Situated Mapping of Sequential Instructions to Actions with Single-step Reward Observation

Alane Suhr and **Yoav Artzi**
Department of Computer Science and Cornell Tech
Cornell University
New York, NY, 10044
{suhr, yoav}@cs.cornell.edu

Abstract

We propose a learning approach for mapping context-dependent sequential instructions to actions. We address the problem of discourse and state dependencies with an attention-based model that considers both the history of the interaction and the state of the world. To train from start and goal states without access to demonstrations, we propose SESTRA, a learning algorithm that takes advantage of single-step reward observations and immediate expected reward maximization. We evaluate on the SCONE domains, and show absolute accuracy improvements of 9.8%-24.9% across the domains over approaches that use high-level logical representations.

1 Introduction

An agent executing a sequence of instructions must address multiple challenges, including grounding the language to its observed environment, reasoning about discourse dependencies, and generating actions to complete high-level goals. For example, consider the environment and instructions in Figure 1, in which a user describes moving chemicals between beakers and mixing chemicals together. To execute the second instruction, the agent needs to resolve *sixth beaker* and *last one* to objects in the environment. The third instruction requires resolving *it* to the rightmost beaker mentioned in the second instruction, and reasoning about the set of actions required to mix the colors in the beaker to brown. In this paper, we describe a model and learning approach to map sequences of instructions to actions. Our model considers previous utterances and the world state to select actions, learns to combine simple actions to achieve complex goals, and can be trained using goal states without access to demonstrations.

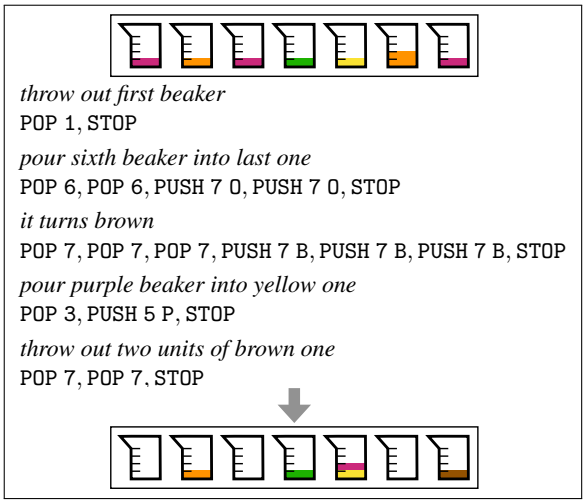

Figure 1: Example from the SCONE (Long et al., 2016) ALCHEMY domain, including a start state (top), sequence of instructions, and a goal state (bottom). Each instruction is annotated with a sequence of actions from the set of actions we define for ALCHEMY.

The majority of work on executing sequences of instructions focuses on mapping instructions to high-level formal representations, which are then evaluated to generate actions (e.g., Chen and Mooney, 2011; Long et al., 2016). For example, the third instruction in Figure 1 will be mapped to $\mathrm{mix}(\mathrm{prev\_arg1})$, indicating that the mix action should be applied to first argument of the previous action (Long et al., 2016; Guu et al., 2017). In contrast, we focus on directly generating the sequence of actions. This requires resolving references without explicitly modeling them, and learning the sequences of actions required to complete high-level actions; for example, that mixing requires removing everything in the beaker and replacing with the same number of brown items.

A key challenge in executing sequences of instructions is considering contextual cues from both the history of the interaction and the state of the world. Instructions often refer to previously

mentioned objects (e.g., *it* in Figure 1) or actions (e.g., *do it again*). The world state provides the set of objects the instruction may refer to, and implicitly determines the available actions. For example, liquid can not be removed from an empty beaker. Both types of contexts continuously change during an interaction. As new instructions are given, the instruction history expands, and as the agent acts the world state changes. We propose an attention-based model that takes as input the current instruction, previous instructions, the initial world state, and the current state. At each step, the model computes attention encodings of the different inputs, and predicts the next action to execute.

We train the model given instructions paired with start and goal states without access to the correct sequence of actions. During training, the agent learns from rewards received through exploring the environment with the learned policy by mapping instructions to sequences of actions. In practice, the agent learns to execute instructions gradually, slowly correctly predicting prefixes of the correct sequences of increasing length as learning progress. A key challenge is learning to correctly select actions that are only required later in execution sequences. Early during learning, these actions receive negative updates, and the agent learns to assign them low probabilities. This results in an exploration problem in later stages, where actions that are only required later are not sampled during exploration. For example, in the ALCHEMY domain shown in Figure 1, the agent behavior early during execution of instructions can be accomplished by only using `POP` actions. As a result, the agent quickly learns a strong bias against `PUSH` actions, which in practice prevents the policy from exploring them again. We address this with a learning algorithm that observes the reward for all possible actions for each visited state, and maximizes the immediate expected reward.

We evaluate our approach on SCONE (Long et al., 2016), which includes three domains, and is used to study recovering predicate logic meaning representations for sequential instructions. We study the problem of generating a sequence of low-level actions, and re-define the set of actions for each domain. For example, we treat the beakers in the ALCHEMY domain as stacks and use only `POP` and `PUSH` actions. Our approach robustly learns to execute sequential instructions with up to 89.1% task-completion accuracy for single instruction, and 62.7% for complete sequences. Our code is available at https://github.com/clic-lab/scone.

2 Technical Overview

Task and Notation Let $\mathcal{S}$ be the set of all possible world states, $\mathcal{X}$ be the set of all natural language instructions, and $\mathcal{A}$ be the set of all actions. An instruction $\bar{x} \in \mathcal{X}$ of length $|\bar{x}|$ is a sequence of tokens $\langle x_1, ...x_{|\bar{x}|} \rangle$. Executing an action modifies the world state following a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. For example, the ALCHEMY domain includes seven beakers that contain colored liquids. The world state defines the content of each beaker. We treat each beaker as a stack. The actions are `POP N` and `PUSH N C`, where $1 \leq$ `N` ≤ 7 is the beaker number and `C` is one of six colors. There are a total of 50 actions, including the `STOP` action. Section 6 describes the domains in detail.

