RESIDUAL LOSS PREDICTION: REINFORCEMENT LEARNING WITH NO INCREMENTAL FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider reinforcement learning and bandit structured prediction problems with very sparse loss feedback: only at the end of an episode. We introduce a novel algorithm, RESIDUAL LOSS PREDICTION (RESLOPE), that solves such problems by automatically learning an internal representation of a denser reward function. RESLOPE operates as a reduction to contextual bandits, using its learned loss representation to solve the credit assignment problem, and a contextual bandit oracle to trade-off exploration and exploitation. RESLOPE enjoys a no-regret reduction-style theoretical guarantee and outperforms state of the art reinforcement learning algorithms in both MDP environments and bandit structured prediction settings.

1 INTRODUCTION

Current state of the art learning-based systems require enormous, costly datasets on which to train supervised models. To progress beyond this requirement, we need learning systems that can interact with their environments, collect feedback (a loss or reward), and improve continually over time. In most real-world settings, such feedback is sparse and delayed: most decisions made by the system will not immediately lead to feedback. Any sort of interactive system like this will face at least two challenges: the credit assignment problem (which decision(s) did the system make that led to the good/bad feedback?); and the exploration/exploitation problem (in order to learn, the system must try new things, but these could be bad).

We consider the question of how to learn in an extremely sparse feedback setting: the environment operates episodically, and the only feedback comes at the *end* of the episode, with *no incremental feedback* to guide learning. This setting naturally arises in many classic reinforcement learning problems (§4): a barista robot will only get feedback from a customer after their cappuccino is finished[1]. It also arises in the context of bandit structured prediction (Sokolov et al., 2016; Chang et al., 2015) (§2.2), where a structured prediction system must produce a single output (e.g., translation) and observes only a scalar loss.

We introduce a novel reinforcement learning algorithm, RESIDUAL LOSS PREDICTION (RESLOPE) (§3), which aims to learn *effective representations of the **loss** signal*. By effective we mean effective in terms of credit assignment. Intuitively, RESLOPE attempts to learn a decomposition of the episodic loss into a sum of per-time-step losses. This process is akin to how a person solving a task might realize before the task is complete when and where they are likely to have made suboptimal choices. In RESLOPE, the per-step loss estimates are conditioned on all the information available up to the current point in time, allowing it to learn a highly non-linear representation for the episodic loss (assuming the policy class is sufficiently complex; in practice, we use recurrent neural network policies). When the system receives the final episodic loss, it uses the *difference* between the observed loss and the cumulative predicted loss to update its parameters.

Algorithmically, RESLOPE operates as a *reduction* (§3.3) to contextual bandits (Langford & Zhang, 2008), allowing the bandit algorithm to handle exploration/exploitation and focusing only on the credit assignment problem. RESIDUAL LOSS PREDICTION is theoretically motivated by the need for variance reduction techniques when estimating counterfactual costs (Dudík et al., 2014) and enjoys a

[1]This problem can be—and to a large degree *has* been—mitigated through the task-specific and complex process of reward engineering and reward shaping. Indeed, we were surprised to find that many classic RL algorithms fail badly when incremental rewards disappear. We aim to make such problems disappear.

no-regret bound (§3.3) when the underlying bandit algorithm is no-regret. Experimentally, we show the efficacy of RESLOPE on four benchmark reinforcement problems and three bandit structured prediction problems (§ 5.1), comparing to several reinforcement learning algorithms: Reinforce, Proximal Policy Optimization and Advantage Actor-Critic.

2 PROBLEM FORMULATION AND BACKGROUND

We focus on finite horizon, episodic Markov Decision Processes (MDPs) in this paper, which captures *both* traditional reinforcement learning problems (§ 4) *and* bandit structured prediction problems (§ 2.2). Our solution to this problem, RESIDUAL LOSS PREDICTION (§ 3) operates in a *reduction* framework. Specifically, we assume there exists "some" machine learning problem that we know how to solve, and can treat as an oracle. Our reduction goal is to develop a procedure that takes the reinforcement learning problem described above and map it to this oracle, so that a good solution to the oracle guarantees a good solution to our problem. The specific oracle problem we consider is a contextual bandit learning algorithm, relevant details of which we review in §2.1.

Formally, we consider a (possibly virtual) learning agent that interacts directly with its environment. The interaction between the agent and the environment is governed by a restricted class of finite-horizon Markov Decision Processes (MDP), defined as a tuple $\{\mathcal{S}, s_0, \mathcal{A}, \mathcal{P}, \mathcal{L}, H\}$ where: $\mathcal{S}$ is a large but finite state space, typically $\mathcal{S} \subset \mathbb{R}^d$; $s_0 \in \mathcal{S}$ is a start state; $\mathcal{A}$ is a finite action space[2] of size K; $\mathcal{P} = \{ \mathcal{P}(s'|s,a) : s, s' \in \mathcal{S}, a \in \mathcal{A} \}$ is the set of Markovian transition probabilities; $\mathcal{L} \in \mathbb{R}^{|\mathcal{S}|}$ is the state dependent loss function, defined only at terminal states $s \in \mathcal{S}$; H is the horizon (maximum length of an episode).

The goal is to learn a policy π, which defines the behavior of the agent in the environment. We consider policies that are potentially functions of entire trajectories[3], and potentially produce distributions over actions: $\pi(s) \in \Delta^{\mathcal{A}}$, where $\Delta^{\mathcal{A}}$ is the $\mathcal{A}$-dimensional probability simplex. However, to ease exposition, we will present the background in terms of policies that depend only on states; this can be accomplished by simply blowing up the state space.

Let d_h^π denote the distribution of states visited at time step h when starting at state s_0 and operating according to π: $d_{h+1}^\pi(s') = \mathbb{E}_{s_h \sim d_h^\pi, a_h \sim \pi(s_h)} \mathcal{P}(s' \mid s = s_h, a = a_h)$ The quality of the policy π is quantified by its value function or q-value function: $V^\pi(s) \in \mathbb{R}$ associates each state with the expected future loss for starting at this state and following π afterwards; $Q^\pi(s,a) \in \mathbb{R}$ associates each state/action pair with the same expected future loss: $V^\pi(s_h) = \mathbb{E}_{s_H \sim d_H^\pi \mid s_h} \mathcal{L}(s_H)$ and $Q^\pi(s_h, a_h) = \mathbb{E}_{s_H \sim d_H^\pi \mid s_h, a_h} \mathcal{L}(s_H)$ The learning goal is to estimate a policy π from a hypothesis class of policies Π with minimal expected loss: $J(\pi) = V^\pi(s_0)$.

2.1 CONTEXTUAL BANDITS

The contextual bandit learning problem (Langford & Zhang, 2008) can be seen as a tractable special case of reinforcement learning in which the time horizon $H = 1$. In particular, the world operates episodically. At each round t, the world reveals a context $\boldsymbol{x}_t \in \mathcal{X}$; the system chooses an action a_t; the world reveals a scalar loss $\ell_t(\boldsymbol{x}_t, a_t) \in \mathbb{R}^+$, a loss *only* for the selected action that may depend stochastically on $\boldsymbol{x}_t$ and a_t. The total loss for a system over T rounds is the sum of losses: $\sum_{t=1}^T \ell_t(\boldsymbol{x}_t, a_t)$. The goal in policy optimization is to learn a policy $\pi : \boldsymbol{x} \to \mathcal{A}$ from a policy class Π that has low *regret* with respect to the best policy in this class. Assuming the learning algorithm produces a sequence of policies $\pi_1, \pi_2, \ldots, \pi_T$, its regret is: *Regret* $\left(\langle \pi_t \rangle_{t=1}^T\right) = \sum_{t=1}^T \ell(\boldsymbol{x}_t, \pi_t(\boldsymbol{x}_t)) - \min_{\pi^* \in \Pi} \sum_{t=1}^T \ell(\boldsymbol{x}_t, \pi^*(\boldsymbol{x}_t))$. The particular contextual bandit algorithms we will use in this paper perform a second level of reduction: they assume access to an oracle supervised learning algorithm that can optimize a cost-sensitive loss (Appendix C), and transform the contextual bandit problem to a cost-sensitive classification problem. Algorithms in this family typically vary along two axes: how to explore (faced with a new $\boldsymbol{x}$ how does the algorithm choose which action to take); and how to update (Given the observed loss ℓ_t, how does the algorithm construct a supervised training example on which to train). More details are in Appendix A.

[2]In some problems the set of actions available will depend on the current state.

[3]Policies could choose to ignore all but the most recent state, for instance in fully observable environments, though this may be insufficient in partially observable environments (Littman & Sutton, 2002).

2.2 Bandit Structured Prediction via Learning to Search

In structured prediction, we observe structured input sequences $\boldsymbol{x}^{\text{SP}} \in \mathcal{X}$ and the goal is to predict a set of correlated output variables $\boldsymbol{y}^{\text{SP}} \in \mathcal{Y}$. For example, in machine translation, the input $\boldsymbol{x}^{\text{SP}}$ is a sentence in an input language (e.g., Tagalog) and the output $\boldsymbol{y}^{\text{SP}}$ is a sentence in an output language (e.g., Chippewa). In the fully supervised setting, we have access to samples $(\boldsymbol{x}^{\text{SP}}, \boldsymbol{y}^{\text{SP}})$ from some distribution $\mathcal{D}$ over input/output pairs. Structured prediction problems typically come paired with a structured loss $\ell(\boldsymbol{y}^{\text{SP}}, \hat{\boldsymbol{y}}^{\text{SP}}) \in \mathbb{R}^+$ that measures the fidelity of a predicted output $\hat{\boldsymbol{y}}^{\text{SP}}$ to the "true" output $\boldsymbol{y}^{\text{SP}}$. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ with low expected loss under $\mathcal{D}$: $\mathbb{E}_{(\boldsymbol{x}^{\text{SP}}, \boldsymbol{y}^{\text{SP}}) \sim \mathcal{D}} \ell(\boldsymbol{y}^{\text{SP}}, f(\boldsymbol{x}^{\text{SP}}))$. Recently, it has become popular to solve structured prediction problems incrementally using some form of recurrent neural network (RNN) model. When the output $\boldsymbol{y}^{\text{SP}}$ contains multiple parts (e.g., words in a translation), the RNN can predict each word in sequence, conditioning each prediction on all previous decisions. Although typically such models are trained to maximize cross-entropy with the gold standard output (in a fully supervised setting), there is mounting evidence that this has similar drawbacks to pre-RNN techniques, such as overfitting to gold standard prefixes (the model never learns what to do once it has made an error) and sensitivity to errors of different severity (due to error compounding). In order to achieve this we must formally map from the structured prediction problem to the MDP setting; this mapping is natural and described in detail in Appendix B.

