Structured Prediction via Learning to Search under Bandit Feedback

Amr Sharaf and **Hal Daumé III**
University of Maryland, College Park
{amr,hal}@cs.umd.edu

Abstract

We present an algorithm for structured prediction under online bandit feedback. The learner repeatedly predicts a sequence of actions, generating a structured output. It then observes feedback for that output and no others. We consider two cases: a pure bandit setting in which it only observes a loss, and more fine-grained feedback in which it observes a loss for every action. We find that the fine-grained feedback is necessary for strong empirical performance, because it allows for a robust variance-reduction strategy. We empirically compare a number of different algorithms and exploration methods and show the efficacy of BLS on sequence labeling and dependency parsing tasks.

1 Introduction

In structured prediction the goal is to jointly predict the values of a collection of variables that interact. In the usual "supervised" setting, at training time, you have access to ground truth outputs (e.g., dependency trees) on which to build a predictor. We consider the substantially harder case of online *bandit* structured prediction, in which the system never sees supervised training data, but instead must make predictions and then only receives feedback about the quality of that *single* prediction. The model we simulate (Figure 1) is:

1. the world reveals an input (e.g., a sentence)
2. the algorithm produces a *single* structured prediction (e.g., full parse tree);
3. the world provides a loss (e.g., overall quality rating) and possibly a small amount of additional feedback;
4. the algorithm updates itself

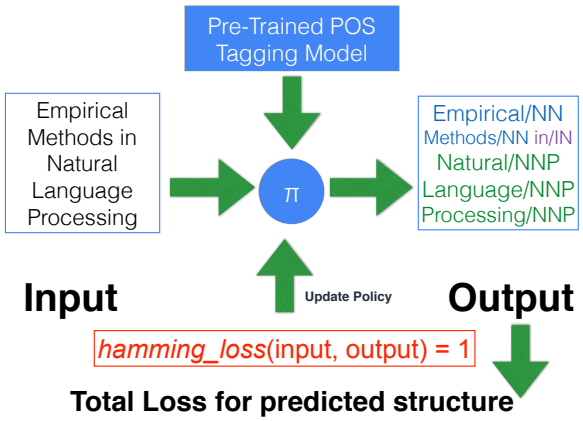

Figure 1: BLS for learning POS tagging. We learn a policy π, whose output a user sees. The user views predicted tags and provides a loss (and possibly additional feedback, such as which words are labeled incorrectly). This is used to update π.

The goal of the system is to minimize it's cumulative loss over time, using the feedback to improve itself. This introduces a fundamental exploration-versus-exploitation trade-off, in which the system must try new things in hopes that it will learn something useful, but also in which it is penalized (by high cumulative loss) for exploring too much.[1]

One natural question we explore in this paper is: in addition to the loss, what forms of feedback are both easy for a user to provide and useful for a system to utilize? At one extreme, one can solicit no additional feedback, which makes the learning problem very difficult. We describe *Bandit Learning to Search*, BLS[2], an approach for improving joint predictors from different types of bandit feedback. The algorithm predicts outputs and observes the loss of the predicted struc-

[1] Unlike active learning—in which the system chooses which examples it wants feedback on—in our setting the system is beholden to the human's choice in inputs.

[2] Our implementation will be made freely available.

Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing, pages 17–26
Copenhagen, Denmark, September 7–11, 2017.

ture; but then it uses a regression strategy to estimate *counterfactual* costs of (some) other structures that it did *not* predict. This variance reduction technique (§2.2) is akin to doubly-robust estimation in contextual bandits. The trade-off is that in order to accurately train these regressors, BLS requires per-action feedback from the user (e.g., which words were labeled incorrectly). It appears that this feedback is necessary; with out it, accuracy *degrades* over time. Additionally, we consider several forms of exploration beyond a simple ϵ-greedy strategy, including Boltzmann exploration and Thompson sampling (§2.4). We demonstrate the efficacy of these developments on POS tagging, syntactic chunking and dependency parsing (§ 3), in which we show improvements over both LOLS (Chang et al., 2015) and Policy Gradient (Sutton et al., 1999).

2 Learning with Bandit Feedback

We operate in the learning to search framework, a family of algorithms for solving structured prediction problems, which essentially train a *policy* to make sequence of decisions that are stitched together into a final structured output. Such algorithms decompose a joint prediction task into a sequence of action predictions, such as predicting the label of the next word in sequence labeling or predicting a shift/reduce action in dependency parsing[3]; these predictions are tied by features and/or internal state. Algorithms in this family have recently met with success in neural networks (Bengio et al., 2015; Wiseman and Rush, 2016), though date back to models typically based on linear policies (Collins and Roark, 2004; Daumé III and Marcu, 2005; Xu et al., 2007; Daumé III et al., 2009; Ross et al., 2010; Ross and Bagnell, 2014; Doppa et al., 2014; Chang et al., 2015).

