Scaling Neural Programmer-Interpreter For Real-Life Tasks*

Himadri Mishra[1] and K.K. Shukla[1]

***Abstract*— We demonstrate the application of Neural Programmer-Interpreters(NPI) to real-life tasks and how it can be scaled to unseen problems as a generic framework. NPI is class of models that learn to represent hierarchical tasks as programs and execute them. Till date, NPI has only been demonstrated on simple toy tasks with little or no resemblance to real-life tasks such as addition, sorting, canonicalizing 3D models, etc. Unlike the established belief, with concrete representations, NPI can easily generalize to a variable number of entities as well as unseen tasks. NPI has great potential in learning hierarchical tasks, which most of the real-life tasks are. Thus, leveraging this fact, we demonstrate the potential of NPI on Block Stacking task on fetch robot and how this can be extended to other problems. NPI shows strong generalization with initial configuration and variation in final goal and gives promising results in an unseen number of blocks. We hope to experiment with harder tasks and build a generic framework around NPI, which can help us incorporate robots into our daily life.**

I. INTRODUCTION

Learning how to perform real-life tasks has been a challenge from the emergence of the Computer era. By "real-life tasks," we mean the class of problems which are hard to be learned by traditional Machine Learning algorithms such as learning to solve a class of mathematical problem, driving a car, intelligent care-giving robot or learning any algorithm. Kentridge tackled this problem in the early 90s and proposed that the problem can be solved by making two part in the system; one which learns a representation of the problem and other which refines those representation [1]. He pointed out the need for a teacher for any system to work. After that, there had been several supervised (imitation learning) and reinforcement learning based methods to learn how to perform tasks, but they all lacked on the same grounds of generalization beyond the training examples/explored space (i.e., on unseen cases). Researchers have argued from time to time that there is a need for representing task into a hierarchy and then applying the learning algorithms. Salakhutdinov et al. emphasize on learning hierarchical features and using them to improve classification and similar tasks [2]. There have been several other attempts in hierarchical learning from demonstrations via reinforcement learning and imitation learning [3], [4], [5].

Meta-Learning, aka Learning to Learn is the process of automatically acquiring the representation and the algorithms to perform given tasks. As authors of "The Data Mining and Knowledge Discovery Handbook" quote, "Meta-learning has as one of its primary goals as the understanding of the interaction between the mechanism of learning and the concrete contexts in which that mechanism is applicable "[6]. There have been several contributions made in meta-learning in recent years in the form of Neural Turing Machine (NTM), Model Agnostic Meta Learning (MAML), Neural GPU, Multi-Level Discovery of Deep Options (DDO), Neural Programmer-Interpreter (NPI) etc. [7], [8], [9], [10].

Neural Programmer-Interpreter (NPI) are class of meta-learning models that learn to represent a task as a collection of lower level programs and compose these low-level programs into high-level programs. NPI consists of three components

- a task-independent recurrent core, which given some representation of current state of problem, produces the next program to be called conditioned over, previous program function called. This is the most important part of NPI and makes NPI capable of learning Hierarchical tasks.
- a key-value program memory, which is a learn-able program embedding, which is used to represent a program function call efficiently. It also facilitates next program prediction by predicting in embedding space and calling the program most similar to predicted vector.
- domain specific encoder, which provide the generalization capabilities to NPI across various tasks by giving each problem an efficient representation, which can be used directly by recurrent core.

NPI has been shown to perform well in tasks such as Learning grade school addition, sorting, canonicalizing 3D models with strong generalization (generalization beyond the training scenario, on unseen settings). A further addition of recursion to NPI model made it one of the most promising models for learning real-life tasks [11]. One of the most interesting aspect of NPI is being able to predict next action based on its current state conditioned on the past actions taken. There are some limitations such as the arguments are assumed to be discrete (given as a tuple of integers). Also, the existing research doesn't talk about the variations of real-life tasks, their richness and also how NPI can be used as it is to model those tasks. There have been attempts to Learn from Demonstrations and generalize to unseen cases in a few examples, particularly the One-Shot Imitation Learning and Neural Task Programming research [12], [13]. However, the aim of both of these research works is to perform "few-shot learning from demonstration". In one-shot imitation learning framework, a large number of unique task specifications are considered. A neural network is trained on pairs of

*Extended Abstract

[1]Himadri Mishra and K.K.Shukla are with Department of Computer Science and Engineering, IIT BHU, Varanasi, India (email: himadri.mishra.cse13@iitbhu.ac.in; kkshukla.cse@iitbhu.ac.in)

demonstrations; it is presented with first demonstration and a state from second demonstration and it has to predict the correct action for that state. At test time, a demonstration (for any seen/novel instance of problem) is provided, and based on it, the model has to predict correct action for any state of new instance of the same task, i.e. one-shot imitation. Similarly, Neural Task Programming (NTP) learns to generate a policy (here a neural program) from single demonstration and use it to perform any new instance of the same task, i.e., one-shot imitation.

Summary of our contributions: We first build upon the past research and note the different classes of problems (based on available data/demonstrations) that one may encounter in real-life. Then, we present a direct solution to one of the classes using NPI and propose a solution to another class with modified NPI. Handling the remaining class is kept for future work.