Our focus in this paper is on the recently proposed *bandit* structured prediction setting (Chang et al., 2015; Sokolov et al., 2016), at training time, we only have access to input $\boldsymbol{x}^{\text{SP}}$ from the marginal distribution $\mathcal{D}^{\mathcal{X}}$. For example, a Chippewa speaker sees an article in Tagalog, and asks for a translation. A system then produces a *single* translation $\hat{\boldsymbol{y}}^{\text{SP}}$, on which a single "bandit" loss $\ell(\hat{\boldsymbol{y}}^{\text{SP}} \mid \boldsymbol{x}^{\text{SP}})$ is observed. Given only this bandit feedback, without ever seeing the "true" translation, the system must learn.

3 Proposed Approach

Our goal is to learn a good policy in a Markov Decision Process (§2) in which losses only arrive at the end of episodes. Our solution, Residual Loss Prediction (Reslope), automatically deduces per-step losses based only on the episodic loss. To gain an intuition for how this works, suppose you are at work and want to meet a colleague at a nearby coffee shop. In hopes of finding a more efficient path to the coffee shop, you take a different path than usual. While you're on the way, you run into a friend and talk to them for a few minutes. You then arrive at the coffee shop and your colleague tells you that you are ten minutes late. To estimate the value of the different path, you wonder: how much of this ten minutes is due to taking the different path vs talking to a friend. If you can accurately estimate that you spent seven minutes talking to your friend (you lost track of time), you can conclude that the disadvantage for the different path is three minutes.

Reslope addresses the problem of **sparse reward signals** and **credit assignment** by learning a decomposition of the reward signal, essentially doing automatic reward shaping (evaluated in §5.3). Finally, it addresses the problem of **exploration vs exploitation** by relying on a strong underlying contextual bandit learning algorithm with provably good exploration behavior.

3.1 Key Idea: Residual Loss Prediction

Akin to the coffee shop example, Reslope learns a decomposition of the episodic loss (i.e total time spent from work to the coffee shop) into a sum of per-time-step losses (i.e. timing activities along the route). Reslope operates as a reduction from reinforcement learning with episodic loss to contextual bandits. In this way, Reslope solves the *credit assignment* problem by predicting residual losses, and relies on the underlying contextual bandit oracle to solve explore/exploit. Reslope operates online, incrementally updating a policy π^{learn} once per episode. In the structured contextual bandit setting, we assume access to a *reference policy*, π^{ref}, that was perhaps pretrained on supervised data, and which we wish to improve; a hyperparameter β controls how much we trust π^{ref}. As π^{learn} improves, we replace π^{ref} with π^{learn}. In the RL setting, we set $\beta = 0$.

We initially present a simplified variant of Reslope that mostly follows the learned policy (and the reference policy as appropriate), except for a *single* deviation per episode. This algorithm closely

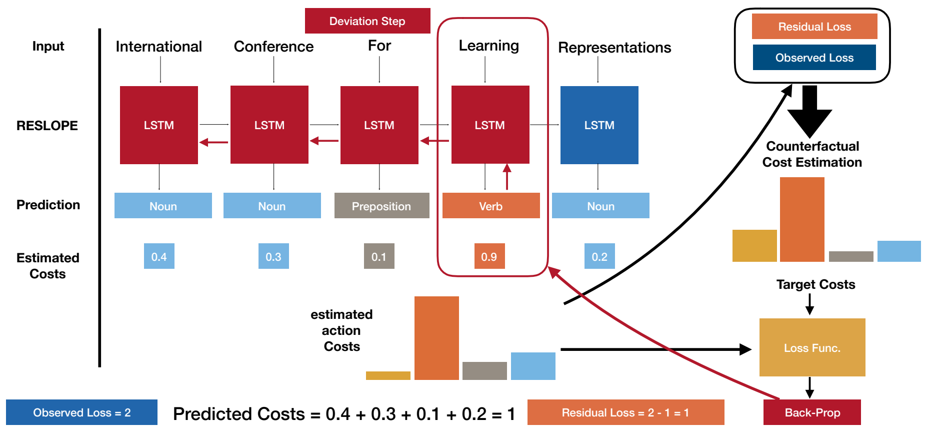


Figure 1: RESIDUAL LOSS PREDICTION: the system assigns a part-of-speech tag sequence to the sentence "International Conference for Learning Representations". Each state represents a partial labeling. The end state $e = [\text{Noun}, \text{Noun}, \text{Preposition}, \text{Verb}, \text{Noun}]$. The end state e is associated with an episodic loss $\ell(e)$, which is the total hamming loss in comparison to the optimal output structure $e^* = [\text{Adjective}, \text{Noun}, \text{Preposition}, \text{Noun}, \text{Noun}]$. We emphasize that our algorithm doesn't assume access to neither the optimal output structure, nor the hamming loss for every time step. Only the total hamming loss is observed in this case ($\ell(e) = 2$).

follows the bandit version of the Locally Optimal Learning to Search (LOLS) approach of Chang et al. (2015), with three key differences: (1) residual loss prediction; (2) alternative exploration strategies; (3) alternative parameter update strategies. We assume access to a contextual bandit oracle CB that supports the following API:

1. CB.ACT($\pi^{\text{learn}}, \boldsymbol{x}$), where $\boldsymbol{x}$ is the input example; this returns a tuple (a, p), where a is the selected action, and p is the probability with which that action was selected.
2. CB.COST($\pi^{\text{learn}}, \boldsymbol{x}, a$) returns the estimated cost of taking action a in the context.
3. CB.UPDATE($\pi^{\text{learn}}, \boldsymbol{x}, a, p, c$), where $\boldsymbol{x}$ is the input example, $a \in [K]$ is the selected action, $p \in (0, 1]$ is the probability of that action, and $c \in \mathbb{R}$ is the target cost.

The requirement that the contextual bandit algorithm also predicts costs (CB.COST) is somewhat non-standard, but is satisfied by many contextual bandit algorithms in practice, which often operate by regressing on costs and picking the minimal predicted cost action. We describe the specific contextual bandit approaches we use in §3.2.

Algorithm 1 shows how our reduction is constructed formally. It uses a MAKEENVIRONMENT(t) function to construct a new environment (randomly in RL and by selecting the tth example in bandit structured prediction). To learn a good policy, RESLOPE reduces long horizon trajectories to single-step contextual bandit training examples. In each episode, RESLOPE picks a single time step to deviate. Prior to the deviation step, it executes π^{learn} as a roll-in policy and after the deviation step, it executes a β mixture of π^{learn} and π^{ref}. *At* the deviation step, it calls CB.ACT to handle the exploration and choose an action. At *every* step, it calls CB.COST to estimate the cost of that action. Finally, it constructs a single contextual bandit training example for the deviation step, whose input was the observation at that step, whose action and probability are those that were selected by CB.ACT, and whose cost is the observed total cost *minus* the cost of every *other* action taken in this trajectory. This example is sent to CB.UPDATE. When the contextual bandit policy is an RNN (as in our setting), this will then compute a loss which is back-propagated through the RNN.

Interaction between RESLOPE and the environment takes place via the RESLOPEPOLICY procedure, which is given state and returns the action a predicted by RESLOPE. Internally, RESLOPEPOLICY performs a standard learning-to-search prediction step. A mixture policy is used to generate the roll-out trajectories (Figure 5). The mixture policy either executes π^{ref} with probability β or π^{learn} with probability $1 - \beta$. At the deviation step, the contextual bandit exploration algorithm is used to pick an action (§3.2). Regardless of how the action is chosen, the action's cost is estimated using the contextual bandit oracle.

Algorithm 1 RESIDUAL LOSS PREDICTION (RESLOPE) with *single* deviations

```
Require: Reference policy $\pi^{\text{ref}}$, mixture parameter $\beta \in [0,1]$, contextual bandit oracle CB,
    MAKEENVIRONMENT to build new enviornments
 1: Initialize a policy $\pi_0^{\text{learn}}$ {either randomly or from a pretrained model}
 2: for all episodes $t = 1 \dots T$ do
 3:     env ← MAKEENVIRONMENT($t$)
 4:     Initialize variables: example $\boldsymbol{x}^{\text{dev}}$, action $a^{\text{dev}}$, probability $p^{\text{dev}}$
 5:     Initialize cost vector $\hat{c}_h^{\text{dev}} = 0$ for $h = 1 \dots \text{env}.H$
 6:     Choose deviation step $h^{\text{dev}} \leftarrow \text{UNIFORM}(\text{env}.H)$
 7:     Choose rollout policy $\pi^{\text{mix}}$ to be $\pi^{\text{ref}}$ with probability $\beta$ or $\pi_{t-1}^{\text{learn}}$ with probability $1-\beta$
 8:     for all time steps $h = 1 \dots \text{env}.H$ do
 9:         $\boldsymbol{x} \leftarrow \text{env.STATEFEATURES}$ {computed by an RNN}
10:         if $h \neq h^{\text{dev}}$ { no deviation } then
11:             $a \leftarrow \begin{cases} \pi_{t-1}^{\text{learn}}(\boldsymbol{x}) & \text{if } h < h^{\text{dev}} \\ \pi^{\text{mix}}(\boldsymbol{x}) & \text{if } h > h^{\text{dev}} \end{cases}$
12:         else if $h = h^{\text{dev}}$ { deviation } then
13:             $(a^{\text{dev}}, p^{\text{dev}}) \leftarrow \text{CB.ACT}(\pi^{\text{learn}}, \boldsymbol{x})$
14:             $\boldsymbol{x}^{\text{dev}} \leftarrow \boldsymbol{x}$
15:             $a \leftarrow a^{\text{dev}}$
16:         end if
17:         $\hat{c}_h^{\text{dev}} \leftarrow \text{CB.COST}(\pi_{t-1}^{\text{learn}}, \boldsymbol{x}, a)$
18:         env.STEP(a) {updates environment and internal state of the RNN }
19:     end for
20:     $\ell^{\text{residual}} \leftarrow \text{env.FINALLOSS} - \sum_{h \neq h^{\text{dev}}} \hat{c}_h^{\text{dev}}$
21:     $\pi_t^{\text{learn}} \leftarrow \text{CB.UPDATE}(\pi_{t-1}^{\text{learn}}, \boldsymbol{x}^{\text{dev}}, a^{\text{dev}}, p^{\text{dev}}, \ell^{\text{residual}})$
22: end for
23: Return average policy $\bar{\pi} = \frac{1}{T}\sum_t \pi_t^{\text{learn}}$
```

3.2 CONTEXTUAL BANDIT ORACLE

The contextual bandit oracle receives examples where the cost for only one predicted action is observed, but no others. It learns a policy for predicting actions minimizing expected loss by estimating the unobserved target costs for the unpredicted actions and exploring different actions to balance the exploitation exploration trade-off (§ 3.2). The contextual bandit oracle then calls a cost-sensitive multi-class oracle (Appendix C) given the target costs and the selected action.

