Supplementary Material: Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning

Junhyuk Oh [1] Satinder Singh [1] Honglak Lee [1 2] Pushmeet Kohli [3]

A. Experimental Setting for Parameterized Tasks

The episode terminates after 50 steps for 'Independent' and 'Object-dependent' cases and 70 steps for 'Inter/Extrapolation' case. The agent receives a positive reward $(+1)$ when it successfully finishes the given task and receives a time penalty of -0.1 for each step. The details of each generalization scenario are described below.

Independent. The semantics of the tasks are consistent across all types of target objects. The training set of tasks is shown in Table 1. Examples of analogies used in this experiment are:

- [Visit, X] : [Visit, Y] :: [Transform, X] : [Transform, Y]
- [Visit, X] : [Visit, Y] $\neq$ [Transform, X] : [Pick up, Y]
- [Visit, X] $\neq$ [Visit, Y],

where X and Y can be any object type (X $\neq$ Y).

Object-dependent. We divided objects into two groups: Group A and Group B. Given 'Interact with' action, Group A should be picked up, whereas Group B should be transformed by the agent. The training set of tasks with groups for each object is shown in Table 2. Examples of analogies used in this experiment are:

- [Visit, Sheep] : [Visit, Cat] :: [Interact with, Sheep] : [Interact with, Cat]
- [Visit, Sheep] : [Visit, Greenbot] $\neq$ [Interact with, Sheep] : [Interact with, Greenbot]
- [Visit, Horse] : [Visit, Greenbot] :: [Interact with, Horse] : [Interact with, Greenbot]

Note that the second example implies that Sheep and Greenbot should be treated in a different way because they belong to different groups. Given such analogies, the agent can learn to interact with Cat as it interacts with Sheep and interact with Greenbot as it interacts with Horse.

Inter/Extrapolation. In this experiment, a task is defined by three parameters: action, object, and number. The agent should repeat the same subtask for a given number of times. The agent is trained on all configurations of actions and target objects. However, only a subset of numbers is used during training. In order to interpolate and extrapolate, we define analogies based on simple arithmetic such as:

- [Pick up, X, 1] : [Pick up, X, 6] :: [Pick up, X, 2] : [Pick up, X, 7]
- [Pick up, X, 3] : [Pick up, X, 6] :: [Transform, Y, 4] : [Transform, Y, 7]
- [Pick up, X, 1] : [Pick up, X, 3] $\neq$ [Transform, Y, 2] : [Transform, Y, 3]

[1]University of Michigan [2]Google Brain [3]Microsoft Research. Correspondence to: Junhyuk Oh <junhyuk@umich.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

	Visit	Pick up	Transform
Sheep	✓		✓
Horse	✓		✓
Pig	✓	✓	
Box	✓	✓	
Cat		✓	✓
Greenbot		✓	✓

Table 1: Training set of tasks for the ‘Independent’ case.

	Visit	Pick up	Transform	Interact with
Sheep (A)	✓	✓	✓	✓
Horse (B)	✓	✓	✓	✓
Pig (A)	✓	✓	✓	✓
Box (B)	✓	✓	✓	✓
Cat (A)	✓	✓	✓	
Greenbot (B)	✓	✓	✓	

Table 2: Training set of tasks for the ‘Object-dependent’ case.

B. Experimental Setting for Instruction Execution

The episode terminates after 80 steps during training of the meta controller. A box appears with probability of 0.03 and disappears after 20 steps. During evaluation on longer instructions, the episode terminates after 600 steps. We constructed a training set of instructions and an unseen set of instructions as described in Table 3. We generated different sequences of instructions for training and evaluation by sampling instructions from such sets of instructions.

	Visit	Pick up	Pick up 2	Pick up 3	Transform	Transform 2	Transform 3
Sheep	✓				✓	✓	✓
Horse	✓				✓	✓	✓
Pig	✓	✓	✓	✓			
Cat		✓	✓	✓	✓	✓	✓
Greenbot		✓	✓	✓	✓	✓	✓

Table 3: Training set of instructions.

