Evolving Robots: Robots Adapting in a Real-World Environment

Daniel Palacios, George K. Thiruvathukal
Loyola University of Chicago
dpalacios2@luc.edu

Abstract

Neuroevolutionary methods have proved to be an influential change for solving challenging optimization problems. However, the power of evolutionary algorithms is yet to be explored by other researchers. The deep learning community looks down on evolutionary algorithms because of the expensive computational time complexity that it takes to find a solution. However, some deep learning models exceed human design capability which makes it a compelling domain for evolutionary techniques. Using methods to evolve an overall topology and routings between modules, this paper applies this approach for live-object detection towards autonomous driving. This evaluation demonstrates how evolution can help further solve complicated computer vision architectures as well as advancing complex systems and designs of deep neural networks in general.

Keywords

Multi-Task Learning (MTL), Neuroevolution of Augmenting Topologies (NEAT), Genetic Algorithms (GA), CoDeepNeat, Computer Vision, Soft Layer Ordering

1 Introduction

In Multitask learning (MTL), we can have a Neural Network simultaneously train on several different tasks at once.[1] For example, given an image, we can recognize the scenery and develop a verbal caption for it. MTL's ability to learn several tasks at once can drastically improve performance in each of the tasks. How we align layers that are shared with different tasks depends on the architecture. Soft layer ordering learns how shared layers are applied in different ways for different tasks and shows that deep MTL can be improved towards our understanding of complex real-world processes. [2] Much of the research in deep learning for camera-based autonomous vehicles revolves around creating a large and complex network architecture. However, the size and complexity of this problem can and does exceed human design and ability which makes it a compelling domain for evolutionary methods for optimization. This paper evaluates an automated, flexible approach for evolving architectures. i.e hyperparameters, modules, and module routings of deep multitask networks. Previous work uses the soft ordering method [3] as a starting point and extends it with CoDeepNEAT where both modules and task routings are coevolved (CMTR).[4] This extension demonstrates that evolutionary techniques and MTL can make a large difference in performance and tasks. When both ideas are combined, it then creates a particularly powerful approach and can solve complicated real-world issues. This paper explores this method to find synergies in different tasks in vision. Since related work has shown that tasks do not have to be closely related to gain benefit in MTL [5], we want to see if it is possible to express the contents of an image in words to help object recognition. This approach is evaluated with a simplified autonomous vehicle using the CIFAR-10 dataset. We observe if a camera-based autonomous vehicle can respond to live object-detection with the evolution of modules and topologies. The results will demonstrate that since live object detection is a crucial aspect for autonomous driving, using evolutionary methods to develop powerful architectures can expand artificial intelligence towards autonomous software. It is hypothesized that if computer vision is applied and expanded with evolutionary algorithms, then this will allow autonomous robots to react and adapt to any given scenario. The rest of the paper is organized as follows: In Section 2, previous work

on CMTR and other neural architectures are summarized. In Section 3, the key contributions of this paper are extended with live object detection and methods of how genetic algorithms (GA) can evolve in real-time [6,7]. Finally, in Section 4 and Section 5 experimental results on a live autonomous vehicle are presented and analyzed.

2 Related Work

Before applying methods to find synergies in vision in Section 3, this section reviews methods and relationships for evolving the modules and overall topology of the soft ordering architecture.

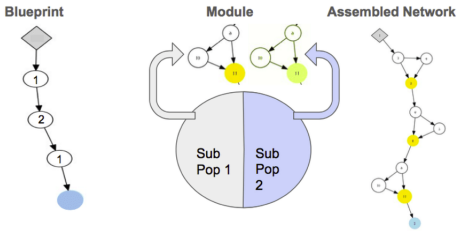


Figure 1: A visualization of how CoDeepNEAT assembles networks for fitness evaluation. Modules and Blueprints are assembled into a network through the replacement of blueprint nodes with corresponding modules. This approach allows evolving repetitive and deep structures seen in many recent DNNs

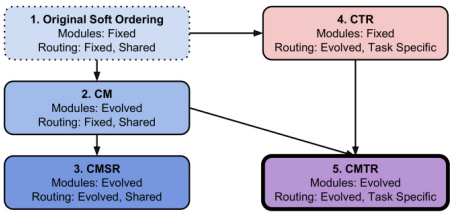


Figure 2: Relation between the 5 methods as the original soft ordering architecture as the starting point. This architecture is then extended with CoDeepNEAT on the left and task-specific routing on the right. CMTR on the bottom right is then the combination of the two main ideas.

