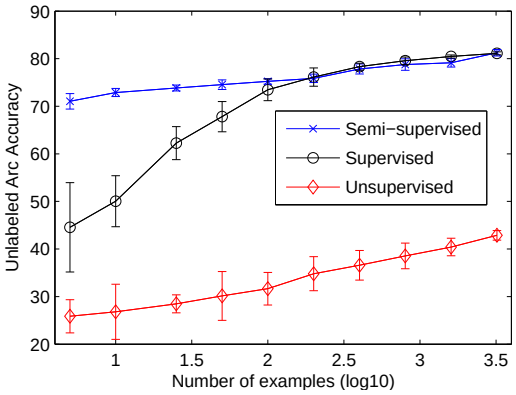

Figure 3. Parsing accuracy for semi-supervised, supervised and unsupervised SEARN. X-axis is: (semi/sup) # of labeled examples; (unsup) # of unlabeled examples.

In Figure 3, we present results on dependency parsing. We show learning curves for unsupervised, fully supervised and semi-supervised models. The x-axis shows the number of examples used; in the unsupervised and supervised cases, this is the total number of examples; in the semi-supervised case, it is the number of labeled examples. Error bars are two standard deviations. Somewhat surprisingly, with only five labeled examples, the semi-supervised approach achieves an accuracy of over 70%, only about 10% behind the fully supervised approach with 5182 labeled examples. Eventually the supervised model catches up (at about 250 labeled examples). The performance of the unsupervised model continues to grow as more examples are provided, but never reaches anywhere close to the supervised or semi-supervised models.

8. Conclusions

We have described the application of a search-based structured prediction algorithm, SEARN, to unsupervised learning. This answers positively an open question in the field of learning reductions (Beygelzimer et al., 2005): can unsupervised learning be reduced to supervised learning? We have shown a near-equivalence between the resulting algorithm and the forward-backward algorithm in hidden Markov models. We have shown an application of this algorithm to unsupervised dependency parsing in a shift-reduce framework. This provides the first example of unsupervised learning for dependency parsing in a non-

probabilistic model and shows that unsupervised shift-reduce parsing is possible. One obvious extension of this work is to structured prediction problems with additional latent structure, such as in machine translation. Instead of using the predict-self methodology, one could directly apply a predict-target methodology.

The view of "predict the input" for unsupervised learning is implicit in many unsupervised learning approaches, including standard models such as restricted Boltzmann machines and Markov random fields. This is made most precise in the wake-sleep algorithm (Hinton et al., 1995), which explicitly trains a neural network to reproduce its own input. The wake-sleep algorithm consists of two phases: the wake phase, where the latent layers are produced, and the sleep phase, where the input is (re-)produced. These two phases are analogous to the predict-structure phase and the predict-words phase in unsupervised SEARN.

Acknowledgements. Thanks for Ryan McDonald and Joakim Nivre for discussions related to dependency parsing algorithms. Comments from 5 (!) anonymous reviewers were incredibly helpful. This was partially supported by NSF grant IIS-0712764.

References

Beygelzimer, A., Dani, V., Hayes, T., Langford, J., & Zadrozny, B. (2005). Error limiting reductions between classification tasks. *Proc. Int'l Conf. on Machine Learning* (pp. 49–56).

Brown, P., Della Pietra, S., Della Pietra, V., & Mercer, R. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*, 263–311.

Daumé III, H., Langford, J., & Marcu, D. (2009 (to appear)). Search-based structured prediction. *Machine Learning J.*.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society*, *B39*, 1–38.

Hall, J., Nivre, J., & Nilsson, J. (2006). Discriminative classifiers for determining dependency parsing. *Proc. Conf. of the Assoc. for Computational Linguistics* (pp. 316–323).

Hinton, G., Dayan, P., Frey, B., & Neal, R. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, *26*, 1158–1161.

Klein, D., & Manning, C. (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. *Proc. Conf. of the Assoc. for Computational Linguistics* (pp. 478–485).

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. Int'l Conf. on Machine Learning* (pp. 282–289).

Liang, P., & Klein, D. (2008). Analyzing the errors of unsupervised learning. *Proc. Assoc. for Computational Linguistics* (pp. 879–887).

Marcus, M., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*, 313–330.

McDonald, R., & Satta, G. (2007). On the complexity of non-projective data-driven dependency parsing. *Int'l Wk. on Parsing Technologies* (pp. 121–132).

Ng, A., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. *Proc. Converence on Uncertainty in Artificial Intelligence* (pp. 406–415).

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. *Int'l Wk. on Parsing Technologies* (pp. 149–160).

Paskin, M. A. (2001). Grammatical bigrams. *Advances in Neural Info. Processing Systems* (pp. 91–97).

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* (pp. 257–285).

Ramanan, D., Forsyth, D., & Zisserman, A. (2005). Strike a pose: Tracking people by finding stylized poses. *Computer Vision and Pattern Recognition* (pp. 271–278).

Sagae, K., & Lavie, A. (2005). A classifier-based parser with linear run-time complexity. *Int'l Wk. on Parsing Technologies*.

Smith, N. A., & Eisner, J. (2005). Guiding unsupervised grammar induction using contrastive estimation. *IJCAI Wk. on Grammatical Inference Apps* (pp. 73–82).

Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: A large margin approach. *Proc. Int'l Conf. on Machine Learning* (pp. 897–904).

Titov, I., & Henderson, J. (2007). A latent variable model for generative dependency parsing. *Int'l Conf. on Parsing Technologies*.

Vogel, S., Ney, H., & Tillmann, C. (1996). HMM-based word alignment in statistical translation. *Proc. Int'l Conf. on Computational Linguistics* (pp. 836–841).