C. Experiment on 2D Grid-World

Environment. To see how our approach can be generally applied to different domains, we developed a 2D grid-world based on MazeBase (Sukhbaatar et al., 2015) where the agent can interact with many objects, as illustrated in Figure 1. Unlike the original MazeBase, an observation is represented as a binary 3D tensor: $\mathbf{x}_t \in \mathbb{R}^{18\times10\times10}$ where 18 is the number of object types and 10×10 is the size of the grid world. Each channel is a binary mask indicating the presence of each object type. There are agent, blocks, water, and 15 types of objects with which the agent can interact, and all of them are randomly placed for each episode.

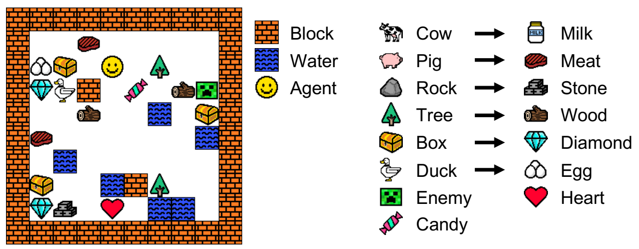


Figure 1: Example of 2D grid-world with object specification. The arrows represent the outcome of object transformation. Objects without arrows disappear when transformed. The agent is not allowed to go through blocks and gets a penalty for going through water.

The agent has 13 primitive actions: *No-operation*, *Move* (North/South/West/East, referred to as “NSWE”), *Pick up* (NSWE), and *Transform* (NSWE). *Move* actions move the agent by one cell in the specified direction. *Pick up* actions remove the adjacent object in the corresponding relative position, and *Transform* actions either remove it or transform it to another object depending on the object type, as shown in Figure 1.

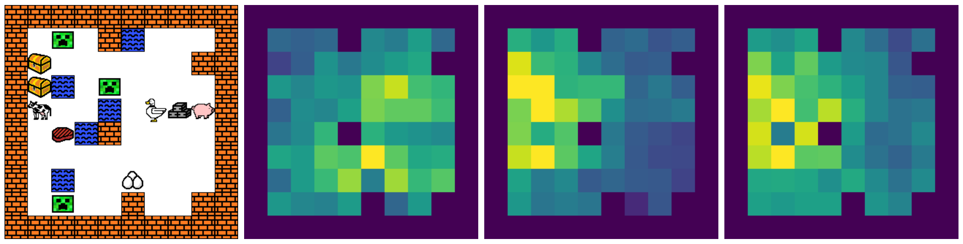
The agent receives a time penalty (-0.1) for each time-step. Water cells act as obstacles which give -0.3 when the agent visits them. The agent receives $+1$ reward when it finishes all instructions in the correct order. Throughout the episode, an enemy randomly appears, moves, and disappears after 10 steps. Transforming an enemy gives $+0.9$ reward.

Evaluation on Parameterized Tasks. As in the main experiment, we evaluated the parameterized skill on seen and unseen parameterized tasks separately using the same generalization scenarios. As shown in Table 4, the parameterized skill with analogy-making can successfully generalize to unseen tasks both in independent and object-dependent scenarios.

Scenario	Analogy	Train	Unseen
Independent	×	**0.56** (99.9%)	-1.88 (49.6%)
	✓	**0.56** (99.9%)	**0.55** (99.6%)
Object-dependent	×	**0.55** (99.9%)	-3.23 (43.2%)
	✓	**0.55** (99.9%)	**0.55** (99.5%)

Table 4: Performance on parameterized tasks. Each entry shows 'Average reward (Success rate)'. We assume an episode is successful only if the agent successfully finishes the task and its termination predictions are correct throughout the whole episode.

We visualized the value function learned by the critic network of the parameterized skill in Figure 2. As expected from its generalization performance, our parameterized skill trained with analogy-making objective learned high values around the target objects given unseen tasks.



