Variance Reduction for Structured Prediction with Bandit Feedback

Amr Sharaf
University of Maryland, College Park

Hal Daumé III
University of Maryland, College Park

Abstract

We present BANDITLOLS, an algorithm for learning to make joint predictions from bandit feedback. The learner repeatedly predicts a *sequence* of actions, corresponding to either a structured output or control behavior, and observes feedback for that single output and no others. To address this limited feedback, we design a structured cost-estimation strategy for predicting the costs of some unobserved structures. We demonstrate the practical importance of this strategy: without it, performance degrades dramatically over time due to high variance. Furthermore, we empirically compare a number of different exploration methods (ϵ-greedy, Boltzmann sampling, and Thompson sampling) and show the efficacy of the proposed algorithm. Our approach yields improved performance on three natural language processing tasks.

1 INTRODUCTION

The challenge of making a collection of predictions simultaneously and consistently arises in a variety of settings, most notably structured prediction (e.g., image segmentation, machine translation) and control (e.g., self-driving cars). For problems like these, it is often possible to build an initial, baseline system learned from a small amount of fully supervised data (parallel translations or driving demonstrations). Unfortunately, the performance of such a system is often limited to: the domain it was built on, the loss function it was optimized for, the small hypothesis class chosen (because of small amounts of data), an upper bound of performance based on the quality of the training data or demonstrator.

Our goal is to develop an algorithm that can take such a baseline system, and improve its performance over time based on interactions with the world. Because this improvement is through natural interactions, it is guaranteed to be in the right domain, and guaranteed to match real-world rewards. This also means the system will naturally improve over time, beyond the performance of the initial data. In particular, we assume a very weak form of *bandit feedback*: the system learns a *policy* that makes a joint prediction (segmentation, translation or trajectory) and receives a small amount of feedback about the quality of its prediction. Such bandit feedback is very limited: the system never gets to observe the "correct" output. This means that it faces a fundamental exploration/exploitation tradeoff, where the learning problem requires balancing reward maximization based on the knowledge already acquired (exploitation) with attempting new behavior to further increase knowledge (exploration).

We describe BANDITLOLS, an approach for improving joint predictors from bandit feedback. BANDITLOLS is an extention of the recently-proposed LOLS algorithm (Chang et al., 2015) for addressing the *structured contextual bandit* learning problem (§ 2). BANDITLOLS extends LOLS algorithm in two crucial ways. **First**, during an exploration phase, BANDITLOLS employs a doubly-robust strategy of estimating costs of unobserved outcomes in order to reduce variance (§ 3.1). In order to accomplish this, we learn a separate regressor for predicting unknown costs, which requires some additional feedback for estimation. Experimentally, we found this to be crucial (§4): without it, the bandit feedback often led to *decreased* rather than *increased* accuracy. **Second**, BANDITLOLS employs alternative forms of exploration, based on the predictions and uncertainties of the underlying policy (§ 3.2). We demonstrate the efficacy of these developments on several challenging natural language processing applications (§4). The experimental setup we consider is the online learning setting: what is the loss of the deployed system that faces (simulated) users as it balances exploration and exploitation. Our implementation will be made freely available.

Our approach connects to existing work (§5) in several ways. In the structured prediction setting, the setting we address is the *structured contextual bandit* framework introduced by Chang et al. (2015); in the control setting, it is closely related to the heuristic used by AlphaGo (Silver et al., 2016) in which imitation learning is used to initialize a good policy for playing Go, after which reinforcement learning is used to improve that policy. We jointly address both the structured prediction case and the control case by casting structured prediction as a learning to search problem (Daumé III et al., 2009), a view that has had great success recently in neural network sequence-to-sequence models (Bengio et al., 2015; Wiseman and Rush, 2016).

2 BACKGROUND

We operate in the learning to search framework, a style of solving joint prediction problems that subsumes both structured prediction and control. This family includes a number of specific algorithms including LaSO (Daumé III and Marcu, 2005; Xu et al., 2007; Wiseman and Rush, 2016), Searn (Daumé III et al., 2009), DAgger (Ross et al., 2010; Bengio et al., 2015), Aggrevate (Ross and Bagnell, 2014), LOLS (Chang et al., 2015), and others (Doppa et al., 2014). These approaches all decompose a joint prediction task into a sequence of smaller prediction tasks[1], which are tied by features and/or internal state.

