Reinforcement Learning for Mapping Instructions to Actions

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{branavan, harr, lsz, regina}@csail.mit.edu

Abstract

In this paper, we present a reinforcement learning approach for mapping natural language instructions to sequences of executable actions. We assume access to a reward function that defines the quality of the executed actions. During training, the learner repeatedly constructs action sequences for a set of documents, executes those actions, and observes the resulting reward. We use a policy gradient algorithm to estimate the parameters of a log-linear model for action selection. We apply our method to interpret instructions in two domains — Windows troubleshooting guides and game tutorials. Our results demonstrate that this method can rival supervised learning techniques while requiring little or no annotated data.

1 Introduction

In this paper, we consider the problem of relating linguistic analysis and automatic control — specifically, mapping natural language instructions to executable actions. This problem is directly relevant to the automatic interpretation of guides for configuring software, manuals for operating simulators, or tutorials for playing games. These texts are written in natural language, but describe sequences of actions that should be executed in their respective software environments. Solving this problem will enable the automation of tasks that currently require human participation.

For concreteness, consider instructions from a Windows troubleshooting guide on deleting temporary folders, shown in Figure 1. We aim to map this text to low-level commands and corresponding parameters. For example, properly interpreting the third instruction requires clicking on a tab, finding the appropriate option in a tree control, and clearing its associated checkbox. Since actions alter the environment (e.g., a dialog box appears), each new instruction must be interpreted in the context that results from previous actions.

The main contribution of this paper is the use of interaction with a dynamic environment to find mappings from natural language instructions to concrete action sequences. Given a candidate mapping, we can execute the corresponding actions in the environment and assess their effects. This form of supervision is especially useful when hand-labeled annotations of action sequences are not readily available, preventing the use of standard supervised techniques.

We formalize this idea in a reinforcement learning framework (Sutton and Barto, 1998). We assume access to a reward function that defines the quality of executed actions. For instance, in the example above, the reward may assess whether the goal described in the instructions is achieved, i.e., the folder is deleted. During training, the learner repeatedly constructs action sequences for a set of given documents, executes those actions, and observes the resulting reward. The learner’s goal

Figure 1: A Windows troubleshooting article describing how to remove “msdownld.tmp” temporary folders.
is to estimate a policy — a distribution over actions given instruction text and environment state — that maximizes future expected reward.

Our language interpretation task leads to a challenging reinforcement learning problem. While the state of the environment is observable, it exhibits complex structure. Therefore, we cannot simply learn a lookup table from states to actions. Instead, we model the policy as a log-linear distribution with features encoding both instruction text and environment characteristics. This setting is particularly suitable for policy gradient methods, which learn effectively while exploring only a small part of the state space.

We evaluate our method in two domains. First, we apply it to documents that describe user-interface operations for performing tasks in the Windows operating system. The second domain is a computer puzzle game, where tutorial documents describe how to complete each level. Using only task completion as the reward, our method solves 54.0% of the puzzles by following tutorial instructions. On the Windows domain, the method correctly identifies 64.7% of actions. Moreover, when the reward is augmented with a few annotated training examples, performance rivals a fully supervised learner.

2 Related Work

Grounded Language Acquisition Our work fits into a broader class of approaches that aim to learn language from a situated context (Mooney, 2008a; Mooney, 2008b; Siskind, 2001; Oates, 2001). Instances of such approaches include work on inferring the meaning of words from video data (Roy and Pentland, 2002; Barnard and Forsyth, 2001) or interpreting the commentary of a simulated soccer game (Chen and Mooney, 2008). Most of these approaches assume some form of parallel data, and learn perceptual co-occurrence patterns. In contrast, our emphasis is on a learner that can proactively interact with an external environment to learn language.

Reinforcement Learning for Language Processing Reinforcement learning has been previously applied to the problem of dialogue management (Singh et al., 1999; Litman et al., 2000; Roy et al., 2000; Scheffler and Young, 2002). These systems converse with a human user by taking actions that emit natural language utterances. The reinforcement learning state space encodes information about the goals of the user and what they say at each time step. The learning problem is to find an optimal policy that maps states to actions, through a trial-and-error process of repeated interaction with the user.

Reinforcement learning is applied very differently in dialogue systems versus our setup. In some respects, our task is more easily amenable to reinforcement learning. For instance, we are not interacting with a human user, so the cost of interaction is lower. However, the main challenge of our application is the substantially larger state space, which is determined by the complexity of the underlying environment. We address this complexity by developing a policy gradient algorithm that learns efficiently while exploring a small subset of the state space.

