Mapping Instructions and Visual Observations to Actions with Reinforcement Learning (Supplementary Material)

Dipendra Misra[†], John Langford[‡], and Yoav Artzi[†]

[†] Dept. of Computer Science and Cornell Tech, Cornell University, New York, NY 10044
{dkm, yoav}@cs.cornell.edu

[‡] Microsoft Research, New York, NY 10011
jcl@microsoft.com

A Reward Shaping Theorems

In Section 6, we introduce two reward shaping terms. We follow the safe-shaping theorems of Ng et al. (1999) and Wiewiora et al. (2003). The theorems outline potential-based terms that realize sufficient conditions for *safe* shaping. Applying safe terms guarantees the order of policies according to the original problem reward does not change. While the theory only applies when optimizing the total reward, we show empirically the effectiveness of the safe shaping terms in a contextual bandit setting. For convenience, we provide the definitions of potential-based shaping terms and the theorems introduced by Ng et al. (1999) and Wiewiora et al. (2003) using our notation. We refer the reader to the original papers for the full details and proofs.

The distance-based shaping term F_1 is defined based on the theorem of Ng et al. (1999):

Definition. *A shaping term $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is potential-based if there exists a function $\phi : \mathcal{S} \rightarrow \mathbb{R}$ such that, at time j, $F(s_j, a_j, s_{j+1}) = \gamma\phi(s_{j+1}) - \phi(s_j)$, $\forall s_j, s_{j+1} \in \mathcal{S}$ and $a_j \in \mathcal{A}$, where $\gamma \in [0, 1]$ is a future reward discounting factor. The function ϕ is the potential function of the shaping term F.*

Theorem. *Given a reward function $R(s_j, a_j)$, if the shaping term is potential-based, the shaped reward $R_F(s_j, a_j, s_{j+1}) = R(s_j, a_j) + F(s_j, a_j, s_{j+1})$ does not modify the total order of policies.*

In the definition of F_1, we set the discounting term γ to 1.0 and omit it.

The trajectory-based shaping term F_2 follows the shaping term introduced by Brys et al. (2015). To define it, we use the look-back advice shaping term of Wiewiora et al. (2003), who extended the potential-based term of Ng et al. (1999) for terms that consider the previous state and action:

Definition. *A shaping term $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is potential-based if there exists a function $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that, at time j, $F(s_{j-1}, a_{j-1}, s_j, a_j) = \gamma\phi(s_j, a_j) - \phi(s_{j-1}, a_{j-1})$, $\forall s_j, s_{j-1} \in \mathcal{S}$ and $a_j, a_{j-1} \in \mathcal{A}$, where $\gamma \in [0, 1]$ is a future reward discounting factor. The function ϕ is the potential function of the shaping term F.*

Theorem. *Given a reward function $R(s_j, a_j)$, if the shaping term is potential-based, the shaped reward $R_F(s_{j-1}, a_{j-1}, s_j, a_j) = R(s_j, a_j) + F(s_{j-1}, a_{j-1}, s_j, a_j)$ does not modify the total order of policies.*

In the definition of F_2 as well, we set the discounting term γ to 1.0 and omit it.

B Evaluation Systems

We implement multiple systems for evaluation.

STOP The agent performs the STOP action immediately at the beginning of execution.

RANDOM The agent samples actions uniformly until STOP is sampled or J actions were sampled, where J is the execution horizon.

SUPERVISED Given the training data with N instruction-state-execution triplets, we generate training data of instruction-state-action triplets and optimize the log-likelihood of the data. Formally, we optimize the objective:

$$J = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m^{(i)}} \log \pi(\tilde{s}_j^{(i)}, a_j^{(i)}) \ ,$$

where $m^{(i)}$ is the length of the execution $\bar{e}^{(i)}$, $\tilde{s}_j^{(i)}$ is the agent context at step j in sample i, and $a_j^{(i)}$ is the demonstration action of step j in demonstration execution $\bar{e}^{(i)}$. Agent contexts are generated with the annotated previous actions (i.e., to generate previous images and the previous action). We use minibatch gradient descent with ADAM updates (Kingma and Ba, 2014).

DQN We use deep Q-learning (Mnih et al., 2015) to train a Q-network. We use the architecture described in Section 4, except replacing the task specific part with a single 81-dimension layer. In contrast to our probabilistic model, we do not

decompose block and direction selection. We use the shaped reward function, including both F_1 and F_2. We use a replay memory of size 2,000 and an ϵ-greedy behavior policy to generate rollouts. We attenuate the value of ϵ from 1 to 0.1 in 100,000 steps and use prioritized sweeping for sampling. We also use a target network that is synchronized after every epoch.