Because these approaches decompose a joint prediction task into making a sequence of predictions, it becomes natural to leverage terminology and techniques from reinforcement learning. Learning to search approaches solve the fully supervised structured prediction problem by decomposing the production of the structured output in terms of an explicit search space (observations and actions); and learning hypotheses that control a policy that takes actions in this space. For example, in a neural machine translation setting (Bahdanau et al., 2014), in each step, the predictor observes the input sentence, the previously predicted word, and its own internal state; from this, it predicts the next word. In practice, this policy is typically implemented as a multi-class classifier, where it chooses an action (class) given an observation (features). After the prediction is complete, the world reveals a joint loss over the entire set of predictions.

The key learning challenge is the chicken-and-egg problem regarding measuring the quality of a policy's individual decisions. A good individual decision is one that, together with all the other individual decisions (past and future) made by that policy, yields a low joint loss. Different learning to search algorithms largely vary in terms of how they address this chicken-and-egg problem, typically through forms of iteration. This problem is shared in full reinforcement learning; however, *unlike* full reinforcement learning, learning to search approaches assume access to a **reference policy** that guides behavior at training time, significantly reducing (or sometimes entirely eliminating) the exploration/exploitation trade-off. This reference policy has historically been assumed to be optimal, in which case it is called an oracle policy[2].

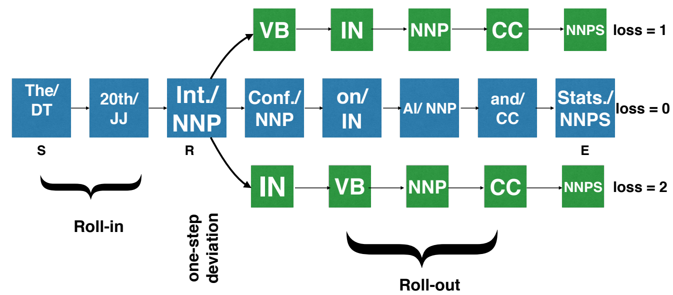

Figure 1: A search space implicitly defined by an imperative program. The system begins at the start state S and chooses the middle among three actions by the rollin policy twice. At state R it considers both the chosen action (middle) and both one-step deviations from that action (top and bottom). Each of these deviations is completed using the rollout policy until an end state is reached, at which point the loss is collected.

A common strategy for solving this chicken-and-egg problem is to consider a form of "one-step deviations." In particular, some policy is used to generate the prefix of a prediction (a "roll-in"), from which all possible actions are considered (one-step deviations), and the joint prediction is completed according to a roll-out policy (see Figure 1). This allows one to address the credit assignment problem (by holding everything fixed except the one-step deviation). Chang et al. (2015) theoretically and experimentally analyze several choices of roll-in and roll-out policies, many of which subsume existing algorithms. They advocate an online learning strategy, in which the current learned policy is used for roll-in. If the reference policy is known to be optimal, they advocate using the reference policy for roll-out (matching the Aggrevate algorithm); if not, they advocate using a mixture of the

[1]Although the decomposition is into a sequence of predictions, such approaches are not limited to "left-to-right" style prediction tasks. The standard way to approach broader classes of problems is to take a well known inference algorithm like belief propagation, linearize its behavior, and train predictors based on that linearization (Ross et al., 2010; Stoyanov et al., 2011).

[2]Occasionally the term "dynamic oracle" is used to refer to an "oracle"; we avoid this term because it is giving an unnecessary new name to an old concept.

Input: Dataset $\{x_i, y_i\}_{i=1}^{N}$ drawn from D and $\beta \geq 0$: a mixture parameter for roll-out.
Initialize a policy π_0;
for *all examples/episodes* $i = 1 \dots N$ **do**
  Generate a reference policy π^{ref} ;
  Initialize $\Gamma = \emptyset$;
  for $t = 0 \dots T-1$ **do**
    Roll-in by executing $\pi_i^{\text{in}} = \hat{\pi}_i$ for t steps and reach s_t;
    for *all* $a \in A(s_t)$ **do**
      Let $\pi_i^{\text{out}} = \pi^{\text{ref}}$ with probability β, otherwise $\hat{\pi}_i$;
      Evaluate cost $c_{i,t}(a)$ by rolling-out with π_i^{out} for $T-t-1$ steps and observing the final loss;
    end
    Generate a feature vector $\Phi(x_i, s_t)$;
    Set $\Gamma = \Gamma \cup \{\langle c_{i,t}, \Phi(x_i, s_t)\rangle\}$;
  end
  $\hat{\pi}_{i+1} \leftarrow Train(\hat{\pi}_i, \Gamma)$ (Update).
end
return *the average policy across* $\hat{\pi}_0, \hat{\pi}_1, \cdots \hat{\pi}_N$