(a) Observation (b) Visit egg (c) Pick up cow (d) Transform meat

Figure 2: Value function visualization given unseen tasks. (b-d) visualizes learned values for each position of the agent in a grid world (a). The agent estimates high values around the target object in the world.

Evaluation on Instruction Execution. There are 5 types of instructions: Visit X, Pick up X, Transform X, Pick up all X, and Transform all X, where 'X' is the target object type. While the first three instructions require the agent to perform the corresponding subtask, the last two instructions require the agent to repeat the same subtask until the target objects completely disappear from the world.

The overall result is consistent with the result on 3D environment as shown in Figure 3 and Table 5. The flat baseline learned a sub-optimal policy that transforms or picks up all target objects in the world even if there is no 'all' adverb in the instructions. This sub-optimal policy unnecessarily removes objects that can be potentially target objects in the future instructions. This is why the performance of the flat baseline drastically decreases as the number of instructions increases in Figure 3. Our architecture with learned time-scale ('Hierarchical-Dynamic') performs much better than 'Hierarchical-Short' baseline which updates the subtask at every time-step. This also suggests that learning to operate in a large-time scale is crucial for dealing with delayed reward. We also observed that our agent learned to deal with enemies whenever they appear, and thus it outperforms the 'Shortest Path' method which is near-optimal in executing instructions while ignoring enemies.

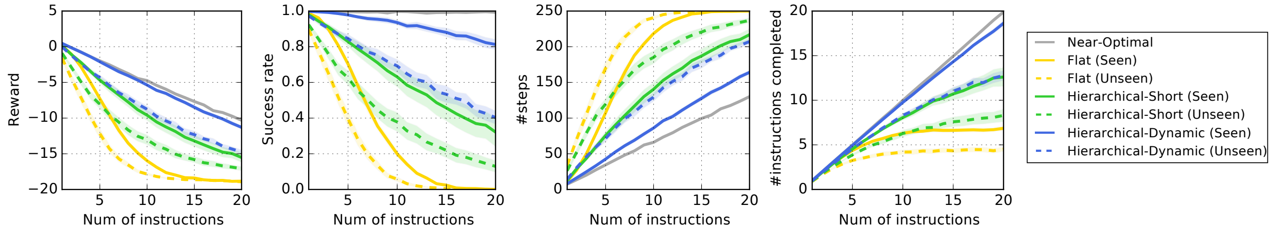


Figure 3: Performance per number of instructions.

	Train	Test (Seen)	Test (Unseen)
#Instructions	4	20	20
Shortest Path	-1.62 (99.7%)	-11.94 (99.4%)	
Near-Optimal	-1.34 (99.5%)	-10.30 (99.3%)	
Flat	-2.38 (76.0%)	-18.83 (0.1%)	-18.92 (0.0%)
Hierarchical-Short	-1.74 (81.0%)	-15.89 (28.0%)	-17.23 (11.3%)
Hierarchical-Dynamic	**-1.26** (95.5%)	**-11.30** (81.3%)	**-14.75** (40.3%)

Table 5: Performance of meta controller. Each entry in the table represents reward with success rate in parentheses averaged over 10-best runs among 20 independent runs. 'Shortest Path' is a hand-designed policy which executes instructions optimally based on the shortest path but ignores enemies. 'Near-Optimal' is a near-optimal policy that executes instructions based the shortest path and transforms enemies when they are close to the agent.

D. Details of Learning Objectives

D.1. Parameterized Skill

The parameterized skill is first trained through *policy distillation* (Rusu et al., 2016; Parisotto et al., 2016) and fine-tuned using actor-critic method (Konda and Tsitsiklis, 1999) with generalized advantage estimation (GAE) (Schulman et al., 2016). The parameterized skill is also trained to predict whether the current state is terminal or not through binary classification objective.