3 Problem Formulation

Our task is to learn a mapping between documents and the sequence of actions they express. Table 1 summarizes the terminology we introduce here.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Sequence of actions $\vec{a}$</td>
</tr>
<tr>
<td>Instruction sentence in $d$</td>
<td>Parameters $\theta$ of policy $p(a</td>
</tr>
<tr>
<td>Sequence of actions $\vec{a}$</td>
<td>Reward function $r(h)$</td>
</tr>
<tr>
<td>(Eq. $a_0, \ldots, a_{n-1}$)</td>
<td>Transition distr. $p(s'</td>
</tr>
<tr>
<td>Action $a = (c, R, W')$</td>
<td>Transition distr. $p(s'</td>
</tr>
<tr>
<td>Command $c$</td>
<td>Policy $p(a</td>
</tr>
<tr>
<td>Vector of command parameters $R$</td>
<td>Environment state transition distr. $p(E'</td>
</tr>
<tr>
<td>Words mapped by action $W'$</td>
<td>Environment state transition distr. $p(E'</td>
</tr>
<tr>
<td>Observed mapping state $E$</td>
<td>History of actions and states $h = (s_0, a_0, s_1, \ldots, a_{n-1}, s_n)$</td>
</tr>
<tr>
<td>Observed environment state $j$</td>
<td>Reward function $r(h)$</td>
</tr>
<tr>
<td>Sentence index into document $d$</td>
<td>Environment state transition distr. $p(E'</td>
</tr>
</tbody>
</table>

Table 1: Summary of notations and tasks.
Figure 2: A four-step mapping from an instruction sentence to a sequence of actions in Windows 2000. For each step, the figure shows the words selected by the action, along with the corresponding system command and its parameters. The notation from Table 1 is used to show mapping progress. In addition, the words of $W$ are highlighted in grey, and the words of $W'$ are underlined.

and executed sequentially.\(^2\) Figure 2 shows how one sentence is mapped to four actions.

An action $a = (c, R, W')$ encompasses a command $c$, the command’s parameters $R$, and the words $W'$ specifying $c$ and $R$. Elements of $R$ refer to objects available in the environment state, as described below.\(^3\)

The Environment The environment state $E$ specifies the set of objects available for interaction, and their properties. In the example, $E$ is shown graphically on the right. $E$ changes in response to the execution of command $c$ with parameters $R$, according to a transition distribution $p(E' | E, c, R)$. This distribution is a priori unknown to the learner. As we will see in Section 5, our approach avoids having to directly estimate this distribution.

State To predict actions sequentially, we need to track the state of the document-to-actions mapping over time. A mapping state $s$ is a tuple $(E, d, j, W)$, where $E$ refers to the current environment state; $j$ is the index of the sentence currently being interpreted in document $d$; and $W$ contains words that were mapped by previous actions for the same sentence. $s$ is observed after each action.

The initial mapping state $s_0$ for document $d$ is $(E_d, d, 0, \emptyset); E_d$ is the unique starting environment state for $d$. Performing action $a$ in state $s = (E, d, j, W)$ leads to a new state $s'$ according to distribution $p(s' | s, a)$, defined as follows: $E$ transitions according to $p(E' | E, c, R)$, $W$ is updated with $a$’s selected words, and $j$ is incremented if all words of the sentence have been mapped. For the applications we consider in this work, environment state transitions, and consequently mapping state transitions, are deterministic.

Training During training, we are provided with a set $D$ of documents, access to the transition distribution, and a reward function $r(h)$. Here, $h = (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$ is a history of states and actions visited while interpreting one document. $r(h)$ outputs a real-valued score that correlates with correct action selection.\(^4\) We con-

\(^2\)That is, action $a_i$ is executed before $a_{i+1}$ is predicted.
\(^3\)Some parameters can also refer to word sequences in the document $d$.

\(^4\)Commonly in reinforcement learning, the reward function is defined over state-action pairs, as $r(s, a)$ — in this case, $r(h) = \sum r(s_i, a_i)$, and our formulation becomes a standard finite-horizon Markov decision process. Policy gradient approaches allow us to learn using the more general case of history-based reward.
sider both immediate reward, which is available after each action, and delayed reward, which does not provide feedback until the last action. For example, task completion is a delayed reward that produces a positive value after the final action only if the task was completed successfully. We will also demonstrate how manually annotated action sequences can be incorporated into the reward.