Most learning to search algorithms operate by considering a search space like that shown in Figure 2. The learning algorithm first *rolls-in* for a few steps using a *roll-in* policy π^{in} to some state R, then considers all possible actions available at state R, and then *rolls out* using a *roll-out* policy π^{out} until the end of the sequence. In the fully supervised case, the learning algorithm can then compute a loss for all possible outputs, and use this loss to drive learning at state R, by encouraging the learner to take the action with lowest cost, updating the learned policy from $\hat{\pi}_i$ to $\hat{\pi}_{i+1}$.

[3] Although the decomposition is into a sequence of predictions, such approaches are not limited to "left-to-right" style prediction tasks (Ross et al., 2010; Stoyanov et al., 2011).

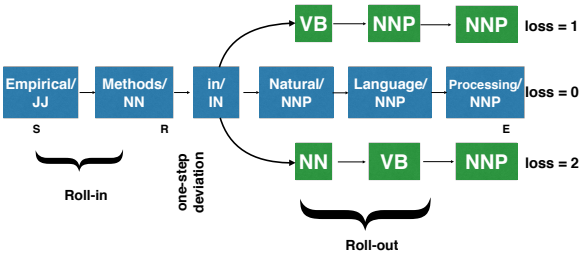

Figure 2: A search space for part of speech tagging, explored by a policy that chooses to "explore" at state R.

In the bandit setting, this is not possible: we can only evaluate one loss; nevertheless, we can follow a similar algorithmic structure. Our specific algorithm, BLS, is summarized in algorithm 1. We start with a pre-trained reference policy π^{ref} and seek to improve it with bandit feedback. On each example, an exploration algorithm (§2.4) chooses whether to explore or exploit. If it chooses to exploit, a random learned policy is used to make a prediction and no updates are made. If, instead, it chooses to explore, it executes a roll-in as usual, a *single* deviation at time t according to the exploration policy, and then a roll-out. Upon completion it suffers a bandit loss for the *entire* complete trajectory. It then uses a *cost estimator* ρ to guess the costs of the un-taken actions. From this it forms a complete cost vector, and updates the underlying policy based on this cost vector. Finally, it updates the cost estimator ρ.

2.1 Cost Estimation by Importance Sampling

The simplest possible method of cost estimation is importance sampling (Horvitz and Thompson, 1952; Chang et al., 2015). If the third action is the one explored with probability p_3 and a cost $\hat{c}_3$ is observed, then the cost vector for all actions is set to $\langle 0, 0, \hat{c}_3/p_3, 0, \ldots, 0\rangle$. This yields an unbiased estimate of the true cost vector because in expectation over all possible actions, the cost vector equals $\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_K\rangle$.

Unfortunately, this type of cost estimate suffers from huge variance (see experiments in §3). If actions are explored uniformly at random, then all cost vectors look like $\langle 0, 0, K\hat{c}_3, 0, \ldots 0\rangle$, which varies quite far from its expectation when K is large. To better understand the variance reduction issue, consider the part of speech tagging exam-

Input: examples $\{x_i\}_{i=1}^N$, reference policy π^{ref}, exploration algorithm $explorer$, and rollout-parameter $\beta \geq 0$
$\pi_0 \leftarrow$ initial policy
$\mathcal{I} \leftarrow \emptyset$
$\rho \leftarrow$ initial cost estimator
for *each* x_i *in training examples* **do**
  if *explorer* chooses not to explore **then**
    $\pi \leftarrow$ *Unif*$(\mathcal{I})$ // pick policy
    $y_i \leftarrow$ predict using π
    $c \leftarrow$ bandit loss of y_i
  else
    $t \leftarrow$ *Unif*$(0, T-1)$ // deviation time
    $\tau \leftarrow$ roll-in with $\hat{\pi}_i$ for t rounds
    $s_t \leftarrow$ final state in τ
    $a_t =$ *explorer*(s_t) // deviation action
    $\pi^{\text{out}} \leftarrow \pi^{\text{ref}}$ with prob β, else $\hat{\pi}_i$
    $y_i \leftarrow$ roll-out with π^{out} from $\tau + a_t$
    $c \leftarrow$ bandit loss of y_i
    $\hat{c} \leftarrow$ *est_cost*$(s_t, \tau, \rho, A(s_t), a_t, c)$
    $\hat{\pi}_{i+1} \leftarrow$ *update*$(\hat{\pi}_i, (\Phi(x_i, s_t), \hat{c}))$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{\hat{\pi}_{i+1}\}$
    update cost estimator ρ
  end
end