CB.UPDATE: Cost Estimation Techniques. The update procedure for our contextual bandit oracles takes an example $\boldsymbol{x}$, action a, action probability p and cost c as input and updates its policy. We do this by reducing to a cost-sensitive classification oracle (Appendix C), which expects an example $\boldsymbol{x}$ and a cost vector $\boldsymbol{y} \in \mathbb{R}^K$ that specifies the cost for *all* actions (not just the selected one). The reduction challenge is constructing this cost-sensitive classification example given the input to CB.UPDATE. We consider three methods: inverse propensity scoring (Horvitz & Thompson, 1952), doubly robust estimation (Dudík et al., 2014) and multitask regression (Langford & Agarwal, 2017).

Inverse Propensity Scoring (IPS): IPS uses the selected action probability p to correct for the shift in action proportions predicted by the policy π^{learn}. IPS estimates the target cost vector $\boldsymbol{y}$ as: $\boldsymbol{y}(i) = \frac{c}{p}\mathbf{1}[i = a]$, where $\mathbf{1}$ is an indicator function and where a is the selected action and c is the observed cost. While IPS yields an unbiased estimate of costs, it typically has a large variance as $p \to 0$.

Doubly Robust Cost Estimation (DR): The doubly robust estimator uses both the observed cost c as well as its own predicted costs $\hat{y}(i)$ for *all* actions, forming a target that combines these two sources of information. DR estimates the target cost vector $\boldsymbol{y}$ as: $\boldsymbol{y}(i) = \hat{y}(i) + \mathbf{1}[i = a](c - \hat{y}(i))/p$. The DR estimator remains unbiased, and the estimated loss $\boldsymbol{y}$ helps decrease its variance.

Multitask Regression (MTR): The multitask regression estimator functions somewhat differently from IPS and DR. Instead of reducing to to cost-sensitive classification, MTR reduces directly to importance-weighted regression. MTR maintains K different regressors for predicting costs given input/action pairs. Given $\boldsymbol{x}, a, c, p$, MTR constructs a *regression* example, whose input is $(\boldsymbol{x}, a)$, whose target output is c and whose importance weight is $1/p$.

CB.ACT: Exploration Strategies. We experiment with three exploration strategies:

Uniform: explores randomly with probability ϵ and otherwise acts greedily (Sutton & Barto, 1998).

Boltzmann: varies action probabilities where action a is chosen with probability proportional to $\exp[-c(a)/\text{temp}]$, where $\text{temp} \in \mathbb{R}^+$ is the temperature, and $c(a)$ is the predicted cost of action a.

Bootstrap Exploration: (Agarwal et al., 2014) trains a bag of multiple policies simultaneously. Each policy in the bag votes once on its predicted action, and an action is sampled from this distribution. To train, each example gets passed to each policy Poisson($\lambda = 1$)-many times, which ensures diversity . Bootstrap can operate in "greedy update" and "greedy prediction" mode (Bietti et al., 2017). In greedy update, we always update the first policy in the bag exactly once. In greedy prediction, we always predict the action from the first policy during exploitation.

3.3 Theoretical Analysis

For simplicity, we first consider the case where we have access to a good reference policy π^{ref} but do not have access to good Q-value estimates under π^{ref}. The only way one can obtain a Q-value estimate is to do a roll-out, but in a non-resettable environment, we can only do this once. We will subsequently consider the case of suboptimal (or missing) reference policies, in which the goal of the analysis will change from competing with π^{ref} to competing with both π^{ref} and a local optimality guarantee. We can show the following (proof in Appendix D).

Theorem 1. *If $\beta = 1$, running* RESLOPE *for N episodes with a contextual bandit algorithm that has regret $\epsilon_{CB}(N)$, the average returned policy $\bar{\pi} = \mathbb{E}_n \pi_n$ has regret equal to the suboptimality of π^{ref}, namely: Regret$(\bar{\pi}) \leq$ Regret$(\pi^{ref}) + \frac{1}{N}\epsilon_{CB}(N)$.*

In the case that π^{ref} is not known to be optimal, or not available, we follow the LOLS analysis and obtain a regret to a convex combination of π^{ref} and the learned policy's one-step deviations (a form of local optimality) and can additionally show the following (proof in Appendix E):

Theorem 2. *For arbitrary β, define the combined regret of $\bar{\pi}$ as: Regret$_\beta(\bar{\pi}) = \beta[J(\bar{\pi}) - J(\pi^{ref})] + (1-\beta)\sum_h [J(\bar{\pi}) - \min_{\pi \in \Pi} \mathbb{E}_{s \sim d^h_{\bar{\pi}}} Q^{\bar{\pi}}(s, \pi)]$. The first term is suboptimality to π^{ref}; the second term is suboptimality to the policy's own realizable one-step deviations. Given a contextual bandit learning algorithm, the combined regret of $\bar{\pi}$ satisfies: Regret$_\beta(\bar{\pi}) \leq \frac{1}{N}\epsilon_{CB}(N)$.*

If the contextual bandit algorithm is no regret, then $\epsilon_{\text{CB}}/N \to 0$ as $N \to \infty$.

3.4 Multi-deviation Residual Loss Prediction

Finally, we present the multiple deviation variant of RESLOPE. Algorithm 2 shows how RESLOPE operates under multiple deviations. The difference between the single and multiple deviation mode is twofold: 1. Instead of deviating at a single time step, *multi*-dev RESLOPE performs deviations at each time step in the horizon; 2. Instead of generating a single contextual bandit example per episode, *multi*-dev RESLOPE generates H examples, where H is the length of the time horizon, effectively updating the policy H times.

These two changes means that we update the learned policy π^{learn} multiple times per episode. Empirically, we found this to be crucial for achieving superior performance. Although, the generated samples for the same episode are not independent, this is made-up for by the huge increase in the number of available samples for training (i.e. $T \times H$ samples for multiple deviations versus only T samples in the single deviation mode). The theoretical analysis that precedes still holds in this case, but only makes sense when $\beta = 0$ because there is no longer any distinction between roll-in and roll-out, and so the guarantee reduces to a local optimality guarantee.

4 Experimental Setup

We conduct experiments on both reinforcement learning and structured prediction tasks. Our goal is to evaluate how quickly different learning algorithms learn from episodic loss. We implement our models on top of the DyNet neural network optimization package (Neubig et al., 2017). The code will be made available upon publication of this work.

Algorithm 2 RESIDUAL LOSS PREDICTION (RESLOPE) with *multiple* deviations

```
Require: Contextual bandit oracle CB, MAKEENVIRONMENT to build new enviornments
 1: Initialize a policy $\pi_0^{\text{learn}}$ {either randomly or from a pretrained model}
 2: for all episodes $t = 1 \dots T$ do
 3:     env ← MAKEENVIRONMENT($t$)
 4:     Initialize variables: examples $\boldsymbol{x}_h^{\text{dev}}$, actions $a_h^{\text{dev}}$, probabilities $p_h^{\text{dev}}$
            and costs $\hat{c}_h^{\text{dev}} = 0$ for $h = 1 \dots \text{env}.H$
 5:     for all time steps $h = 1 \dots \text{env}.H$ do
 6:         $\boldsymbol{x}_h^{\text{dev}} \leftarrow$ env.STATEFEATURES {computed by an RNN}
 7:         $(a_h^{\text{dev}}, p_h^{\text{dev}}) \leftarrow$ CB.ACT($\pi^{\text{learn}}, \boldsymbol{x}_h^{\text{dev}}$)
 8:         $\hat{c}_h^{\text{dev}} \leftarrow$ CB.COST($\pi_{t-1}^{\text{learn}}, \boldsymbol{x}_h^{\text{dev}}, a_h^{\text{dev}}$)
 9:         env.STEP($a_h^{\text{dev}}$) {updates environment and internal state of the RNN }
10:     end for
11:     $\ell_h^{\text{residual}} \leftarrow$ env.FINALLOSS $- \sum_{h' \neq h} \hat{c}^{\text{dev}}(h')$ for all $h$
12:     $\pi_t^{\text{learn}} \leftarrow$ CB.UPDATE($\pi_{t-1}^{\text{learn}}, \boldsymbol{x}_h^{\text{dev}}, a_h^{\text{dev}}, p_h^{\text{dev}}, \ell_h^{\text{residual}}$) for all $h$
13: end for
14: Return average policy $\bar{\pi} = \frac{1}{T}\sum_t \pi_t^{\text{learn}}$
```

Reinforcement Learning Environments We perform experiments in four standard reinforcement learning environments: Blackjack (classic card game), Hex (two-player board game), Cartpole (aka "inverted pendulum") and Gridworld. Our implementations of these environments are described in Appendix F and largely follows the AI Gym (Brockman et al., 2016) implementations. We report results in terms of *cumulative loss*, where loss is $-1\times$*reward* for consistency with the loss-based exposition above and the loss-based evaluation of bandit structured prediction (§2.2).