REINFORCE We use the REINFORCE algorithm (Sutton et al., 1999) to train our agent. REINFORCE performs policy gradient learning with total reward accumulated over the roll-out as opposed to using immediate rewards as in our main approach. REINFORCE samples the total reward using monte-carlo sampling by performing a roll-out. We use the shaped reward function, including both F_1 and F_2 terms. Similar to our approach, we initialize with a SUPERVISED model and regularize the objective with the entropy of the policy. We do not use a reward baseline.

SUPERVISED with Oracle Planner We use a variant of our model assuming a perfect planner. The model predicts the block to move and its target position as a pair of coordinates. We modify the architecture in Section 4 to predict the block to move and its target position as a pair of coordinates. This model assumes that the sequence of actions is inferred from the predicted target position using an oracle planner. We train using supervised learning by maximizing the likelihood of the block being moved and minimizing the squared distance between the predicted target position and the annotated target position.

C Parameters and Initialization

C.1 Architecture Parameters

We use an RGB image of 120x120 pixels, and a convolutional neural network (CNN) with 4 layers. The first two layers apply 32 8×8 filters with a stride of 4, the third applies 32 4×4 filters with a stride of 2. The last layer performs an affine transformation to create a 200-dimension vector. We linearly scale all images to have zero mean and unit norm. We use a single layer RNN with 150-dimensional word embeddings and 250 LSTM units. The dimension of the action embedding ψ_a is 56, including 32 for embedding the block and 24 for embedding the directions. $\mathbf{W}^{(1)}$ is a 506×120 matrix and $\mathbf{b}^{(1)}$ is a 120-dimension vector. $\mathbf{W}^{(D)}$ is 120×20 for 20 blocks, and $\mathbf{W}^{(B)}$ is 120×5 for the four directions (north, south, east, west) and the `STOP` action. We consider $K = 4$ previous images, and use horizon length $J = 40$.

C.2 Initialization

Embedding matrices are initialized with a zero-mean unit-variance Gaussian distribution. All biases are initialized to $\mathbf{0}$. We use a zero-mean truncated normal distribution to initialize the CNN filters (0.005 variance) and CNN weights matrices (0.004 variance). All other weight matrices are initialized with a normal distribution (mean=0.0, standard deviation=0.01). The matrices used in the word embedding function ψ are initialized with a zero-mean normal distribution with standard deviation of 1.0. Action embedding matrices, which are used for ψ_a, are initialized with a zero-mean normal distribution with 0.001 standard deviation. We initialize policy gradient learning, including our approach, with parameters estimated using supervised learning for two epochs, except the direction parameters $\mathbf{W}^{(D)}$ and $\mathbf{b}^{(D)}$, which we learn from scratch. We found this initialization method to provide a good balance between strong initialization and not biasing the learning too much, which can result in limited exploration.

C.3 Learning Parameters

We use the distance error on a small validation set as stopping criteria. After each epoch, we save the model, and select the final model based on development set performance. While this method overfits the development set, we found it more reliable then using the small validation set alone. Our relatively modest performance degradation on the held-out set illustrates that our models generalize well. We set the reward and shaping penalties $\delta = \delta_f = 0.02$. The entropy regularization coefficient is $\lambda = 0.1$. The learning rate is $\mu = 0.001$ for supervised learning and $\mu = 0.00025$ for policy gradient. We clip the gradient at a norm of 5.0. All learning algorithms use a mini-batch of size 32 during training.

D Dataset Comparisons

We briefly review instruction following datasets in Table 1, including: Blocks (Bisk et al., 2016), SAIL (MacMahon et al., 2006; Chen and Mooney, 2011), Matuszek (Matuszek et al., 2012), and Misra (Misra et al., 2015). Overall, Blocks provides the largest training set and a relatively complex environment with well over 2.43^{18} possible

Name	# Samples	Vocabulary Size	Mean Instruction Length	# Actions	Mean Trajectory Length	Partially Observed
Blocks	16,767	1,426	15.27	81	15.4	No
SAIL	3,237	563	7.96	3	3.12	Yes
Matuszek	217	39	6.65	3	N/A	No
Misra	469	775	48.7	> 100	21.5	No

Table 1: Comparison of several related natural language instructions corpora.

states.[1] The most similar dataset is SAIL, which provides only partial observability of the environment (i.e., the agent observes what is around it only). However, SAIL is less complex on other dimensions related to the instructions, trajectories, and action space. In addition, while Blocks has a large number of possible states, SAIL includes only 400 states. The small number of states makes it difficult to learn vision models that generalize well. Misra (Misra et al., 2015) provides a parameterized action space (e.g., $\mathrm{grasp(cup)}$), which leads to a large number of potential actions. However, the corpus is relatively small.

E Common Questions

This is a list of potential questions following various decisions that we made. While we ablated and discussed all the crucial decisions in the paper, we decided to include this appendix to provide as much information as possible.