The goal of training is to estimate parameters $\theta$ of the action selection distribution $p(a|s, \theta)$, called the policy. Since the reward correlates with action sequence correctness, the $\theta$ that maximizes expected reward will yield the best actions.

### 4 A Log-Linear Model for Actions

Our goal is to predict a sequence of actions. We construct this sequence by repeatedly choosing an action given the current mapping state, and applying that action to advance to a new state.

Given a state $s = (E, d, j, W)$, the space of possible next actions is defined by enumerating the subspans of the $j$th sentence of $d$ not in $W$, and the possible commands and parameters in environment state $E$. We model the policy distribution $p(a|s; \theta)$ over this action space in a log-linear fashion (Della Pietra et al., 1997; Lafferty et al., 2001), giving us the flexibility to incorporate a diverse range of features. Under this representation, the policy distribution is:

$$p(a|s; \theta) = \frac{e^{\theta \cdot \phi(s,a)}}{\sum_{a'} e^{\theta \cdot \phi(s,a')}}.$$  

where $\phi(s,a) \in \mathbb{R}^n$ is an $n$-dimensional feature representation. During test, actions are selected according to the mode of this distribution.

### 5 Reinforcement Learning

During training, our goal is to find the optimal policy $p(a|s; \theta)$. Since reward correlates with correct action selection, a natural objective is to maximize expected future reward — that is, the reward we expect while acting according to that policy from state $s$. Formally, we maximize the value function:

$$V_\theta(s) = E_{p(h|\theta)} [r(h)],$$  

where the history $h$ is the sequence of states and actions encountered while interpreting a single document $d \in D$. This expectation is averaged over all documents in $D$. The distribution $p(h|\theta)$ returns the probability of seeing history $h$ when starting from state $s$ and acting according to a policy with parameters $\theta$. This distribution can be decomposed into a product over time steps:

$$p(h|\theta) = \prod_{t=0}^{n-1} p(a_t|s_t; \theta) p(s_{t+1}|s_t, a_t).$$  

### 5.1 A Policy Gradient Algorithm

Our reinforcement learning problem is to find the parameters $\theta$ that maximize $V_\theta$ from equation 2. Although there is no closed form solution, policy gradient algorithms (Sutton et al., 2000) estimate the parameters $\theta$ by performing stochastic gradient ascent. $V_\theta$’s gradient is approximated by interacting with the environment, and the resulting reward is used to update the estimate of $\theta$. Policy gradient algorithms optimize a non-convex objective and are only guaranteed to find a local optimum. However, as we will see, they scale to large state spaces and can perform well in practice.

To find the parameters $\theta$ that maximize the objective, we first compute the derivative of $V_\theta$. Expanding according to the product rule, we have:

$$\frac{\partial}{\partial \theta} V_\theta(s) = E_{p(h|\theta)} \left[ r(h) \sum_t \frac{\partial}{\partial \theta} \log p(a_t|s_t; \theta) \right],$$  

where the inner sum is over all time steps $t$ in the current history $h$. Expanding the inner partial derivative we observe that:

$$\frac{\partial}{\partial \theta} \log p(a_t|s_t; \theta) = \phi(s,a) - \sum_{a'} \phi(s,a') p(a'|s_t; \theta),$$  

which is the derivative of a log-linear distribution.

Equation 5 is easy to compute directly. However, the complete derivative of $V_\theta$ in equation 4 is intractable, because computing the expectation would require summing over all possible histories. Instead, policy gradient algorithms employ stochastic gradient ascent by computing a noisy estimate of the expectation using just a subset of the histories. Specifically, we draw samples from $p(h|\theta)$ by acting in the target environment, and use these samples to approximate the expectation in equation 4. In practice, it is often sufficient to sample a single history $h$ for this approximation.

Algorithm 1 details the complete policy-gradient algorithm. The algorithm performs $T$ iterations over the set of documents $D$. For each document, step 3 samples a history that maps the
Input: A document set $D$, Feature representation $\phi$, Reward function $r(h)$, Number of iterations $T$

Initialization: Set $\theta$ to small random values.