Bandit Structured Prediction Environments We also conduct experiments on structured prediction tasks. The evaluation framework we consider is the fully online setup described in (§2.2), measuring the degree to which various algorithms can effectively improve by observing only the episodic loss, and effectively balancing exploration and exploitation. We learn from one structured example at a time and we do a single pass over the available examples. We measure performance in terms of average cumulative loss on the online examples as well as on a held-out evaluation dataset. The loss on the online examples measures how much the algorithm is penalized for unnecessary exploration. We perform experiments on the three tasks described in detail in Appendix G: English Part of Speech Tagging, English Dependency Parsing and Chinese Part of Speech Tagging.

4.1 COMPARATIVE ALGORITHMS

We compare against three common reinforcement learning algorithms: Reinforce (Williams, 1992) with a baseline whose value is an exponentially weighted running average of rewards; Proximal Policy Optimization (PPO) (Schulman et al., 2017); and Advantage Actor-Critic (A2C) (Mnih et al., 2016). For the structured prediction experiments, since the bandit feedback is simulated based on labeled data, we can also estimate an "upper bound" on performance by running a supervised learning algorithm that uses full information (thus forgoing issues of both exploration/exploitation and credit assignment). We run supervised DAgger to obtain such an upper bound.

4.2 POLICY ARCHITECTURE

In all cases, our policy is a recurrent neural network (Elman, 1990) that maintains a real-valued hidden state and combines: (a) its previous hidden state, (b) the features from the environment (described for each environment in the preceding sections), and (c) an embedding of its previous action. These form a new hidden state, from which a prediction is made. Formally, at time step h, $\boldsymbol{v}_h$ is the hidden state representation, $f(\text{state}_h)$ are the features from the environment and a_h is the action taken. The recursion is:

$$\boldsymbol{v}_0 = \mathbf{const} \qquad ; \qquad \boldsymbol{v}_{h+1} = \textit{ReLU}\left(\mathbf{A}\left[\ \boldsymbol{v}_h\ ,\ \ \boldsymbol{f}(\text{state}_h)\ ,\ \ \mathbf{emb}(a_h)\ \right]\right) \tag{1}$$

Here, $\mathbf{A}$ is a learned matrix, **const** is an initial (learned) state, **emb** is a (learned) action embedding function, and *ReLU* is a rectified linear unit applied element-wise.

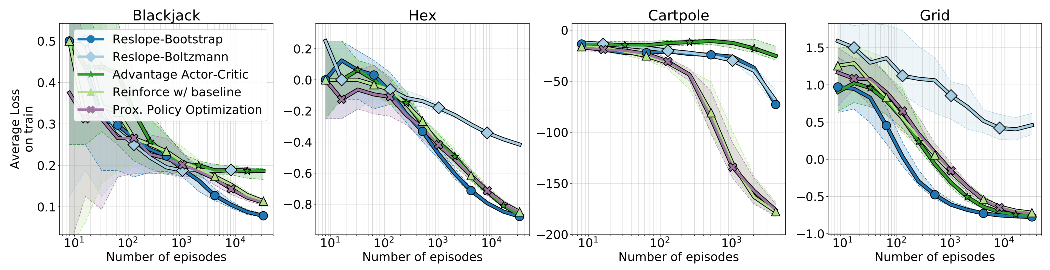


Figure 2: Average loss during learning on the four RL problems. Shaded regions are empirical quartiles over the experimental replicates with different random seeds.

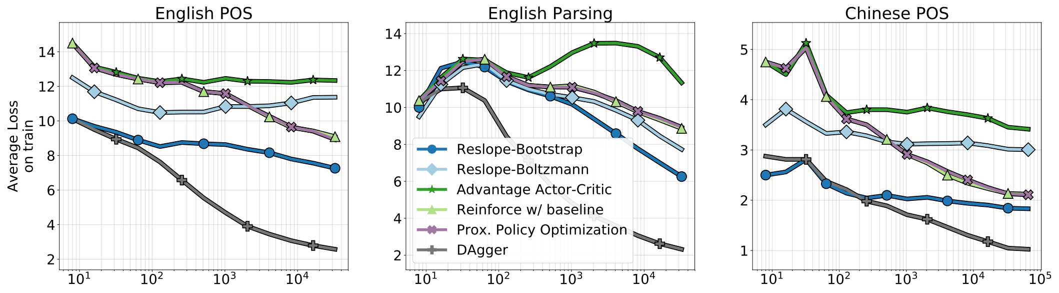


Figure 3: Average loss during learning for three bandit structured prediction problems. Also included are supervised learning results with DAgger.

Given the hidden state $\boldsymbol{v}_h$, an action must be selected. This is done using a simple feedforward network operating on $\boldsymbol{v}_h$ with either no hidden layers (in which case the output vector is $\boldsymbol{o}_h = \mathbf{B}\boldsymbol{v}_h$) or a single hidden layer (where $\boldsymbol{o}_h = \mathbf{B}_2\, \mathit{ReLU}(\mathbf{B}_1 \boldsymbol{v}_h)$). In the case of RESLOPE and DAgger, which expect *cost estimates* as the output of the policy, the output values $\boldsymbol{o}_h$ are used as the predicted costs (and a_h might be the argmin of these costs when operating greedily). In the case of Reinforce, PPO and A2C, which expect action probabilities, these are computed as $\mathit{softmax}(-\boldsymbol{o}_h)$ from which, for instance, an action a_h is sampled.

Details on optimization, hyperparameters and "deep learning tricks" are reported in Appendix H.

5 Experimental Results

We study several questions empirically: 1. How does RESIDUAL LOSS PREDICTION compare to policy gradient methods on reinforcement learning and bandit structured prediction tasks? (§5.1) 2. What's the effect of ablating various parts of the RESLOPE approach, including multiple deviations? (§5.2) 3. Does RESLOPE succeed in learning a good representation of the loss? (§5.3)

5.1 Reinforcement Learning and Bandit Structured Prediction Results

In our first set of experiments, we compare RESLOPE to the competing approaches on the four reinforcement learning tasks described above. Figure 2 shows the results. In Blackjack, Hex and Grid, RESLOPE outperforms all the competing approaches with lower loss earlier in the learning process (though for Hex and Grid they all finish at the same near-optimal policy). For Cartpole, RESLOPE significantly *underperforms* both Reinforce and PPO.[4] Furthermore, in both Blackjack and Grid, the bootstrap exploration significantly improves upon Boltzmann exploration. In general, both RESLOPE and PPO perform quite well, though PPO often takes a bit longer to learn (likely because it is overly conservative).

In our second set of experiments, we compare the same algorithms *plus* the fully supervised DAgger algorithm on the three structured prediction problems; the results are in Figure 3. Here, we can

[4]It is not entirely clear to us yet why this happens. We found that RESLOPE performs as well as Reinforce and PPO if we (a) replace the loss with one centered around zero and (b) replace the RNN policy with a simpler feed-forward network, but we do not include these results in the figure to keep the experiments consistent.

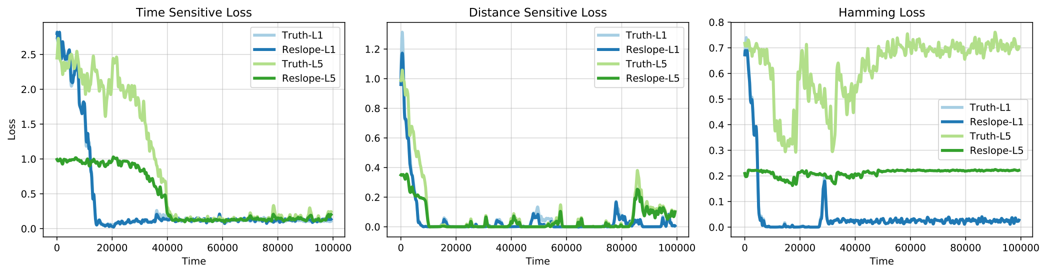


Figure 4: Empirical effect of additive vs non-additive loss functions. Performance is better when the loss is additive (blue) vs non-additive (green). The x-axis shows the number of episodes and the y-axis measures the incremental loss using the true loss function (light colors) and using RESLOPE (dark colors). If RESLOPE worked perfectly, these would coincide.

observe RESLOPE significantly outperforming all alternative algorithms (except, of course, DAgger) on training loss (also on heldout (development) loss; see Figure 8 in the appendix). There is still quite a gap to fully supervised learning, but nonetheless RESLOPE is able to reduce training error significantly on all tasks: by over 25% on English POS, by about half on English dependency parsing, and by about 10% on Chinese POS tagging.

5.2 ABLATION OF RESIDUAL LOSS PREDICTION

In our construction of RESLOPE, there are several tunable parameters: which contextual bandit learner to use (IPS, DR, MTR), which exploration strategy (Uniform, Boltzmann, Bootstrap), and, for Bootstrap, whether to do greedy prediction and/or greedy update. In Table 1 (in the Appendix), we show the results on all tasks for ablating these various parameters. For the purpose of the ablation, we *fix* the "baseline" system as: DR, Bootstrap, and with both greedy prediction and greedy updates, though this is not uniformly the optimal setting (and therefore these numbers may differ slightly from the preceding figures). The primary take-aways from these results are: (1) MTR and DR are competitive, but IPS is *much* worse; (2) Bootstrap is much better than either other exploration method (especially uniform, not surprisingly); (3) Greedy prediction is a bit of a wash, with only small differences either way; (4) Greedy update is important. In Appendix I, we consider the effect of single vs multiple deviations and observe that significant importance of multiple deviations for all algorithms, with Reinforce and PPO behaving quite poorly with only single deviations.