Given a start state s_1 and a sequence of instructions $\langle \bar{x}_1, \ldots, \bar{x}_n \rangle$, our goal is to generate the sequence of actions specified by the instructions starting from s_1. We treat the execution of a sequence of instructions as executing each instruction in turn. The execution $\bar{e}$ of an instruction $\bar{x}_i$ starting at a state s_1 and given the history of the instruction sequence $\langle \bar{x}_1, \ldots, \bar{x}_{i-1} \rangle$ is a sequence of state-action pairs $\bar{e} = \langle (s_1, a_1), ..., (s_m, a_m) \rangle$, where $a_k \in \mathcal{A}$, $s_{k+1} = T(s_k, a_k)$. The final action a_m is the special action `STOP`, which indicates the execution has terminated. The final state is then s_m, as $T(s_k, \texttt{STOP}) = s_k$. Executing a sequence of instructions in order generates a sequence $\langle \bar{e}_1, ..., \bar{e}_n \rangle$, where $\bar{e}_i$ is the execution of instruction $\bar{x}_i$. When referring to states and actions in an indexed execution $\bar{e}_i$, the k-th state and action are $s_{i,k}$ and $a_{i,k}$. We execute instructions one after the other: $\bar{e}_1$ starts at the interaction initial state s_1 and $s_{i+1,1} = s_{i,|\bar{e}_i|}$, where $s_{i+1,1}$ is the start state of $\bar{e}_{i+1}$ and $s_{i,|\bar{e}_i|}$ is the final state of $\bar{e}_i$.

Model We model the agent with a neural network policy (Section 4). At step k of executing the i-th instruction, the model input is the current instruction $\bar{x}_i$, the previous instructions $\langle \bar{x}_1, \ldots, \bar{x}_{i-1} \rangle$, the world state s_1 at the beginning of executing $\bar{x}_i$, and the current state s_k. The model predicts the next action a_k to execute. If $a_k =$ `STOP`, we switch to the next instruction, or if at the end of the instruction sequence, terminate. Otherwise, we update the state to $s_{k+1} = T(s_k, a_k)$. The model uses attention to

process the different inputs and a recurrent neural network (RNN) decoder to generate actions (Bahdanau et al., 2015).

Learning We assume access to a set of N instruction sequences, where each instruction in each sequence is paired with its start and goal states. During training, we create an example for each instruction. Formally, the training set is $\{(\bar{x}_i^{(j)}, s_{i,1}^{(j)}, \langle \bar{x}_1^{(j)}, \ldots, \bar{x}_{i-1}^{(j)} \rangle, g_i^{(j)})\}_{j=1,i=1}^{N,n^{(j)}}$, where $\bar{x}_i^{(j)}$ is an instruction, $s_{i,1}^{(j)}$ is a start state, $\langle \bar{x}_1^{(j)}, \ldots, \bar{x}_{i-1}^{(j)} \rangle$ is the instruction history, $g_i^{(j)}$ is the goal state, and $n^{(j)}$ is the length of the j-th instruction sequence. This training data contains no evidence about the actions and intermediate states required to execute each instruction.[1] We use a learning method that maximizes the expected immediate reward for a given state (Section 5). The reward accounts for task-completion and distance to the goal via potential-based reward shaping.

Evaluation We evaluate exact task completion for sequences of instructions on a test set $\{(s_1^{(j)}, \langle \bar{x}_1^{(j)}, \ldots, \bar{x}_{n_j}^{(j)} \rangle, g^{(j)})\}_{j=1}^{N}$, where $g^{(j)}$ is the oracle goal state of executing instructions $\bar{x}_1^{(j)}, \ldots, \bar{x}_{n_j}^{(j)}$ in order starting from $s_1^{(j)}$. We also evaluate single-instruction task completion using per-instruction annotated start and goal states.

3 Related Work

Executing instructions has been studied using the SAIL corpus (MacMahon et al., 2006) with focus on navigation using high-level logical representations (Chen and Mooney, 2011; Chen, 2012; Artzi and Zettlemoyer, 2013; Artzi et al., 2014) and low-level actions (Mei et al., 2016). While SAIL includes sequences of instructions, the data demonstrates limited discourse phenomena, and instructions are often processed in isolation. Approaches that consider as input the entire sequence focused on segmentation (Andreas and Klein, 2015). Recently, other navigation tasks were proposed with focus on single instructions (Anderson et al., 2018; Janner et al., 2018). We focus on sequences of environment manipulation instructions and modeling contextual cues from both the changing environment and instruction history. Manipulation using single-sentence instructions has been studied using the Blocks domain (Bisk et al., 2016, 2018; Misra et al., 2017; Tan and Bansal, 2018). Our work is related to the work of Branavan et al. (2009) and Vogel and Jurafsky (2010). While both study executing sequences of instructions, similar to SAIL, the data includes limited discourse dependencies. In addition, both learn with rewards computed from surface-form similarity between text in the environment and the instruction. We do not rely on such similarities, but instead use a state distance metric.

Language understanding in interactive scenarios that include multiple turns has been studied with focus on dialogue for querying database systems using the ATIS corpus (Hemphill et al., 1990; Dahl et al., 1994). Tür et al. (2010) surveys work on ATIS. Miller et al. (1996), Zettlemoyer and Collins (2009), and Suhr et al. (2018) modeled context dependence in ATIS for generating formal representations. In contrast, we focus on environments that change during execution and directly generating environment actions, a scenario that is more related to robotic agents than database query.

