Preparation of Papers for IEEE Sponsored Conferences & Symposia*

Samer Sabri[1] and Chao Yu[2]

I. INTRODUCTION

In Zero-Shot Task Generalization with Multi-Task Deep RL, Oh et al built a multi-task RL agent which was able to execute a set of instructions in a specific order. In our project, we replicated the author's hierarchical model to solve a 2D environment they describe in their paper. This was a significant undertaking, since the authors did not publish any code on this, which means we had to rebuild the networks, the learning, and the environment. We also attempted to train the network end-to-end using curriculum learning; this had mixed to negative results. Following Oh et Al's process, we started by building task-specific teacher networks through A3C. We then used policy distillation to build a multi-task agent that initially learns from the teacher policies. This multi-task agent, or parameterized skill architecture uses an analogy-based loss to generalize to unseen tasks in a zero-shot fashion.This enables our parameterized task to perform unseen tasks with a high completion rate (¿90%). Finally, we implemented a meta-controller to teach the model to perform several tasks in order, but we were not able to train it successfully even when using curriculum training.

II. BACKGROUND

A. Zero-shot learning

As shown in [5], in a supervised setting, zero-shot learning refers to a model's ability to correctly classify objects from unseen classes. In a Reinforcement Learning setting, this can be interpreted as an agent's ability to correctly interpret (i.e., choose the optimal actions for) an unseen instruction or ordered set of instructions. Zero-shot learning requires going beyond classification to understanding and exploiting the relationship between classes. To achieve this, Oh et Al rely on two elements: a hierarchical architecture to concentrate the p̈lanningïn the upper l̈evelöf the network (i.e., meta-controller), and an analogy-making approach to generalizing to unseen tasks.

B. Hierarchical reinforcement learning

Hierarchical RL is based on the intuition that humans and possibly animals are able to plan because they can lump small tasks together into bigger wholes. For example, when you think about making a sandwich, you don't need to think about finding the bread, opening the container, etc. In hierarchical RL, a lower-level agent focuses on the small low-level tasks and a high-level agent or master policy choose higher-level, bigger tasks that are fed to the low-level agent. Bringing deep networks to hierarchical RL has been very successful. For instance, in [6], the authors show that a hierarchical Deep Q-Learning agent demonstrated improved exploration and thus better results in a challenging Atari game.

C. Analogy making

Oh et al follow [7]'s method for analogy making. The idea is to enforce the parallelogram rule onto the embedding (i.e., linear layer output) $\phi(x)$ of the task x. If a:b::c:d (read "a is to be what c is to d") then $||\phi(a) - \phi(b) - (\phi(c) - \phi(d))$ should be close to 0. The analogy is enforced by adding simple edge constraints (i.e., the embeddings for any two tasks in the training set should not be too close to each other, and the embeddings of non-anology-following quadruplets of tasks should not be close to 0) and adding an ä̈nalogy-making lossẗo the global loss.

D. Policy distillation

Policy distillation is a method used to extract the policy of one or multiple reinforcement learning agent. As shown in [2], policy distillation can be used to train a new network that performs at the expert level on multiple tasks while being dramatically smaller and more efficient than its teacher networks.

E. Curriculum training

As Bengio et al explain in [9], curriculum training is based on the idea that humans and animals often need to follow a specific process analogous to building scaffolding. We attempted to train a network end-to-end instead of in a hierarchical fashion where the lower levels are trained, fixed, and then used by the higher levels during training.

F. A3C

A3C, or Asynchronous Advantage Actor-Critic , was first presented in [8]. The asynchronous aspect takes advantage of distributed architectures to train multiple networks in parallel while keeping their weights synchronized through a set of shared weights. It uses the Generalized Advantage Estimate (or GAE, see [12]).

III. METHODS

A. Environment

We built an n by n 2-D grid which is shown in Figure 1. The environment features an agent, 4 possible objects (greenball, blueball, tv, and heart), and 13 possible action ("None",

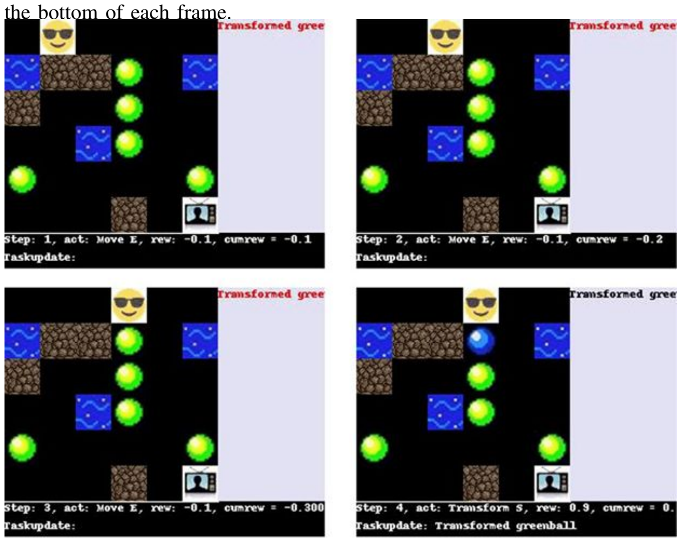
*This work was not supported by any organization