5.3 EVALUATING THE LEARNED LOSS REPRESENTATION

In our final set of experiments, we study RESLOPE's performance under different—and especially non-additive—loss functions. Our goal is to investigate RESLOPE's ability to learn good representations for the episodic loss. We consider the following different incremental loss functions for each time step: Hamming (0/1 loss at each position), Time-Sensitive (cost for an error at position h is equal to h) and Distance-Sensitive (cost for predicting $\hat{a}$ instead of a is $|\hat{a} - a|$). To combine these per-stop losses into a per-trajectory loss $\boldsymbol{\tau}$ of length H, we compute the H-dimensional loss vector $\boldsymbol{\ell}$ suffered by RESLOPE along this trajectory. To consider both additive and non-additive combinations, we consider Lp norms of this loss vector. When the norm is L1, this is simple additive loss. More generally, we consider $\ell(\boldsymbol{\tau}) = \sqrt[p]{\sum_{t=1}^{t=H} \boldsymbol{\ell}^p(t)}$ for any $p > 0$.

We run six different experiments using different incremental and episodic loss functions. For each incremental loss function (i.e. hamming, time sensitive, distance sensitive) we run two experiments: using the total hamming loss (additive) and an Lp norm of five (non-additive). Results are presented in Figure 4. We observe the following. RESLOPE can always learn the optimal representation for the incremental loss when the episodic loss function is additive. This is the case for all the three incremental loss functions: hamming, time sensitive, and distance sensitive. Learning is faster when the episodic loss function is additive. While RESLOPE is still able to learn a good representation even when using the L5 norm loss, this happens much later in comparison to the additive loss function (40k time steps for L5 norm vs 20k for total hamming loss). Not surprisingly, performance degrades as the episodic loss function becomes non-additive. This is most acute when using L-5 norm with the incremental hamming loss. This is expected as in the distance and time sensitive loss functions, RESLOPE observes a smoother loss function and learns to distinguish between different time steps based on the implicit encoding of time and distance information in the observed loss. RESLOPE can still learn a good representation for smoother episodic loss functions. This is shown empirically for time and distance sensitive loss functions.

6 RELATED WORK AND DISCUSSION

RESIDUAL LOSS PREDICTION builds most directly on the bandit learning to search frameworks LOLS (Chang et al., 2015) and BLS (Sharaf & Daumé, 2017). The "bandit" version of LOLS was analyzed theoretically but not empirically in the original paper; Sharaf & Daumé (2017) found that it failed to learn empirically.They addressed this by requiring additional feedback from the user, which worked well empirically but did not enjoy any theoretical guarantees. RESLOPE achieves the best of both worlds: a strong regret guarantee, good empirical performance, and no need for additional feedback. The key ingredient for making this work is using the residual loss structure together with strong base contextual bandit learning algorithms.

A number of recent algorithms have updated "classic" learning to search papers with deep learning underpinnings (Wiseman & Rush, 2016; Leblond et al., 2017). These aim to incorporate sequence-level global loss function to mitigate the mismatch between training and test time discrepancies, but only apply in the fully supervised setting. Mixing of supervised learning and reinforcement signals has become more popular in structured prediction recently, generally to do a better job of tuning for a task-specific loss using either Reinforce (Ranzato et al., 2015) or Actor-Critic (Bahdanau et al., 2016). The bandit variant of the structured prediction problem was studied by Sokolov et al. (2016), who proposed a reinforce method for optimizing different structured prediction models under bandit feedback in a log-linear structured prediction model.

A standard technique for dealing with sparse and episodic reward signals is reward shaping (Ng et al., 1999): supplying additional rewards to a learning agent to guide its learning process, beyond those supplied by the underlying environment. Typical reward shaping is hand-engineered; RESLOPE essentially learns a good task-specific reward shaping automatically. The most successful baseline approach we found is Proximal Policy Optimization (PPO, (Schulman et al., 2017)), a variant of Trust Region Policy Optimization (TRPO, (Schulman et al., 2015)) that is more practical. Experimentally we have seen RESLOPE to typically learn more quickly than PPO. Theoretically both have useful guarantees of a rather incomparable nature.

Since RESLOPE operates as a reduction to a contextual bandit oracle, this allows it to continually improve as better contextual bandit algorithms become available, for instance work of Syrgkanis et al. (2016b) and Agarwal et al. (2014). Although RESLOPE is quite effective, there are a number of shortcomings that need to be addressed in future work. For example, the bootstrap sampling algorithm is prohibitive in terms of both memory and time efficiency. One approach for tackling this would be using the amortized bootstrap approach by Nalisnick & Smyth (2017), which uses amortized inference in conjunction with implicit models to approximate the bootstrap distribution over model parameters. There is also a question of whether the reduction to contextual bandits creates "reasonable" contextual bandit problems in conjunction with RNNs. While some contextual bandit algorithms assume strong convexity or linearity, the ones we employ operate on arbitrary policy classes, provided a good cost-sensitive learner exists. The degree to which this is true will vary by neural network architecture, and what can be guaranteed (e.g., no regret full-information online neural learning). A more significant problem in the multi-deviation setting is that as RESLOPE learns, the residual costs will change, leading to a shifting distribution of *costs*; in principle this could be addressed using CB algorithms that work in adversarial settings (Syrgkanis et al., 2016a;b), but largely remains an open challenge. RESLOPE is currently designed for discrete action spaces. Extension to continuous action spaces (Levine et al., 2016; Lillicrap et al., 2015) remains an open problem.

REFERENCES

Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *In Proceedings of the 31st*

International Conference on Machine Learning (ICML-14, pp. 1638–1646, 2014.

J. A. Bagnell, Sham M Kakade, Jeff G. Schneider, and Andrew Y. Ng. Policy search by dynamic programming. In S. Thrun, L. K. Saul, and P. B. Schölkopf (eds.), *Advances in Neural Information Processing Systems 16*, pp. 831–838. MIT Press, 2004. URL `http://papers.nips.cc/paper/2378-policy-search-by-dynamic-programming.pdf`.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5): 834–846, Sept 1983. ISSN 0018-9472. doi: 10.1109/TSMC.1983.6313077.

Alina Beygelzimer, John Langford, and Bianca Zadrozny. Weighted one-against-all. In *AAAI*, pp. 720–725, 2005.

Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 19–26, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `http://proceedings.mlr.press/v15/beygelzimer11a.html`.

Alberto Bietti, Alekh Agarwal, and John Langford. Vowpal wabbit. *VW*, 2017. URL `https://github.com/JohnLangford/vowpal_wabbit/wiki`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé, III, and John Langford. Learning to search better than your teacher. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2058–2066. JMLR.org, 2015. URL `http://dl.acm.org/citation.cfm?id=3045118.3045337`.

Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, Jun 2009. ISSN 1573-0565. doi: 10.1007/s10994-009-5106-x. URL `https://doi.org/10.1007/s10994-009-5106-x`.

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*, ICML '05, pp. 169–176, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102373. URL `http://doi.acm.org/10.1145/1102351.1102373`.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Doubly robust policy evaluation and optimization. *Statist. Sci.*, 29(4):485–511, 11 2014. doi: 10.1214/14-STS500. URL `https://doi.org/10.1214/14-STS500`.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL `http://www.sciencedirect.com/science/article/pii/036402139090002E`.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

Ryan B Hayward and Jack Van Rijswijck. Hex and combinatorics. *Discrete Mathematics*, 306(19): 2515–2528, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952. doi: 10.1080/01621459.1952.10483446. URL `http://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483446`.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

John Langford and Alekh Agarwal. Vowpal wabbit online learning project, 2017.

John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20*, pp. 817–824. Curran Associates, Inc., 2008. URL `http://papers.nips.cc/paper/3178-the-epoch-greedy-algorithm-for-multi-armed-bandits-with-side-information.pdf`.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. Searnn: Training rnns with global-local losses. *arXiv preprint arXiv:1706.04499*, 2017.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, January 2016. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2946645.2946684`.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Michael L. Littman and Richard S Sutton. Predictive representations of state. In T. G. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems 14*, pp. 1555–1561. MIT Press, 2002. URL `http://papers.nips.cc/paper/1983-predictive-representations-of-state.pdf`.

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.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Eric Nalisnick and Padhraic Smyth. The amortized bootstrap. In *ICML 2017 Workshop on Implicit Models.*, 2017.

John F Nash. Some games and machines for playing them. *Technical Report, D-1164*, 1952.

Gergely Neu and Csaba Szepesvári. Training parsers by inverse reinforcement learning. *Mach. Learn.*, 77(2-3):303–337, December 2009. ISSN 0885-6125. doi: 10.1007/s10994-009-5110-1. URL `https://doi.org/10.1007/s10994-009-5110-1`.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL `http://dl.acm.org/citation.cfm?id=645528.657613`.

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

Olutobi Owoputi, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *In Proceedings of NAACL*, 2013.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D14-1162`.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

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

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `http://proceedings.mlr.press/v15/ross11a.html`.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amr Sharaf and Hal Daumé, III. Structured prediction via learning to search under bandit feedback. In *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*, pp. 17–26, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W17-4304`.

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), 2016. doi: 10.18653/v1/p16-1152. URL `http://dx.doi.org/10.18653/v1/P16-1152`.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Vasilis Syrgkanis, Akshay Krishnamurthy, and Robert Schapire. Efficient algorithms for adversarial contextual learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2159–2168, New York, New York, USA, 20–22 Jun 2016a. PMLR. URL `http://proceedings.mlr.press/v48/syrgkanis16.html`.

Vasilis Syrgkanis, Haipeng Luo, Akshay Krishnamurthy, and Robert E Schapire. Improved regret bounds for oracle-based adversarial contextual bandits. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3135–3143. Curran Associates, Inc., 2016b. URL `http://papers.nips.cc/paper/6400-improved-regret-bounds-for-oracle-based-adversarial-contextual-bandits.pdf`.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1296–1306, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1137`.

Fei Xia. The part-of-speech tagging guidelines for the penn chinese treebank (3.0). *Technical Report*, 2000.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207–238, 2005.

