Gradual Program Induction

Josef Strunc, Joseph R. Davidson, and Sungmin Aum
GoodAI
Prague, Czech Republic
{josef.strunc, joseph.davidson, sungmin.aum}@goodai.com

Abstract

We introduce Operation and Memory Augmented Neural Networks. A work in progress model which can utilise both externally provided operations and external differentiable persistent memory, to perform program induction. The model will be able to gradually add its own learned compositions to the repertoire of available operations, allowing it to construct programs of increasing complexity as it develops an understanding of the task domain. This gradual learning allows it to exploit knowledge in both the externally provided operations, and the solutions to previous tasks and hence reduce the amount of necessary training data and learning time compared to contemporary models.

1 Introduction

There is presently considerable interest in neural network based models which can learn to induce programs that solve some given task. This approach is different from non-neural network based approaches to program induction [11, 14] mainly due to the differentiable memory and controller, which allows gradient-based optimization method to search for the appropriate program.

Memory Augmented Neural Networks [5, 9, 2] resemble traditional computer architectures with a neural network controller playing the role of the CPU and a separate memory structure serving as working memory. This class of models often makes use of differentiable external memories, which allow the model to be end-to-end trainable with standard gradient descent methods. This memory augmentation transforms a traditional neural network into a kind of trainable computer, allowing it to represent and reason about more complex structures [6]. However the state of the art in this area is strongly influenced by the current trend of deep learning - especially with respect to the high volume of data, and long training times required to perform acceptably.

This need for large amounts of data stems from the requirement that the model must learn all of the actions that it should perform from scratch. The CPU controller cannot perform any meaningful action from the outset. Unlike these purely data-driven approaches, we propose the addition of yet another external data source, namely the ability to exploit externally supplied operations of any functionality. How each operation can be used is subsequently learned from data (i.e. presented tasks to be solved), and can be composed with other operations to form a solution.

Furthermore, we propose that the model is able to add to the set of operations, effectively permanently remembering learned solutions to tasks. This approach on an inherently compositional task such as writing programs can conceivably increase data efficiency, and decrease training time.

2 Operation and Memory Augmented Neural Networks

An Operation and Memory Augmented Neural Network (OMANN) is a learning model that combines a neural network controller with both an external differentiable memory store, and a number of external operations which can be called by the controller to manipulate data in memory. When a model finds a beneficial composition of these operations, it saves the composition as another operation, creating a function. As more complex tasks are introduced to the OMANN, the reuse of functions from its library helps prevent the need to relearn previous helpful solutions and thus reduce the search space.

With differentiable memory, the model can be weakly supervised using $\langle input, output \rangle$ pairs as in the neural programmer [10]. This is a necessary feature for the system, so that it is able to induce programs on its own, and allows us to train the model on tasks for which the end solution is known but the path to it is not. Additionally, external persistent memory has been shown to provide the traditional neural network with the ability to memorize specific pieces of information for an extended period of time [5, 6]. This helps the model generalize much better on previously unseen data, and learn to work with variables.

Making use of supplied operations places the burden of learning on finding the required sequence of operations, rather than learning to perform said operations, similar to how Neural Module Networks and Neural Programmer Interpreters [1, 12] operate. The ability to induce programs composed of external operations allows for efficient knowledge reuse and the possibility to supply the system with specialized solutions to sub-problems that are tested and guaranteed to work. The system does not need to learn every, possibly trivial, domain- specific operation from scratch, resulting in improved training efficiency.

The OMANN model is naturally paired with the idea of curriculum based learning [7, 3, 13, 16]. A well constructed curriculum will afford an instance of the model the opportunity to create its library of composed functions, which will grow in complexity as the tasks become more challenging. While it will be technically feasible for an instance of the model to induce complex programs from the start, the search space will be too large for it to effectively find a solution.

3 Current Progress

As it currently stands, our implementation resembles that of the Neural Programmer [10] which supports external operations and memory, can learn multiple programs using one LSTM controller which reads the programs as its input data, but requires strongly supervised learning. We use a number of differentiable stacks [15, 4, 9] and a single controller that learns one particular program which is represented in its weights and which operates on top of the stacks. This model is able to compose externally supplied stack operations (push, pop, addition, subtraction, comparison) to perform simple arithmetic and list transformation tasks such as sorting and reversal, when trained just with input/output pairs.

4 Future Work

Current development of our work immediately focuses on incremental addition of new controllers, which represent newly learned operations. Along with a 'key', which is fixed, all operations will be given a modifiable 'context' embedding generated from its input/output history. In order to retain as much information as possible from previous episodes of learning process, each controller weights are 'frozen' when it first meets stopping criteria for training. A 'patching' network, which is much smaller than the controller, will be updated to adjust changes in contexts, or to be used in conjunction with an existing controller to form a new controller.

References

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799, 2015.

[2] J. Ba, G. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using Fast Weights to Attend to the Recent Past. *ArXiv e-prints*, October 2016.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.

[4] Sreerupa Das, C. Lee Giles, and Guo zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *CONFERENCE OF THE COGNITIVE SCIENCE SOCIETY*, pages 791–795. Morgan Kaufmann Publishers, 1992.

[5] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[6] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio G. Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià P. Badia, Karl M. Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, advance online publication, October 2016.

[7] Çaglar Gülçehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *CoRR*, abs/1301.4083, 2013.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[9] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *CoRR*, abs/1503.01007, 2015.

[10] Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834, 2015.

[11] Roland Olsson. *Inductive Functional Programming Using Incremental Program Transformation*. PhD thesis, University of Oslo, 1994.

[12] Scott E. Reed and Nando de Freitas. Neural programmer-interpreters. *CoRR*, abs/1511.06279, 2015.

[13] Marek Rosa and Jan Feyereisl. Consolidating the search for general ai. 2016.

[14] Jürgen Schmidhuber, Christophe Giraud-carrier, Ricardo Vilalta, and Pavel Brazdil. Optimal ordered problem solver, 2004.

[15] G. Z. Sun, C. L. Giles, H. H. Chen, and Y. C. Lee. The neural network pushdown automaton: Model, stack and learning simulations. Technical report, 1993.

[16] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, and Chris Dyer. Learning the curriculum with bayesian optimization for task-specific word representation learning. *CoRR*, abs/1605.03852, 2016.