Learning to Optimize Memory Allocation on Hardware using Reinforcement Learning

- Published on May 24, 2021

Somdeb Majumdar

Head of Intel AI Lab (US)

Somdeb Majumdar *heads the AI Lab (U.S.) at* ***Intel Labs****. Here he explores a range of topics spanning fundamental representational properties of neural networks, using AI to automate the design of large, complex systems.*

Highlights:

- Deep reinforcement learning (RL) has advanced at a frenetic pace in the last few years and has evolved from game-play benchmarks to solving massive-scale real-world optimization problems.
- We describe a scalable framework that combines Deep RL with genetic algorithms to search in extremely large combinatorial spaces to solve a critical memory allocation problem in hardware.

Reinforcement Learning is an area of machine learning that has seen a massive resurgence in recent years. The combination of RL with the powerful representational properties of deep neural networks (DNNs) has made it possible to solve large-scale problems that were previously considered intractable.

At Intel AI Lab, we have been studying the role of Deep RL in solving real-world problems with real constraints. The most common formulation of Deep RL is when an ***agent*** has access to at least partial observations of its environment at each time step (the state ***s***) and can execute an action (***a)*** in that environment according to a ***policy***. It also gets feedback, or a reward ***(r),*** based on the current state and the action taken. The agent's objective is to take a series of actions to maximize the cumulative sum of rewards over all time steps in that episode. This formulation is shown in Figure 1, where the agent is parameterized as a DNN.

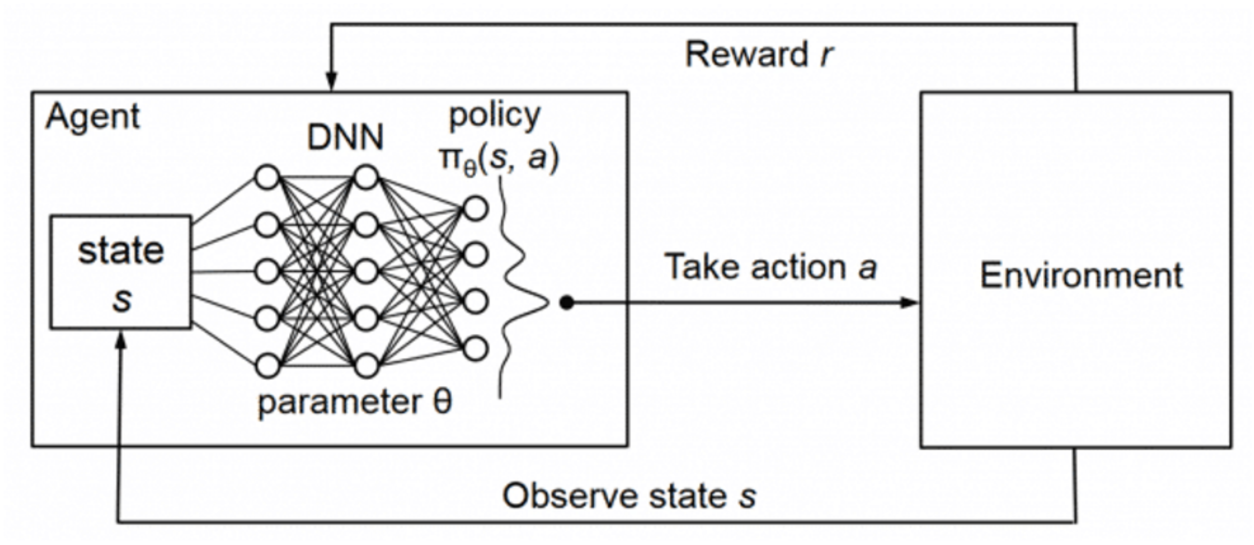

Figure 1: The Deep RL setting (**Mao et al**)

The vast majority of Deep RL training methods utilize this relatively simple and intuitive setting. At Intel AI Lab, we have been investigating real-world settings where this basic assumption does not entirely hold true. For example, a reward may be obtained very rarely, or there may only be a single-step process instead of a multi-step episode making it a noisy learning signal. Such problems are also difficult to scale due to massive state- and action-spaces.

Under such settings, standard RL methods don't work very well. In a paper published at the 2019 International Conference on Machine Learning (ICML), titled "Collaborative Evolutionary Reinforcement Learning" (**CERL**), we showed that combining Deep RL with evolutionary search allowed us to scale to large action spaces and sparse reward settings with minimal hyper-parameter tuning or reward engineering. We were able to accomplish this and still achieve near state-of-the-art performance on several benchmarks. At the 2020 ICML conference, in our paper "Evolutionary Reinforcement Learning for Sample-Efficient Multiagent Coordination" (**MERL**), we demonstrated state-of-the-art results on several multiagent coordination settings.