1. for $i = 1 \cdots T$ do
2.   foreach $d \in D$ do
3.     Sample history $h \sim p(h|\theta)$ where $h = (s_0, a_0, \cdots, a_{n-1}, s_n)$ as follows:
3a. for $t = 0 \cdots n - 1$ do
3b.   Sample action $a_t \sim p(a|s_t; \theta)$
3c.   Execute $a_t$ on state $s_t$: $s_{t+1} \sim p(s|s_t, a_t)$
4.     $\Delta \leftarrow \sum_t (\phi(s_t, a_t) - \sum_{a'} \phi(s_t, a')p(a'|s_t; \theta))$
5.     $\theta \leftarrow \theta + r(h)\Delta$
end
end

Output: Estimate of parameters $\theta$

Algorithm 1: A policy gradient algorithm.

documents to actions. This is done by repeatedly selecting actions according to the current policy, and updating the state by executing the selected actions. Steps 4 and 5 compute the empirical gradient and update the parameters $\theta$.

In many domains, interacting with the environment is expensive. Therefore, we use two techniques that allow us to take maximum advantage of each actual environment interaction. First, a single history $h = (s_0, a_0, \ldots, s_n)$ contains subsequences $(s_i, a_i, \ldots, s_n)$ for $i = 1$ to $n - 1$, each with its own reward value given by the environment as a side effect of executing $h$. We can compute the update from equation 5 for each subsequence. Second, for a sampled history $h$, we can propose alternative histories that result in the same commands and parameters with different underlying word spans. We can again apply equation 5 for each alternative $h'$, weighted by its probability under the current policy, $p(h'|\theta)/p(h|\theta)$.

The algorithm we have presented belongs to a family of policy gradient algorithms that have been successfully used for complex tasks such as robotics control (Ng et al., 2003). Our formulation is unique in how it represents natural language in the reinforcement learning framework.

5.2 Reward Functions and ML Estimation

We can design a range of reward functions to guide learning, depending on the availability of annotated data and environment feedback. Consider the case when every training document $d \in D$ is annotated with its correct sequence of actions, and state transitions are deterministic. Given these observed examples, it is straightforward to construct a reward function that connects policy gradient with maximum likelihood. Specifically, define a reward function $r(h)$ that returns one when $h$ matches the annotation for the document being analyzed, and zero otherwise. Policy gradient performs stochastic gradient ascent on the objective from Equation 2, performing one update per document. For document $d$, this objective becomes:

$$E_{p(h|\theta)}[r(h)] = \sum_h r(h)p(h|\theta) = p(h_d|\theta),$$

where $h_d$ is the history corresponding to the annotated action sequence. Thus, with this reward policy gradient is equivalent to stochastic gradient ascent with a maximum likelihood objective.

At the other extreme, when annotations are completely unavailable, learning is still possible given informative feedback from the environment. Crucially, this feedback only needs to correlate with action sequence quality. We detail environment-based reward functions in the next section. As our results will show, reward functions built using this kind of feedback can provide strong guidance for learning. We will also consider reward functions that combine annotated supervision with environment feedback.

6 Applying the Model

We study two applications of our model: following instructions to perform software tasks, and solving a puzzle game using tutorial guides.

6.1 Microsoft Windows Help and Support

On its Help and Support website, Microsoft publishes a number of articles describing how to perform tasks and troubleshoot problems in the Windows operating systems. Examples of such tasks include installing patches and changing security settings. Figure 1 shows one such article.

Our goal is to automatically execute these support articles in the Windows 2000 environment. Here, the environment state is the set of visible user interface (UI) objects, and object properties such as label, location, and parent window. Possible commands include left-click, right-click, double-click, and type-into, all of which take a UI object as a parameter; type-into additionally requires a parameter for the input text.

\footnote{support.microsoft.com}
Table 2: Example features in the Windows domain. All features are binary, except for the normalized edit distance which is real-valued.

Table 2 lists some of the features we use for this domain. These features capture various aspects of the action under consideration, the current Windows GUI state, and the input instructions. For example, one lexical feature measures the similarity of a word in the sentence to the UI labels of objects in the environment. Environment-specific features, such as whether an object is currently in focus, are useful when selecting the object to manipulate. In total, there are 4,438 features.