2.1 Evolutionary Architecture Search

MTL has led to many successes in deep learning research as it has been applied and improve overall performance to many machine learning applications such as Vision [], Natural Language Processing [], and Reinforcement Learning []. Generally, When optimizing more than one loss function we are effectively application a multi-task learning application. In other words, if multiple tasks are related, the optimal model for those tasks are related as well. The key design for MTL is deciding how parameters (Convolutional layers and weight matrices) are shared across tasks. The most recent approach uses soft layer ordering which avoids alignment by allowing shared layers to be used across different tasks. [2]. As this method has shown to improve deep MTL, one particular area of research is the use of evolutionary algorithms (EA) for architecture and hyperparameter search. This approach works well for these types of problems because EAs can rapidly be applied with no gradient information. The original soft layer ordering uses a fixed architecture for the modules and a fixed routine (i.e topology) that is shared among tasks. The most recent work that takes an evolutionary approach to deep MTL uses soft ordering and demonstrates relationships between five methods that are extended by CoDeepNEAT as shown in figure 1 and figure 2. CoDeepNEAT is extended by coevolving the module architectures (CM) and by coevolving both the module architectures and a single shared routing for all tasks using (CMSR). This relationship also introduces to keep the module architecture fixed but evolves separate routing for each task during training (CTR). Finally, both CM and CTR are combined into the coevolution of modules and task routing (CMTR) where both modules and task routing are coevolved. Narrowing the focus to CMTR, experiments have shown to result in the discovery of a wide variety of complex modules thus concluding that evolution is capable to discover useful architecture hyperparameters that are diverse in structure. The power of CMTR us from evolving different topologies for different tasks and it is crucial for its performance.

2.2 Coevolution of Modules

In the coevolution of modules, CoDeepNEAT is used to search for a promising architecture which is then inserted into an appropriate position to create and enhance the soft ordering network. [4] The process goes as follows:

1. CoDeepNEAT initializes a population of modules (MP). The Blueprints are not used
2. Modules are randomly chosen from each species in MP, grouped into sets (M) and are assembled into enhanced soft ordering networks.
3. Each assembled network is trained/evaluated on some task and its performance is returned as fitness.
4. Fitness is attributed to the modules, and NEAT evolutionary operators are applied to evolve the modules
5. The process is repeated from step 1 until CoDeepNEAT terminates, i.e. no further progress is observed for a given number of generations.

Originally, soft orderings architecture has a fixed number of modules and depth. When CoDeepNEAT is applied, these attributes are evolved as global hyperparameters. However, since the routing layout is still fixed, the blueprint population of CoDeepNEAT is not used. Thus, the key operation of CoDeepNEAT is inserting each module into each node of the blueprint is then skipped; only the module population are coevolved. One the network is assembled, the fitness is assigned to each of the modules in the assembled networks and is averaged into the module fitness. Once the evaluation is complete, the standard NEAT [] mutation, crossover, and speciation operators are applied to create the next generation of the module population.

2.3 Coevolution of Task Routing

In the Covevolution of task routing (CTR), there is n number of modules who share their weights everywhere across all tasks (similar to soft ordering).