Algorithm 1: Bandit Learning to Search (BLS)

Input: current state: s_t; roll-in trajectory: τ; K regression functions (one for every action): ρ; set of allowed actions: $A(s_t)$; roll-out policy: π^{out}; explored action: a_t; bandit loss: c
$t \leftarrow |\tau|$
Initialize $\hat{c}$: a vector of size $|A(s_t)|$
$\hat{c}_0 \leftarrow 0$
for $(a, s) \in \tau$ **do**
  $\hat{c}_0 \leftarrow \hat{c}_0 + \rho_a(\Phi(s))$
end
for $a \in A(s_t)$ **do**
  if $a = a_t$ **then**
    $\hat{c}(a) \leftarrow c$
  else
    $\hat{c}(a) \leftarrow \hat{c}_0$
    $y \leftarrow$ roll-out with π^{out} from $\tau + a$
    for (a', s') *in* y **do**
      $\hat{c}(a) \leftarrow \hat{c}(a) + \rho_{a'}(\Phi(s'))$
    end
end
return $\hat{c}$: *a vector of size* $|A(s_t)|$, *where* $\hat{c}(a)$ *is the estimated cost for action* a *at state* s_t.

Algorithm 2: *est_cost*

ple from Figure 2. As in the figure, suppose that the deviation occurs at time step 3 and that during roll-in, the first two words are tagged correctly by the roll-in policy. At $t = 3$, there are 45 possible actions (each possible part of speech) to take from the deviation state, of which three are shown; each action (under uniform exploration) will be taken with probability $1/45$. If the first is taken, a loss of one will be observed, if the second, a loss of zero, and if the third, a loss of two. When a fair coin is flipped, perhaps the third choice is selected, which will induce a cost vector of $\vec{c} = \langle 0, 0, 90, 0, \ldots \rangle$. In expectation over this random choice, we have $\mathbb{E}_a[c]$ is correct, implying unbiasedness, but the variance is very large: $\mathcal{O}((Kc_{\max})^2)$.

This problem is exacerbated by the fact that many learning to search algorithms define the cost of an action a to be the *difference* between the cost of a and the minimum cost. This is desirable because when the policy is predicting greedily, it should choose the action that *adds* the least cost; it should not need to account for already-incurred cost. For example, suppose the first two words in Figure 2 were tagged incorrectly. This would add a loss of 2 to any of the estimated costs, but could be very difficult to fit because this loss was based on past actions, not the current action.

2.2 Doubly Robust Cost Estimation

To address the challenge of high variance cost estimates, we adopt a strategy similar to the doubly-robust estimation used in the (non-structured) contextual bandit setting (Dudik et al., 2011). In particular, we train a separate set of regressors to estimate the total costs of any action, which we use to impute a counterfactual cost for untaken actions.

Algorithm 2 spells this out, taking advantage of an action-to-cost regressor, ρ. To estimate the cost of an un-taken action a' at a deviation, we simulate the execution of a', followed by the execution of the current policy through the end. The cost of that entire trajectory is estimated by summing ρ over all states along the path. For example, in the part of speech tagging example above, we learn 45 regressors: one for each part of speech. The cost of a roll-out is estimated as the sum of these regressors over the entire predicted sequence.

Using this doubly-robust estimate strategy addresses *both* of the problems mentioned in § 2.1.

First, this is able to provide a cost estimate for *all* actions. Second, because ρ is deterministic, it will give the same cost to the common prefix of all trajectories, thus resolving credit assignment.

The remaining challenge is: how to estimate these regressors. Unfortunately, this currently comes at an additional cost to the user: we must observe per-action feedback. In particular, when the user views a predicted output (e.g., part of speech sequence), we ask for a binary signal for each action whether the predicted action was right or wrong. Thus, for a sentence of length T, we generate T training examples for every time step $1 \leq t \leq T$. Each training example has the form: (a_t, c_t), where a_t is the predicted action at time t, and c_t is a binary cost, either 1 if the predicted action was correct, or zero otherwise. This amounts to a user "crossing out" errors, which hopefully incurs low overhead. Using these T training examples, we can effectively train the K regressors for estimating the cost of unobserved actions.