Algorithm 1: Locally Optimal Learning to Search

learned policy and the reference policy for roll-out.

This yields LOLS, shown in Algorithm 1. In LOLS, the system follows policy $\pi \in \Pi$, which chooses an action $a \in A(s)$ at each non-terminal state s (corresponding to the observations at each time step). An action specifies the next state from s. A trajectory is a complete sequence of state/action pairs from the starting state to an end state e. Without loss of generality, we assume the lengths of trajectories are fixed and equal to T. The expected loss of a policy is the expected loss of the end state of the trajectory $e \sim \pi$, where $e \in E$ is an end state reached by following the policy.

LOLS assumes access to a cost-sensitive classification algorithm. A cost-sensitive classifier predicts a label $\hat{y}$ given an example x, and receives a loss $c_x(\hat{y})$, where c_x is a vector containing the cost for each possible label. For each decision point $t \in [0, T)$, LOLS executes π^{in} for t rounds reaching a state s_t. A cost-sensitive multiclass example is generated using the features $\Phi(x_i, s_t)$. Classes in the multiclass example correspond to available actions in state s_t. The cost $c(a)$ assigned to action a is the difference in loss, ℓ, between taking action a and the best action.

$$c(a) = \ell(y_{e(a)}, y_i) - \min_{a'} \ell(y_{e(a')}, y_i) \qquad (1)$$

where $e(a)$ is the end state reached with rollout by π^{out} after taking action a in state s_t. LOLS collects the T examples from the different roll-out points and feeds the set of examples Γ into an online cost-sensitive multiclass learner, thereby updating the learned policy from $\hat{\pi}_i$ to $\hat{\pi}_{i+1}$.

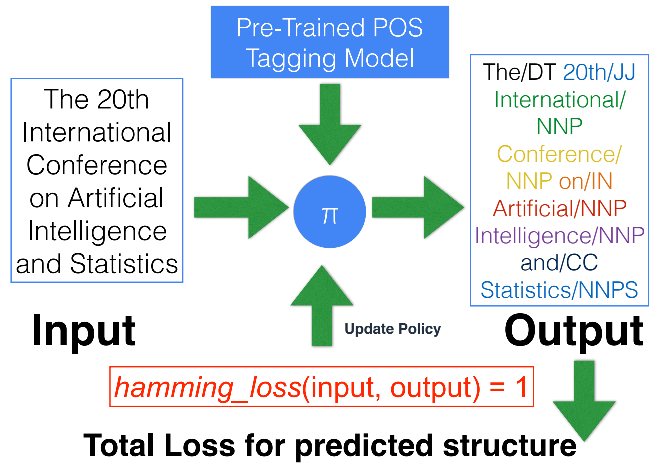

Figure 2: BanditLOLS for learning a POS tagging model: a pre-trained tagging model provides an initial policy π, which faces a user. The user views predicted tags and provides, for instance, the total hamming loss in the output. This is used to update the policy.

LOLS enjoys a compelling regret bound (see Theorem 3 and Corollary 1 of (Chang et al., 2015)), which ensures that if the underlying cost-sensitive learner is no-regret, then: (a) if the reference policy is optimal and in the hypothesis class, and $\beta = 1$, then the learned policy will have low regret with respect to the optimal policy; (b) if the reference policy is optimal and *not* in the hypothesis class, and $0 < \beta < 1$, then the learned policy will have low regret with respect to its own one-step deviations, a form of local optimality; (c) if the reference policy is very suboptimal, then the learned policy is guaranteed to improve on the reference.