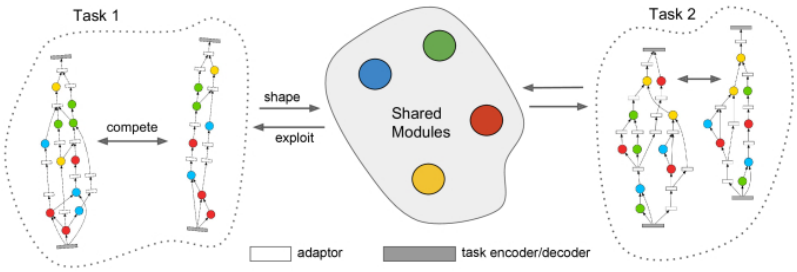


Figure 3: An instance of CTR with three tasks and four modules that are shared across all tasks. As each individual is assembled in different ways through gradient-based training, they exploit the shared resources to compete within a task and develop mutualistic relationships with other tasks with their use of the shared modules.

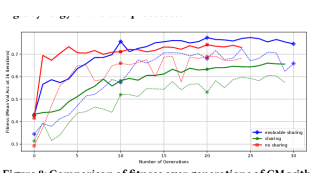
And like in CoDeepNEAT's blueprint evolution, CTR also searches for the best ways to assemble its modules into a complete network. However, unlike the blueprint, CTR urges to search for a distinct module routing scheme for each task. Once found, it is then trained on a single set of modules throughout evolution, and for good reason. As modules are trained within different locations, their functionality should become increasingly general and thus becoming easier for it to adapt to the needs of a new location. This means that CTR has no additional need to perform backpropagation over a training single fixed-topology multitask model. Figure 3 will demonstrate an instance of CTR and how tasks and modules are shared across all tasks.

2.4 Coevolution of Modules and Task Routing

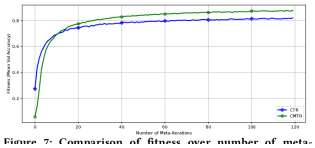
As both methods, CM and CTR have shown to improve performance for soft ordering - combining both ideas forms an even more powerful algorithm named coevolution of modules and task routings. This approach can overcome weaknesses in both CM and CTR: CM's inability to create a customized routine for each task and CTR's inability to search for better module architectures.

Here is an overview of how this evolutionary loop works that were used in [4]:

1. CoDeepNEAT initializes a population of modules MP. The blueprints are not used.
2. Modules are randomly chosen from each species in MP and grouped into sets of modules (Mk).
3. Each set of modules is given to CTR, which assembles the modules by evolving task-specific routings. The performance of the evolved routines on a task is returned as fitness.
4. Fitness is attributed to the modules, and NEATs evolutionary operators applied to evolve the modules
5. The process repeats from step 1 until CoDeepNEAT terminates, i.e. no improvement for a given number of generations.



(a) Comparison of fitness for 3000 iterations



(b) Comparisons of fitness over number of meta iterations

Algorithm	Val Accuracy (%)	Test Accuracy (%)
1. Single Task [29]	63.59 (0.53)	60.81 (0.50)
2. Soft Ordering [29]	67.67 (0.74)	66.59 (0.71)
3. CM	80.38 (0.36)	81.33 (0.27)
4. CMSR	83.69 (0.21)	83.82 (0.18)
5. CTR	82.48 (0.21)	82.36 (0.19)
6. CMTR	**88.20** (1.02)	**87.82** (1.02)

Table 1: Average validation and test accuracy over 20 tasks for each algorithm. CMTR performs the best as it combines

Figure 4: Average validation and test accuracy over 20 tasks for each algorithim. CMTR performs the best as it combines both module and routing evolution

The difference between CM and CMTR os that each of its module's final convolutional layer has additional hyperparameters to evolve such as kernel size, activation function, and dropout rate. Past experiments [4] have shown that CMTR requires more complex final layers thus evolving the complexity of the final convolutional layer. In Figures 4, 5, and 6 Jason Liang, Elliot Meyerson, and Risto Miikkulainen conducted and demonstrated experimental results that used the Omniglot dataset which consisted of 50 alphabets of handwritten characters. To our knowledge, this was the first paper that took an evolutionary approach to deep MTL. As the experiments show that CMTR performs the best when it is combined with both module and task routing evolution.