Is it possible to manually engineer a competitive reward function without shaping? Shaping is a principled approach to add information to a problem reward with relatively intuitive potential functions. Our experiments demonstrate its effectiveness. Investing engineering effort in designing a reward function specifically designed to the task is a potential alternative approach.

Are you using beam search? Why not? While using beam search can probably increase our performance, we chose to avoid it. We are motivated by robotic scenarios, where implementing beam search is a challenging task and often not possible. We distinguish between beam search and backtracking. Beam search is also incompatible with common assumptions of reinforcement learning, although it is often used during test with reinforcement learning systems.

Why are you using the mean of the LSTM hidden states instead of just the final state? We empirically tested both options. Using the mean worked better. This was also observed by Narasimhan et al. (2015). Understanding in which scenarios one technique is better than the other is an important question for future work.

Can you provide more details about initialization? Please see Appendix C.

Does the agent in the block world learn to move obstacles and other blocks? While the agent can move any block at any step, in practice, it rarely happens. The agent prefers to move blocks around obstacles rather than moving other blocks and moving them back into place afterwards. This behavior is learned from the data and shows even when we use only very limited amount of demonstrations. We hypothesize that in other tasks the agent is likely to learn that moving obstacles is advantageous, for example when demonstrations include moving obstacles.

Does the agent explicitly mark where it is in the instruction? We estimate that over 90% of the instructions describe the target position. Therefore, it is often not clear how much of the instruction was completed during the execution. The agent does not have an explicit mechanism to mark portions of the instruction that are complete. We briefly experimented with attention, but found that empirically it does not help in our domain. Designing an architecture to allows such considerations is an important direction for future work.

Does the agent know which blocks are present? Not all blocks are included in each task. The agent must infer which blocks are present from the image and instruction. The set of possible actions, which includes moving all possible blocks, does not change between tasks. If the agent chooses to move a block that is not present, the world state does not change.

Did you experiment with executing sequences of instruction? The Bisk et al. (2016) includes such instructions, right? The majority of existing corpora, including SAIL (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Mei et al., 2016), provide segmented sequences of instructions. Existing approaches take advantage of this segmentation during training. For example, Chen and Mooney (2011), Artzi and Zettlemoyer (2013), and Mei et al. (2016) all train on seg-

[1] We compute this loose lower bound on the number of states in the block world as $20! = 2.43^{18}$ (the number of block permutations). This is a very loose lower bound.

mented data and test on sequences of instructions by doing inference on one sentence at a time. We are also able to do this. Similar to these approaches, we will likely suffer from cascading errors. The multi-instruction paragraphs in the Bisk et al. (2016) data are an open problem and present new challenges beyond just instruction length. For example, they often merge multiple block placements in one instruction (e.g, *put the SRI, HP, and Dell blocks in a row*). Since the original corpus does not provide trajectories and our automatic generation procedure is not able to resolve which block to move first, we do not have demonstrations for this data. The instructions also present a significantly more complex task. This is an important direction for future work, which illustrates the complexity and potential of the domain.

Potential-based shaping was proven to be safe when maximizing the total expected reward. Does this apply for the contextual bandit setting, where you maximize the immediate reward? The safe shaping theorems (Appendix A) do not hold in our contextual bandit setting. We show empirically that shaping works in practice. However, how and if it changes the order of policies is an open question.

How long does it take to train? How many frames the agent observes? The agent observes about 2.5 million frames. It takes 16 hours using 50% capacity of an Nvidia Pascal Titan X GPU to train using our approach. DQN takes more than twice the time for the same number of epochs. Supervised learning takes about 9 hours to converge. We also trained DQN for around four days, but did not observe improvement.

Did you consider initializing DQN with supervised learning? Initializing DQN with the probabilistic supervised model is challenging. Since DQN is not probabilistic it is not clear what this initialization means. Smart initialization of DQN is an important problem for future work.

References

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics* 1:49–62. http://aclweb.org/anthology/Q13-1005.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. https://doi.org/10.18653/v1/N16-1089.

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. 2015. Reinforcement learning from demonstration through shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

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*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

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*.

Cynthia Matuszek, Evan Herbst, Luke S. Zettlemoyer, and Dieter Fox. 2012. Learning to parse natural language commands to a robot control system. In *Proceedings of the International Symposium on Experimental Robotics*.

Hongyuan Mei, Mohit Bansal, and R. Matthew Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. https://doi.org/10.18653/v1/N16-1086.

Kumar Dipendra Misra, Kejia Tao, Percy Liang, and Ashutosh Saxena. 2015. Environment-driven lexicon induction for high-level instructions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. https://doi.org/10.3115/v1/P15-1096.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540).

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/D15-1001.

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.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*.

Eric Wiewiora, Garrison W. Cottrell, and Charles Elkan. 2003. Principled methods for advising reinforcement learning agents. In *Proceedings of the International Conference on Machine Learning*.