The SCONE corpus (Long et al., 2016) was designed to reflect a broad set of discourse context-dependence phenomena. It was studied extensively using logical meaning representations (Long et al., 2016; Guu et al., 2017; Fried et al., 2018). In contrast, we are interested in directly generating actions that modify the environment. This requires generating lower-level actions and learning procedures that are otherwise hardcoded in the logic (e.g., mixing action in Figure 1). Except for Fried et al. (2018), previous work on SCONE assumes access only to the initial and final states during training. This form of supervision does not require operating the agent manually to acquire the correct sequence of actions, a difficult task in robotic agents with complex control. Goal state supervision has been studied for instructional language (e.g., Branavan et al., 2009; Artzi and Zettlemoyer, 2013; Bisk et al., 2016), and more extensively in question answering when learning with answer annotations only (e.g., Clarke et al., 2010; Liang et al., 2011; Kwiatkowski et al., 2013; Berant et al., 2013; Berant and Liang, 2014, 2015; Liang et al., 2017).

4 Model

We map sequences of instructions $\langle \bar{x}_1, \ldots, \bar{x}_n \rangle$ to actions by executing the instructions in or-

[1] This training set is a subset of the data used in previous work (Section 6, Guu et al., 2015), in which training uses all instruction sequences of length 1 and 2.

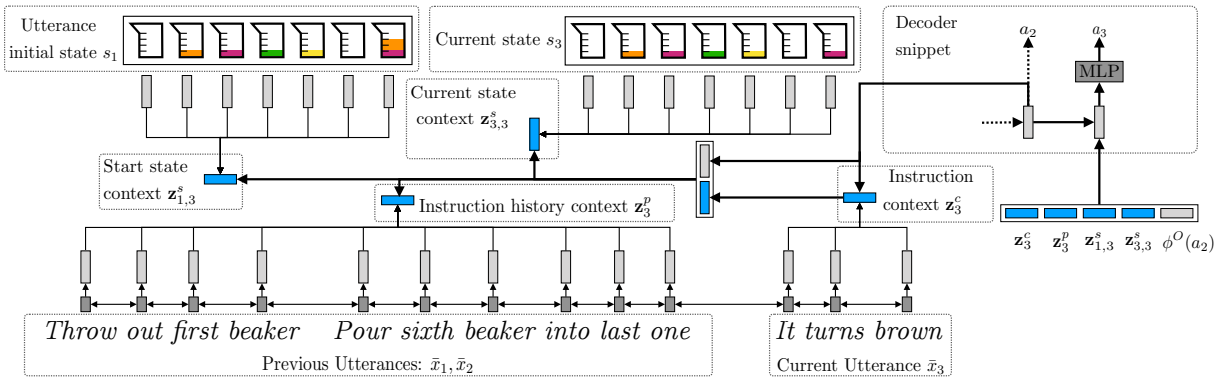

Figure 2: Illustration of the model architecture while generating the third action a_3 in the third utterance $\bar{x}_3$ from Figure 1. Context vectors computed using attention are highlighted in blue. The model takes as input vector encodings from the current and previous instructions $\bar{x}_1$, $\bar{x}_2$, and $\bar{x}_3$, the initial state s_1, the current state s_3, and the previous action a_2. Instruction encodings are computed with a bidirectional RNN. We attend over the previous and current instructions and the initial and current states. We use an MLP to select the next action.

der. The model generates an execution $\bar{e} = \langle (s_1, a_1), \ldots, (s_{m_i}, a_{m_i}) \rangle$ for each instruction $\bar{x}_i$. The agent context, the information available to the agent at step k, is $\tilde{s}_k = (\bar{x}_i, \langle \bar{x}_1, \ldots, \bar{x}_{i-1} \rangle, s_k, \bar{e}[:k])$, where $\bar{e}[:k]$ is the execution up until but not including step k. In contrast to the world state, the agent context also includes instructions and the execution so far. The agent policy $\pi_\theta(\tilde{s}_k, a)$ is modeled as a probabilistic neural network parametrized by θ, where $\tilde{s}_k$ is the agent context at step k and a is an action. To generate executions, we generate one action at a time, execute the action, and observe the new world state. In step k of executing the i-th instruction, the network inputs are the current utterance $\bar{x}_i$, the previous instructions $\langle \bar{x}_1, \ldots, \bar{x}_{i-1} \rangle$, the initial state s_1 at beginning of executing $\bar{x}_i$, and the current state s_k. When executing a sequence of instructions, the initial state s_1 is either the state at the beginning of executing the sequence or the final state of the execution of the previous instruction. Figure 2 illustrates our architecture.

We generate continuous vector representations for all inputs. Each input is represented as a set of vectors that are then processed with an attention function to generate a single vector representation (Luong et al., 2015). We assume access to a domain-specific encoding function ENC(s) that, given a state s, generates a set of vectors S representing the objects in the state. For example, in the ALCHEMY domain, a vector is generated for each beaker using an RNN. Section 6 describes the different domains and their encoding functions.

We use a single bidirectional RNN with a long short-term memory (LSTM; Hochreiter and Schmidhuber, 1997) recurrence to encode the instructions. All instructions $\bar{x}_1, \ldots, \bar{x}_i$ are encoded with a single RNN by concatenating them to $\bar{x}'$. We use two delimiter tokens: one separates previous instructions, and the other separates the previous instructions from the current one. The forward LSTM RNN hidden states are computed as:[2]

$$\overrightarrow{\mathbf{h}_{j+1}} = \overrightarrow{\text{LSTM}^E}\left(\phi^I(x'_{j+1}); \overrightarrow{\mathbf{h}_j}\right) \ ,$$

where ϕ^I is a learned word embedding function and $\overrightarrow{\text{LSTM}^E}$ is the forward LSTM recurrence function. We use a similar computation to compute the backward hidden states $\overleftarrow{\mathbf{h}_j}$. For each token x'_j in $\bar{x}'$, a vector representation $\mathbf{h}'_j = \left[\overrightarrow{\mathbf{h}_j}; \overleftarrow{\mathbf{h}_j}\right]$ is computed. We then create two sets of vectors, one for all the vectors of the current instruction and one for the previous instructions:

$$X^c = \{\mathbf{h}'_j\}_{j=J}^{J+|\bar{x}_i|}$$
$$X^p = \{\mathbf{h}'_j\}_{j=0}^{j<J}$$

where J is the index in $\bar{x}'$ where the current instruction $\bar{x}_i$ begins. Separating the vectors to two sets will allows computing separate attention on the current instruction and previous ones.