The idea of policy distillation is to first train separate teacher policies ($\pi_T^g(a|s)$) for each task (g) through reinforcement learning and train a multi-task policy ($\pi_\phi^g(a|s)$) to mimic teachers' behavior by minimizing KL divergence between them as follows:

$$\nabla_\phi \mathcal{L}_{RL} = \mathbb{E}_{g\sim\mathcal{U}}\left[\mathbb{E}_{s\sim\pi_\phi^g}\left[\nabla_\phi D_{KL}\left(\pi_T^g || \pi_\phi^g\right) + \alpha \nabla_\phi \mathcal{L}_{term}\right]\right], \tag{1}$$

where $D_{KL}\left(\pi_T^g||\pi_\phi^g\right) = \sum_a \pi_T^g(a|s)\log\frac{\pi_T^g(a|s)}{\pi_\phi^g(a|s)}$ and $\mathcal{U} \subset \mathcal{G}$ is the training set of tasks. $\mathcal{L}_{term} = -\log\beta_\phi\left(s_t, g\right) = -\log P_\phi\left(s_t \in \mathcal{T}_g\right)$ is the cross-entropy loss for termination prediction. Intuitively, we sample a mini-batch of tasks (g), use the parameterized skill to generate episodes, and train it to predict teachers' actions. This method has been shown to be efficient for multi-task learning.

After policy distillation, the parameterized skill is fine-tuned through actor-critic with generalized advantage estimation (GAE) (Schulman et al., 2016) as follows:

$$\nabla_\phi \mathcal{L}_{RL} = \mathbb{E}_{g\sim\mathcal{U}}\left[\mathbb{E}_{s\sim\pi_\phi^g}\left[-\nabla_\phi \log\pi_\phi\left(a_t|s_t,g\right)\hat{A}_t^{(\gamma,\lambda)} + \alpha\nabla_\phi\mathcal{L}_{term}\right]\right], \tag{2}$$

where $\hat{A}_t^{(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V$ and $\delta_t^V = r_t + \gamma V^\pi(s_{t+1};\phi') - V^\pi(s_t;\phi')$. ϕ' is optimized to minimize $\mathbb{E}\left[\left(R_t - V^\pi(s_t;\phi')\right)^2\right]$. $\gamma,\lambda \in [0,1]$ are a discount factor and a weight for balancing between bias and variance of the advantage estimation.

The final update rule for the parameterized skill is:

$$\Delta\phi \propto -\left(\nabla_\phi\mathcal{L}_{RL} + \xi\nabla_\phi\mathcal{L}_{AM}\right), \tag{3}$$

where $\mathcal{L}_{AM} = \mathcal{L}_{sim} + \rho_1\mathcal{L}_{dis} + \rho_2\mathcal{L}_{diff}$ is the analogy-making regularizer defined as the weighted sum of three objectives described in the main text. ρ_1, ρ_2, ξ are hyperparameters for each objective.

D.2. Meta Controller

Actor-critic method with GAE is used to update the parameter of the meta controller as follows:

$$\nabla_\theta\mathcal{L}_{RL} = -\begin{cases} \mathbb{E}\left[c_t\left(\sum_i \nabla_\theta\log\pi_\theta\left(g_t^{(i)}|\mathbf{h}_t,\mathbf{r}_t\right) + \nabla_\theta\log P\left(\mathbf{l}_t|\mathbf{h}_t\right)\right)\hat{A}_t^{(\gamma,\lambda)}\right. & \\ \quad\left. + \nabla_\theta\log P\left(c_t|\mathbf{s}_t,\mathbf{h}_{t-1}\right)\hat{A}_t^{(\gamma,\lambda)} + \eta\nabla_\theta\left\|\sigma\left(\varphi^{update}\left(\mathbf{s}_t,\mathbf{h}_{t-1}\right)\right)\right\|_1\right] & \text{(Hard)} \\ \mathbb{E}\left[\sum_i\nabla_\theta\log\left[c_t\pi_\theta\left(g_t^{(i)}|\tilde{\mathbf{h}}_t,\tilde{\mathbf{r}}_t\right) + (1-c_t)g_{t-1}^{(i)}\right]\hat{A}_t^{(\gamma,\lambda)}\right] & \text{(Soft)}, \end{cases} \tag{4}$$