A MORE DETAILS ON CONTEXTUAL BANDIT ALGORITHMS

We assume that contexts are chosen i.i.d from an unknown distribution $\mathcal{D}(\boldsymbol{x})$, the actions are chosen from a finite action set $\mathcal{A}$, and the distribution over loss $\mathcal{D}(\ell|a, \boldsymbol{x})$ is fixed over time, but is unknown. In this context, the key challenge in contextual bandit learning is the exploration/exploitation problem. Classic algorithms for the contextual bandit problem such as EXP4.P (Beygelzimer et al., 2011) can achieve a $\sqrt{T}$ regret bound; in particular:

$$R\,(\text{EXP4}) \in O\left(\sqrt{TK \log |\Pi|}\right) \tag{2}$$

where $K = |A|$. When the regret is provably sublinear in T, such algorithms are often called "no regret" because their average regret per time step goes to zero as $T \to \infty$.

The particular contextual bandit algorithms we will use in this paper perform a second level of reduction: they assume access to an oracle supervised learning algorithm that can optimize a cost-sensitive loss, and transform the contextual bandit problem to a cost-sensitive classification problem. Algorithms in this family typically vary along two axes:

1. How to explore? I.e., faced with a new $\boldsymbol{x}$ how does the algorithm choose which action to take;
2. How to update? Given the observed loss ℓ_t, how does the algorithm construct a supervised training example on which to train.

As a simple example, an algorithm might explore uniformly at random on 10% of the examples and return the best guess action on 90% of examples (ϵ-greedy exploration). A single round to such an algorithm consists of a tuple $(\boldsymbol{x}, a, p)$, where p is the probability with which the algorithm took action a. (In the current example, this would be $\frac{0.1}{K}$ for all actions except $\pi(\boldsymbol{x})$ and $0.9 + \frac{0.1}{K}$ for $a = \pi(\boldsymbol{x})$.) If the update rule were "inverse propensity scaling" (IPS) (Horvitz & Thompson, 1952), the generated cost-sensitive learning example would have $\boldsymbol{x}$ as an input, and a cost vector $\boldsymbol{c} \in \mathbb{R}^K$ with zeros everywhere except in position a where it would take value $\frac{\ell}{p}$. The justification for this scaling is that in expectation over $a \sim p$, the expected value of this cost vector is equal to the true costs for each action. Neither of these choices is optimal (IPS has very high variance as p gets small); we discuss alternative exploration strategies and variance reduction strategies (§3.2).

B BANDIT STRUCTURED PREDICTION

Recently, it has become popular to solve structured prediction problems incrementally using some form of recurrent neural network (RNN) model. When the output $\boldsymbol{y}$ contains multiple parts (e.g., words in a translation), the RNN can predict each word in sequence, conditioning each prediction on all previous decisions. Although typically such models are trained to maximize cross-entropy with the gold standard output (in a fully supervised setting), there is mounting evidence that this has similar drawbacks to pre-RNN techniques, such as overfitting to gold standard prefixes (the model never learns what to do once it has made an error) and sensitivity to errors of different severity (due to error compounding).

By casting the structured prediction problem explicitly a sequential decision making problem (Daumé & Marcu, 2005; Daumé et al., 2009; Ross et al., 2011; Neu & Szepesvári, 2009) by, we can avoid these problems by applying imitation-learning style algorithms to their solution. This "Learning to Search" framework (Figure 5) solves structured prediction problems by:

1. converting structured and control problems to search problems by defining a search space of states $\mathcal{S}$ and an action set $\mathcal{A}$;
2. defining structured features over each state to capture the inter-dependency between output variables;
3. constructing a reference policy π^{ref} based on the supervised training data;
4. learning a policy π^{learn} that imitates or improves upon the reference policy.

In the *bandit* structured prediction setting, this maps nicely to the type of MDPs described at the beginning of this section. The formal reduction, following (Daumé & Marcu, 2005) is to ignore the

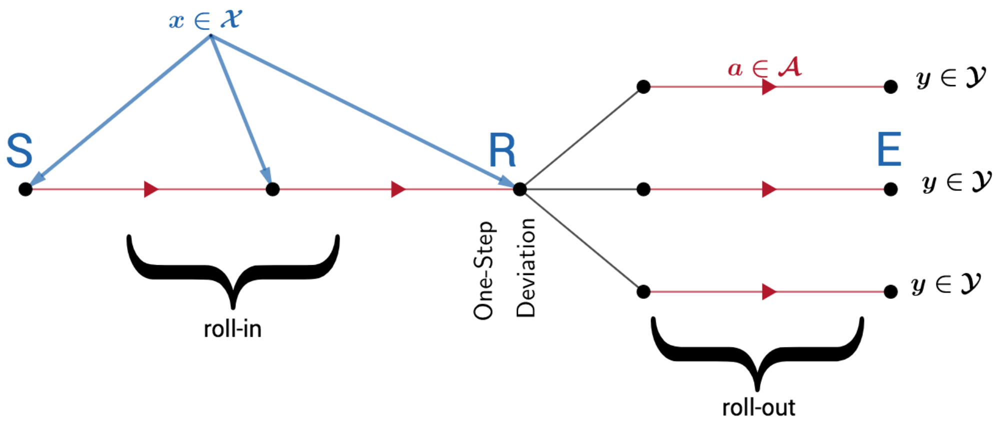


Figure 5: An example for a search space defined by a Learning to Search (L2S) algorithm. A search space is defined in terms of the set of states $\mathcal{X}$, and the set of action $\mathcal{A}$. The agent starts at the initial state S, and queries the roll-in policy π^{in} twice, next, at state R, the agent considers all three actions as possible one-step deviations. The agent queries the roll-out policy π^{out} to generate three different trajectories from the set of possible output structures $\mathcal{Y}$.

first action a_0 and to transition to an "initial state" s_1 by drawing an input $\boldsymbol{x}^{\text{SP}} \sim \mathcal{D}^{\mathcal{X}}$. The search space of the structured prediction task then generates the remainder of the state/action space for this example. The episode terminates when a state, s_H that corresponds to a "final output" is reached, at which point the structured prediction loss $\ell(\hat{\boldsymbol{y}}_{s_H} \mid \boldsymbol{x}^{\text{SP}})$ is computed on the output that corresponds to s_H. This then becomes the loss function $\mathcal{L}$ in the MDP. Clearly, learning a good policy under this MDP is equivalent to learning a structured prediction model with low expected loss.

C Cost-sensitive Classification

Many of the contextual bandit approaches we use in turn reduce the contextual bandit problem to a cost-sensitive classification problem. Cost-sensitive classification problems are defined by inputs $\boldsymbol{x}$ and cost vectors $\boldsymbol{y} \in \mathbb{R}^K$, where $\boldsymbol{y}(i)$ is the cost of choosing class i on this example. The goal in cost-sensitive classification is to learn a classifier $f : \boldsymbol{x} \to [K]$ such that $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\big[\boldsymbol{y}(f(\boldsymbol{x}))\big]$ is small. A standard strategy for solving cost-sensitive classification is via reduction to regression in a one-against-all framework (Beygelzimer et al., 2005). Here, a regression function $g(\boldsymbol{x}, i) \in \mathbb{R}$ is learned that predicts costs given input/class pairs. A predicted class on an input $\boldsymbol{x}$ is chosen as $\operatorname{argmin}_i g(\boldsymbol{x}, i)$. This cost-sensitive one-against-all approach achieves low regret when the underlying regressor is good. In practice, we use regression against Huber loss.

D Proof of Theorem 1

In a now-classic lemma, (Kakade et al., 2003; Bagnell et al., 2004) show that the difference in total loss between two policies can be computed exactly as a sum of per-time-step advantages of one over the other:

Lemma 1 ((Bagnell et al., 2004; Kakade et al., 2003)). *For all policies π and π':*

$$J(\pi) - J(\pi') = \sum_{h=1}^{H} \mathbb{E}_{s_h \sim d_\pi^h} \left[Q^{\pi'}(s_h, \pi) - V^{\pi'}(s_h) \right] \tag{3}$$

Theorem 1. Let π_n be the nth learned policy and $\bar{\pi}$ be the average learned policy. We wish to bound $J(\bar{\pi}) - J(\pi^*)$. We proceed as follows, largely following the AggreVaTe analysis (Ross & Bagnell, 2014). We begin by noting that $J(\bar{\pi}) - J(\pi^*) = J(\bar{\pi}) - J(\pi^{\text{ref}}) + J(\pi^{\text{ref}}) - J(\pi^*)$ and will concern ourselves with bounding the first difference.

$$J(\bar{\pi}) - J(\pi^{\text{ref}}) = \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) \right] \tag{4}$$

Fix an n, and consider the sum above for a fixed deviation time step h^{dev}:

$$\sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) \right] \tag{5}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \left(Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) - \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s', \pi_n) - Q^{\pi^{\text{ref}}}(s', \pi^{\text{ref}}) \right] \right) \right] \tag{6}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \left(\mathbb{E}_{s_H \sim \pi^{\text{ref}} \mid s_h = s} \ell(s_H) - \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} A^{\text{ref}}(s', \pi_n) \right) \right] \tag{7}$$

where $A^{\text{ref}}(s', \pi_n) = Q^{\pi^{\text{ref}}}(s', \pi_n) - Q^{\pi^{\text{ref}}}(s', \pi^{\text{ref}})$ is the policy (dis)advantage. The term in the round parentheses $(\dots)$ is exactly the expected value of the target of the contextual bandit cost; therefore:

$$\textit{Regret}(\bar{\pi}) \leq \textit{Regret}(\pi^{\text{ref}}) + \frac{1}{N} \epsilon_{\text{CB}}(N) \tag{8}$$

If the CB algorithm has regret sublinear in N, the second term goes to zero as $N \to \infty$. $\square$

E Proof of Theorem 2

Proof. The proof follows a combination of the above analysis with the LOLS analysis. Using the same notation as before, additionally let π_n^{out} be the mixture of π_n with π^{ref} for rollout.