II. CLASSIFICATION OF REAL-LIFE PROBLEMS BASED ON AVAILABLE SUPERVISION

Real-life tasks can be mostly represented as a sequence of hierarchical commands. Be it some numerical time series (position data from the problem), a sequence of images (for demonstration) or rough description (high-level description), all of them can be clubbed together to represent some primitive function or non-primitive function which itself is a collection of multiple primitive functions. Based on the availability of sequence of states and corresponding function/primitive, the tasks can be classified into following types:

- Full execution traces - These are the class of problems for which we have full state trajectory available along with the corresponding trace labels. These can be easily modeled using NPI as is. Examples of such problems are Block Stacking or any other task whose algorithm exists or is well known.
- State trajectory without traces - These are the class of problems where we only have demonstration of the task in the form of state trajectories, such as a sequence of images and corresponding action taken. Sometimes, with some small hacks, these class of problems can be converted into full execution traces problem. Example of such problems includes driving a car autonomously. Researchers are still looking for solutions to this class of problems.
- Partial execution traces - These are the class of problems where state trajectory is hard to collect, and all we have is a high level and rough description of the task. Most of the real-life tasks which haven't been tackled or which are hard to be demonstrated, come in this category. There hasn't been much progress in research for these class of problems.

We now show an application of NPI to the first class of problem and also propose a solution (active work, without verification) for the second class of problems. The third class of problems is still kept as a future research direction.

III. BLOCK STACKING TASK

Block stacking problem is one of the common problems to tackle for meta-learning and imitation learning [12], [13], [14]. Unlike the established belief that NPI can't generalize to unseen task objective, we demonstrate that with concrete representation, NPI can easily generalize to a variable number of blocks as well as unseen tasks in block stacking. The target state is given as a string on a scratch pad (for example "abc de fgh"). We define primitives such as GOTO, which moves the arm to specified location; GRIPPER, which controls the arms grasping and PTR, which controls the pointer motion as well as non-primitives such as MAKE_STACK, which builds each of the stacks; PUT, which puts a block onto the current stack.

We train NPI with different number of examples (5000, 10000) and test it on four different verification settings (referred as VSets in the table): (i) Same traces as that of training data, (ii) Traces with the same number of blocks, same target state, but varying initial configuration, (iii) Traces with the same number of blocks, different target state, (iv) Traces with a varying number of blocks (both less than and greater than that trained on).

We refer each of the above sets as Verification Sets numbered 1-4 respectively. The results are summarized in table 1. We see that NPI can achieve near perfect generalization accuracy in block stacking task across varying initial configuration of blocks and final target configuration.

IV. CONCLUSION AND FUTURE WORK

In this work, we define different types of real-life problems based on available supervision and how Neural Programmer Interpreters can be used as is to model any problem where full execution traces are available. NPI is a class of strong models that can be efficiently used to model real-life tasks with enormous generalization capability (beyond seen tasks and problem size). We see that NPI shows strong generalization with initial configuration and variation in the task and also gives promising results in an unseen number of blocks.

For future work, we consider the problem, where the state trajectories (demonstrations) are available as a sequence of images. We can use weak logic to convert the set of demonstrations along with actions taken and convert it to a sequence of hierarchical program structure. We can then train a fine-tuned, pre-trained, behavior cloning model for the set of demonstrations and use this fine-tuned network as a task-specific encoder. Above task-specific encoder can be coupled with NPI to predict the action taken. This is a work in progress. We also aim to target real-life problems with partial execution traces available using some modified form of NPI itself.

TABLE I
BLOCK STACKING ACCURACY USING NPI (%)

# of train traces	VSet 1	VSet 2	VSet 3	VSet 4
5000	100	100	100	21.6
10000	100	100	100	36.3

REFERENCES

[1] R. Kentridge, "Neural networks for learning in the real world: representation, reinforcement and dynamics," *Parallel Computing*, vol. 14, no. 3, pp. 405 – 414, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016781919090089R

[2] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba, "Learning with hierarchical-deep models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1958–1971, Aug 2013.

[3] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, "Multi-level discovery of deep options," *CoRR*, vol. abs/1703.08294, 2017. [Online]. Available: http://arxiv.org/abs/1703.08294

[4] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, "DDCO: Discovery of deep continuous options for robot learning from demonstrations," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 418–437. [Online]. Available: http://proceedings.mlr.press/v78/krishnan17a.html

[5] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical multiscale recurrent neural networks," *CoRR*, vol. abs/1609.01704, 2016. [Online]. Available: http://arxiv.org/abs/1609.01704

[6] O. Maimon and L. Rokach, *The Data Mining and Knowledge Discovery Handbook*, 01 2005, vol. 1.

[7] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014. [Online]. Available: http://arxiv.org/abs/1410.5401

[8] L. Kaiser and I. Sutskever, "Neural gpus learn algorithms," *CoRR*, vol. abs/1511.08228, 2015. [Online]. Available: http://arxiv.org/abs/1511.08228

[9] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *CoRR*, vol. abs/1703.03400, 2017. [Online]. Available: http://arxiv.org/abs/1703.03400

[10] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *CoRR*, vol. abs/1710.09767, 2017. [Online]. Available: http://arxiv.org/abs/1710.09767

[11] J. Cai, R. Shin, and D. Song, "Making neural programming architectures generalize via recursion," *CoRR*, vol. abs/1704.06611, 2017. [Online]. Available: http://arxiv.org/abs/1704.06611

[12] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," *CoRR*, vol. abs/1710.01813, 2017. [Online]. Available: http://arxiv.org/abs/1710.01813

[13] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," *CoRR*, vol. abs/1703.07326, 2017. [Online]. Available: http://arxiv.org/abs/1703.07326

[14] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," *CoRR*, vol. abs/1709.10089, 2017. [Online]. Available: http://arxiv.org/abs/1709.10089