To compute each input representation during decoding, we use a bi-linear attention function (Luong et al., 2015). Given a set of vectors H, a query vector $\mathbf{h}^q$, and a weight matrix $\mathbf{W}$, the attention function ATTEND$(H, \mathbf{h}^q, \mathbf{W})$ computes a context vector $\mathbf{z}$:

$$\alpha_i \propto \exp(\mathbf{h}_i^\intercal \mathbf{W} \mathbf{h}^q) : i = 0, \ldots, |H|$$
$$\mathbf{z} = \sum_{i=1}^{|H|} \alpha_i \mathbf{h}_i \ .$$

[2] To simplify the notation, we omit the memory cell (often denoted as $\mathbf{c}_j$) from all LSTM descriptions. We use only the hidden state $\mathbf{h}_j$ to compute the intended representations (e.g., for the input text tokens). All LSTMs in this paper use zero vectors as initial hidden state $\mathbf{h}_0$ and initial cell memory $\mathbf{c}_0$.

We use a decoder to generate actions. At each time step k, we compute an input representation using the attention function, update the decoder state, and compute the next action to execute. Attention is first computed over the vectors of the current instruction, which is then used to attend over the other inputs. We compute the context vectors $\mathbf{z}_k^c$ and $\mathbf{z}_k^p$ for the current instruction and previous instructions:

$$\begin{aligned}\mathbf{z}_k^c &= \text{ATTEND}(X^c, \mathbf{h}_{k-1}^d, \mathbf{W}^c) \\ \mathbf{z}_k^p &= \text{ATTEND}(X^p, [\mathbf{h}_{k-1}^d, \mathbf{z}_k^c], \mathbf{W}^p) \ ,\end{aligned}$$

where $\mathbf{h}_{k-1}^d$ is the decoder hidden state for step $k-1$, and X^c and X^p are the sets of vector representations for the current instruction and previous instructions. Two attention heads are used over both the initial and current states. This allows the model to attend to more than one location in a state at once, for example when transferring items from one beaker to another in ALCHEMY. The current state is computed by the transition function $s_k = T(s_{k-1}, a_{k-1})$, where s_{k-1} and a_{k-1} are the state and action at step $k-1$. The context vectors for the initial state s_1 and the current state s_k are:

$$\begin{aligned}\mathbf{z}_{1,k}^s &= [\text{ATTEND}(\text{ENC}(s_1), [\mathbf{h}_{k-1}^d, \mathbf{z}_k^c], \mathbf{W}^{s_b,1}); \\ &\quad\ \text{ATTEND}(\text{ENC}(s_1), [\mathbf{h}_{k-1}^d, \mathbf{z}_k^c], \mathbf{W}^{s_b,2})] \\ \mathbf{z}_{k,k}^s &= [\text{ATTEND}(\text{ENC}(s_k), [\mathbf{h}_{k-1}^d, \mathbf{z}_k^c], \mathbf{W}^{s_c,1}); \\ &\quad\ \text{ATTEND}(\text{ENC}(s_k), [\mathbf{h}_{k-1}^d, \mathbf{z}_k^c], \mathbf{W}^{s_c,2})] \ ,\end{aligned}$$

where all $\mathbf{W}^{*,*}$ are learned weight matrices.

We concatenate all computed context vectors with an embedding of the previous action a_{k-1} to create the input for the decoder:

$$\begin{aligned}\mathbf{h}_k &= \tanh([\mathbf{z}_k^c; \mathbf{z}_k^p; \mathbf{z}_{1,k}^s; \mathbf{z}_{k,k}^s; \phi^O(a_{k-1})]\mathbf{W}^d + \mathbf{b}^d) \\ \mathbf{h}_k^d &= \text{LSTM}^D\left(\mathbf{h}_k; \mathbf{h}_{k-1}^d\right) \ ,\end{aligned}$$

where ϕ^O is a learned action embedding function and LSTM^D is the LSTM decoder recurrence.

Given the decoder state $\mathbf{h}_k^d$, the next action a_k is predicted with a multi-layer perceptron (MLP). The actions in our domains decompose to an action type and at most two arguments.[3] For example, the action `PUSH 1 B` in ALCHEMY has the type `PUSH` and two arguments: a beaker number and a color. Section 6 describes the actions of each domain. The probability of an action is:

$$\begin{aligned}\mathbf{h}_k^a &= \tanh(\mathbf{h}_k^d \mathbf{W}^a) \\ s_{k,a_T} &= \mathbf{h}_k^a \mathbf{b}_{a_T} \\ s_{k,a_1} &= \mathbf{h}_k^a \mathbf{b}_{a_1} \\ s_{k,a_2} &= \mathbf{h}_k^a \mathbf{b}_{a_2} \\ p(a_k = a_T(a_1, a_2) \mid \tilde{s}_k; \theta) &\propto \\ &\exp(s_{k,a_T} + s_{k,a_1} + s_{k,a_2}) \ ,\end{aligned}$$

where a_T, a_1, and a_2 are an action type, first argument, and second argument. If the predicted action is `STOP`, the execution is complete. Otherwise, we execute the action a_k to generate the next state s_{k+1}, and update the agent context $\tilde{s}_k$ to $\tilde{s}_{k+1}$ by appending the pair (s_k, a_k) to the execution $\bar{e}$ and replacing the current state with s_{k+1}.