6.1.1 Reward Function

Environment feedback can be used as a reward function in this domain. An obvious reward would be task completion (e.g., whether the stated computer problem was fixed). Unfortunately, verifying task completion is a challenging system issue in its own right. Instead, we rely on a noisy method of checking whether execution can proceed from one sentence to the next: at least one word in each sentence has to correspond to an object in the environment. For instance, in the sentence from Figure 2 word “Run” matches the Run...menu item. If no words in a sentence match a current environment object, then one of the previous sentences was analyzed incorrectly, so the history should receive negative reward. This reward is not guaranteed to penalize all incorrect histories, because there may be false positive matches between the sentence and the environment.

For a document \( d = (u_1, \ldots, u_n) \) and candidate history \( h \), \( r(h) = \sum_{i=1}^{n-1} r_d(u_i, u_{i+1}) \), where

\[ r_d(u_i, u_{i+1}) = \begin{cases} |W|/|u_i| & \text{if } u_{i+1} \text{ has word matches,} \\ 1 - n_c/|u_i| & \text{if } u_{i+1} \text{ has word matches,} \\ -1 & \text{otherwise.} \end{cases} \]

Here, \( n_c \) is the number of commands executed in the environment for sentence \( u_i \), and \( W \) is the words in the current history corresponding to those commands. The first case of the reward encourages longer sentences to map to more actions.

6.2 Crossblock: A Puzzle Game

Our second application is to a puzzle game called Crossblock, available online as a Flash game. Each of 50 puzzles is played on a grid, where some grid positions are filled with squares. The object of the game is to clear the grid by drawing vertical or horizontal line segments that remove groups of squares. Each segment must exactly cross a specific number of squares, ranging from two to seven, depending on the puzzle. Humans players have found this game challenging and engaging enough to warrant posting textual tutorials. A sample puzzle and tutorial are shown in Figure 3. We aim to learn how to interpret these novice-written tutorials to play Crossblock.

The environment here is defined by the state of the grid; in particular, we can view the set of objects as all the valid line segments, of which there can be dozens. The only command in this environment is clear, which takes a parameter specifying which line segment to remove. The challenge in this domain is identifying the correct object from references such as “the bottom four from the second column from the left.”

Features in this domain are defined as follows. For each pair \( (w, a) \) of vocabulary word \( w \) and
candidate action \( a \), we define a set of features capturing the orientation and location of action \( a \)'s selected line segment. These features are nonzero only if \( w \) appears in the current sentence. When \( w \) does appear, orientation is encoded with two binary features, one for vertical and one for horizontal; location is encoded with a set of features that measure the line’s horizontal and vertical distances from the edges and center of the entire grid. There are a total of 39 individual features per vocabulary term.

6.2.1 Reward Function

Unlike in the Windows domain, task completion for Crossblock is easy to verify directly. We define \( r(h) = 1 \) if \( h \) ends in a state where the puzzle is completed (i.e., the grid is entirely cleared), and zero otherwise.

7 Experimental Setup

Datasets For the Windows domain, our dataset consists of 128 documents, divided into 70 for training, 18 for development, and 40 for test. In the puzzle game domain, we use 50 tutorials, divided into 30 for training and 20 for test.\(^9\) Statistics for the datasets are shown below.

<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>Puzzle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # of documents</td>
<td>128</td>
<td>50</td>
</tr>
<tr>
<td>Total # of words</td>
<td>5562</td>
<td>994</td>
</tr>
<tr>
<td>Vocabulary size</td>
<td>610</td>
<td>46</td>
</tr>
<tr>
<td>Avg. words per sentence</td>
<td>9.93</td>
<td>19.88</td>
</tr>
<tr>
<td>Avg. sentences per document</td>
<td>4.38</td>
<td>1.00</td>
</tr>
<tr>
<td>Avg. actions per document</td>
<td>10.37</td>
<td>5.86</td>
</tr>
</tbody>
</table>

The data exhibits certain qualities that make for a challenging learning problem. For instance, there are a surprising variety of linguistic constructs — as Figure 4 shows, in the Windows domain even a simple command is expressed in at least six different ways.

Experimental Framework To apply our algorithm to the Windows domain, we use the Win32 application programming interface to simulate human interactions with the user interface and gather environment state information. The operating system environment is hosted within a virtual machine,\(^10\) allowing us to rapidly save and reset system state snapshots. For the puzzle game domain, we replicated the game with an implementation that facilitates automatic play.

As is commonly done in reinforcement learning, we use an exploration parameter to smooth the policy distribution (Sutton and Barto, 1998), set to 0.1 in our experiments. For Windows, the development set is used to select the best parameters. For Crossblock, we choose the parameters that produce the highest reward during training. During evaluation, we use these parameters to predict mappings for the test documents.