2.3 Theoretical Analysis

In order to analyze the variance of the BLS estimator (in particular to demonstrate that it has lower variance than importance sampling), we provide a reduction to contextual bandits in an i.i.d setting. Dudík et al. (2014) studied the contextual bandit setting, which is similar to out setting but without the complexity of *sequences* of actions. (In particular, if $T = 1$ then our setting is exactly the contextual bandit setting.) They studied the task of off-policy evaluation and optimization for a target policy ν using doubly robust estimation given historic data from an exploration policy μ consisting of contexts, actions, and received rewards. They prove that this approach yields accurate value estimates when there is either a good (but not necessarily consistent) model of rewards or a good (but not necessarily consistent) model of past policy. In particular, they show:

Theorem. *Let $\Delta(x, a)$ and $\rho_k(x, a)$ denote, respectively, the additive error of the reward estimator $\hat{r}$ and the multiplicative error of the action probability estimator $\hat{\mu}_k$. If exploration policy μ and the estimator $\hat{\mu}_k$ are stationary, and the target policy ν is deterministic, then the variance of the doubly robust estimator $\mathbb{V}_\mu[\hat{V}_{DR}]$ is:*

$$\frac{1}{n}(\mathbb{V}_{(x,a)\sim\nu}[r^*(x,a) + (1-\rho_1(x,a))\Delta(x,a)])$$
$$+\mathbb{E}_{(x,a)\sim\nu}\big[\frac{1}{\hat{\mu}_1(a|x)}\rho_1(x,a)\mathbb{V}_{r\sim D(\cdot|x,a)}[r]\big]$$
$$+\mathbb{E}_{(x,a)\sim\nu}\big[\frac{1-\mu_1(a|x)}{\hat{\mu}_1(a|x)}\rho_1(x,a)\Delta(x,a)^2\big]\big]$$

The theorem show that the variance can be decomposed into three terms. The first term accounts for the randomness in the context features. The second term accounts for randomness in rewards and disappears when rewards are deterministic functions of the context and actions. The last term accounts for the disagreement between actions taken by the exploration policy μ and the target policy ν. This decomposition shows that doubly robust estimation yields accurate value estimates when there is either a good model of rewards or a good model of the exploration policy.

We can build on this result for BLS to show an *identical* result under the following reduction. The exploration policy μ in our setting is defined as follows: for every exploration round, a randomly selected time-step is assigned a randomly chosen action, and a deterministic reference policy is used to generate the roll-in and roll-out trajectories. Our goal is to evaluate and optimize a better target policy ν. Under this setting, and assuming that the structures are generated i.i.d from a fixed but unknown distribution, the structured prediction problem will be equivalent to a contextual bandit problem were we consider the roll-in trajectory as part of the context.

2.4 Options for Exploration Strategies

In addition to the ϵ-greedy exploration algorithm, we consider the following exploration strategies:

Boltzmann (Softmax) Exploration. Boltzmann exploration varies the action probabilities as a graded function of estimated value. The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their cost estimates; action a is chosen with probability proportional to $\exp\left[\frac{1}{\text{temp}}c(a)\right]$, where "temp" is a positive parameter called the temperature, and $c(a)$ is the current predicted cost of taking action a. High temperatures cause the actions to be all (nearly) equiprobable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates.

Thompson Sampling estimates the following elements: a set Θ of parameters μ; a prior distribution $P(\mu)$ on these parameters; past observations D consisting of observed contexts and rewards; a likelihood function $P(r|b, \mu)$, which gives the probability of reward given a context b and a parameter μ; In each round, Thompson Sampling selects an action according to its posterior probability of having the best parameter μ. This is achieved by taking a sample of parameter for each action, using the posterior distributions, and selecting that action that produces the best sample (Agrawal and Goyal, 2013; Komiyama et al., 2015). We use Gaussian likelihood function and Gaussian prior for the Thompson Sampling algorithm. In addition, we make a linear payoff assumption similar to (Agrawal and Goyal, 2013), where we assume that there is an unknown underlying parameter $\mu_a \in R^d$ such that the expected cost for each action a, given the state s_t and context x_i is $\Phi(x_i, s_t)^T \mu_a$.

3 Experimental Results

The evaluation framework we consider is the fully online setup described in the introduction, measuring the degree to which various algorithms can effectively improve upon a reference policy by observing only a partial feedback signal, and effectively balancing exploration and exploitation. We learn from one structured example at every time step, and we do a single pass over the available examples. We measure loss as the average cumulative loss over time, thus algorithms are appropriately "penalized" for unnecessary exploration.