The model parameters θ include: the embedding functions ϕ^I and ϕ^O; the recurrence parameters for $\overrightarrow{\text{LSTM}^E}$, $\overleftarrow{\text{LSTM}^E}$, and LSTM^D; $\mathbf{W}^C$, $\mathbf{W}^P$, $\mathbf{W}^{s_b,1}$, $\mathbf{W}^{s_b,2}$, $\mathbf{W}^{s_c,1}$, $\mathbf{W}^{s_c,2}$, $\mathbf{W}^d$, $\mathbf{W}^a$, and $\mathbf{b}^d$; and the domain dependent parameters, including the parameters of the encoding function ENC and the action type, first argument, and second argument weights $\mathbf{b}_{a_T}$, $\mathbf{b}_{a_1}$, and $\mathbf{b}_{a_2}$.

5 Learning

We estimate the policy parameters θ using an exploration-based learning algorithm that maximizes the immediate expected reward. Broadly speaking, during learning, we observe the agent behavior given the current policy, and for each visited state compute the expected immediate reward by observing rewards for all actions. We assume access to a set of training examples $\{(\bar{x}_i^{(j)}, s_{i,1}^{(j)}, \langle \bar{x}_1^{(j)}, \ldots, \bar{x}_{i-1}^{(j)} \rangle, g_i^{(j)})\}_{j=1,i=1}^{N,n^{(j)}}$, where each instruction $\bar{x}_i^{(j)}$ is paired with a start state $s_{i,1}^{(j)}$, the previous instructions in the sequence $\langle \bar{x}_1^{(j)}, \ldots, \bar{x}_{i-1}^{(j)} \rangle$, and a goal state $g_i^{(j)}$.

Reward The reward $R_i^{(j)} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined for each example j and instruction i:

$$R_i^{(j)}(s, a, s') = P_i^{(j)}(s, a, s') + \phi_i^{(j)}(s') - \phi_i^{(j)}(s) \ ,$$

where s is a source state, a is an action, and s' is a target state.[4] $P_i^{(j)}(s, a, s')$ is a problem reward and $\phi_i^{(j)}(s') - \phi_i^{(j)}(s)$ is a shaping term. The problem reward $P_i^{(j)}(s, a, s')$ is positive for stopping at the goal $g_i^{(j)}$ and negative for stopping in an incorrect

[3] We use a `NULL` argument for unused arguments.

[4] While the reward function is defined for any state-action-state tuple, in practice, it is used during learning with tuples that follow the system dynamics, $s' = T(s, a)$.

Algorithm 1 SESTRA: **S**ingle-**st**ep **R**eward Observa**t**ion.

Input: Training data $\{(\bar{x}_i^{(j)}, s_{i,1}^{(j)}, \langle \bar{x}_1^{(j)}, \ldots, \bar{x}_{i-1}^{(j)} \rangle, g_i^{(j)})\}_{j=1,i=1}^{N,n^{(j)}}$, learning rate μ, entropy regularization coefficient λ, episode limit horizon M.

Definitions: π_θ is a policy parameterized by θ, `BEG` is a special action to use for the first decoder step, and `STOP` indicates end of an execution. $T(s, a)$ is the state transition function, H is an entropy function, $R_i^{(j)}(s, a, s')$ is the reward function for example j and instruction i, and RMSPROP divides each weight by a running average of its squared gradient (Tieleman and Hinton, 2012).

Output: Parameters θ defining a learned policy π_θ.

1: **for** $t = 1, \ldots, T, j = 1, \ldots, N$ **do**
2: **for** $i = 1, \ldots, n^{(j)}$ **do**
3: $\bar{e} \leftarrow \langle\,\rangle, k \leftarrow 0, a_0 \leftarrow$ `BEG`
4: » Rollout up to `STOP` or episode limit.
5: **while** $a_k \neq$ `STOP` $\wedge\ k < M$ **do**
6: $k \leftarrow k + 1$
7: $\tilde{s}_k \leftarrow (\bar{x}_i, \langle \bar{x}_1, \ldots, \bar{x}_{i-1} \rangle, s_k, \bar{e}[:k])$
8: » Sample an action from policy.
9: $a_k \sim \pi_\theta(\tilde{s}_k, \cdot)$
10: $s_{k+1} \leftarrow T(s_k, a_k)$
11: $\bar{e} \leftarrow [\bar{e}; \langle (s_k, a_k) \rangle]$
12: $\Delta \leftarrow \bar{0}$
13: **for** $k' = 1, \ldots, k$ **do**
14: » Compute the entropy of $\pi_\theta(\tilde{s}_{k'}, \cdot)$.
15: $\Delta \leftarrow \Delta + \lambda \nabla_\theta H(\pi_\theta(\tilde{s}_{k'}, \cdot))$
16: **for** $a \in \mathcal{A}$ **do**
17: $s' \leftarrow T(s_{k'}, a)$
18: » Compute gradient for action a.
19: $\Delta \leftarrow \Delta + R_i^{(j)}(s_{k'}, a, s') \nabla_\theta \pi_\theta(\tilde{s}_{k'}, a)$
20: $\theta \leftarrow \theta + \mu \text{RMSPROP}\left(\frac{\Delta}{k}\right)$
21: **return** θ

state or taking an invalid action:

$$P_i^{(j)}(s, a, s') = \begin{cases} 1.0 & a = \texttt{STOP} \wedge s' = g_i^{(j)} \\ -1.0 & a = \texttt{STOP} \wedge s' \neq g_i^{(j)} \\ -1.0 - \delta & s = s' \\ -\delta & \text{otherwise} \end{cases},$$

where δ is a verbosity penalty. The case $s = s'$ indicates that a was invalid in state s, as in this domain, all valid actions except `STOP` modify the state. We use a potential-based shaping term $\phi_i^{(j)}(s') - \phi_i^{(j)}(s)$ (Ng et al., 1999), where $\phi_i^{(j)}(s) = -||s - g_i^{(j)}||$ computes the edit distance between the state s and the goal, measured over the objects in each state. The shaping term densifies the reward, providing a meaningful signal for learning in nonterminal states.