3 BANDIT LOLS

One key property of LOLS, and many other learning to search algorithms, is that they assume that one can make *many* predictions on the same example/episode. In particular, if the trajectory length is T and the number of actions at each step is A, LOLS will evaluate the cost of AT different trajectories. In a bandit setting, we can evaluate precisely one. Figure 2 shows an overview for how BanditLOLS works for POS tagging. A pre-trained tagging model provides an initial policy π, which faces a user. The user views predicted tags and provides, for instance, the total hamming loss in the output. This is used to update the policy.

Chang et al. (2015) describe a variant of LOLS that requires only one evaluation, which they refer to as "Structured Contextual Bandit Learning." This approach modifies LOLS (Algorithm 1) in three ways:

1. Instead of evaluating deviations at *all* T time steps, a single time step t is chosen uniformly at random.
2. Instead of evaluating all A actions at t, only one action, a, is chosen. ϵ-greedy exploration is employed: with $1-\epsilon$ probability, the current policy is followed (exploitation), and with ϵ probability an action a is chosen uniformly at random.
3. Upon exploration, the cost vector $c_{i,t}(a')$ is set to 0 for all $a' \neq a$ and set to $K = |A(s_t)|$ times the observed loss for that trajectory (the K factor is compensation for importance sampling).

Although Chang et al. (2015) are able to obtain a regret bound for this bandit algorithm, we found that, in practice, it is ineffective. In fact, the bandit feedback often serves to make the predictions worse, not better, over time (see § 4)! Below we describe the two key issues with the algorithm, and at the end present an improved algorithm, BanditLOLS, which is effective in practice. The first issue is that in modification (3) above, the cost vector estimate remains unbiased, but has incredibly high variance (§ 3.1). The second issue is that in modification (2) above, an ϵ-greedy strategy is insufficient to explore the space well, and more nuanced exploration strategies are necessary (§ 3.2).

Input: current state: s_t, roll-in trajectory: τ, K regression functions (0ne for every action): ρ, set of allowed actions: $A(s_t)$, roll-out policy: π^{out}, explored action: a_t, observed cost: $e(a_t)$

$t \leftarrow |\tau|$;
Initialize $\hat{c}$: a vector of size $|A(s_t)|$;
roll_in_cost = 0;
for $(a, s) \in \tau$ **do**
  roll_in_cost $+=$ $\rho_a(\Phi(s))$;
end
for $a \in A(s_t)$ **do**
  if $a = a_t$ **then**
    $\hat{c}(a) = e(a_t)$;
  else
    action_cost = roll_in_cost;
    append action a to roll-in trajectory;
    roll-out with π^{out} for $T - t - 1$ steps;
    for (a', s') *in roll-out trajectory* **do**
      action_cost $+=$ $\rho_{a'}(\Phi(s'))$;
    end
    $\hat{c}(a)$ = action_cost;
end
return $\hat{c}$: *a vector of length* $|A(s_t)|$, *where* $\hat{c}(a)$ *is the estimated cost for the action* a *at state* s_t.

Algorithm 2: *estimate_cost*

3.1 Doubly-Robust Variance Reduction

To better understand the variance reduction issue, consider the part of speech tagging example from Figure 1. Suppose the deviation occurs at time step 3, as in that figure, and that during roll-in, the first two words are tagged correctly by the roll-in policy. 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 $\vec{c} = \langle 0, 0, 90, 0, \dots \rangle$. Clearly, in expectation over this random choice, we have $\mathbb{E}_a[c] = \langle 1, 0, 2, \dots \rangle$, implying unbiasedness, but the variance is clearly very large: $\mathcal{O}((Kc_{\max})^2)$.

This problem is exacerbated by the fact that LOLS, like other learning to search algorithms, typically defines the cost of an action a to be the *difference* between the cost of a and the minimum cost (see Eq. 1). 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 1 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.

To address this challenge, we adopt a strategy similar to the doubly-robust estimation used in the vanilla (non-structured) contextual bandit setting (Dudik et al., 2011). In particular, we estimate the cost function for each action using the predicted trajectories and use this estimate in place of the actual cost for the unobserved actions.

Algorithm 2 shows how this works. We assume access to a action-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.

This modification fixes *both* of the problems mentioned above. 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 solving the credit assignment issue.