Figure 2 shows the basic idea. A population of ***learners***, codified as DNNs, train using fast policy gradients. Each learner represents a partial and suboptimal solution to the problem. It's suboptimal because they train on sparse reward signals obtained only once at the end of an episode. Another population of neural network policies trains completely gradient-free using an evolutionary algorithm (EA). EAs are known to converge to the optimal solution given infinite time. To speed up the EA, we inject the partial solutions from the learner population into the EA population, essentially biasing the EA process to search around them. We show that this significantly accelerates EA search and that overall, this process requires a smaller number of samples to reach state-of-the-art performance even when we accumulate all samples consumed by all members of both populations.

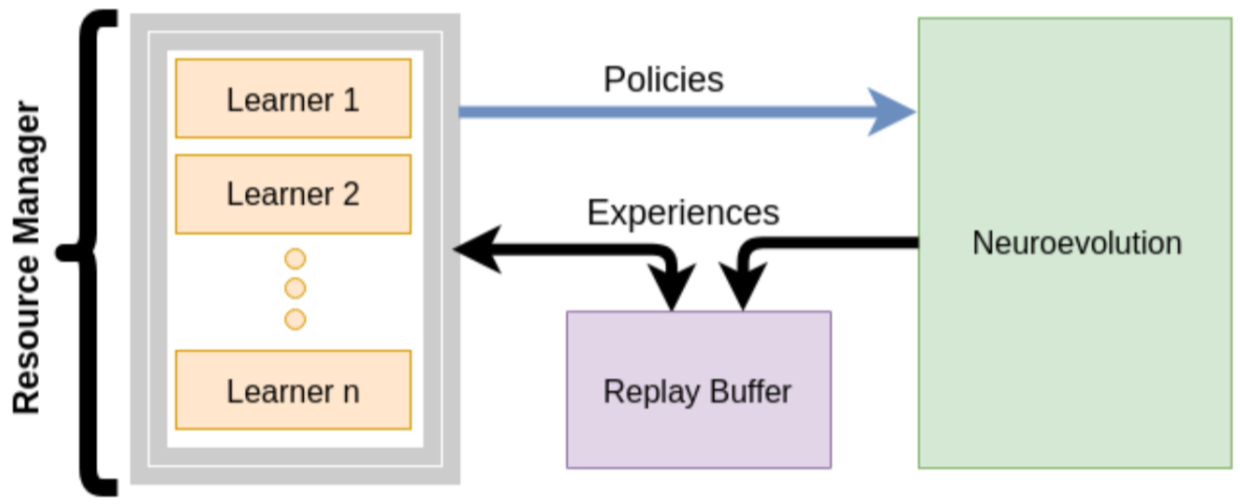

Figure 2: Collaborative Evolutionary Reinforcement Learning (CERL)

In our latest paper that was presented at the 2021 International Conference on Learning Representatives (**ICLR**), titled "Optimizing Memory Placement using Evolutionary Graph Reinforcement Learning" (**EGRL**), we extend this idea. We look at the problem of optimally mapping and executing portions of a workload on a hardware target to various onboard memory components. We call this the memory placement problem.

The hardware we investigated is the Intel® Neural Network Processor – Inference (Intel® NNP-I), a custom accelerator that executes DNN inference. NNP-I has a typical memory hierarchy, a fast SRAM that sits close to the compute cores but is only 4MB, a 10x slower shared lower-level cache (LLC) core, which is 24MB, and a 100x slower DRAM, which sits farthest from the compute cores but has a much larger capacity at 32GB.