Objective We maximize the immediate expected reward over all actions and use entropy regularization. The gradient is approximated by sampling an execution $\bar{e} = \langle (s_1, a_1), \ldots, (s_k, a_k) \rangle$ using our current policy:

$$\nabla_\theta \mathcal{J} = \frac{1}{k} \sum_{k'=1}^{k} \left(\sum_{a \in \mathcal{A}} R\left(s_k, a, T(s_k, a)\right) \nabla_\theta \pi(\tilde{s}_k, a) + \lambda \nabla_\theta H(\pi(\tilde{s}_k, \cdot)) \right),$$

where $H(\pi(\tilde{s}_k, \cdot)$ is the entropy term.

Algorithm Algorithm 1 shows the Single-step Reward Observation (SESTRA) learning algorithm. We iterate over the training data T times (line 1). For each example j and turn i, we first perform a rollout by sampling an execution $\bar{e}$ from π_θ with at most M actions (lines 5-11). If the rollout reaches the horizon without predicting `STOP`, we set the problem reward $P_i^{(j)}$ to -1.0 for the last step. Given the sampled states visited, we compute the entropy (line 15) and observe the immediate reward for all actions (line 19) for each step. Entropy and rewards are used to accumulate the gradient, which is applied to the parameters using RMSPROP (Dauphin et al., 2015) (line 20).

Discussion Observing the rewards for all actions for each visited state addresses an on-policy learning exploration problem. Actions that consistently receive negative reward early during learning will be visited with very low probability later on, and in practice, often not explored at all. Because the network is randomly initialized, these early negative rewards are translated into strong general biases that are not grounded well in the observed context. Our algorithm exposes the agent to such actions later on when they receive positive rewards even though the agent does not explore them during rollout. For example, in ALCHEMY, `POP` actions are sufficient to complete the first steps of good executions. As a result, early during learning, the agent learns a strong bias against `PUSH` actions. In practice, the agent then will not explore `PUSH` actions again. In our algorithm, as the agent learns to roll out the correct `POP` prefix, it is then exposed to the reward for the first `PUSH` even though it likely sampled another `POP`. It then unlearns its bias towards predicting `POP`.

Our learning algorithm can be viewed as a cost-sensitive variant of the oracle in DAGGER (Ross et al., 2011), where it provides the rewards for all actions instead of an oracle action. It is also related to Locally Optimal Learning to Search (LOLS; Chang et al., 2015) with two key distinctions: (a) instead of using different roll-in and roll-out policies, we use the model policy; and (b) we branch at each step, instead of once, but do not rollout

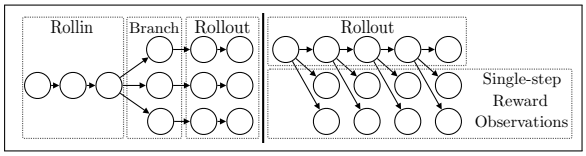

Figure 3: Illustration of LOLS (left; Chang et al., 2015) and our learning algorithm (SESTRA, right). LOLS branches a single time, and samples complete rollout for each branch to obtain the trajectory loss. SESTRA uses a complete on-policy rollout and single-step branching for all actions in each sample state.

	ALC	SCE	TAN
# Sequences (train)	3657	3352	4189
# Sequences (dev)	245	198	199
# Sequences (test)	899	1035	800
Mean instruction length	8.0±3.2	10.5±5.5	5.4±2.4
Vocabulary size	695	816	475

Table 1: Data statistics for ALCHEMY (ALC), SCENE (SCE), and TANGRAMS (TAN).

	Refs/Ex		1	2	3	4
ALCHEMY	1.4	Coref.	28	7	2	0
		Ellipsis	0	0	3	1
SCENE	2.4	Coref.	49	16	5	3
		Ellipsis	0	0	0	0
TANGRAMS	1.7	Coref.	25	14	2	1
		Ellipsis	4	0	0	0

Table 2: Counts of discourse phenomena in SCONE from 30 randomly selected development interactions for each domain. We count occurrences of coreference between instructions (e.g., *he leaves* in SCENE) and ellipsis (e.g., *then, drain 2 units* in ALCHEMY), when the last explicit mention of the referent was 1, 2, 3, or 4 turns in the past. We also report the average number of multi-turn references per interaction (Refs/Ex).

from branched actions since we only optimize the immediate reward. Figure 3 illustrates the comparison. Our summation over immediate rewards for all actions is related the summation of estimated Q-values for all actions in the Mean Actor-Critic algorithm (Asadi et al., 2017). Finally, our approach is related to Misra et al. (2017), who also maximize the immediate reward, but do not observe rewards for all actions for each state.

6 SCONE Domains and Data

SCONE has three domains: ALCHEMY, SCENE, and TANGRAMS. Each interaction contains five instructions. Table 1 shows data statistics. Table 2 shows discourse reference analysis. State encodings are detailed in the Supplementary Material.

ALCHEMY Each environment in ALCHEMY contains seven numbered beakers, each containing up to four colored chemicals in order. Figure 1 shows an example. Instructions describe pouring chemicals between and out of beakers, and mixing beakers. We treat all beakers as stacks. There are two action types: `PUSH` and `POP`. `POP` takes a beaker index, and removes the top color. `PUSH` takes a beaker index and a color, and adds the color at the top of the beaker. To encode a state, we encode each beaker with an RNN, and concatenate the last output with the beaker index embedding. The set of vectors is the state embedding.