3.1 Tasks, Policy Classes and Data Sets

We experiment with the following three tasks. For each, we briefly define the problem, describe the policy class that we use for solving that problem in a learning to search framework (we adopt a similar setting to that of (Chang et al., 2016), who describes the policies in more detail), and describe the data sets that we use. The regression problems are solved using squared error regression, and the classification problems (policy learning) is solved via cost-sensitive one-against-all.

Part-Of-Speech Tagging over the 45 Penn Treebank (Marcus et al., 1993) tags. To simulate a domain adaptation setting, we train a reference policy on the TweetNLP dataset (Owoputi et al., 2013), which achieves good accuracy in domain, but does poorly out of domain. We simulate bandit feedback over the entire Penn Treebank Wall Street Journal (sections 02–21 and 23), comprising $42k$ sentences and about one million words. (Adapting *from* tweets *to* WSJ is nonstandard; we do it here because we need a large dataset on which to simulate bandit feedback.) The measure of performance is average per-word accuracy (one minus Hamming loss).

Noun Phrase Chunking is a sequence segmentation task, in which sentences are divided into base noun phrases.We solve this problem using a sequence span identification predictor based on Begin-In-Out encoding, following (Ratinov and Roth, 2009), though applied to chunking rather than named-entity recognition. We used the CoNLL-2000 datasetfor training and testing. We used the smaller test split ($2,012$ sentences) for training a reference policy, and used the training split ($8,500$ sentences) for online evaluation. Performance was measured by F-score over predicted noun phrases (for which one has to predict the entire noun phrase correctly to get any points).

Dependency Parsing is a syntactic analysis task, in which each word in a sentence gets assigned a grammatical head (or "parent"). The experimental setup is similar to part-of-speech tagging. We train an arc-eager dependency parser (Nivre, 2003), which chooses among (at most) four actions at each state: Shift, Reduce, Left or Right. As in part of speech tagging, the reference policy is trained on the TweetNLP dataset (using an oracle due to (Goldberg and Nivre, 2013)), and evaluated on the Penn Treebank corpus (again, sections $02-21$ and section 23). The loss is unlabeled attachment score (UAS), which measures the fraction of words that pick the correct parent.

3.2 Main Results

Here, we describe experimental results (Table 1) comparing several algorithms: (line B) The bandit variant of the LOLS algorithm, which uses importance sampling and ϵ-greedy exploration; (lines C-F) BLS, with bandit feedback and per-word error correction, with variance reduction and four exploration strategies: ϵ-greedy, Boltzmann, Thompson, and "oracle" exploration in which case the oracle action is always chosen during exploration; (line G) The Policy Gradient reinforcement learning algorithm, with ϵ-greedy exploration on one-step deviations; and (line H) A fully super-

	Algorithm	Variant	POS Acc	DepPar UAS	Chunk F-Scr
A.	Reference	-	47.24	44.15	74.73
B.	LOLS	ϵ-greedy	2.29	18.55	31.76
C.	BLS	ϵ-greedy	86.55	56.04	90.03
D.		Boltz.	**89.62**	**57.20**	**90.91**
E.		Thomp.	89.37	56.60	90.06
F.		Oracle	89.23	56.60	90.58
G.	Policy∇	ϵ-greedy	75.10	-	90.07
H.	DAgger	Full Sup	96.51	90.64	95.29

Table 1: Total progressive accuracies for the different algorithms on the three natural language processing tasks. LOLS uniformly *decreases* performance over the Reference baseline. BLS, which integrates cost regressors, uniformly improves, often quite dramatically. The overall effect of the exploration mechanism is small, but in all cases Boltzmann exploration is statistically significantly better than the other options at the $p < 0.05$ level (because the sample size is so large). Policy Gradient for dependency parsing is missing because after processing $\frac{1}{4}$ of the data, it was substantially subpar.

vised "upper bound" trained with DAgger.

From these results, we draw the following conclusions (the rest of this section elaborates on these conclusions in more detail):

1. The original LOLS algorithm is ineffective at improving the accuracy of a poor reference policy (A vs B);
2. Collecting additional per-word feedback in BLS allows the algorithm to drastically improve on the reference (A vs C) and on LOLS (B vs C); we show in §3.3 that this happens because of variance reduction;
3. Additional leverage can be gained by varying the exploration strategy, and in general Boltzmann exploration is effective (C,D,E), but the Oracle exploration strategy is *not* optimal (F vs D); see §3.4;
4. For large action spaces like POS tagging, the BLS-type updates outperform Policy Gradient-type updates, when the exploration strategy is held constant (G vs D), see §3.5.
5. Bandit feedback is less effective than full feedback (H vs D) (§3.6).