where $c_t \sim P\left(c_t|\mathbf{s}_t, \mathbf{h}_{t-1}\right) \propto \sigma\left(\varphi^{update}\left(\mathbf{s}_t, \mathbf{h}_{t-1}\right)\right)$, and $P\left(\mathbf{l}_t|\mathbf{h}_t\right) \propto \text{Softmax}\left(\varphi^{shift}(\mathbf{h}_t)\right)$. We applied L1-penalty to the probability of update to penalize too frequent updates, and η is a weight for the update penalty.

The final update rule for the meta controller is:

$$\Delta\theta \propto -\left(\nabla_\theta \mathcal{L}_{RL} + \xi \nabla_\theta \mathcal{L}_{AM}\right), \tag{5}$$

where $\mathcal{L}_{AM}$ is the analogy-making objective.

E. Architectures and Hyperparameters

Background: Multiplicative Interaction For combining condition variables into a neural network (e.g., combining task embedding into the convolutional network in the parameterized skill), we used a form of multiplicative interaction instead of concatenating such variables as suggested by (Memisevic and Hinton, 2010; Oh et al., 2015). This is also related to *parameter prediction* approaches where the parameters of the neural network is produced by condition variables (e.g., exempler, class embedding). This approach has been shown to be effective for achieving zero-shot and one-shot generalization in image classification problems (Lei Ba et al., 2015; Bertinetto et al., 2016). More formally, given an input ($\mathbf{x}$), the output ($\mathbf{y}$) of a convolution and a fully-connected layer with parameters predicted by a condition variable ($\mathbf{g}$) can be written as:

$$\text{Convolution: } \mathbf{y} = \varphi\left(\mathbf{g}\right) * \mathbf{x} + \mathbf{b}$$
$$\text{Fully-connected: } \mathbf{y} = \mathbf{W}'\text{diag}\left(\varphi\left(\mathbf{g}\right)\right)\mathbf{W}\mathbf{x} + \mathbf{b},$$

where φ is the embedding of the condition variable learned by a multi-layer perceptron (MLP). Note that we use matrix factorization (similar to (Memisevic and Hinton, 2010)) to reduce the number of parameters for the fully-connected layer. Intuitively, the condition variable is converted to the weight of the convolution or fully-connected layer through multiplicative interactions. We used this approach both in the parameterized skill and the meta controller. The details are described below.

Parameterized skill. The teacher architecture used for policy distillation is Conv1(32x8x8-4)-Conv2(64x5x5-2)-LSTM(64).[1] The network has two fully-connected output layers for actions and value (baseline) respectively. The parameterized skill architecture consists of Conv1(16x8x8-4)-Conv2(32x1x1-1)-Conv3(32x1x1-1)-Conv4(32x5x5-2)-LSTM(64). The parameterized skill takes two task parameters ($\mathbf{g} = \left[g^{(1)}, g^{(2)}\right]$) as additional input and computes $\varphi(\mathbf{g}) = \text{ReLU}(\mathbf{W}^{(1)}g^{(1)} \odot \mathbf{W}^{(2)}g^{(2)})$ to compute the subtask embedding. It is further linearly transformed into the weights of Conv3 and the (factorized) weight of LSTM through multiplicative interaction as described above. Finally, the network has three fully-connected output layers for actions, termination probability, and baseline, respectively.

We used RMSProp optimizer with the smoothing parameter of 0.97 and epsilon of $1e-6$. When training the teacher policy through actor-critic, we used a learning rate of $2.5e-4$. For training the parameterized skill, we used a learning rate of $2.5e-4$ and $1e-4$ for policy distillation and actor-critic fine-tuning respectively. We used $\tau_{dis} = \tau_{diff} = 3, \alpha = 0.1$ for analogy-making regularization and the termination prediction objective respectively. $\gamma = 0.99$ and $\lambda = 0.96$ are used as a discount factor and a balancing weight for GAE. 16 threads with batch size of 8 are used to run 16×8 episodes in parallel, and the parameter is updated after each run (1 iteration = 16×8 episodes). For better exploration, we applied entropy regularization with a weight of 0.1 and linearly decreased it to zero for the first 7500 iterations. The total number of iterations was 15,000 for both policy distillation and actor-critic fine-tuning.