The remaining question is: how to estimate these regressors. Currently, this comes at an additional cost to the user: we must observe per-action feedback[3]. In particular, when the user views a predicted output (e.g., part of speech sequence), we ask for a binary signal for each word whether the predicted part of speech 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 in many application settings. Using these T training examples, we can effectively train the 45 regression functions for estimating the cost of unobserved actions. Such regression estimators have provably low variance whenever the regression function is a good estimate of the true cost (an analysis is provided for the contextual bandit case in Dudik et al. (2011)).

3.2 Improved Exploration Strategies

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

3.2.1 Boltzmann (Softmax) Exploration

Although ϵ-greedy exploration is an effective and popular method for balancing the exploration / exploitation trade-off, exploration is performed by choosing an action equally among all the available actions. This means that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action. In tasks where there is a wide distinction between the different actions, this could be unsatisfactory.

Boltzmann exploration targets this problem by varying 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. In the limit as temp $\to 0$, softmax action selection becomes the same as greedy action selection.

[3]We leave open the question of how to remove this.

3.2.2 Thompson Sampling

Thompson Sampling has recently generated significant interest after several studies demonstrated it to have better empirical performance compared to other exploration strategies. Recent theoretical advances have also shown the effectiveness of this exploration approach (Agrawal and Goyal, 2013; Komiyama et al., 2015).

The general structure of Thompson sampling involves 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.

We use Gaussian likelihood function and Gaussian prior for the Thompson Sampling algorithm, which is a common choice in practice. 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 $\Phi(x_i, s_t)^T \mu_a$.

3.3 Putting it All Together

The complete BanditLOLS algorithm is shown in Algorithm 3, working roughly as follows. On each example it chooses whether to explore or exploit. This choice matters only for ϵ-greedy; all the other exploration methods always explore. Upon exploitation, it simply predicts accoding to the current policy. Otherwise, it picks a timestep uniformly at random, rolls-in to that time step using the learned policy, and allows the explorer to choose a one-step deviation. It then generates a rollout to comptute the cost of the chosen deviation. All other costs are estimated using the doubly robust regressors ρ. The underlying policy is updated according to a cost-sensitive classification update based on this full cost vector.

4 EXPERIMENTAL RESULTS

In this section, we show that BanditLOLS is able to improve upon a suboptimal reference policy under bandit feedback setting, where only a loss for a single predicted structure is observed. We conducted experiments using three exploration algorithms: ϵ-greedy, softmax (Boltzmann) exploration, and Thompson

Input: Examples $\{x_i\}_{i=1}^N$, reference policy π^{ref}, exploration algorithm *explorer*, and $\beta \geq 0$: a mixture parameter for roll-out.

Initialize a policy π_0 and set $\mathcal{I} = \Phi$;
Initialize cost estimator ρ;
for *all* $i \in \{1, 2, \cdots, N\}$ *(loop over each instance)* **do**
    Observe the example x_i;
    set $n_i = |\mathcal{I}|$;
    if *explore(explorer)* **then**
        Pick random time $t \in \{0, 1, \cdots, T-1\}$;
        Roll-in by executing $\pi_i^{\text{in}} = \hat{\pi}_{n_i}$ for t rounds to compute the roll-in trajectory τ and reach s_t;
        $a_t = \mathit{choose\_action}(\mathit{explorer}, s_t)$;
        Let $\pi_i^{\text{out}} = \pi^{\text{ref}}$ with probability β, else $\hat{\pi}_{n_i}$;
        Roll-out with π^{out} for $T - t - 1$ steps and observe total cost $e(a_t)$;
        train cost estimator ρ from observed trajectory;
        Estimate cost vector: $\hat{c} = \mathit{estimate\_cost}(s_t, \tau, \rho, A(s_t), \pi^{\text{out}}, a_t, e(a_t))$;
        Generate a feature vector $\Phi(x_i, s_t)$;
        $\hat{\pi}_{i+1} \leftarrow \mathit{Train}(\hat{\pi}_{n_i}, \hat{c}, \Phi(x_i, s_t))$ (Update);
        Augment $\mathcal{I} = \mathcal{I} \cup \{\hat{\pi}_{i+1}\}$
    else
        Follow the trajectory of a policy π drawn randomly from $\mathcal{I}$ to an end state e, predict the corresponding structured output y_{ie}.
    end
end

Algorithm 3: BanditLOLS

sampling. We report results on three natural language processing tasks: Part-of-speech tagging, dependency parsing, and noun phrase chunking.