First, we observe (LOLS Eq 6):

$$J(\bar{\pi}) - J(\pi^{\text{ref}}) = \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} [Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}})] \tag{9}$$

Then (LOLS Eq 7):

$$\sum_h \left[J(\bar{\pi}) - \min_{\pi \in \Pi} \mathbb{E}_{s \sim d^h_{\bar{\pi}}} Q^{\bar{\pi}}(s, \pi) \right] \leq \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi_n}(s, \pi_n) - \min_a Q^{\pi_n}(s, a) \right] \tag{10}$$

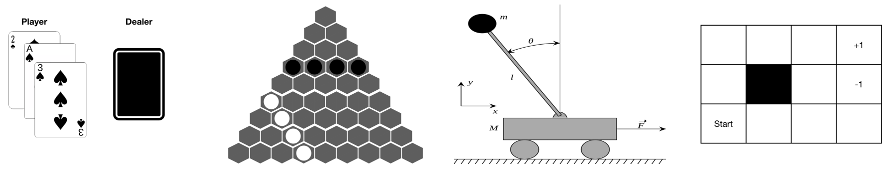
So far nothing has changed. It will be convenient to define $Q^{\pi_n}_\beta(s) = \beta \min_a Q^{\pi^{\text{ref}}}(s, a) + (1 - \beta) \min_a Q^{\pi_n}(s, a)$. For each n fix the deviation time step h^{dev}_n. We plug these together ala LOLS and get:

$$\beta \Big(J(\bar{\pi}) - J(\pi^{\text{ref}}) \Big) + (1 - \beta) \Big(J(\bar{\pi}) - \min_{\pi \in \Pi} \mathbb{E}_{s \sim d^h_{\bar{\pi}}} Q^{\bar{\pi}}(s, \pi) \Big) \tag{11}$$

$$\leq \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s, \pi_n) - \beta \min_a Q^{\pi^{\text{ref}}}(s, a) - (1 - \beta) \min_a Q^{\pi_n}(s, a) \right] \tag{12}$$

$$= \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s, \pi_n) - Q^{\pi_n}_\beta(s) \right] \tag{13}$$

$$= \mathbb{E}_n \mathbb{E}_{s^{\text{dev}} \sim d^{h^{\text{dev}}_n}_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s^{\text{dev}}, \pi_n) - \left(Q^{\pi_n}_\beta(s^{\text{dev}}) - \sum_{h \neq h^{\text{dev}}_n} \mathbb{E}_{s_h \sim d^h_{\pi_n}} \left(Q^{\pi^{\text{out}}_n}(s_h, \pi_n) - Q^{\pi_n}_\beta(s_h) \right) \right) \right] \tag{14}$$



(a) Blackjack (b) Hex (c) Cart Pole (d) Grid World

Figure 6: Reinforcement Learning Tasks

$$=\mathbb{E}_n \mathbb{E}_{s^{\text{dev}} \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s^{\text{dev}}, \pi_n) - \left(\mathbb{E}_{s_H \sim d^H_{\pi_n} \mid s_{h^{\text{dev}}_n} = s^{\text{dev}}} \mathcal{L}_n(s_H) - \sum_{h \neq h^{\text{dev}}} \text{CB.COST}(\pi_n, s_h) \right) \right] \tag{15}$$

$$=\frac{1}{N}\epsilon_{\text{CB}}(N) \tag{16}$$

The penultimate step follows because the inner-most expectation is exactly what the contextual bandit algorithm is estimating, and $Q^{\pi_n}_{\beta}(s^{\text{dev}})$ is exactly the expectation of the observed loss The final step follows because the costs observed by the CB algorithm are exactly those in $(\dots)$, and so if the CB algorithm is no-regret, then it follows that RESLOPE is also no-regret. □

F Details on Reinforcement Learning Environments

Blackjack is a card game where the goal is to obtain cards that sum to as near as possible to 21 without going over. Players play against a fixed dealer who hits until they have at least 17. Face cards (Jack, Queen, King) have a point value of 10. Aces can either count as 11 or 1, and a card is called “usable” at 11. The reward for winning is $+1$, drawing is 0, and losing is -1. The world is partially visible: the player can see one their own cards and one of the two initial dealer cards.

Hex is a classic two-player board game invented by Piet Hein and independently by John Nash (Hayward & Van Rijswijck, 2006; Nash, 1952). The board is an $n \times n$ rhombus of hexagonal cells. Players alternately place a stone of their color on any empty cell. To win, a player connects her two opposing sides with her stones. We use $n = 5$; the world is fully visible to the agent, with each hexagon showing as unoccupied, occupied with white or occupied with black. The reward is $+1$ for winning and -1 for losing.

Cart Pole is a classic control problem variously referred to as the “cart-pole”, “inverted pendulum”, or “pole balancing” problem (Barto et al., 1983). Is is an example of an inherently unstable dynamic system, in which the objective is to control translational forces that position a cart at the center of a finite width track while simultaneously balancing a pole hinged on the cart’s top. In this task, a pole is attached by a joint to a cart which moves along a frictionless track (Figure 6c). The system is controlled by applying a force of $+1$ or -1 to the cart, thus, we operate in a discrete action space with only two actions. The pendulum starts upright, and the goal is to prevent it from falling over. The episode ends when the pole is more than 15 degrees from the vertical axis, or the cart moves more than 2.4 units from the center. The state is represented by four values indicating the poles position, angle to the vertical axis, and the linear and angular velocities. The total cumulative reward at the end of the episode is the total number of time steps the pole remained upright before the episode terminates.

Grid World consists of a simple 3×4 grid, with a $+1$ reward in the upper-right corner and -1 reward immediately below it; the cell at $(1, 1)$ is blocked (Figure 6d). The agent starts at a random unoccupied square. Each step costs 0.05 and the agent has a 10% chance of misstepping. The agent only gets partial visibility of the world: it gets an indicator feature specifying which directions it can step. The only reward observed is the complete sum of rewards over an episode.

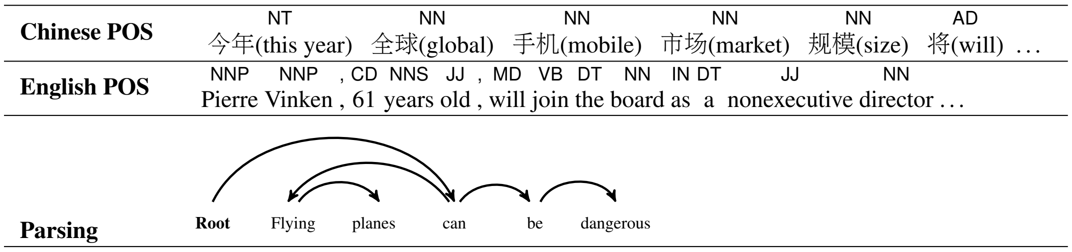

Chinese POS
NT NN NN NN NN AD
今年(this year) 全球(global) 手机(mobile) 市场(market) 规模(size) 将(will) . . .
English POS
NNP NNP , CD NNS JJ , MD VB DT NN IN DT JJ NN
Pierre Vinken , 61 years old , will join the board as a nonexecutive director . . .
Parsing
Root Flying planes can be dangerous

Figure 7: Example inputs for part of speech tagging and dependency parsing.

G Structured Prediction Data Sets

English POS Tagging we conduct POS tagging experiments over the 45 Penn Treebank (Marcus et al., 1993) tags. We simulate a domain adaptation setting by training a reference policy on the TweetNLP dataset (Owoputi et al., 2013) which achieves good accuracy in domain, but performs badly out of domain. We simulate bandit episodic loss over the entire Penn Treebank Wall Street Journal (sections $02 \to 21$ and 23), comprising 42k sentences and about one million words. The measure of performance is the average Hamming loss. We define the search space by sequentially selecting greedy part-of-speech tags for words in the sentence from left to right.

Chinese POS Tagging we conduct POS tagging experiments over the Chinese Penn Treebank (3.0) (Xia, 2000) tags. We simulate a domain adaptation setting by training a reference policy on the Newswire domain from the Chinese Treebank Dataset (Xue et al., 2005) and simulate bandit episodic feedback from the spoken conversation domain. We simulate bandit episodic loss over 40k sentences and about 300k words. The measure of performance is the average Hamming loss. We define the search space by sequentially selecting greedy part-of-speech tags for words in the sentence from left to right.

English Dependency Parsing For this task, we assign a grammatical head (i.e. parent) for each word in the sentence. We train an arc-eager dependency parser (Nivre, 2003) which chooses among (at most) four actions at each state: Shift, Reduce, Left or Right. The reference policy is trained on the TweetNLP dataset and evaluated on the Penn Treebank corpus. The loss is the unlabeled attachment score (UAS), which measures the fraction of words that are assigned the correct parent.

In all structured prediction settings, the feature representation begins with pretrained (and non-updated) embeddings. For English, these are the 6gb Glove embeddings (Pennington et al., 2014); for Chinese, these are the FastText embeddings (Joulin et al., 2016). We then run a bidirectional LSTM (Hochreiter & Schmidhuber, 1997) over the input sentence. The input features for labeling the nth word in POS tagging experiments are the biLSTM representations at position n. The input features for dependency actions are a concatenation of the biLSTM features of the next word on the buffer and the two words on the top of the stack.

H Optimization, Hyperparameter Selection and "Tricks"

We optimize all parameters of the model using the Adam[5] optimizer (Kingma & Ba, 2014), with a tuned learning rate, a moving average rate for the mean of $\beta_1 = 0.9$ and for the variance of $\beta_2 = 0.999$; epsilon (for numerical stability) is fixed at $1e - 8$ (these are the DyNet defaults). The learning rate is tuned in the range $\{0.050.01, 0.005, 0.001, 0.0005, 0.0001\}$.

For the structured prediction experiments, the following input features hyperparameters are tuned:

- Word embedding dimension $\in \{50, 100, 200, 300\}$ (for the Chinese embeddings, which come only in 300 dimensional versions, we took the top singular vectors to reduce the dimensionality).

[5]We initially experimented also with RMSProp (Tieleman & Hinton, 2012) and AdaGrad (Duchi et al., 2011) but Adam consistently performed as well or better than the others on all tasks.

- BiLSTM dimension $\in \{50, 150, 300\}$
- Number of BiLSTM layers $\in \{1, 2\}$
- Pretraining: DAgger or AggreVaTe initialization with probability of rolling in with the reference policy $\in \{0.0, 0.999^N, 0.99999^N, 1.0\}$, where N is the number of examples
- Policy RNN dimension $\in \{50, 150, 300\}$
- Number of policy layers $\in \{1, 2\}$
- Roll-out probability $\beta \in \{0.0, 0.5, 1.0\}$

For each task, the network architecture that was optimal for supervised pretraining was fixed and used for all bandit learning experiments[6].