[1]H. Kwakernaak is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands `h.kwakernaak at papercept.net`

[2]P. Misra is with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA `p.misra at ieee.org`

as well as each of "Visit," "Pick up," and "Transform" in each of 4 directions NSWE). Transform can only applied on greenball and tv, which respectively turn into blueball and heart. There are also concrete blocks which are impossible to pass through and water blocks decrease reward by -.3 when the agent passes through them. Reward decreases by .1 at every timestep, and it increases by 1 when all the instructions are executed.

The environment is represented as a $grid_n \times grid_n \times num_objects$ binary tensor signaling the presence of the absence of an object in a location. The environment is generated according to a random procedure.



```
Step: 1, act: Move E, rew: -0.1, cumrew = -0.1
Taskupdate:
```

```
Step: 2, act: Move E, rew: -0.1, cumrew = -0.2
Taskupdate:
```

```
Step: 3, act: Move E, rew: -0.1, cumrew = -0.300
Taskupdate:
```

```
Step: 4, act: Transform S, rew: 0.9, cumrew = 0.
Taskupdate: Transformed greenball
```

Fig. 1. Frames from a four step episode. The task is described on the right. The agent's actions, rewards, and cumulative rewards are shown at the bottom of each frame.

B. Overall architecture

As previously explained, we use a hierarchical architecture and train the lower levels first. The lowest level of our process are the teacher networks, which each learn a specific task. The parameterized task learns from them through policy distillation, then directly from the environment to become a strong multi-policy network. Finally, this layer is fixed and appended onto a meta-controller network that learns to choose the next task through learning to read instructions from memory. We kept instructions represented as task one-hot vectors to simplify our work.

C. Teacher Networks: one task at a time

The teachers are trained through A3C, using 8 threads with 1 episode per iteration. The value loss is given by:

$$ValueLoss = \sum_t (R_t - V(s_t))$$

where t is the timesteps, R_t is the total reward from timestep t onward in a given episode, and V is the value network.

The Policy Loss uses the advantage $A(S)$:

$$PolicyLoss = -log(\pi(s)) * A(s)$$

where π is the policy network.

The network architecture is as follows:

- 2 convolutional layers with kernel sizes 2 and 1, number of channels 256 and 128 respectively.
- 2 linear layers with 1600 then 800 hidden units.
- 2 separate linear layers from the output of the preceding layers to the actor (13 output units) and the critic (1 output unit).

Even though the task is simple, we found that smaller networks had trouble generalizing to many tasks; they only did well when the maze was fixed or changed little throughout training.

D. The parameterized skill: one multitask policy

The network architecture is initially trained through policy distillation, where the loss is given in [1] by:

$$E_{g\sim U}[E_{s\sim\pi^g_\phi}[\bigtriangledown_\phi D_{KL}(\pi^g_T||\pi^g_\phi) + \alpha \bigtriangledown_\phi (-log\beta_\phi(s_t, g))]]$$

where D_{KL} is the KL divergence (over all actions) between the teacher policy π_T and the distilled multi-task policy π_ϕ. The term after the second $\bigtriangledown$ is the termination loss, simply a cross-entropy loss wrt the networkśbinary termination prediction.

In [2], the authors use a temperature $T = .01$ to sharpen the teacher networks. We found that this just introduce false precision and reduces the multitask policy's performance.

The parameterized skill undergoes fine-tuning using the loss from A3C in addition to an analogy loss and the termination loss.

The network structure was similar to the teacher network but it also incorporates the task, which is represented as a 7-D one-hot vector (3 for verbs + 4 for objects):

- As task embedding from the one-hot-vector to a 72-dimensional embedding, using:
  - one linear layer from the 3 verb inputs to 72 dimensions
  - on linear layer from the 4 action inputs to 72 dimensions
  - component-wise multiplication of those layers' outputs
- A convolution of this embedding with the observation
- Two convolutional layers with kernel sizes 2 and 1, and 256 and 128 features respectively
- Two linear layers with 1600 and 800 hidden units
- Three linear layers for value, action, and pterm (sigmoid of one unit).

E. The meta controller: choosing which task to execute

The meta-controllers learns a mapping from:

- Memory read
- Current observation
- The parameterized skill's termination signal
- Its own internal variables

to:

- a binary value c_t which determines whether the output task changes

- a new task g_t which is fed to the parameterized skill if c_t is 1
- memory read updates

For this phase, we implemented Algorithm 1 shown in [1].