3.3 Effect of Variance Reduction

Table 1 shows the progressive validation accuracies for all three tasks for a variety of algorithmic settings. To understand the effect of variance, it is enough to compare the performance of the Reference policy (the policy learned from the out of domain data) with that of LOLS. In all of these cases, running LOLS substantially decreases performance. Accuracy drops by 45% for POS tagging, 26% for dependency parsing and 43% for noun phrase chunking. For POS tagging, the LOLS accuracy falls *below* the accuracy one would get for random guessing (which is approximately 14% on this dataset for `NN`)!

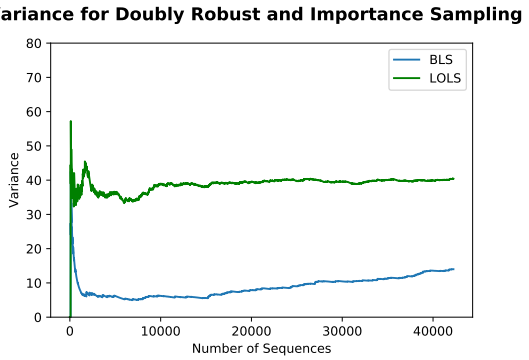

Figure 3: Analyzing the variance of the the cost estimates from LOLS and BLS over a run of the algorithm for POS; the x-axis is number of sentences processed, y-axis is empirical variance.

When the underlying algorithm changes from LOLS to BLS, the overall accuracies go up significantly. Part of speech tagging accuracy increases from 47% to 86%; dependency parsing accuracy from 44% to 57%; and chunking F-score from 74% to 90%. These numbers naturally fall below state of the art for *fully supervised* learning on these data sets, precisely because these results are based only on bandit feedback (see §3.6).

3.4 Effect of Exploration Strategy

Figure 4 shows the effect of the choice of ϵ for ϵ-greedy exploration in BLS. Overall, best results are achieved with *remarkably* high epsilon, which is possibly counter-intuitive. The reason this happens is because BLS only explores on one out of T time steps, of which there are approximately 30 in each of these experiments (the sentence lengths). This means that even with $\epsilon = 1$, we only take a random action roughly 3.3% of the time. It is therefore not surprising that large ϵ is the most effective strategy. Overall, although the differences are small, the best choice of ϵ across these different tasks is ≈ 0.6.

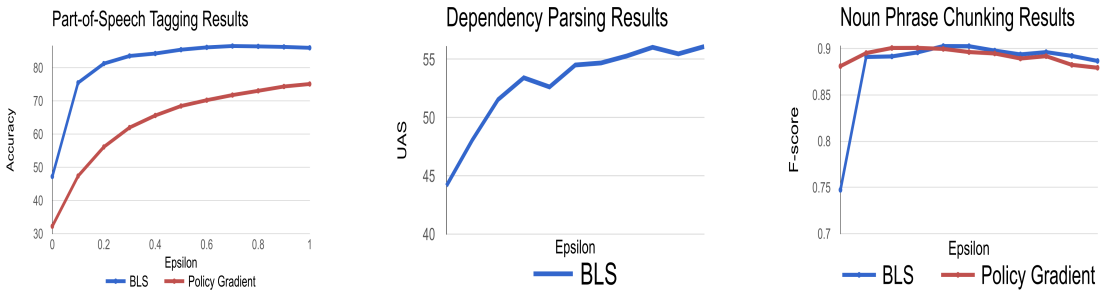

Figure 4: Analyzing the effect of ϵ in exploration/exploitation trade-off. Overall, large values of ϵ are strongly preferred.

Returning to Table 1, we can consider the effect of different exploration mechanisms: ϵ-greedy, Boltzmann (or softmax) exploration, and Thompson sampling. Overall, Boltzmann exploration was the most effective strategy, gaining about 3% accuracy in POS tagging, just over 1% in dependency parsing, and just shy of 1% in noun phrase chunking. Although the latter two effects are small, they are statistically significant, which is measurable due to the fact that the evaluation sets are very large. In general, Thompson sampling is also effective, though worse than Boltzmann.

Finally, we consider a variant in which whenever BLS requests exploration, the algorithm "cheats" and chooses the gold standard decision at that point. This is the "oracle exploration" line in Table 1. We see that this does *not* improve overall quality, which suggests that a good exploration strategy is not one that always does the right thing, but one that also explored bad—but useful-to-learn-from—options.

3.5 Policy Gradient Updates

A natural question is: how does bandit structured prediction compare to more standard approaches to reinforcement learning (we revisit the question of how these problems differ in § 4). We chose Policy Gradient (Sutton et al., 1999) as a point of comparison. The main question we seek to address is how the BLS update rule compares to the Policy Gradient update rule. In order to perform this comparison, we hold the exploration strategy *fixed*, and implement the Policy Gradient update rule inside our system.