4.1 Experimental Setup

In our experiments, we demonstrate the effectiveness of BanditLOLS in two ways:

1. We show that BanditLOLS can improve effectively upon a reference policy by observing only a partial feedback signal.
2. We show that BanditLOLS handles the exploration / exploitation trade-off effectively.

To capture both notions, we used an online learning setting for evaluation. We learn from one structured example at every time step, and we do a single pass over the available examples. Online evaluation using progressive validation is particularly suitable for measuring the effectiveness of the exploration algorithm, since the decision on whether to exploit or explore at earlier time steps will affect the performance on the observed examples in the future.

We extended the Vowpal Wabbit open source machine learning system (VW) (Langford et al., 2007) to include the BanditLOLS algorithm. We did not do any parameter tuning of the underlying learners. The regression problems are estimated using squared error regression, and the classification problems (policy learning) is solved via cost-sensitive one-against-all.

4.2 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 describe the policies in more detail), and describe the data sets that we use.

Part-Of-Speech Tagging is a sequence labeling task (see Figure 3, top), in which one assigns a part of speech tag (from a set of 45 possible tags defined by the Penn Treebank tagset (Marcus et al., 1993)) to each word in an input sequence. We treat this as a sequence labeling problem, and decompose the prediction into a greedy left-to-right policy. We use stardard features: words and their affixes in a window around the current word, together with the predicted tags for the most recent words. 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 (see Figure 3, middle). 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 dataset[4] for 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 mea-

[4] `http://www.cnts.ua.ac.be/conll2000/chunking/`

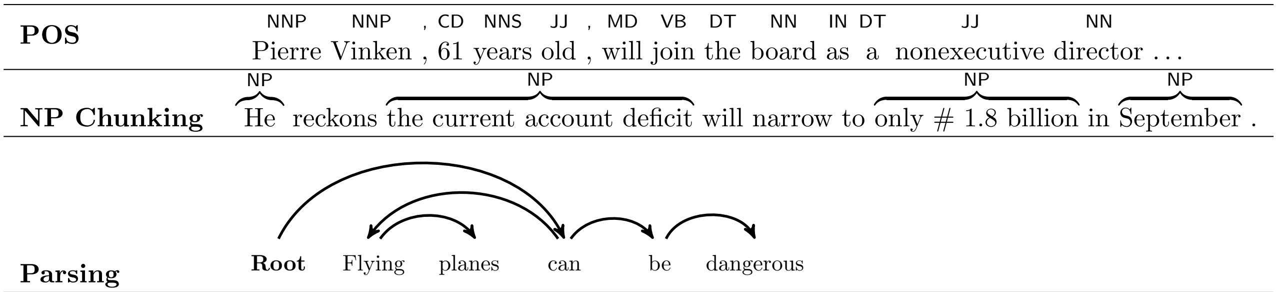

Figure 3: Example inputs (below) and desired outputs (above) for part of speech tagging task, noun phrase chunking, and dependency parsing.

Algorithm	Exploration	POS Accuracy	Dependency UAS	Chunking F-Score
Reference	-	47.24	44.15	74.73
LOLS	ϵ-greedy	2.29	18.55	31.76
BanditLOLS	ϵ-greedy	86.55	56.04	90.03
	Boltzmann	**89.62**	**57.20**	**90.91**
	Thompson	89.37	56.60	90.06

Table 1: Total progressive accuracies for the different algorithms on the three natural language processing tasks. LOLS uniformly *decreases* performance over the Reference baseline. BanditLOLS, which integrates cost regressors, uniformly improves, often quite dramatically. The overall effect of the exploration mechanism is small, but in all cases Boltzmann exploration statistically significantly better than the other options at the $p < 0.05$ level (because the same size is so large).

sured 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"). An example is shown in Figure 3, bottom. 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 evaluated in UAS (unlabeled attachment score), which measures the fraction of words that pick the correct parent.

4.3 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. In fact, for POS tagging, the LOLS accuracy falls *below* the accuracy one would get for random guessing (which is approximately 14% on this dataset for always guessing `NN`)!

When the underlying algorithm changes from LOLS to BanditLOLS, 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.