SCENE Each environment in SCENE contains ten positions, each containing at most one person defined by a shirt color and an optional hat color. Instructions describe adding or removing people, moving a person to another position, and moving a person's hat to another person. There are four action types: `ADD_PERSON`, `ADD_HAT`, `REMOVE_PERSON`, and `REMOVE_HAT`. `ADD_PERSON` and `ADD_HAT` take a position to place the person or hat and the color of the person's shirt or hat. `REMOVE_PERSON` and `REMOVE_HAT` take the position to remove a person or hat from. To encode a state, we use a bidirectional RNN over the ordered positions. The input for each position is a concatenation of the color embeddings for the person and hat. The set of RNN hidden states is the state embedding.

TANGRAMS Each environment in TANGRAMS is a list containing at most five unique objects. Instructions describe removing or inserting an object into a position in the list, or swapping the positions of two items. There are two action types: `INSERT` and `REMOVE`. `INSERT` takes the position to insert an object, and the object identifier. `REMOVE` takes an object position. We embed each object by concatenating embeddings for its type and position. The resulting set is the state embedding.

7 Experimental Setup

Evaluation Following Long et al. (2016), we evaluate task completion accuracy using exact match between the final state and the annotated goal state. We report accuracy for complete interactions (5utts), the first three utterances of each interaction (3utts), and single instructions (Inst). For single instructions, execution starts from the annotated start state of the instruction.

Systems We report performance of ablations and two baseline systems: POLICYGRADIENT: policy gradient with cumulative episodic reward without a baseline, and CONTEXTUALBANDIT: the contextual bandit approach of Misra et al. (2017). Both systems use the reward with the

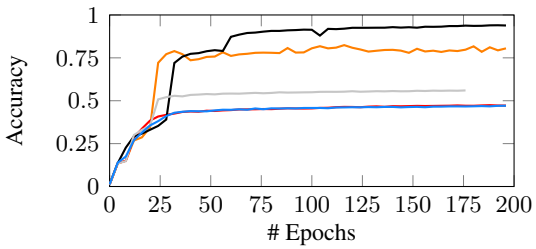

Figure 4: Instruction-level training accuracy per epoch when training five models on SCENE, demonstrating the effect of randomization in the learning method. Three of five experiments fail to learn effective models. The red and blue learning trajectories are overlapping.

shaping term and our model. We also report supervised learning results (SUPERVISED) by heuristically generating correct executions and computing maximum-likelihood estimate using context-action demonstration pairs. Only the supervised approach uses the heuristically generated labels. Although the results are not comparable, we also report the performance of previous approaches to SCONE. All three approaches generate logical representations based on lambda calculus. In contrast to our approach, this requires an ontology of hand built symbols and rules to evaluate the logical forms. Fried et al. (2018) uses supervised learning with annotated logical forms.

Training Details For test results, we run each experiment five times and report results for the model with best validation interaction accuracy. For ablations, we do the same with three experiments. We use a batch size of 20. We stop training using a validation set sampled from the training data. We hold the validation set constant for each domain for all experiments. We use patience over the average reward, and select the best model using interaction-level (5utts) validation accuracy. We tune λ, δ, and M on the development set. The selected values and other implementation details are described in the Supplementary Material.

8 Results

Table 3 shows test results. Our approach significantly outperforms POLICYGRADIENT and CONTEXTUALBANDIT, both of which suffer due to biases learned early during learning, hindering later exploration. This problem does not appear in TANGRAMS, where no action type is dominant at the beginning of executions, and all methods perform well. POLICYGRADIENT completely fails to learn ALCHEMY and SCENE due to observing only negative total rewards early during learning. Using a baseline, for example with an actor-critic method, will potentially close the gap to CONTEXTUALBANDIT. However, it is unlikely to address the on-policy exploration problem.

Table 4 shows development results, including model ablation studies. Removing previous instructions (– previous instructions) or both states (– current and initial state) reduces performance across all domains. Removing only the initial state (– initial state) or the current state (– current state) shows mixed results across the domains. Providing access to both initial and current states increases performance for ALCHEMY, but reduces performance on the other domains. We hypothesize that this is due to the increase in the number of parameters outweighing what is relatively marginal information for these domains. In our development and test results we use a single architecture across the three domains, the full approach, which has the highest interactive-level accuracy when averaged across the three domains (62.7 5utts). We also report mean and standard deviation for our approach over five trials. We observe exceptionally high variance in performance on SCENE, where some experiments fail to learn and training performance remains exceptionally low (Figure 4). This highlights the sensitivity of the model to the random effects of initialization, dropout, and ordering of training examples.

We analyze the instruction-level errors made by our best models when the agent is provided the correct initial state for the instruction. We study fifty examples in each domain to identify the type of failures. Table 5 shows the counts of major error categories. We consider multiple reference resolution errors. State reference errors indicate a failure to resolve a reference to the world state. For example, in ALCHEMY, the phrase *leftmost red beaker* specifies a beaker in the environment. If the model picked the correct action, but the wrong beaker, we count it as a state reference. We distinguish between multi-turn reference errors that should be feasible, and these that that are impossible to solve without access to states before executing previous utterances, which are not provided to our model. For example, in TANGRAMS, the instruction *put it back in the same place* refers to a previously-removed item. Because the agent only has access to the world state after following this instruction, it does not observe what kind of item was previously removed, and cannot identify the item to add. We

System	ALCHEMY			SCENE			TANGRAMS		
	Inst	3utts	5utts	Inst	3utts	5utts	Inst	3utts	5utts
Long et al. (2016)	–	56.8	52.3	–	64.9	27.6	–	23.2	14.7
Guu et al. (2017)	–	66.9	52.9	–	65.8	37.1	–	64.8	46.2
Fried et al. (2018)	–	–	72.0	–	–	72.7	–	–	69.6
SUPERVISED	89.4	73.3	62.3	88.8	78.9	66.4	86.6	81.4	60.1
POLICYGRADIENT	0.0	0.0	0.0	0.0	1.3	0.2	84.1	77.4	54.9
CONTEXTUALBANDIT	73.8	36.0	25.7	15.1	2.9	4.4	84.8	76.9	57.9
Our approach	89.1	74.2	62.7	87.1	73.9	62.0	86.6	80.8	62.4

Table 3: Test accuracies for single instructions (Inst), first-three instructions (3utts), and full interactions (5utts).