F. End to end training

For this part of the training, we attempted to train the whole architecture end-to-end ob 1-task instruction sets first. We also tried to provide easy instructions (i.e., V̈isit)̈ first. We enforced the analogy loss internally.

IV. Results

A. Teacher Policy

We tested our models with two different sets of tasks. Each task was defined as easy or hard based on the Manhattan distance between the agent and the closest target at the starting point. If the Manhattan distance between them was lower than 3, the task was easy. If the Manhattan distance was greater than or equal to 3, then the task was hard. The results are shown below.

	$SuccessRate$	$AverageSteps$
$Visitedgreenball$	0.99	5.65
$Visitedblueball$	0.99	5.58
$Visitedtv$	1	5.19
$Visitedheart$	0.96	6.48
$Pickedupgreenball$	0.92	15.53
$Pickedupblueball$	0.89	16.48
$Pickeduptv$	0.9	14.33
$Pickedupheart$	0.94	11.88
$Transformedgreenball$	0.92	13.35
$Transformedtv$	0.88	12.85

TABLE I
TEACHER MODELS FOR EASY TASK

	$SuccessRate$	$AverageSteps$
$Visitedgreenball$	0.93	11.81
$Visitedblueball$	0.93	13.14
$Visitedtv$	0.93	11.54
$Visitedheart$	0.94	16.22
$Pickedupgreenball$	0.85	23.81
$Pickedupblueball$	0.83	24.05
$Pickeduptv$	0.83	23.02
$Pickedupheart$	0.83	22.79
$Transformedgreenball$	0.86	22.56
$Transformedtv$	0.86	23.18

TABLE II
TEACHER MODELS FOR HARD TASK

From tables 1 and 2, while comparing the average reward, we can see that the "Visited" task is easier for the agent to learn than the "Picked up" and "Transformed" tasks. We explain this by the fact that a v̈isitedẗeacher can focus on the 4 movement actions and ignore the 9 other actions.

B. Parameterized skill

The parameterized skill was trained on every task in the table above except V̈isited TVänd P̈icked up blueball.T̈he output of policy distillation performed decently but could not do better than a 0.75 success rate. The fine-tuned network performed similarly whether it was initialized from scratch or from a distilled network. It was not as stable as the teachers, but it was able to reach Sucess rates around 0.85-0.9 on seen tasks and 0.8-0.95 on unseen tasks on certain iterations.

Unfortunately, we failed at training the termination signal. We found that the α parameter from section III D was very sensitive and hard to set, and the network inevitably started marking all states as terminal or all of them as nonterminal.

C. Meta-Controller

Due to the issue with the termination signal, we were not able to train the meta-controller.

D. E2E training

We were not able to train the network end-to-end, in part because of our inability to set α and possibly due to insufficient training time.

V. DISCUSSION

Unfortunately, due to parameter-tuning issues, we feel our idea was not given a true chance, and it is difficult to assess whether this kind of multi-task zero-shot RL can be trained end-to-end. We are hoping to clean up our code and continue trying to make it work.

We were most pleased with the analogy loss: the network had this loss very close to 0 very early on in training. This shows that the constraint is easy to enforce and can be used in many settings even in supervised and unsupervised learning. Our parameterized skill generalizes well to unseen tasks without much parameter tuning.

We found that during policy distillation and fine-tuning, different tasks tended to c̈annibalizeëach other: the network either did better on v̈isitedäctions or on the others, and it seemed to only hit the s̈weet spotv̈ery rarely. It would be interesting to find a way to incorporate this need to balance different tasks through in the loss. Often, distilled networks solve this problem by using different output layers for different tasks, but this is not possible in this zero-shot generalization setting.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

References

[1] Oh, J., Singh, S., Lee, H., & Kohli, P. (2017). Zero-shot task generalization with multi-task deep reinforcement learning. arXiv preprint arXiv:1706.05064.

[2] 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.

[3] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In ICRA, 2017.

[4] W. Zaremba and I. Sutskever. Reinforcement learning neu- ral turing machines. arXiv preprint arXiv:1505.00521, 2015.

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

[6] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. arXiv preprint arXiv:1604.06057, 2016.

[7] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee. Deep visual analogy-making. In NIPS, 2015.

[8] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning (pp. 1928-1937).

[9] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009, June). Curriculum learning. In Proceedings of the 26th annual international conference on machine learning (pp. 41-48). ACM.

[10] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. ”Playing atari with deep reinforcement learning.” arXiv preprint arXiv:1312.5602 (2013).

[11] Oh, Junhyuk, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. ”Control of memory, active perception, and action in minecraft.” arXiv preprint arXiv:1605.09128 (2016).

[12] Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.