4.4 Effect of Epsilon

Figure 4 shows the effect of the choice of ϵ for ϵ-greedy exploration in BanditLOLS. Overall, best results are achieved with *remarkably* high epsilon, which is possibly counter-intuitive. The reason this happens is because BanditLOLS 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.

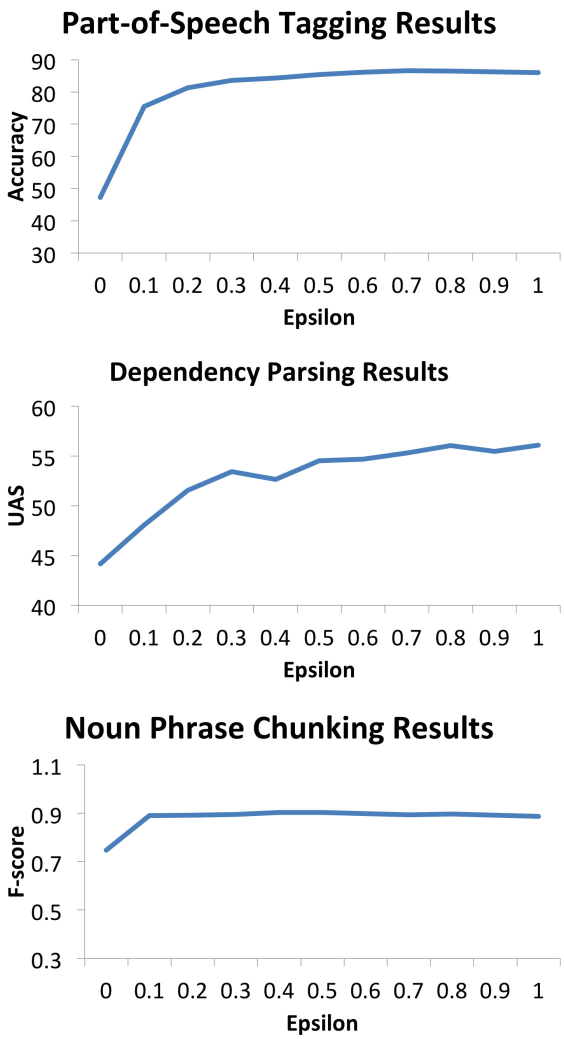

Figure 4: Analyzing the effect of ϵ in exploration/exploitation trade-off. Overall, large values of ϵ are strongly preferred.

4.5 Effect of Exploration Strategy

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 exploration.

5 DISCUSSION

Learning from partial feedback has generated a vast amount of work in the literature, dating back to the seminal introduction of multiarmed bandits by (Robbins, 1985). However, the vast number of papers on this topic do 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*. The system observes (bandit) feedback for every decision it makes. Other forms of contextual bandits on structured problems have been considered recently. For example, (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. This setting has applications to network routing, crowd-sourcing, personalized search, and many other domains, but differs from our setting in that we do not assume a linear (or other) connection between individual actions and the overall loss.

The most similar work to ours is that of (Sokolov et al., 2016a) and (Sokolov et al., 2016b). They propose a policy gradient-like method for optimizing log-linear models under bandit feedback. Although the approach works in practice, the gradient updates used appear to be biased, potentially leading to an algorithm that is inconsistent. They evaluated their approach most impressively to 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 the sample complexity is fairly large.

In this paper, we presented a computationally efficient algorithm for structured contextual bandits, Bandit-LOLS, 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 true bandit setting. In particular, they must mark each decision as correct or incorrect (notably, in the case of incorrect decisions, they do not provide a correction). 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, along the lines of selective sampling (Shi et al., 2015), using active learning techniques.

References

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

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

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

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

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. *arXiv preprint arXiv:1502.02206*, 2015.

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

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

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

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

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

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

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

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

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

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

John Langford, Lihong Li, and Alex Strehl. Vowpal wabbit online learning project, 2007.

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

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

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

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

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

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

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

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

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

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. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Artem Sokolov, Julia Kreutzer, Christopher Lo, and Stefan Riezler. 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), 2016a. doi: 10.18653/v1/p16-1152. URL `http://dx.doi.org/10.18653/v1/P16-1152`.

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

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

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

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

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

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