Meta Controller. The meta controller consists of Conv1(16x8x8-4)-Conv2(32x1x1-1)-Conv3(32x1x1-1)-Pool(5)-LSTM(256). The embedding of previously selected subtask ($\varphi(\mathbf{g}_{t-1})$), the previously retrieved instruction ($\mathbf{r}_{t-1}$), and the subtask termination (b_t) are concatenated and given as input for one-layer MLP to compute a 256-dimensional joint embedding. This is further linearly transformed into the weights of Conv3 and LSTM through multiplicative interaction. The output of FC1 is used as the context vector ($\mathbf{h}_t$). We used the bag-of-words (BoW) representation as a sentence embedding which computes the sum of all word embeddings in a sentence: $\varphi^w\left(\mathbf{m}_i\right) = \sum_{j=1}^{|\mathbf{m}_i|} \mathbf{W}^m w_j$ where $\mathbf{W}^m$ is the word embedding matrix, each of which is 256-dimensional. An MLP with one hidden layer with 256 units is for φ^{shift}, a fully-connected layer is used for φ^{update}. φ^{goal} is an MLP with one hidden layer with 256 units that takes the concatenation of $\mathbf{r}_t$

[1]For convolution layers, NxKxK-S represents N kernels with size of KxK and stride of S. The number in LSTM represents the number of hidden units.

and $\mathbf{h}_t$ as an input and computes the probabilities over subtask parameters as the outputs. The baseline network is a linear regression from the concatenation of the memory pointer $\mathbf{p}_t$, a binary mask indicating the presence of given instruction, and the final hidden layer (256 hidden units in φ^{goal}).

We used the same hyperparameters used in the parameterized skill except that the batch size was 32 (1 iteration = 16×32 episodes). We trained the soft-architecture (with soft-update) with a learning rate of $2.5e-4$ using curriculum learning for 15,000 iterations and a weight of 0.015 for entropy regularization, and fine-tuned it with a learning rate of $1e-4$ without curriculum learning for 5,000 iterations. Finally, we initialized the hard-architecture (with hard-update) to the soft-architecture and fine-tuned it using a learning rate of $1e-4$ for 5,000 iterations. $\eta = 0.001$ is used to penalize frequent update decision in Eq (4).

Flat Controller. The flat controller architecture consists of the same layers used in the meta controller with the following differences. The previously retrieved instruction ($\mathbf{r}_{t-1}$) is transformed through an MLP with two hidden layers to compute the weight of Conv3 and LSTM. The output is probabilities over primitive actions.

Curriculum Learning. For training all architectures, we randomly sampled the size of the world from $\{5, 6, 7, 8\}$, the density of walls are sampled from $[0, 0.1]$, and the density of objects are sampled from $[0.1, 0.8]$ during training of parameterized skill and $[0, 0.15]$ during training of the meta controller. We sampled the number of instructions from $\{1, 2, 3, 4\}$ for training the meta controller. The sampling range was determined based on the success rate of the agent.

References

L. Bertinetto, J. F. Henriques, J. Valmadre, P. H. Torr, and A. Vedaldi. Learning feed-forward one-shot learners. *arXiv preprint arXiv:1606.05233*, 2016.

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *NIPS*, 1999.

J. Lei Ba, K. Swersky, S. Fidler, et al. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *ICCV*, 2015.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010.

J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, 2015.

E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *ICLR*, 2016.

A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. In *ICLR*, 2016.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

S. Sukhbaatar, A. Szlam, G. Synnaeve, S. Chintala, and R. Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.