For the reinforcement learning experiments, we tuned:

- Policy RNN dimension $\in \{20, 50, 100\}$
- Number of policy layers $\in \{1, 2\}$

Some parameters we do not tune: the nonlinearities used, the size of the action embeddings (we use 10 in all cases), the input RNN form for the text experiments (we always use LSTM instead of RNN or GRU based on preliminary experiments). We do not regularize our models (weight shrinkage only reduced performance in initial experiments) nor do we use dropout. Pretraining of the structured prediction models ran for 20 passes over the data with early stopping based on held-out loss. The state of the optimizer was reset once bandit learning began.

The variance across difference configurations was relatively small across RL tasks, so we chose a two layer policy with 20 dimensional vectors for all RL tasks.

Each algorithm also has a set of hyperparameters; we tune them as below:

- Reinforce: with baseline or without baseline
- A2C: a multiplier on the relative importance of actor loss and critic loss $\in \{0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$
- PPO: with baseline or without baseline; and epsilon parameter $\in \{0.01, 0.05, 0.1, 0.2, 0.4, 0.8\}$
- RESLOPE: update strategy (IPS, DR, MTR) and exploration strategy (uniform, Boltzmann or Bootstrap)

In each reinforcement/bandit experiment, we *optimistically* pick algorithm hyperparameters and learning rate based on final evaluation criteria, noting that this likely provides unrealistically optimistic performance for *all* algorithms. We perform 100 replicates of every experiment in the RL setting and 20 replicates in the structured prediction setting. We additionally ablate various aspects of RESLOPE in §5.2.

We employ only two "tricks," both of which are defaults in dynet: gradient clipping (using the default dynet settings) and smart parameter initialization (dynet uses Glorot initialization (Glorot & Bengio, 2010)).

I EFFECT OF SINGLE VS MULTIPLE DEVIATIONS

Next, we consider the single-deviation version of RESLOPE (1) versus the multiple-deviation version (2). To enable comparison with alternative algorithms, we also experiment with variants of Reinforce, PPO and DAgger that are only allowed single deviations as well (also chosen uniformly at random). The results are shown in Figure 9. Not surprisingly, all algorithms suffer when only allowed single deviations. PPO makes things worse over time (likely because its updates are very conservative, such that even in the original PPO paper the authors advocate multiple runs over the

[6]English POS tagging and dependency parsing: DAgger 0.99999^N, 300 dim embeddings, 300 dim 1 layer LSTM, 2 layer 300 dimensional policy; Chinese POS tagging: DAgger 0.999^N, 300 dim embeddings, 50 dim 2 layer LSTM, 1 layer 50 dimensional policy).

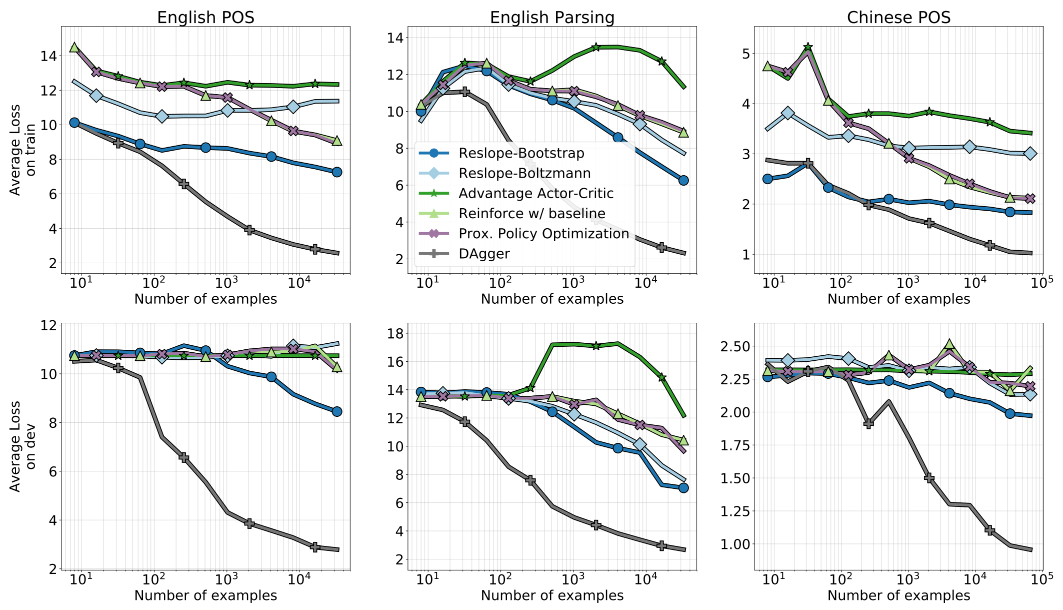


Figure 8: Average loss (top) and heldout loss (bottom) during learning for three bandit structured prediction problems. Also included are supervised learning results with DAgger. The main difference between the training loss and the development loss is that in the development data, the system needn't explore, and so the gaps in algorithms which explore different amounts (e.g., especially on English POS tagging) disappear.

	Reinforcement Learning				**Bandit SP**		
	Blackjack	**Cartpole**	**Grid**	**Hex**	**Zh-POS**	**En-Dep**	**En-POS**
total loss	0.17	−28.0	0.69	−0.88	1.8	6.3	7.3
loss std	0.021	23.0	0.74	0.008	0.019	0.58	0.77
→ MTR	−1.55	−0.105	−0.783	2.88	0.023	1.56	0.661
→ IPS	−1.81	0.77	−0.28	0.427	282.0	13.2	17.6
→ Boltzmann	2.85	0.263	0.184	54.8	275.0	14.1	18.3
→ Uniform	10.8	0.28	0.566	104.0	285.0	16.1	13.8
– g-predict	−0.638	0.362	−0.31	−0.151	0.236	0.314	0.596
– g-update	1.03	0.508	−0.158	2.24	7.11	3.87	2.79

Table 1: Results of ablating various parts of the RESIDUAL LOSS PREDICTION approach. Columns are tasks. The first two rows are the cumulative average loss over multiple runs and its standard deviation. The numbers in the rest of the column measure how much it hurts (positive number) or helps (negative number) to ablate the corresponding parameter. To keep the numbers on a similar scale, the changes are reported as *multiples* of the standard deviation. So a value of 2.0 means that the cumulative loss gets worse by an additive factor of two standard deviations.

same data), as does Reinforce. DAgger still learns, though more slowly, when only allowed a single deviation. RESLOPE behaves similarly though not quite as poorly. Overall, this suggests that even though the samples generated with multiple deviations by RESLOPE are no longer independent, the gain in number of samples more than makes up for this.

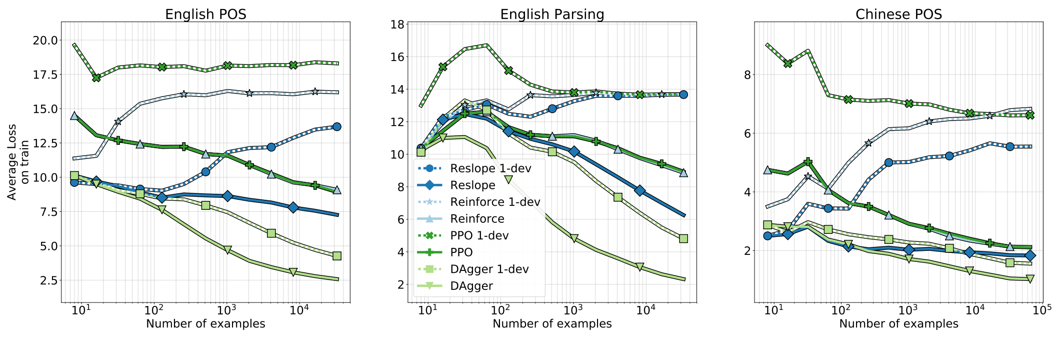


Figure 9: The empirical effect of multiple deviations for different algorithms.

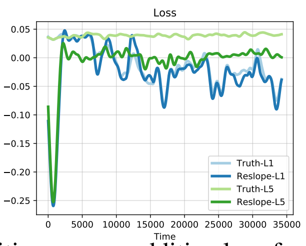


Figure 10: Empirical effect of additive vs non-additive loss functions. Performance is better when the loss is additive (blue) vs non-additive (green). The x-axis shows the number of episodes and the y-axis measures the incremental loss using the true loss function (light colors) and using RESLOPE (dark colors). If RESLOPE worked perfectly, these would coincide.

J SYNTHETIC DATA FOR EVALUATING THE LEARNED LOSS REPRESENTATION

Experiments were conducted on a synthetic sequence labeling dataset. Input sequences are random integers (between one and ten) of length 6. The ground truth label for the hth word is the corresponding input mod 4. We generate 16k training sequences for this experiment. We run RESLOPE with bootstrap sampling in multiple deviation mode. We use the MTR cost estimator, and optimize the policies using ADAM with a learning rate of 0.01.

K EVALUATING THE LEARNED LOSS REPRESENTATION FOR GRID WORLD

In this section, we study RESLOPE's performance under different—and especially non-additive—loss functions. This experiment is akin to the experimental setting in section 5.3, however it's performed on the grid world reinforcement learning environment, where the quantitative aspects of the loss function is well understood.

We study a simple 4×4 grid, with a $+1$ reward in the upper-right corner and -1 reward immediately below it; the cells at $(1, 1)$ and $(2, 1)$ are blocked. The agent starts at a random position in the grid. Each step costs $+0.05$ and the probability of success is 0.9. The agent has full visibility of the world: it knows its horizontal and vertical position in the grid.

We consider two different episodic reward settings:

1. The only reward observed is the complete sum of losses over an episode. (additive setting);
2. The only reward observed is the L5 norm of the vector of losses over an episode (non-additive setting).

Results are shown in Figure 10. Results are very similar to the structured prediction setting (section 5.3). Performance is better when the loss is additive (blue) vs non-additive (green).