System	ALCHEMY			SCENE			TANGRAMS		
	Inst	3utts	5utts	Inst	3utts	5utts	Inst	3utts	5utts
SUPERVISED	92.0	83.3	71.4	85.3	72.7	60.6	86.1	81.9	58.3
POLICYGRADIENT	0.0	0.0	0.0	0.9	1.0	0.5	85.2	74.9	52.3
CONTEXTUALBANDIT	58.8	6.9	5.7	12.0	0.5	1.5	85.6	78.4	52.6
Our approach	**92.1**	**82.9**	**71.8**	**83.9**	**68.7**	56.1	88.5	82.4	60.3
– previous instructions	90.1	77.1	66.1	79.3	60.6	45.5	76.4	55.8	27.6
– current and initial state	25.7	4.5	3.3	17.5	0.0	0.0	45.4	15.1	3.5
– current state	89.8	78.0	62.9	83.0	**68.7**	54.0	87.6	78.4	60.8
– initial state	81.1	68.6	42.9	82.7	67.7	**57.1**	**88.6**	**82.9**	**63.3**
Our approach ($\mu \pm \sigma$)	91.5 $\pm$1.4	80.4 $\pm$2.6	69.5 $\pm$5.0	62.9 $\pm$17.7	37.8 $\pm$23.5	29.0 $\pm$21.1	88.2 $\pm$0.6	80.8 $\pm$2.8	59.2 $\pm$2.3

Table 4: Development results, including model ablations. We also report mean μ and standard deviation σ for all metrics for our approach across five experiments. We bold the best performing variations of our model.

Class	ALC	SCE	TAN
State reference	23	13	7
Multi-turn reference	12	5	13
Impossible multi-turn reference	2	5	13
Ambiguous or incorrect label	2	19	12

Table 5: Common error counts in the three domains.

also find a significant number of errors due to ambiguous or incorrect instructions. For example, the SCENE instruction *person in green appears on the right end* is ambiguous. In the annotated goal, it is interpreted as referring to a person already in the environment, who moves to the 10th position. However, it can also be interpreted as a new person in green appearing in the 10th position.

We also study performance with respect to multi-turn coreference by observing whether the model was able to identify the correct referent for each occurrence included in the analysis in Table 2. The models were able to correctly resolve 92.3%, 88.7%, and 76.0% of references in ALCHEMY, SCENE, and TANGRAMS respectively.

Finally, we include attention visualization for examples from the three domains in the Supplementary Material.

9 Discussion

We propose a model to reason about context-dependent instructional language that display strong dependencies both on the history of the interaction and the state of the world. Future modeling work may include using intermediate world states from previous turns in the interaction, which is required for some of the most complex references in the data. We propose to train our model using SESTRA, a learning algorithm that takes advantage of single-step reward observations to overcome learned biases in on-policy learning. Our learning approach requires additional reward observations in comparison to conventional reinforcement learning. However, it is particularly suitable to recovering from biases acquired early during learning, for example due to biased action spaces, which is likely to lead to incorrect blame assignment in neural network policies. When the domain and model are less susceptible to such biases, the benefit of the additional reward observations is less pronounced. One possible direction for future work is to use an estimator to predict rewards for all actions, rather than observing them.

Acknowledgements

This research was supported by the NSF (CRII-1656998), Schmidt Sciences, and cloud computing credits from Amazon. We thank John Langford and Dipendra Misra for helpful and insightful discussions with regards to our learning algorithm. We also thank the anonymous reviewers for their helpful comments.

References

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 2018. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Jacob Andreas and Dan Klein. 2015. Alignment-based compositional semantics for instruction following. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Yoav Artzi and Luke S. Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics*, 1:49–62.

Kavosh Asadi, Cameron Allen, Melrose Roderick, Abdel-rahman Mohamed, George Konidaris, and Michael L. Littman. 2017. Mean actor critic. *CoRR*, abs/1709.00503.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Jonathan Berant and Percy Liang. 2015. Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558.

Yonatan Bisk, Kevin Shih, Yejin Choi, and Daniel Marcu. 2018. Learning interpretable spatial operations in a rich 3D blocks world. In *Proceedings of the Thirty-Second Conference on Artificial Intelligence*.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016. Natural language communication with robots. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

S.R.K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the AFNLP*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé, and John Langford. 2015. Learning to search better than your teacher. In *Proceedings of the International Conference on Machine Learning*.

David Chen. 2012. Fast online lexicon learning for grounded language acquisition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Proceedings of the Workshop on Human Language Technology*.

Yann Dauphin, Harm de Vries, , and Yoshua Bengio. 2015. Equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified pragmatic models for generating and following instructions. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Proceedings of the DARPA speech and natural language workshop*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. 2018. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6:49–61.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Matthew MacMahon, Brian Stankiewics, and Benjamin Kuipers. 2006. Walk the talk: Connecting language, knowledge, action in route instructions. In *Proceedings of the National Conference on Artificial Intelligence*.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Association for the Advancement of Artificial Intelligence*.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Dipendra Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Hao Tan and Mohit Bansal. 2018. Source-target inference models for spatial instruction understanding. In *AAAI Conference on Artificial Intelligence*.

Tijmen Tieleman and Geoffrey Hinton. 2012. 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.

Gökhan Tür, Dilek Hakkani-Tür, and Larry Heck. 2010. What is left to be understood in ATIS? In *Proceedings of the Spoken Language Technology Workshop*.

Adam Vogel and Daniel Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Luke S. Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*.