More formally, the policy gradient optimization is similar to that used in BLS. PG maintains a policy π_θ, which is parameterized by a set of parameters θ. Features are extracted from each state s_t to construct the feature vectors $\phi(s_t)$, and linear function approximation models the probability of selecting action a_t at state s_t under π_θ: $\pi_\theta(a_t|s_t) \propto \exp(\theta_{a_t}^T \phi(s_t))$, where K is the total number of actions. PG maximizes the total expected return under the distribution of trajectories sampled from the policy π_θ.

To balance the exploration / exploitation trade-off, we use exactly the same epsilon greedy technique used in BLS (Algorithm 1). For each trajectory τ sampled from π_θ, a state is selected uniformly at random, and an action is selected greedily with probability ϵ. The policy π_θ is used to construct the roll-in and roll-out trajectories. For every trajectory τ, we collect the same binary grades from the user as in BLS, and use them to train a regression function to estimate the per-step reward. These estimates are then be summed up to compute the total return G_t from time step t onwards (Algorithm 2).

We use standard policy gradient update for optimizing the policy θ based on the observed rewards:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log(\pi_\theta(s_t, a_t)) G_t \qquad (1)$$

The results of this experiment are shown in line G of Table 1. Here, we see that on POS tagging, where the number of actions is very large, PG significantly underperforms BLS. Our initial experiments in dependency parsing showed the PG significantly underperformed BLS after processing $\frac{1}{4}$ of the data. The difference is substantially smaller in chunking, where PG is on part with BLS with ϵ-greedy exploration. Figure 4 shows the effect of ϵ on PG, where we see that it also prefers large values of ϵ, but its performance saturates as $\epsilon \to 1$.

3.6 Bandit Feedback vs Full Feedback

Finally, we consider the trade-off between bandit feedback in BLS and full feedback. To make

this comparison, we run the fully supervised algorithm DAgger (Ross et al., 2010) which is effectively the same algorithm as LOLS and BLS under full supervision. In Table 1, we can see that full supervision dramatically improves performance from around 90% to 97% in POS tagging, 57% to 91% in dependency parsing, and 91% to 95% in chunking. Of course, achieving this improved performance comes at a high labeling cost: a human has to provide exact labels for each decision, not just binary "yes/no" labels.

4 Discussion & Conclusion

The most similar algorithm to ours is the bandit version of LOLS (Chang et al., 2015) (which is analyzed theoretically but not empirically); the key differences between BLS and LOLS are: (a) BLS employs a doubly-robust estimator for "guessing" the costs of counterfactual actions; (b) BLS employs alternative exploration strategies; (c) BLS is effective in practice at improving the performance of an initial policy.

In the NLP community, Sokolov et al. (2016a) and Sokolov et al. (2016b) have proposed a policy gradient-like method for optimizing log-linear models like conditional random fields (Lafferty et al., 2001) under bandit feedback. Their evaluation is most impressive on the problem of domain adaptation of a machine translation system, in which they show that their approach is able to learn solely from bandit-style feedback, though requiring a large number of samples.

In the learning-to-search setting, the difference between *structured prediction under bandit feedback* and *reinforcement learning* gets blurry. A distinction in the problem definition is that the world is typically assumed to be fixed and stochastic in RL, while the world is both deterministic and *known* (conditioned on the input, which is random) in bandit structured prediction: given a state and action, the algorithm always knows what the next state will be. A difference in solution is that there has been relatively little work in reinforcement learning that explicitly begins with a reference policy to improve and often assumes an *ab initio* training regime. In practice, in large state spaces, this makes the problem almost impossible, and practical settings like AlphaGo (Silver et al., 2016) require imitation learning to initialize a good policy, after which reinforcement learning is used to improve that policy.

Learning from partial feedback has generated a vast amount of work in the literature, dating back to the seminal introduction of multi-armed bandits by (Robbins, 1985). However, the vast number of papers on this topic does not consider joint prediction tasks; see (Auer et al., 2002; Auer, 2003; Langford and Zhang, 2008; Srinivas et al., 2009; Li et al., 2010; Beygelzimer et al., 2010; Dudik et al., 2011; Chapelle and Li, 2011; Valko et al., 2013) and references *inter alia*. There, the system observes (bandit) feedback for every decision.