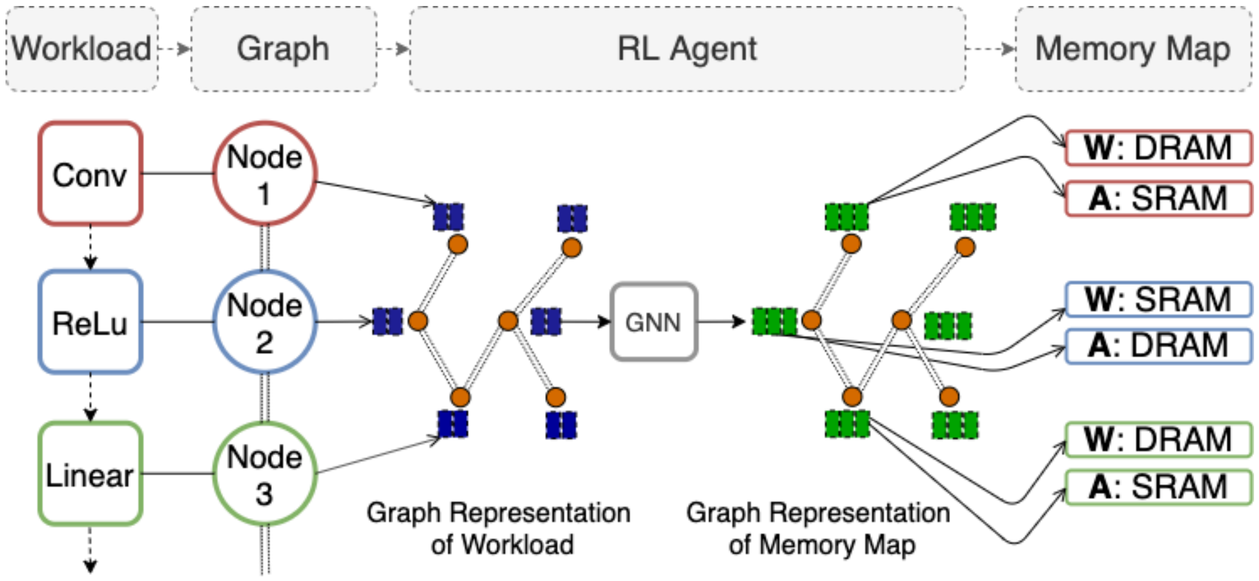

Figure 3: Evolutionary Graph Reinforcement Learning (EGRL)

The workload executed on this hardware is a feed-forward neural network, as shown in Figure 3. We need to map the weight and activation tensors to one of these three memory components for each layer of this network. Since not all of them can be mapped to the tiny but fast SRAM, the optimization problem is to learn the trade-off between memory capacity and memory bandwidth in a way that maximizes the overall throughput.

This kind of scheduling problem is usually solved using heuristics. We explored the idea of training an RL agent to solve this problem in a scalable manner. Since the input is a computational graph, we found that a graph neural network (GNN) formulation naturally fit the problem. Here, each layer of the input workload is codified as a node in the input graph. The GNN agent is trained exactly as in CERL. The state observations comprise the hyper-parameters of each layer, and the action space per layer is made up of the possible memory allocations for its weights and activations.

The primary challenge for the agent is to simultaneously map the weights and activations of all layers in the input workload. For example, the largest workload we tried was BERT with 376 layers. This number represents a total combinatorial complexity of 3^(2*376) since we take one of three decisions per tensor per layer, roughly equating to 10^358.

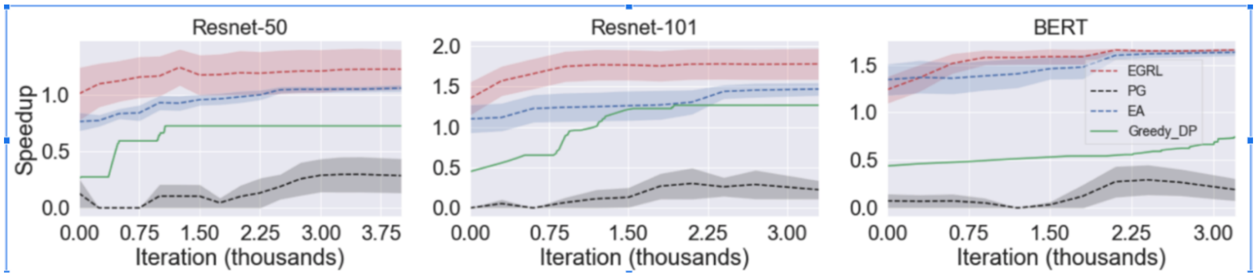

Figure 4: Performance of EGRL vs. baselines on three large workloads

The entire training pipeline happens on hardware with a direct measurement of throughput serving as a reward for a given mapping. We find that EGRL outperformed the baselines, achieving a 78% speedup over the native compiler on BERT, the largest network we tested. We also found that mapping solutions learned on one workload remained highly performant on a previously unseen workload. Figure 4 shows these results. We invite you to read the paper for more extensive analyses.

At Intel AI Lab, we continue to push this idea to even larger combinatorics like chip design and molecular discovery for material science. As this and **other recent works** have shown, we are entering the age of using Deep RL for massive-scale problems relevant to the real world.