Evaluation Metrics For evaluation, we compare the results to manually constructed sequences of actions. We measure the number of correct actions, sentences, and documents. An action is correct if it matches the annotations in terms of command and parameters.\(^11\) A sentence is correct if all of its actions are correctly identified, and analogously for documents. Statistical significance is measured with the sign test.

Baselines We consider the following baselines to characterize the performance of our approach.

- **Full Supervision** Sequence prediction problems like ours are typically addressed using supervised techniques. We can get an idea of how a standard supervised approach would perform on this task, by using a reward signal based on manual annotations, as defined in Section 5.2. As shown there, policy gradient with this reward effectively becomes stochastic gradient ascent with a maximum likelihood objective.
- **Partial Supervision** We consider the case when only a subset of training documents is annotated, and environment reward is used for the remainder. Our method seamlessly combines these two kinds of rewards.
- **Random and Majority (Windows)** We also consider two naïve baselines. Both scan through each sentence from left to right. A command \( c \) is executed on the object whose

---

\(^9\) For Crossblock, because the number of puzzles is limited, we did not hold out a separate development set, and report averaged results over five training/test splits.

\(^10\) VMware Workstation, available at www.vmware.com

\(^11\) In these tasks, each action depends on the correct execution of all previous actions, so a single error can render the remainder of that document’s mapping incorrect.
<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>Puzzle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random baseline</td>
<td>0.128</td>
<td>0.101</td>
</tr>
<tr>
<td>Majority baseline</td>
<td>0.287</td>
<td>0.197</td>
</tr>
<tr>
<td>Environment reward</td>
<td>∗ 0.647</td>
<td>∗ 0.590</td>
</tr>
<tr>
<td>Partial supervision</td>
<td>○ 0.723</td>
<td>+ 0.702</td>
</tr>
<tr>
<td>Full supervision</td>
<td>○ 0.756</td>
<td>0.714</td>
</tr>
</tbody>
</table>

Table 3: Performance on the test set for our model with different reward signals and two baselines. Our evaluation measures the proportion of correct actions, sentences, and documents. Note the puzzle domain has only single-sentence documents, so its sentence and document scores are identical. The partial supervision line refers to 20 out of 70 annotated training documents for Windows, and 10 out of 30 for the puzzle. Each result marked with ∗ or ○ is a statistically significant improvement over the result immediately above it. ∗ indicates $p < 0.01$ and ○ indicates $p < 0.05$.

name is encountered first in the sentence. This command $c$ is either selected randomly, or set to the majority command, which is left-click. This procedure is repeated until no more words match environment objects.

- **Random (Puzzle)** We consider a baseline that randomly selects among the actions that are valid in the current game state.\(^\text{12}\)

### 8 Results

Table 3 presents evaluation results on the test sets. There are several indicators of the difficulty of this task. The random and majority baselines’ poor performance in both domains indicates that naïve approaches are inadequate for these tasks. The performance of the fully supervised approach provides further evidence that the task is challenging. This difficulty can be attributed in part to the large branching factor of possible actions at each step — on average, there are 27.14 choices per action in the Windows domain, and 9.78 in Crossblock.

In both domains, the learners relying only on environment reward perform well. In fact, in the puzzle domain its performance matches the accuracy of the fully supervised learner in terms of actions correct. Here, the reward function is very informative because task completion can be measured exactly. In the Windows domain, the fully supervised approach outperforms the environment-based learner by 11% in terms of action accuracy. However, annotating just 20 out of 70 training documents and adding them to the environment-based learner reduces this gap to 3%. Figure 5 shows the overall tradeoff between annotation effort and system performance. The ability to make this tradeoff is one of the advantages of our approach. The figure also shows that augmenting annotated documents with additional environment-reward documents invariably improves performance.

### 9 Conclusions

In this paper, we presented a reinforcement learning approach for inducing a mapping between instructions and actions. This approach is able to use environment-based rewards, such as task completion, to learn to analyze text. We showed that having reward can significantly reduce the need for annotations.

An interesting direction for future work is to explore how this approach can be applied to more complex text analysis tasks in the context of a virtual world. It could be particularly useful in scenarios where manual annotation is prohibitively expensive, but reward is readily available.

---

\(^\text{12}\)Since action selection is among objects, there is no natural majority baseline for the puzzle.
References