Other forms of contextual bandits on structured problems have been considered recently. Kalai and Vempala (2005) studied the structured problem of online shortest paths, where one has a directed graph and a fixed pair of nodes (s, t). Each period, one has to pick a path from s to t, and then the times on all the edges are revealed. The goal of the learner is to improve it's path predictions over time. Relatedly, Krishnamurthy et al. (2015) studied a variant of the contextual bandit problem, where on each round, the learner plays a sequence of actions, receives a score for each individual action, and obtains a final reward that is a linear combination to those scores.

In this paper, we presented a computationally efficient algorithm for structured contextual bandits, BLS, by combining: locally optimal learning to search (to control the structure of exploration) and doubly robust cost estimation (to control the variance of the cost estimation). This provides the first practically applicable learning to search algorithm for learning from bandit feedback. Unfortunately, this comes at a cost to the user: they must make more fine-grained judgments of correctness than in a full bandit setting. In particular, they must mark each decision as correct or incorrect:it is an open question whether this feedback can be removed without incurring a substantially larger sample complexity. A second large open question is whether the time step at which to deviate can be chosen more intelligently, similar to selective sampling (Shi et al., 2015), using active learning.

Acknowledgements

We thank the anonymous reviewers for many helpful comments. This work was supported by NSF grant IIS-1618193 and LTS grant DO-0032. Opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the view of the sponsor(s).

References

Shipra Agrawal and Navin Goyal. 2013. Thompson sampling for contextual bandits with linear payoffs. In *ICML (3)*. pages 127–135.

Peter Auer. 2003. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research* 3:397–422.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multi-armed bandit problem. *SIAM Journal on Computing* 32(1):48–77.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*. pages 1171–1179.

Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert E Schapire. 2010. Contextual bandit algorithms with supervised learning guarantees. *arXiv preprint arXiv:1002.4058* .

Kai-Wei Chang, He He, Hal Daumé III, John Langford, and Stéphane Ross. 2016. A credit assignment compiler for joint prediction. In *NIPS*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to search better than your teacher. In *Proceedings of The 32nd International Conference on Machine Learning*. pages 2058–2066.

Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*. pages 2249–2257.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, page 111.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning* 75(3):297–325.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd international conference on Machine learning*. ACM, pages 169–176.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2014. Hc-search: A learning framework for search-based structured prediction. *J. Artif. Intell. Res.(JAIR)* 50:369–407.

Miroslav Dudík, Dumitru Erhan, John Langford, Lihong Li, et al. 2014. Doubly robust policy evaluation and optimization. *Statistical Science* 29(4):485–511.

Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. 2011. Efficient optimal learning for contextual bandits. *arXiv preprint arXiv:1106.2369* .

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the ACL* 1.

Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47(260):663–685.

Adam Kalai and Santosh Vempala. 2005. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences* 71(3):291–307.

Junpei Komiyama, Junya Honda, and Hiroshi Nakagawa. 2015. Optimal regret analysis of thompson sampling in stochastic multi-armed bandit problem with multiple plays. *arXiv preprint arXiv:1506.00779* .

Akshay Krishnamurthy, Alekh Agarwal, and Miroslav Dudik. 2015. Efficient contextual semi-bandit learning. *arXiv preprint arXiv:1502.05890* .

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data .

John Langford and Tong Zhang. 2008. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*. pages 817–824.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, pages 661–670.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *IWPT*. pages 149–160.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. Association for Computational Linguistics.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *CoNLL*.

Herbert Robbins. 1985. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, Springer, pages 169–177.

Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979* .

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. 2010. A reduction of imitation learning and structured prediction to no-regret online learning. *arXiv preprint arXiv:1011.0686* .

Tianlin Shi, Jacob Steinhardt, and Percy Liang. 2015. Learning where to sample in structured prediction. In *AISTATS*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Artem Sokolov, Julia Kreutzer, Christopher Lo, and Stefan Riezler. 2016a. Learning structured predictors from bandit feedback for interactive NLP. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/p16-1152.

Artem Sokolov, Stefan Riezler, and Tanguy Urvoy. 2016b. Bandit structured prediction for learning from partial feedback in statistical machine translation. *arXiv preprint arXiv:1601.04468* .

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* .

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. 2011. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS*. pages 725–733.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*. volume 99, pages 1057–1063.

Michal Valko, Nathaniel Korda, Rémi Munos, Ilias Flaounas, and Nelo Cristianini. 2013. Finite-time analysis of kernelised contextual bandits. *arXiv preprint arXiv:1309.6869* .

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960* .

Yuehua Xu, Alan Fern, and Sung Wook Yoon. 2007. Discriminative learning of beam-search heuristics for planning. In *IJCAI*. pages 2041–2046.